The 2009 AP® Computer Science A Released Exam

Contains:

- Multiple-Choice Questions, Answer Key, and Diagnostic Guide
- Free-Response Questions with:
  - Scoring Guidelines
  - Sample Student Responses
  - Scoring Commentary
- Statistical Information About Student Performance on the 2009 Exam

;hapter I: The AP® Process

; chapter will give you a brief overview of the
:lopment and scoring processes for the AP Computer
nce A Exam. You can find more detailed information
P Central® (apcentral.collegeboard.com).

ıat Is the Purpose of the ' Computer Science A Exam?

AP Computer Science A Exam is designed to allow
lents to demonstrate that their knowledge, under-
ding, and skills are equivalent to those gained by
lents who have successfully completed an introductory
:ge-level course in computer science. The AP Computer
nce A Exam includes material that is covered in a
-semester computer science course in which the Java
gramming language is used to illustrate an object-
nted approach to problem solving and algorithm
:lopment.

he multiple-choice section of the exam is designed to
:r the breadth of the curriculum in terms of knowledge,
ıciples, and conceptual understanding. The free-
'onse section requires students to solve problems
)lving more extended reasoning. In both sections,
lents must demonstrate their ability to design, write,
iyze, and document programs and subprograms.
ılifying scores on the AP Computer Science A Exam
' allow students to begin their college careers with credit
ın introductory computer science course and/or
ribution credit for a computer science course.

Who Develops the Exam?

The AP Computer Science Development Committee, working with Assessment Specialists at ETS, develops the exam. This committee is appointed by the College Board and is composed of six teachers from secondary schools, colleges, and universities in the United States. The members provide different perspectives: high school teachers offer valuable advice regarding realistic expectations when matters of content coverage, skills required, and clarity of phrasing are addressed. College and university faculty members ensure that the questions are at the appropriate level of difficulty for students planning to continue their studies at colleges and universities. Committee members typically serve one to four years.

Also aiding in the exam development process is the Chief Reader, the college computer science professor responsible for supervising the scoring of the free-response questions at the AP Reading. The Chief Reader attends every committee meeting to ensure that the free-response questions selected for the exam can be scored reliably. The expertise of the Chief Reader and the committee members who have scored exams in past years is notable: they bring to bear their valuable experience from past AP Readings and suggest changes to improve the quality and the performance of the questions. In addition, ETS computer science Assessment Specialists offer their guidance and advice.

How Is the Exam Developed?

The Development Committee sets the exam specifications, determining what will be tested and how it will be tested. It also determines the appropriate level of difficulty for the exam, based on its understanding of the level of competence required for success in introductory computer science courses in colleges and universities. Each AP Computer Science A Exam is the result of several stages of development that together span two or more years.

Section I—Multiple Choice

1. Development Committee members and other faculty write and submit multiple-choice questions directed to the major topics outlined in the *AP Computer Science A Course Description*.
2. ETS Assessment Specialists perform preliminary reviews to ensure that the multiple-choice questions are worded clearly and concisely.

3. At the committee meetings, which are held two or three times a year, the committee members review, revise, and approve the draft questions for use on future exams. They ensure that the questions are clear and unambiguous, that each question has only one correct answer, and that the difficulty level of the questions is appropriate.
4. Most of the approved draft questions are pretested in college classes to obtain some estimate of the questions' level of difficulty.
5. From the pool of approved questions, ETS Assessment Specialists select an appropriate mix of materials for the multiple-choice section of an exam, making sure that the questions are distributed across the content areas as specified by the Development Committee in the *AP Computer Science A Course Description*.
6. The committee thoroughly reviews the draft exam in various stages of its development, revising the individual questions and the mix of questions until it is satisfied with the result.

The committee controls the difficulty level of the multiple-choice section by selecting a wide range of questions, a subset of which has been used in an earlier form of the exam.

Section II—Free Response

1. Well in advance of the exam administration, the members of the Development Committee write free-response questions for the exam. These are assembled into a free-response question pool.
2. From this pool, the committee selects an appropriate combination of questions for a particular exam. It reviews and revises these questions at all stages of development of that exam to ensure that they are of the highest possible quality. The committee considers, for example, whether the questions will offer an appropriate level of difficulty and whether they will elicit answers that will allow Readers, the high school and college computer science teachers who score the free-response questions, to discriminate among the responses along the scoring guidelines used for the different questions. An ideal question enables the stronger students to demonstrate their accomplishments while revealing the limitations of less proficient students.

Question Types

The 2009 AP Computer Science A Exam contains a 75-minute multiple-choice section consisting of 40 questions and a 105-minute free-response section consisting of 4 questions. The two sections are designed to complement each other and to measure a wide range of skills.

Multiple-choice questions are useful for measuring a student's level of competence in a variety of contexts. In addition, they have three other strengths:

1. They are highly reliable. Reliability, or the likelihood tha students of similar ability levels taking a different form o the exam will receive the same scores, is controlled mor effectively with multiple-choice questions than with free-response questions.
2. They allow the Development Committee to include a selection of questions at various levels of difficulty, there ensuring that the measurement of differences in student achievement is optimized. For AP Exams, the most important distinctions are between students earning scores of 2 and 3 and those earning scores of 3 and 4. These distinctions are usually best accomplished by usir many questions of middle difficulty.
3. They allow comparison of the ability level of the current students with those from another year. A number of questions from an earlier exam are included in the curr one, allowing comparisons to be made between the scor of the earlier group of students and those of the current group. This information, along with other data, is used t establish AP scores that reflect the competence demand by the Advanced Placement Program® and that can be legitimately compared with scores from earlier years.

Free-response questions on the AP Computer Science A Exam require students to use their analytical and organizational skills to reason about and write program fragments that fit the specifications given. They allow students to demonstrate their understanding of program structure and their ability to translate that understanding into a concrete solution. The free-response format allows for the presentation of uncommon yet correct responses and permits students to demonstrate their mastery of computer science by a show of creativity.

The free-response and multiple-choice sections are designed to complement each other and to meet the overal course objectives and exam specifications. After each exam administration, the questions in each section are analyzed both individually and collectively, and the findings are use to improve the following year's exam.

Scoring the Exam

Who Scores the AP Computer Science A Exam?

The multiple-choice answer sheets are machine scored. The teachers who score the free-response section of the AP Computer Science A Exam are known as Readers. The majority of these Readers are experienced faculty membe

ɔ teach either a high school AP Computer Science A
.rse or an equivalent course at a college or university.
:at care is taken to obtain a broad and balanced group of
ıders. Among the factors considered before appointing
ıeone to the role are school locale and setting (urban,
al, and so on), as well as the potential Reader's gender,
nicity, and years of teaching experience. University and
h school computer science teachers who are interested
ıpplying to be a Reader at a future AP Reading can
ıplete and submit an online application via AP Central
central.collegeboard.com/readers) or request more
ɔrmation by e-mailing apreader@ets.org.
n June 2009, approximately 130 computer science
:hers and professors gathered at the Duke Energy Center
Cincinnati, Ohio, to participate in the scoring session
the AP Computer Science A Exam. Some of the most
erienced members of this group were invited to serve as
estion Leaders and Table Leaders, and they arrived at the
ıding early to help prepare for the scoring session. The
ıaining Readers were divided into groups, with each
up advised and supervised by a Table Leader. Under the
dance of the Chief Reader, Question Leaders and Table
ders assisted in establishing scoring guidelines, selecting
ıple student responses that exemplified the guidelines,
l preparing for Reader training. All of the free-response
:stions on the 2009 AP Computer Science A Exam were
luated by the Readers at this single, central scoring
ıion under the supervision of the Chief Reader.

.suring Accuracy

: primary goal of the scoring process is to have all
ıders score their sets of responses fairly, consistently, and
h the same guidelines as the other Readers. This goal is
ieved through the creation of detailed scoring guidelines,
thorough training of all Readers, and the various checks
l balances that are applied throughout the AP Reading.

ıw the Scoring Guidelines Are Created

As the questions are being developed and reviewed, the Development Committee and the Chief Reader discuss the scoring of the free-response questions to ensure that the questions can be scored validly and reliably. Before the AP Reading, the Chief Reader prepares a draft of the scoring guidelines for each free-response question. In the case of AP Computer Science A, a 10-point scale (0–9) is used. A score of 0 means the student received no credit for the problem.

The Chief Reader, Exam Leaders, Question Leaders, Table Leaders, and ETS Assessment Specialists meet at the Reading site a few days before the Reading begins. They discuss, review, and revise the draft scoring guidelines, and test them by pre-scoring randomly selected student responses. If problems or ambiguities become apparent, the scoring guidelines are revised and refined until a final consensus is reached.

3. Once the scoring of student responses begins, no changes or modifications in the guidelines are made. Given the expertise of the Chief Reader and the analysis of many student responses by Question Leaders and Table Leaders in the pre-Reading period, these guidelines can be used to cover the whole range of student responses. Each Question Leader and Table Leader devotes a great deal of time and effort during the first day of the Reading to teaching the scoring guidelines for that particular question and to ensuring that everyone evaluating responses for that question understands the scoring guidelines and can apply them reliably.

Training Readers to Apply the Scoring Guidelines

Because Reader training is so vital in ensuring that students receive an AP score that accurately reflects their performance, the process is thorough:

1. On the first day of the Reading, the Chief Reader provides an overview of the exam and the scoring process to the entire group of Readers. The Readers then break into smaller groups, with each group working on a particular question for which it receives specific training.
2. Question Leaders direct a discussion of the assigned question, commenting on the question requirements and student performance expectations. The scoring guidelines for the question are explained and discussed.
3. The Readers are trained to apply the scoring guidelines by reading and evaluating samples of student answers that were selected at the pre-Reading session as clear examples of the various score points and the kinds of responses Readers are likely to encounter. Question and Table Leaders explain why the responses received particular scores.
4. When the Question Leader is convinced the Readers understand the scoring guidelines and can apply them uniformly, the scoring of student responses begins. Reading teams are formed by pairing a more experienced Reader with a new or less experienced Reader. Each team is given a set of student exams to score. Both members of the team score each student response independently. The resulting scores and differences in judgment are discussed until agreement is reached, with the Table Leader, Question Leader, or Chief Reader acting as arbitrator when needed.

5. After a team shows consistent agreement on its scores, its members proceed to score papers individually. Readers are encouraged to seek advice from each other, the Table Leader, Question Leader, or Chief Reader when in doubt about a score. Throughout the course of the Reading, a student response that is problematic or inappropriate receives multiple readings and evaluations.

Maintaining the Scoring Guidelines

Throughout the Reading, Table Leaders continue to reinforce the use of the scoring guidelines by asking their groups to review sample responses that have already been discussed as clear examples of particular scores, or to score new samples and discuss those scores with them. This procedure helps the Readers adhere to the standards of the group and helps to ensure that a student response will get the same score whether it is evaluated at the beginning, middle, or end of the Reading.

A potential problem is that a Reader could unintentionally score a student response higher or lower than it deserves because that same student performed well or poorly on other questions. The following steps are taken to prevent this so-called halo effect:

- A different Reader scores each question and the student's identity is unknown to the Reader. Thus, each Reader can evaluate student responses without being prejudiced by knowledge about individual students.
- No marks of any kind are made on the students' papers. The Readers record the scores on a form that is identified only by the student's AP number. Readers are unable to see the scores that have been given to other responses in the exam booklet.

Here are some other methods that help ensure that everyone is adhering closely to the scoring guidelines:

- Table Leaders backread (reread) a portion of the student responses from each of the Readers in that Leader's group. This approach allows Table Leaders to guide their Readers toward appropriate and consistent interpretations of the scoring guidelines.
- Readers are paired, so that every Reader has a partner with whom to check for consistency and to discuss problem cases. Table Leaders and Question Leaders are also paired up to help each other on questionable calls.
- The Chief Reader and the Exam Leaders monitor use of the full range of the scoring scale for the group and for each Reader by checking data on score distributions, and they randomly read selected papers to check for scoring consistency.
- Reliability data are periodically collected by having Readers unknowingly rescore booklets. Scores resultin from the first and second readings are then compared and analyzed.

Preparing Students for the Exam

The AP Computer Science A course is designed to be comparable to a typical introductory computer science course taught in a college or university department of computer science. The course emphasizes object-oriented programming methodology with an emphasis on problen solving and algorithm development, and is meant to be th equivalent of a first-semester course in computer science. also includes the study of data structures and abstraction. For a list of the topics covered, see the topic outline in the *AP Computer Science A Course Description*. The following goals apply to the AP Computer Science A course:

- Students should be able to design and implement solutions to problems by writing, running, and debugging computer programs.
- Students should be able to use and implement commonly-used algorithms and data structures.
- Students should be able to develop and select appropri algorithms and data structures to solve problems.
- Students should be able to code fluently in an object-oriented paradigm using the programming language Java. Students are expected to be familiar with and be able to use standard Java library classes from the AP Java subset.
- Students should be able to read and understand a large program consisting of several classes and interacting objects. Students should be able to read and understan a description of the design and development process leading to such a program. (An example of such a program is the *AP Computer Science Case Study.*)
- Students should recognize the ethical and social implications of computer use.

As teachers focus on the above goals, they will not only be preparing their students for the AP Computer Science A Exam, they will be preparing them for future study and applications of computer science.

Essential Features of Student Responses

There is far more to computer science than just the syntax of a programming language. Similarly, there is much mor to the exam than testing syntax. Student responses are evaluated as "good first drafts" of the solution to a probler In a world where so many students (as well as professiona

ıputer scientists) use tools such as IDEs to help in their gramming, it is foolish to expect students to worry about ing punctuation exactly right. As part of evaluating :-response questions, Readers are provided with General ring Guidelines that describe how to penalize (or ignore) ıor errors, such as missing braces, confusing = and ==, using a class name as an instance of that class. In ring questions, Readers focus more on whether a lent response adequately addresses the major issues of question (for example, array access, looping, class ıipulation, etc.) than more minor issues (is the code ımented, are the braces aligned, are parentheses used ead of brackets, etc.).

Additionally, the ability of students in college classes is :ssed by programming projects as well as test questions. ny professors consider programming projects to be more ıortant than test questions because they are more repre-tative of how computer science is actually done. Since Computer Science A students are assessed with only questions, these questions may have more structure n those on a college test. For example, students on a ege test may be asked simply to write code to determine ıe values in an array are in increasing order. Instead omething so straightforward, a question on the Computer Science A Exam may pose the problem as ting code to determine if a student's scores have im-ved over the course of a term, where the scores are kept n array. This more complicated question can then assess udents can do basic array manipulation and how they ıld perform on programming projects.

Teaching Free-Response Writing

It is important for teachers to devote some course time to having students answer questions like those on the free-response section in a timed environment. Students accustomed to doing all their work in IDEs may have trouble writing code "from scratch." It is also important to have students experience the challenge of reading a description of a problem and understanding exactly what is being asked in the problem. Often, students are asked to use code for which they are given just an interface, not a complete implementation. They need to have experience with reading descriptions of code before taking the AP Exam.

Many successful AP Computer Science A teachers use questions from past free-response exams on a regular basis and go over the scoring in class. The most recent questions, along with the scoring guidelines used, are available on AP Central.

Chapter II: The 2009 AP Computer Science A Exam

Exam Content and Format

The 2009 AP Computer Science A Exam is 3 hours in length and has two sections:

- A 75-minute multiple-choice section consisting of 40 questions accounting for 50 percent of the final score.
- A 105-minute free-response section consisting of 4 questions accounting for 50 percent of the final score.

2009 AP Computer Science A Exam Format

Multiple Choice (Section I)

40 questions ..... 75 minutes

Free Response (Section II)

4 questions ..... 105 minutes

Giving a Practice Exam

The following pages contain the instructions as they appeared in the 2009 *AP Examination Instructions* for administering the AP Computer Science A Exam. Following these instructions are a blank 2009 answer sheet and the 2009 AP Computer Science A Exam. If you plan to use this released exam to test your students, you may wish to use these instructions to create an exam situation that closely resembles an actual administration. If so, read only the indented, boldface directions to the students; all other instructions are for the person administering the exam and need not be read aloud. Some instructions, such as those referring to the date, the time, and page numbers, are no longer relevant and should be ignored. References to the AP Computer Science AB Exam, which was discontinued after the 2009 administration, should also be ignored. The term "grades," which appears in the exam and exam instructions that follow, refers to AP Exam scores of 1, 2, 3, 4, or 5.

Another publication you might find useful is the *Packet of 10*—ten copies of the 2009 AP Computer Science A Exam, each with a blank answer sheet. You can order this title online at the College Board Store (store.collegeboard.com).

Instructions for Administering the Exam
(from the 2009 *AP Examination Instructions* booklet)

The Computer Science A exam and the Computer Science AB exam should be administered simultaneously. They may be administered in separate rooms, or in the same room if it is more convenient.

SECTION I: Multiple-Choice Questions

Do not begin the exam instructions below until you have completed the appropriate General Instructions for your group.

Make sure you begin the exam at the designated time. When you have completed the General Instructions, say:

It is Tuesday morning, May 5, and you will be taking either the AP Computer Science A Exam or the AP Computer Science AB Exam. In a moment, you will open the packet that contains your exam materials. By opening this packet, you agree to all of the AP Program's policies and procedures outlined in the 2008-09 *Bulletin for AP Students and Parents*. Please check to make sure you have the correct exam: Computer Science A or Computer Science AB. Raise your hand if you do not have the correct exam. . . .

You may now open your exam packet and take out the Section I booklet, but do not open the booklet or the shrinkwrapped Section II materials. Put the white seals aside. Read the statements on the front cover of Section I and look up when you have finished. . . .

Now sign your name and write today's date. Look up when you have finished. . . .

Now print your full legal name where indicated. Are there any questions? . . .

Answer any questions. Then say:

Now turn to the back cover and read it completely. Look up when you have finished. . . .

Are there any questions? . . .

Answer any questions. Then say:

Section I is the multiple-choice portion of the exam. You may never discuss these multiple-choice questions at any time in any form with anyone, including your teacher and other students. If you disclose these questions through any means, your AP Exam grade will be canceled. Are there any questions? . . .

Answer any questions. Then say:

You must complete the answer sheet using a No. 2 pencil only. Mark all of your responses on your answer sheet, one response per question. Completely fill in the ovals. There are more answer ovals on the answer sheet than there are questions, so you will have unused ovals when you reach the end. Your answer sheet will be scored by machine; any stray marks

or smudges could be read as answers. If you need to erase, do so carefully and completely. No credit will be given for anything written in the exam booklet. Scratch paper is not allowed, but you may use the margins or any blank space in the exam booklet for scratch work. Are there any questions? . . .

Answer all questions regarding procedure. Then say:

You may use the orange appendix booklet throughout the exam. Appendix A contains the AP Java Quick Reference, and appendixes B through G contain case study reference material. In the Computer Science A exam, the case study questions are 21 through 25. In the Computer Science AB exam, the case study questions are 20 through 25. You have 1 hour and 15 minutes for Section I. Open your Section I booklet now and begin.

Note Start Time here _____. Note Stop Time here _____. You and your proctors should make sure students are marking their answers in pencil on their answer sheets, and that they are not looking at their shrinkwrapped Section II booklets. After 1 hour and 15 minutes, say:

Stop working. Close both your exam booklet and orange appendix booklet and put your answer sheet on your desk, face up. I will now collect your answer sheet.

After you have collected an answer sheet from each student, say:

Take your seals and press one on each area of your exam booklet marked "PLACE SEAL HERE." Fold them over the open edges and press them to the back cover. When you have finished, place the exam booklet on your desk with the cover face up. Keep your orange appendix booklet; you will need it for Section II of the exam. . . .

I will now collect your Section I booklet.

As you collect the sealed Section I booklets, check to be sure that each student has signed the front cover. There is a 10-minute break between Sections I and II. When all Section I materials have been collected and accounted for and you are ready for the break, say:

Please listen carefully to these instructions before we take a break. Everything you placed under your chair at the beginning of the exam must remain there. You are not allowed to consult teachers, other students, or textbooks about the exam materials during the break. You may not make phone calls, send text messages, check e-mail, access a computer, calculator, cell phone, PDA, MP3 player, e-mail/messaging device, or any other electronic or communication device. Remember, you are not allowed to discuss the multiple-choice section of this exam with anyone at any time. Failure to adhere to any of these rules could result in cancellation of your grade. Please leave your shrinkwrapped Section II package and your orange appendix booklet on top of your desk during the break. You may get up, talk, go to the restroom, or get a drink. Are there any questions? . . .

Answer all questions regarding procedure. Then say:

Let's begin our break. Testing will resume at _____.

SECTION II: Free-Response Questions

After the break, say:

May I have everyone's attention? Place your Student Pack on your desk. . . .

You may now open the shrinkwrapped Section II package. . . .

Read the bulleted statements on the front cover of the pink booklet. Look up when you have finished. . . .

Now place an AP number label on the shaded box. If you don't place an AP number label on this box, it may be impossible to identify your booklet, which could delay or jeopardize your AP grade. If you don't have any AP number labels, write your AP number in the box. Look up when you have finished. . . .

Read the last statement. . . .

Using a pen with black or dark blue ink, print the first, middle, and last initials of your legal name in the boxes and print today's date where indicated. This constitutes your signature and your agreement to the conditions stated on the front cover. . . .

Turn to the back cover and read Item 1 under "Important Identification Information." Print your identification information in the boxes. Note that you must print the first two letters of your last name and the first letter of your first name. Look up when you have finished. . . .

In Item 2, print your date of birth in the boxes. . . .

Read Item 3 and copy the school code you printed on the front of your Student Pack into the boxes. . . .

Read Item 4. . . .

Are there any questions? . . .

Answer all questions regarding procedure. Then say:

I need to collect the Student Pack from anyone who will be taking another AP Exam. If you are taking another AP Exam, put your Student Pack on your desk. You may keep it only if you are not taking any other AP Exams this year. If you have no other AP Exams to take, place your Student Pack under your chair now. . . .

While Student Packs are being collected, read the "At a Glance" column and the instructions for Section II on the back cover of the pink booklet. Do not open the booklet until you are told to do so. Look up when you have finished. . . .

Collect the Student Packs. Then say:

Are there any questions? . . .

Answer all questions regarding procedure. Then say:

Now open the Section II booklet and tear out the green insert that is in the center of the booklet. In the upper right-hand corner of the cover, print your name, your teacher's name, and your school's name. . . .

Read the information on the front cover of the green insert. Look up when you have finished. . . .

You will need the orange appendix booklet for Question 2 in both the A exam and the AB exam, although you may use it at any time during this period. You may make notes in the green insert, but you must write your answers in the pink booklet using a No. 2 pencil. You are responsible for pacing yourself, and may proceed freely from one question to the next. If you need more paper during the exam, raise your hand. At the top of each extra piece of paper you use, be sure to write your AP number and the number of the question you are working on. You have 1 hour and 45 minutes for Section II. Are there any questions? . . .

Answer any questions. Then say:

You may begin.

 Note Start Time here _____. Note Stop Time here _____. You and your proctors should make sure students are writing their answers in their Section II booklets. After 1 hour and 35 minutes, say:

There are 10 minutes remaining.

After 10 minutes, say:

Stop working and close your exam booklet, the orange appendix booklet, and the green insert. Put your pink booklet, your orange appendix booklet, and your green insert on your desk, face up. Remain in your seat, without talking, while the exam materials are collected. . . .

Collect a pink Section II booklet, an orange appendix booklet, and a green insert from every student. Check for the following:

- Section II booklet front cover: The student placed an AP number label in the shaded box, and printed his or her initials and today's date.
- Section II booklet back cover: The student completed the "Important Identification Information" area.
- The student wrote answers in the pink booklet and not in the orange appendix booklet or green insert.

The green inserts must be stored securely for no fewer than two school days. After the two-day holding time, the green inserts may be given to the appropriate AP teacher(s) for return to the students. **The orange appendix booklets must be returned to the AP Program.** When all exam materials have been collected and accounted for, say:

Your teacher will return your green inserts to you in about two days. You may not discuss the free-response questions with anyone until that time. Remember that the multiple-choice questions may never be discussed or shared in any way at any time. You should receive your grade report in the mail about the third week of July. You are now dismissed.

Exam materials should be put in locked storage until they are returned to the AP Program after your school's last administration. Before storing materials, check your list of students who are eligible for fee reductions and fill in the appropriate oval on their registration answer sheets. To receive a separate AP Instructional Planning Report or student grade roster for each AP class taught, fill in the appropriate oval in the "School Use Only" section of the answer sheet. See "Post-Exam Activities" in the 2009 *AP Coordinator's Manual.*

USE PENCIL ONLY FOR THE ENTIRE ANSWER SHEET.

NAME AND EXAM AREA – COMPLETE THIS AREA AT EVERY EXAM.

To maintain the security of the exam and the validity of my AP grade, I will allow no one else to see the multiple-choice questions. I will seal the multiple-choice booklet when asked to do so, and I will not discuss these questions with anyone at any time after the completion of the section. I am aware of and agree to the AP Program's policies and procedures as outlined in the 2008-09 *Bulletin for AP Students and Parents*, including using testing accommodations (e.g., extended time, computer, etc.) only if I have been preapproved by College Board Services for Students with Disabilities.

A. SIGNATURE _____ Sign your legal name as it will appear on your college applications.

CollegeBoard AP **Answer Sheet for May 2009, Form 4FBP** **PAGE 1**

B123456789T

PLACE YOUR AP® NUMBER LABEL OR WRITE YOUR AP NUMBER HERE AT EVERY EXAM.

B. LEGAL NAME Omit apostrophes, Jr., or II.

Legal Last Name—first 15 letters | Legal First Name—first 12 letters | MI

C. YOUR AP NUMBER

D. DATE

MAY

E. TIME OF DAY

- 6 AM
- 7 AM
- 8 AM
- 9 AM
- 10 AM
- 11 AM
- 12 PM
- 1 PM
- 2 PM
- 3 PM
- 4 PM
- 5 PM

STUDENT INFORMATION AREA—COMPLETE THIS AREA ONLY ONCE.

I. SEX

1. Male
2. Female

J. CURRENT GRADE LEVEL

1. Not yet in 9th grade
2. 9th grade
3. 10th grade
4. 11th grade
5. 12th grade
6. No longer in high school

K. DATE OF BIRTH

Month	Day	Year
01 Jan.		
02 Feb.		
03 Mar.	0 0	0 0
04 Apr.	1 1	1 1
05 May	2 2	2 2
06 June	3 3	3 3
07 July	4 4	4 4
08 Aug.	5 5	5 5
09 Sept.	6 6	6 6
10 Oct.	7 7	7 7
11 Nov.	8 8	8 8
12 Dec.	9 9	9 9

L. SOCIAL SECURITY NUMBER (Optional, but preferred)

M. ETHNICITY/RACE

1. American Indian or Alaska Native
2. Asian, Asian American, or Pacific Islander
3. Black or African American
4. Mexican or Mexican American
5. Puerto Rican
6. Other Hispanic, Latino, or Latin American
7. White
8. Other

N. EXPECTED DATE OF COLLEGE ENTRANCE

F Fall	0 2009
W Winter/Spring	1 2010
S Summer	2 2011
U Undecided	3 2012

O. WHAT LANGUAGE DO YOU KNOW BEST?

1. English
2. English and another language about the same
3. Another language

P. Complete ONLY if you are a SOPHOMORE or a JUNIOR.

I want the College Board to send information about me to colleges, universities, and government scholarship programs interested in students like me.

○ Yes ○ No

F. AP EXAM I AM TAKING USING THIS ANSWER SHEET

Print exam name: _____ Print form code (e.g., 4FBP-R) from M-C booklet: _____

Fill in the oval below that corresponds to the exam that you are taking.

07 ○ U.S. History 4FBP-Q	43 ○ European History	78 ○ Physics B
07 ○ U.S. History 4FBP-R	48 ○ French Language	80 ○ Physics C: Mech.
13 ○ Art History	51 ○ French Literature	82 ○ Physics C: E & M
14 ○ Art: Studio Drawing	53 ○ Geography: Human	85 ○ Psychology
15 ○ Art: Studio 2-D Design	55 ○ German Language	87 ○ Spanish Language
16 ○ Art: Studio 3-D Design	57 ○ Gov. & Pol.: U.S.	89 ○ Spanish Literature
20 ○ Biology	58 ○ Gov. & Pol.: Comp.	90 ○ Statistics
25 ○ Chemistry	60 ○ Latin: Vergil	93 ○ World History
28 ○ Chinese Lang. & Culture	61 ○ Latin Literature	
31 ○ Computer Science A	62 ○ Italian Lang. & Culture	
33 ○ Computer Science AB	64 ○ Japanese Lang. & Culture	
34 ○ Economics: Micro	66 ○ Calculus AB 4FBP-Q	
35 ○ Economics: Macro	66 ○ Calculus AB 4FBP-R	
36 ○ Eng. Language & Comp.	68 ○ Calculus BC 4FBP-Q	
37 ○ Eng. Literature & Comp.	68 ○ Calculus BC 4FBP-R	
40 ○ Environmental Science	75 ○ Music Theory	

G. ONLINE PROVIDER CODE

H. MULTIPLE-CHOICE BOOKLET SERIAL NUMBER

S

748408

Q. PARENTAL EDUCATION LEVEL

Father / Male Guardian	Mother / Female Guardian	
○	○	Grade school
○	○	Some high school
○	○	High school diploma or equivalent
○	○	Business or trade school
○	○	Some college
○	○	Associate or two-year degree
○	○	Bachelor's or four-year degree
○	○	Some graduate or professional school
○	○	Graduate or professional degree

SCHOOL USE ONLY

Instructional Planning Report/Student Grade Roster 1 2 3 4 5 6 7 8 9

Fee Reduction Granted ○ Option 1 ○ Option 2

ETS USE ONLY

Exam 1 2 | Exam 1 2

3659 • 00657 • TF12E4000 Q3144/06-09 lig/mh08260

R. This section is for the survey questions in the AP Student Pack. (Do not put responses to exam questions in this section.) Be sure each mark is dark and completely fills the oval.

1 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ Ⓕ Ⓖ	4 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ Ⓕ Ⓖ	7 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ Ⓕ Ⓖ
2 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ Ⓕ Ⓖ	5 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ Ⓕ Ⓖ	8 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ Ⓕ Ⓖ
3 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ Ⓕ Ⓖ	6 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ Ⓕ Ⓖ	9 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ Ⓕ Ⓖ

S. LANGUAGE—Do not complete this section unless instructed to do so.

If this answer sheet is for the Chinese Language and Culture, French Language, French Literature, German Language, Italian Language and Culture, Japanese Language and Culture, Spanish Language, or Spanish Literature Exam, please answer the following questions. (Your responses will not affect your grade.)

1. Have you lived or studied for one month or more in a country where the language of the exam you are now taking is spoken? ◯ Yes ◯ No
2. Do you regularly speak or hear the language at home? ◯ Yes ◯ No

Indicate your answers to the exam questions in this section. If a question has only four answer options, do not mark option E. Your answer sheet will be scored by machine. Use only No. 2 pencils to mark your answers on pages 2 and 3 (one response per question). After you have determined your response, be sure to completely fill in the oval corresponding to the number of the question you are answering. Stray marks and smudges could be read as answers, so erase carefully and completely. Any improper gridding may affect your grade. Answers written in the multiple-choice booklet will not be scored.

1 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	26 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	51 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
2 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	27 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	52 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
3 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	28 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	53 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
4 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	29 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	54 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
5 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	30 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	55 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
6 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	31 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	56 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
7 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	32 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	57 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
8 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	33 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	58 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
9 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	34 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	59 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
10 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	35 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	60 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
11 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	36 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	61 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
12 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	37 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	62 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
13 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	38 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	63 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
14 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	39 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	64 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
15 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	40 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	65 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
16 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	41 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	66 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
17 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	42 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	67 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
18 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	43 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	68 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
19 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	44 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	69 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
20 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	45 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	70 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
21 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	46 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	71 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
22 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	47 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	72 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
23 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	48 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	73 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
24 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	49 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	74 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ
25 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	50 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ	75 Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

FOR QUESTIONS 76-151, SEE PAGE 3.

DO NOT WRITE IN THIS AREA.

Be sure each mark is dark and completely fills the oval. If a question has only four answer options, do not mark option E.

76	A	B	C	D	E	101	A	B	C	D	E	126	A	B	C	D	E
77	A	B	C	D	E	102	A	B	C	D	E	127	A	B	C	D	E
78	A	B	C	D	E	103	A	B	C	D	E	128	A	B	C	D	E
79	A	B	C	D	E	104	A	B	C	D	E	129	A	B	C	D	E
80	A	B	C	D	E	105	A	B	C	D	E	130	A	B	C	D	E
81	A	B	C	D	E	106	A	B	C	D	E	131	A	B	C	D	E
82	A	B	C	D	E	107	A	B	C	D	E	132	A	B	C	D	E
83	A	B	C	D	E	108	A	B	C	D	E	133	A	B	C	D	E
84	A	B	C	D	E	109	A	B	C	D	E	134	A	B	C	D	E
85	A	B	C	D	E	110	A	B	C	D	E	135	A	B	C	D	E
86	A	B	C	D	E	111	A	B	C	D	E	136	A	B	C	D	E
87	A	B	C	D	E	112	A	B	C	D	E	137	A	B	C	D	E
88	A	B	C	D	E	113	A	B	C	D	E	138	A	B	C	D	E
89	A	B	C	D	E	114	A	B	C	D	E	139	A	B	C	D	E
90	A	B	C	D	E	115	A	B	C	D	E	140	A	B	C	D	E
91	A	B	C	D	E	116	A	B	C	D	E	141	A	B	C	D	E
92	A	B	C	D	E	117	A	B	C	D	E	142	A	B	C	D	E
93	A	B	C	D	E	118	A	B	C	D	E	143	A	B	C	D	E
94	A	B	C	D	E	119	A	B	C	D	E	144	A	B	C	D	E
95	A	B	C	D	E	120	A	B	C	D	E	145	A	B	C	D	E
96	A	B	C	D	E	121	A	B	C	D	E	146	A	B	C	D	E
97	A	B	C	D	E	122	A	B	C	D	E	147	A	B	C	D	E
98	A	B	C	D	E	123	A	B	C	D	E	148	A	B	C	D	E
99	A	B	C	D	E	124	A	B	C	D	E	149	A	B	C	D	E
100	A	B	C	D	E	125	A	B	C	D	E	150	A	B	C	D	E
												151	A	B	C	D	E

ETS USE ONLY

28, 48, 55, 62, 64, 75, 87

	R	W	O
PT02			
PT03			
PT04			

OTHER

	R	W	O
TOTAL			

DO NOT WRITE IN THIS AREA.

T. YOUR MAILING ADDRESS

HOME ADDRESS AND SCHOOL AREA—COMPLETE THIS AREA ONLY ONCE.

- Your grade report will be mailed to this address in July.
- Using the abbreviations given in your AP Student Pack, print your address in the boxes below. If your international address does not fit, see item V. below.
- Indicate a space in your address by leaving a blank box — do not grid that column. Only one oval is to be filled in for each column.

Street Address (street number and name)

City

STATE

1	AL	28	NE
2	AK	29	NV
3	AZ	30	NH
4	AR	31	NJ
5	CA	32	NM
6	CO	33	NY
7	CT	34	NC
8	DE	35	ND
9	DC	36	OH
10	FL	37	OK
11	GA	38	OR
12	HI	39	PA
13	ID	40	RI
14	IL	41	SC
15	IN	42	SD
16	IA	43	TN
17	KS	44	TX
18	KY	45	UT
19	LA	46	VT
20	ME	47	VA
21	MD	48	WA
22	MA	49	WV
23	MI	50	WI
24	MN	51	WY
25	MS	52	Puerto Rico
26	MO		
27	MT	53	Other

ZIP OR POSTAL CODE

U. COUNTRY CODE

W. TELEPHONE NUMBER

Area Code

INTERNATIONAL TELEPHONE

If your international telephone number is longer than 10 digits, write the entire number below.

FOR STUDENTS OUTSIDE THE UNITED STATES ONLY

V. If the address gridded above is not complete enough for delivery of your grade report, please fill in this oval and print your complete address below. ⟶ ○

Address

Address

City — State or Province

Country — ZIP or Postal Code

X. SCHOOL YOU ATTEND

School Code

School Name, City, and State

← Make sure you have correctly entered your School Code, filled in the appropriate ovals, and completed the information below.

School Name

City

State

Y. COLLEGE TO RECEIVE YOUR AP GRADE REPORT

College Code

College Name and Address

Using the College Code list in the AP Student Pack, indicate the **one** college that you want to receive your AP Grade Report by writing in the college code number, gridding the appropriate ovals, and completing the information below.

College Name

City

State

AP® Computer Science A Exam

SECTION I: Multiple-Choice Questions

2009

DO NOT OPEN THIS BOOKLET UNTIL YOU ARE TOLD TO DO SO.

At a Glance

Total Time
1 hour, 15 minutes
Number of Questions
40
Percent of Total Grade
50%
Writing Instrument
Pencil required
Electronic Device
None allowed

Instructions

The orange Appendix booklet is provided for use in both Section I and Section II.

Section I of this exam contains 40 multiple-choice questions. Fill in only the ovals for numbers 1 through 40 on your answer sheet.

Indicate all of your answers to the multiple-choice questions on the answer sheet. No credit will be given for anything written in this exam booklet, but you may use the booklet for notes or scratch work. After you have decided which of the suggested answers is best, completely fill in the corresponding oval on the answer sheet. Give only one answer to each question. If you change an answer, be sure that the previous mark is erased completely. Here is a sample question and answer.

Sample Question	Sample Answer
Chicago is a (A) state (B) city (C) country (D) continent (E) village	Ⓐ ● Ⓒ Ⓓ Ⓔ

Use your time effectively, working as quickly as you can without losing accuracy. Do not spend too much time on any one question. Go on to other questions and come back to the ones you have not answered if you have time. It is not expected that everyone will know the answers to all of the multiple-choice questions.

About Guessing

Many students wonder whether or not to guess the answers to questions about which they are not certain. In this section of the exam, as a correction for random guessing, one-fourth of the number of questions you answer incorrectly will be subtracted from the number of questions you answer correctly. If you are not sure of the best answer but have some knowledge of the question and are able to eliminate one or more of the answer choices, your chance of answering correctly is improved, and it may be to your advantage to answer such a question.

Section I

COMPUTER SCIENCE A
SECTION I
Time—1 hour and 15 minutes
Number of questions—40
Percent of total grade—50

Directions: Determine the answer to each of the following questions or incomplete statements, using the available space for any necessary scratch work. Then decide which is the best of the choices given and fill in the corresponding oval on the answer sheet. No credit will be given for anything written in the examination booklet. Do not spend too much time on any one problem.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Assume that declarations of variables and methods appear within the context of an enclosing class.
- Assume that method calls that are not prefixed with an object or class name and are not shown within a complete class definition appear within the context of an enclosing class.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null`.

1. Consider the following code segment.

```
int value = 15;
while (value < 28)
{
  System.out.println(value);
  value++;
}
```

What are the first and last numbers output by the code segment?

	First	Last
(A)	15	27
(B)	15	28
(C)	16	27
(D)	16	28
(E)	16	29

2. A teacher put three bonus questions on a test and awarded 5 extra points to anyone who answered all three bonus questions correctly and no extra points otherwise. Assume that the `boolean` variables `bonusOne`, `bonusTwo`, and `bonusThree` indicate whether a student has answered the particular question correctly. Each variable was assigned `true` if the answer was correct and `false` if the answer was incorrect.

   Which of the following code segments will properly update the variable `grade` based on a student's performance on the bonus questions?

   I.
   ```
   if (bonusOne && bonusTwo && bonusThree)
     grade += 5;
   ```

   II.
   ```
   if (bonusOne || bonusTwo || bonusThree)
     grade += 5;
   ```

   III.
   ```
   if (bonusOne)
     grade += 5;
   if (bonusTwo)
     grade += 5;
   if (bonusThree)
     grade += 5;
   ```

   (A) I only

   (B) II only

   (C) III only

   (D) I and III

   (E) II and III

GO ON TO THE NEXT PAGE

Assume that an array of integer values has been declared as follows and has been initialized.

```
int[] arr = new int[10];
```

Which of the following code segments correctly interchanges the value of `arr[0]` and `arr[5]` ?

(A)
```
arr[0] = 5;
arr[5] = 0;
```

(B)
```
arr[0] = arr[5];
arr[5] = arr[0];
```

(C)
```
int k = arr[5];
arr[0] = arr[5];
arr[5] = k;
```

(D)
```
int k = arr[0];
arr[0] = arr[5];
arr[5] = k;
```

(E)
```
int k = arr[5];
arr[5] = arr[0];
arr[0] = arr[5];
```

4. Consider the following code segment.

```
ArrayList<String> items = new ArrayList<String>();
items.add("A");
items.add("B");
items.add("C");
items.add(0, "D");
items.remove(3);
items.add(0, "E");
System.out.println(items);
```

What is printed as a result of executing the code segment?

(A) `[A, B, C, E]`

(B) `[A, B, D, E]`

(C) `[E, D, A, B]`

(D) `[E, D, A, C]`

(E) `[E, D, C, B]`

---

5. When designing a class hierarchy, which of the following should be true of a superclass?

(A) A superclass should contain the data and functionality that are common to all subclasses that inherit from the superclass.

(B) A superclass should be the largest, most complex class from which all other subclasses are derived.

(C) A superclass should contain the data and functionality that are only required for the most complex class.

(D) A superclass should have public data in order to provide access for the entire class hierarchy.

(E) A superclass should contain the most specific details of the class hierarchy.

GO ON TO THE NEXT PAGE.

Questions 6-7 refer to the following code segment.

```
int k = a random number such that 1 ≤ k ≤ n ;

for (int p = 2; p <= k; p++)
  for (int r = 1; r < k; r++)
    System.out.println("Hello");
```

6. What is the minimum number of times that `Hello` will be printed?

(A) 0

(B) 1

(C) 2

(D) $n - 1$

(E) $n - 2$

---

7. What is the maximum number of times that `Hello` will be printed?

(A) 2

(B) $n - 1$

(C) $n - 2$

(D) $(n - 1)^2$

(E) n^2

8. Consider the following instance variable and incomplete method. The method `calcTotal` is intended to return the sum of all values in `vals`.

```
private int[] vals;

public int calcTotal()
{
  int total = 0;

  /* missing code */

  return total;
}
```

Which of the code segments shown below can be used to replace `/* missing code */` so that `calcTotal` will work as intended?

I.
```
for (int pos = 0; pos < vals.length; pos++)
{
  total += vals[pos];
}
```

II.
```
for (int pos = vals.length; pos > 0; pos--)
{
  total += vals[pos];
}
```

III.
```
int pos = 0;
while (pos < vals.length)
{
  total += vals[pos];
  pos++;
}
```

(A) I only

(B) II only

(C) III only

(D) I and III

(E) II and III

). Consider the following code segment.

```
String str = "abcdef";
for (int rep = 0; rep < str.length() - 1; rep++)
{
  System.out.print(str.substring(rep, rep + 2));
}
```

What is printed as a result of executing this code segment?

(A) `abcdef`

(B) `aabbccddeeff`

(C) `abbccddeef`

(D) `abcbcdcdedef`

(E) Nothing is printed because an `IndexOutOfBoundsException` is thrown.

10. Consider the following method.

```
public void numberCheck(int maxNum)
{
  int typeA = 0;
  int typeB = 0;
  int typeC = 0;

  for (int k = 1; k <= maxNum; k++)
  {
    if (k % 2 == 0 && k % 5 == 0)
      typeA++;
    if (k % 2 == 0)
      typeB++;
    if (k % 5 == 0)
      typeC++;
  }

  System.out.println(typeA + " " + typeB + " " + typeC);
}
```

What is printed as a result of the call `numberCheck(50)` ?

(A) `5 20 5`

(B) `5 20 10`

(C) `5 25 5`

(D) `5 25 10`

(E) `30 25 10`

GO ON TO THE NEXT PAG

1. Consider the following method that is intended to modify its parameter `nameList` by replacing all occurrences of `name` with `newValue`.

```
public void replace(ArrayList<String> nameList,
                    String name, String newValue)
{
  for (int j = 0; j < nameList.size(); j++)
  {
    if ( /* expression */ )
    {
      nameList.set(j, newValue);
    }
  }
}
```

Which of the following can be used to replace `/* expression */` so that `replace` will work as intended?

(A) `nameList.get(j).equals(name)`

(B) `nameList.get(j) == name`

(C) `nameList.remove(j)`

(D) `nameList[j] == name`

(E) `nameList[j].equals(name)`

Section I

12. Consider the following incomplete method.

```
public int someProcess(int n)
{
  /* body of someProcess */
}
```

The following table shows several examples of input values and the results that should be produced by calling `someProcess`.

Input Value n	Value Returned by someProcess(n)
3	30
6	60
7	7
8	80
9	90
11	11
12	120
14	14
16	160

Which of the following code segments could be used to replace `/* body of someProcess */` so that the method will produce the results shown in the table?

I.
```
if ((n % 3 == 0) && (n % 4 == 0))
  return n * 10;
else
  return n;
```

II.
```
if ((n % 3 == 0) || (n % 4 == 0))
  return n * 10;

return n;
```

II.
```
if (n % 3 == 0)
  if (n % 4 == 0)
    return n * 10;

return n;
```

A) I only

B) II only

C) III only

D) I and III

E) II and III

13. Consider the following method.

```
// precondition: x >= 0
public void mystery(int x)
{
  if ((x / 10) != 0)
  {
    mystery(x / 10);
  }

  System.out.print(x % 10);
}
```

Which of the following is printed as a result of the call `mystery(123456)` ?

(A) `16`

(B) `56`

(C) `123456`

(D) `654321`

(E) Many digits are printed due to infinite recursion.

GO ON TO THE NEXT PAG

Consider the following instance variables and incomplete method that are part of a class that represents an item. The variables `years` and `months` are used to represent the age of the item, and the value for `months` is always between 0 and 11, inclusive. Method `updateAge` is used to update these variables based on the parameter `extraMonths` that represents the number of months to be added to the age.

```
private int years;
private int months;  // 0 <= months <= 11

// precondition: extraMonths >= 0
public void updateAge(int extraMonths)
{
  /* body of updateAge */
}
```

Which of the following code segments could be used to replace `/* body of updateAge */` so that the method will work as intended?

I.
```
int yrs = extraMonths % 12;
int mos = extraMonths / 12;
years = years + yrs;
months = months + mos;
```

II.
```
int totalMonths = years * 12 + months + extraMonths;
years = totalMonths / 12;
months = totalMonths % 12;
```

III.
```
int totalMonths = months + extraMonths;
years = years + totalMonths / 12;
months = totalMonths % 12;
```

(A) I only

(B) II only

(C) III only

(D) II and III only

(E) I, II, and III

15. Consider the following method.

```
public String inRangeMessage(int value)
{
  if (value < 0 || value > 100)
    return "Not in range";
  else
    return "In range";
}
```

Consider the following code segments that could be used to replace the body of `inRangeMessage`.

I.
```
if (value < 0)
{
  if (value > 100)
    return "Not in range";
  else
    return "In range";
}
else
  return "In range";
```

II.
```
if (value < 0)
  return "Not in range";
else if (value > 100)
  return "Not in range";
else
  return "In range";
```

III.
```
if (value >= 0)
  return "In range";
else if (value <= 100)
  return "In range";
else
  return "Not in range";
```

Which of the replacements will have the same behavior as the original version of `inRangeMessage` ?

(A) I only

(B) II only

(C) III only

(D) I and III

(E) II and III

Consider the following class declaration.

```
public class SomeClass
{
  private int num;

  public SomeClass(int n)
  {
    num = n;
  }

  public void increment(int more)
  {
    num = num + more;
  }

  public int getNum()
  {
    return num;
  }
}
```

The following code segment appears in another class.

```
SomeClass one = new SomeClass(100);
SomeClass two = new SomeClass(100);
SomeClass three = one;

one.increment(200);

System.out.println(one.getNum() + "   " + two.getNum() + "   " +
                   three.getNum());
```

What is printed as a result of executing the code segment?

(A) 100 100 100

(B) 300 100 100

(C) 300 100 300

(D) 300 300 100

(E) 300 300 300

17. The following incomplete method is intended to sort its array parameter `arr` in increasing order.

```
// postcondition: arr is sorted in increasing order
public static void sortArray(int[] arr)
{
  int j, k;

  for (j = arr.length - 1; j > 0; j--)
  {
    int pos = j;

    for ( /* missing code */ )
    {
      if (arr[k] > arr[pos])
      {
        pos = k;
      }
    }
    swap(arr, j, pos);
  }
}
```

Assume that `swap(arr, j, pos)` exchanges the values of `arr[j]` and `arr[pos]`. Which of the following could be used to replace `/* missing code */` so that executing the code segment sorts the values in array `arr`?

(A) `k = j - 1; k > 0; k--`

(B) `k = j - 1; k >= 0; k--`

(C) `k = 1; k < arr.length; k++`

(D) `k = 1; k > arr.length; k++`

(E) `k = 0; k <= arr.length; k++`

3. Assume that `x` and `y` are `boolean` variables and have been properly initialized.

```
(x && y) || !(x && y)
```

The result of evaluating the expression above is best described as

(A) always `true`

(B) always `false`

(C) `true` only when `x` is `true` and `y` is `true`

(D) `true` only when `x` and `y` have the same value

(E) `true` only when `x` and `y` have different values

19. Assume that the following variable declarations have been made.

```
double d = Math.random();
double r;
```

Which of the following assigns a value to `r` from the uniform distribution over the range $0.5 \leq r < 5.5$?

(A) `r = d + 0.5;`

(B) `r = d + 0.5 * 5.0;`

(C) `r = d * 5.0;`

(D) `r = d * 5.0 + 0.5;`

(E) `r = d * 5.5;`

. Consider the following instance variables and method that appear in a class representing student information.

```
private int assignmentsCompleted;
private double testAverage;

public boolean isPassing()
{ /* implementation not shown */ }
```

A student can pass a programming course if at least one of the following conditions is met.

- The student has a test average that is greater than or equal to 90.
- The student has a test average that is greater than or equal to 75 and has at least 4 completed assignments.

Consider the following proposed implementations of the `isPassing` method.

I.
```
if (testAverage >= 90)
  return true;
if (testAverage >= 75 && assignmentsCompleted >= 4)
  return true;
return false;
```

II.
```
boolean pass = false;
if (testAverage >= 90)
  pass = true;
if (testAverage >= 75 && assignmentsCompleted >= 4)
  pass = true;
return pass;
```

III.
```
return (testAverage >= 90) ||
       (testAverage >= 75 && assignmentsCompleted >= 4);
```

Which of the implementations will correctly implement method `isPassing` ?

(A) I only

(B) II only

(C) I and III only

(D) II and III only

(E) I, II, and III

Questions 21-25 refer to the code from the GridWorld case study. A copy of the code is provided in the Appendix.

21. Consider the following code segment.

```
Location loc1 = new Location(3, 3);
Location loc2 = new Location(3, 2);

if (loc1.equals(loc2.getAdjacentLocation(Location.EAST)))
  System.out.print("aaa");

if (loc1.getRow() == loc2.getRow())
  System.out.print("XXX");

if (loc1.getDirectionToward(loc2) == Location.EAST)
  System.out.print("555");
```

What will be printed as a result of executing the code segment?

(A) aaaXXX555

(B) aaaXXX

(C) XXX555

(D) 555

(E) aaa

. A `RightTurningBug` behaves like a `Bug`, except that when it turns, it turns 90 degrees to the right. The declaration for the `RightTurningBug` class is as follows.

```
public class RightTurningBug extends Bug
{
  public void turn()
  {
    /* missing implementation */
  }
}
```

Consider the following suggested replacements for `/* missing implementation */`.

I.
```
int desiredDirection = (getDirection() + Location.RIGHT)
                       % Location.FULL_CIRCLE;

while (getDirection() != desiredDirection)
{
  super.turn();
}
```

II.
```
super.turn();
super.turn();
```

III.
```
setDirection(getDirection() + Location.RIGHT);
```

Which of the replacements will produce the desired behavior?

(A) I only

(B) II only

(C) I and II only

(D) I and III only

(E) I, II, and III

23. Consider the following declarations.

```
Actor a = new Actor();
Bug b = new Bug();
Rock r = new Rock();
Critter c = new Critter();
```

Consider the following lines of code.

```
Line 1:   int dir1 = c.getDirection();
Line 2:   int dir2 = a.getDirection();
Line 3:   int dir3 = b.getDirection();
Line 4:   ArrayList<Location> rLoc = r.getMoveLocations();
Line 5:   ArrayList<Location> cLoc = c.getMoveLocations();
```

Which of the lines of code above will cause a compile time error?

(A) Line 1 only

(B) Lines 2 and 3 only

(C) Line 4 only

(D) Line 5 only

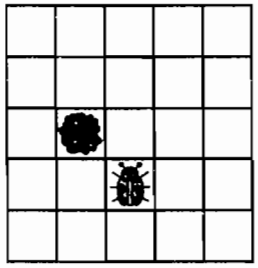
(E) Lines 4 and 5 only

. Consider the following `TestBug` class declaration.

```
public class TestBug extends Bug
{
  public void act()
  {
    if (canMove())
    {
      move();
      if (canMove())
        move();
    }
    else
    {
      setDirection(getDirection() + Location.HALF_CIRCLE);
    }
  }
}
```

The following code segment will produce a grid that has a `Rock` object and a `TestBug` object placed as shown.

```
Grid<Actor> g = new BoundedGrid<Actor>(5, 5);
Rock r = new Rock();
r.putSelfInGrid(g, new Location(2, 1));
Bug t = new TestBug();
t.putSelfInGrid(g, new Location(3, 2));
```



Which of the following best describes what the `TestBug` object `t` does as a result of calling `t.act()` ?

(A) Moves forward two locations and remains facing current direction

(B) Moves forward two locations and turns 180 degrees

(C) Moves forward one location and remains facing current direction

(D) Moves forward one location and turns 180 degrees

(E) Stays in the same location and turns 180 degrees

25. A `DancingCritter` is a `Critter` that moves in the following manner. The `DancingCritter` makes a left turn if at least one of its neighbors is another `DancingCritter`. It then moves like a `Critter`. If none of its neighbors are `DancingCritter` objects, it moves like a `Critter` without making a left turn. In all other respects, a `DancingCritter` acts like a `Critter` by eating neighbors that are not rocks or critters. Consider the following implementations.

I.
```
public class DancingCritter extends Critter
{
  public ArrayList<Actor> getActors()
  {
    ArrayList<Actor> actors = new ArrayList<Actor>();
    for (Actor a : getGrid().getNeighbors(getLocation()))
    {
      if (a instanceof DancingCritter)
        actors.add(a);
    }
    return actors;
  }

  public void processActors(ArrayList<Actor> actors)
  {
    if (actors.size() > 0)
    {
      setDirection(getDirection() + Location.LEFT);
    }
    super.processActors(actors);
  }
}
```

II.
```
public class DancingCritter extends Critter
{
  public void processActors(ArrayList<Actor> actors)
  {
    boolean turning = false;
    for (Actor a : actors)
    {
      if (a instanceof DancingCritter)
        turning = true;
    }
    if (turning)
    {
      setDirection(getDirection() + Location.LEFT);
    }
  }
}
```

I.
```
public class DancingCritter extends Critter
{
  public void makeMove(Location loc)
  {
    boolean turning = false;
    for (Actor a : getGrid().getNeighbors(getLocation()))
    {
      if (a instanceof DancingCritter)
        turning = true;
    }
    if (turning)
    {
      setDirection(getDirection() + Location.LEFT);
    }
    super.makeMove(loc);
  }
}
```

/hich of the proposed implementations will correctly implement the `DancingCritter` class?

A) I only

B) II only

C) III only

D) I and II only

E) I, II, and III

26. Consider the following code segment.

```
int k = 0;
while (k < 10)
{
  System.out.print((k % 3) + "  ");
  if ((k % 3) == 0)
    k = k + 2;
  else
    k++;
}
```

What is printed as a result of executing the code segment?

(A) 0 2 1 0 2

(B) 0 2 0 2 0 2

(C) 0 2 1 0 2 1 0

(D) 0 2 0 2 0 2 0

(E) 0 1 2 1 2 1 2

GO ON TO THE NEXT PAG

. Consider the following method. Method `allEven` is intended to return `true` if all elements in array `arr` are even numbers; otherwise, it should return `false`.

```
public boolean allEven(int[] arr)
{
  boolean isEven = /* expression */ ;

  for (int k = 0; k < arr.length; k++)
  {
    /* loop body */
  }

  return isEven;
}
```

Which of the following replacements for `/* expression */` and `/* loop body */` should be used so that method `allEven` will work as intended?

	/* expression */	/* loop body */
(A)	false	`if ((arr[k] % 2) == 0)` `  isEven = true;`

(B)	false	`if ((arr[k] % 2) != 0)`
		`  isEven = false;`
		`else`
		`  isEven = true;`

(C)	true	`if ((arr[k] % 2) != 0)` `  isEven = false;`

(D)	true	`if ((arr[k] % 2) != 0)`
		`  isEven = false;`
		`else`
		`  isEven = true;`

(E)	true	`if ((arr[k] % 2) == 0)`
		`  isEven = false;`
		`else`
		`  isEven = true;`

28. Consider the following code segment.

```
int x = /* some integer value */ ;
int y = /* some integer value */ ;

boolean result = (x < y);

result = ( (x >= y) && !result );
```

Which of the following best describes the conditions under which the value of `result` will be `true` after the code segment is executed?

(A) Only when `x < y`

(B) Only when `x >= y`

(C) Only when `x` and `y` are equal

(D) The value will always be `true`.

(E) The value will never be `true`.

---

29. Consider the following code segment.

```
for (int outer = 0; outer < n; outer++)
{
  for (int inner = 0; inner <= outer; inner++)
  {
    System.out.print(outer + " ");
  }
}
```

If `n` has been declared as an integer with the value 4, what is printed as a result of executing the code segment?

(A) `0 1 2 3`

(B) `0 0 1 0 1 2`

(C) `0 1 2 2 3 3 3`

(D) `0 1 1 2 2 2 3 3 3 3`

(E) `0 0 1 0 1 2 0 1 2 3`

GO ON TO THE NEXT PAG

30. Consider the following class declarations.

```
public class Base
{
  private int myVal;

  public Base()
  {  myVal = 0;  }

  public Base(int x)
  {  myVal = x;  }
}


public class Sub extends Base
{
  public Sub()
  {  super(0);  }
}
```

Which of the following statements will NOT compile?

(A) `Base b1 = new Base();`

(B) `Base b2 = new Base(5);`

(C) `Base s1 = new Sub();`

(D) `Sub s2 = new Sub();`

(E) `Sub s3 = new Sub(5);`

---

31. Assume that `a` and `b` are variables of type `int`. The expression

```
!(a < b) && !(a > b)
```

is equivalent to which of the following?

(A) `true`

(B) `false`

(C) `a == b`

(D) `a != b`

(E) `!(a < b) && (a > b)`

32. Consider the following code segment.

```
int a = 24;
int b = 30;
while (b != 0)
{
  int r = a % b;
  a = b;
  b = r;
}

System.out.println(a);
```

What is printed as a result of executing the code segment?

(A) 0

(B) 6

(C) 12

(D) 24

(E) 30

GO ON TO THE NEXT PAGE

3. Consider the following method.

```
public int sol(int lim)
{
  int s = 0;

  for (int outer = 1; outer <= lim; outer++)
  {
    for (int inner = outer; inner <= lim; inner++)
    {
      s++;
    }
  }

  return s;
}
```

What value is returned as a result of the call `sol(10)` ?

(A) 20

(B) 45

(C) 55

(D) 100

(E) 385

34. Consider the following incomplete method. Method `findNext` is intended to return the index of the first occurrence of the value `val` beyond the position `start` in array `arr`.

```
// returns index of first occurrence of val in arr
// after position start;
// returns arr.length if val is not found
public int findNext(int[] arr, int val, int start)
{
  int pos = start + 1;

  while ( /* condition */ )
    pos++;

  return pos;
}
```

For example, consider the following code segment.

```
int[] arr = {11, 22, 100, 33, 100, 11, 44, 100};

System.out.println(findNext(arr, 100, 2));
```

The execution of the code segment should result in the value 4 being printed.

Which of the following expressions could be used to replace `/* condition */` so that `findNext` will work as intended?

(A) `(pos < arr.length) && (arr[pos] != val)`

(B) `(arr[pos] != val) && (pos < arr.length)`

(C) `(pos < arr.length) || (arr[pos] != val)`

(D) `(arr[pos] == val) && (pos < arr.length)`

(E) `(pos < arr.length) || (arr[pos] == val)`

5. Consider the following code segments.

I.
```
int k = 1;
while (k < 20)
{
  if (k % 3 == 1)
    System.out.print( k + "  ");

  k = k + 3;
}
```

II.
```
for (int k = 1; k < 20; k++)
{
  if (k % 3 == 1)
    System.out.print( k + "  ");
}
```

III.
```
for (int k = 1; k < 20; k = k + 3)
{
  System.out.print( k + "  ");
}
```

Which of the code segments above will produce the following output?

```
1  4  7  10  13  16  19
```

(A) I only

(B) II only

(C) I and II only

(D) II and III only

(E) I, II, and III

36. Consider the following two methods that appear within a single class.

```
public void changeIt(int[] list, int num)
{
  list = new int[5];
  num = 0;

  for (int x = 0; x < list.length; x++)
    list[x] = 0;
}


public void start()
{
  int[] nums = {1, 2, 3, 4, 5};
  int value = 6;

  changeIt(nums, value);

  for (int k = 0; k < nums.length; k++)
    System.out.print(nums[k] + " ");

  System.out.print(value);
}
```

What is printed as a result of the call `start()` ?

(A) `0 0 0 0 0 0`

(B) `0 0 0 0 0 6`

(C) `1 2 3 4 5 6`

(D) `1 2 3 4 5 0`

(E) `changeIt` will throw an exception.

7. Consider the following declaration of the class `NumSequence`, which has a constructor that is intended to initialize the instance variable `seq` to an `ArrayList` of `numberOfValues` random floating-point values in the range [0.0, 1.0).

```
public class NumSequence
{
  private ArrayList<Double> seq;

  // precondition:  numberOfValues > 0
  // postcondition: seq has been initialized to an ArrayList of
  //                length numberOfValues; each element of seq
  //                contains a random Double in the range [0.0, 1.0)
  public NumSequence(int numberOfValues)
  {
    /* missing code */
  }
}
```

Which of the following code segments could be used to replace `/* missing code */` so that the constructor will work as intended?

I.
```
ArrayList<Double> seq = new ArrayList<Double>();
for (int k = 0; k < numberOfValues; k++)
  seq.add(new Double(Math.random()));
```

II.
```
seq = new ArrayList<Double>();
for (int k = 0; k < numberOfValues; k++)
  seq.add(new Double(Math.random()));
```

III.
```
ArrayList<Double> temp = new ArrayList<Double>();
for (int k = 0; k < numberOfValues; k++)
  temp.add(new Double(Math.random()));

seq = temp;
```

(A) II only

(B) III only

(C) I and II

(D) I and III

(E) II and III

38. Consider the following code segment.

```
double a = 1.1;
double b = 1.2;

if ((a + b) * (a - b) != (a * a) - (b * b))
{
  System.out.println("Mathematical error!");
}
```

Which of the following best describes why the phrase `"Mathematical error!"` would be printed? (Remember that mathematically $(a + b) * (a - b) = a^2 - b^2$.)

(A) Precedence rules make the `if` condition true.

(B) Associativity rules make the `if` condition true.

(C) Roundoff error makes the `if` condition true.

(D) Overflow makes the `if` condition true.

(E) A compiler bug or hardware error has occurred.

---

39. Consider the following recursive method.

```
public static String recur(int val)
{
  String dig = "" + (val % 3);

  if (val / 3 > 0)
    return dig + recur(val / 3);

  return dig;
}
```

What is printed as a result of executing the following statement?

```
System.out.println(recur(32));
```

(A) 20

(B) 102

(C) 210

(D) 1020

(E) 2101

GO ON TO THE NEXT PAG

0. Consider the following method.

```
public String goAgain(String str, int index)
{
  if (index >= str.length())
    return str;

  return str + goAgain(str.substring(index), index + 1);
}
```

What is printed as a result of executing the following statement?

```
System.out.println(goAgain("today", 1));
```

(A) `today`

(B) `todayto`

(C) `todayoday`

(D) `todayodayay`

(E) `todayodaydayayy`

END OF SECTION I

**IF YOU FINISH BEFORE TIME IS CALLED,
YOU MAY CHECK YOUR WORK ON THIS SECTION.**

DO NOT GO ON TO SECTION II UNTIL YOU ARE TOLD TO DO SO.

NO TEST MATERIAL ON THIS PAGE

Content of Appendixes

Appendix A — A Exam Java Quick Reference

Accessible Methods from the Java Library That May Be Included on the Exam

class java.lang.Object

- `boolean equals(Object other)`
- `String toString()`

class java.lang.Integer

- `Integer(int value)`
- `int intValue()`

class java.lang.Double

- `Double(double value)`
- `double doubleValue()`

class java.lang.String

- `int length()`
- `String substring(int from, int to)` // returns the substring beginning at `from` // and ending at `to-1`
- `String substring(int from)` // returns `substring(from, length())`
- `int indexOf(String str)` // returns the index of the first occurrence of `str`; // returns `-1` if not found
- `int compareTo(String other)` // returns a value < 0 if `this` is less than `other` // returns a value = 0 if `this` is equal to `other` // returns a value > 0 if `this` is greater than `other`

class java.lang.Math

- `static int abs(int x)`
- `static double abs(double x)`
- `static double pow(double base, double exponent)`
- `static double sqrt(double x)`
- `static double random()` // returns a `double` in the range [0.0, 1.0)

class java.util.ArrayList<E>

- `int size()`
- `boolean add(E obj)` // appends `obj` to end of list; returns `true`
- `void add(int index, E obj)` // inserts `obj` at position `index` (0 ≤ `index` ≤ size), // moving elements at position `index` and higher // to the right (adds 1 to their indices) and adjusts size
- `E get(int index)`
- `E set(int index, E obj)` // replaces the element at position `index` with `obj` // returns the element formerly at the specified position
- `E remove(int index)` // removes element from position `index`, moving elements // at position `index + 1` and higher to the left // (subtracts 1 from their indices) and adjusts size // returns the element formerly at the specified position

Appendix B — Testable API

`info.gridworld.grid.Location` class (`implements Comparable`)

`public Location(int r, int c)`
constructs a location with given row and column coordinates

`public int getRow()`
returns the row of this location

`public int getCol()`
returns the column of this location

`public Location getAdjacentLocation(int direction)`
returns the adjacent location in the direction that is closest to `direction`

`public int getDirectionToward(Location target)`
returns the closest compass direction from this location toward `target`

`public boolean equals(Object other)`
returns `true` if `other` is a `Location` with the same row and column as this location; `false` otherwise

`public int hashCode()`
returns a hash code for this location

`public int compareTo(Object other)`
returns a negative integer if this location is less than `other`, zero if the two locations are equal, or a positive integer if this location is greater than `other`. Locations are ordered in row-major order.
Precondition: `other` is a `Location` object.

`public String toString()`
returns a string with the row and column of this location, in the format (row, col)

Compass directions:

```
public static final int NORTH = 0;
public static final int EAST = 90;
public static final int SOUTH = 180;
public static final int WEST = 270;
public static final int NORTHEAST = 45;
public static final int SOUTHEAST = 135;
public static final int SOUTHWEST = 225;
public static final int NORTHWEST = 315;
```

Turn angles:

```
public static final int LEFT = -90;
public static final int RIGHT = 90;
public static final int HALF_LEFT = -45;
public static final int HALF_RIGHT = 45;
public static final int FULL_CIRCLE = 360;
public static final int HALF_CIRCLE = 180;
public static final int AHEAD = 0;
```

info.gridworld.grid.Grid<E> interface

`int getNumRows()`
returns the number of rows, or -1 if this grid is unbounded

`int getNumCols()`
returns the number of columns, or -1 if this grid is unbounded

`boolean isValid(Location loc)`
returns `true` if `loc` is valid in this grid, `false` otherwise
Precondition: `loc` is not `null`

`E put(Location loc, E obj)`
puts `obj` at location `loc` in this grid and returns the object previously at that location (or `null` if the location was previously unoccupied).
Precondition: (1) `loc` is valid in this grid (2) `obj` is not `null`

`E remove(Location loc)`
removes the object at location `loc` from this grid and returns the object that was removed (or `null` if the location is unoccupied)
Precondition: `loc` is valid in this grid

`E get(Location loc)`
returns the object at location `loc` (or `null` if the location is unoccupied)
Precondition: `loc` is valid in this grid

`ArrayList<Location> getOccupiedLocations()`
returns an array list of all occupied locations in this grid

`ArrayList<Location> getValidAdjacentLocations(Location loc)`
returns an array list of the valid locations adjacent to `loc` in this grid
Precondition: `loc` is valid in this grid

`ArrayList<Location> getEmptyAdjacentLocations(Location loc)`
returns an array list of the valid empty locations adjacent to `loc` in this grid
Precondition: `loc` is valid in this grid

`ArrayList<Location> getOccupiedAdjacentLocations(Location loc)`
returns an array list of the valid occupied locations adjacent to `loc` in this grid
Precondition: `loc` is valid in this grid

`ArrayList<E> getNeighbors(Location loc)`
returns an array list of the objects in the occupied locations adjacent to `loc` in this grid
Precondition: `loc` is valid in this grid

info.gridworld.actor.Actor class

`public Actor()`
constructs a blue actor that is facing north

`public Color getColor()`
returns the color of this actor

`public void setColor(Color newColor)`
sets the color of this actor to `newColor`

`public int getDirection()`
returns the direction of this actor, an angle between 0 and 359 degrees

`public void setDirection(int newDirection)`
sets the direction of this actor to the angle between 0 and 359 degrees that is equivalent to `newDirection`

`public Grid<Actor> getGrid()`
returns the grid of this actor, or `null` if this actor is not contained in a grid

`public Location getLocation()`
returns the location of this actor, or `null` if this actor is not contained in a grid

`public void putSelfInGrid(Grid<Actor> gr, Location loc)`
puts this actor into location `loc` of grid `gr`. If there is another actor at `loc`, it is removed.
Precondition: (1) This actor is not contained in a grid (2) `loc` is valid in `gr`

`public void removeSelfFromGrid()`
removes this actor from its grid.
Precondition: this actor is contained in a grid

`public void moveTo(Location newLocation)`
moves this actor to `newLocation`. If there is another actor at `newLocation`, it is removed.
Precondition: (1) This actor is contained in a grid (2) `newLocation` is valid in the grid of this actor

`public void act()`
reverses the direction of this actor. Override this method in subclasses of `Actor` to define types of actors with different behavior

`public String toString()`
returns a string with the location, direction, and color of this actor

info.gridworld.actor.Rock class (extends Actor)

`public Rock()`
constructs a black rock

`public Rock(Color rockColor)`
constructs a rock with color `rockColor`

`public void act()`
overrides the `act` method in the `Actor` class to do nothing

info.gridworld.actor.Flower class (extends Actor)

`public Flower()`
constructs a pink flower

`public Flower(Color initialColor)`
constructs a flower with color `initialColor`

`public void act()`
causes the color of this flower to darken

Appendix C — Testable Code for APCS A/AB

3ug.java

```
ackage info.gridworld.actor;

mport info.gridworld.grid.Grid;
mport info.gridworld.grid.Location;

mport java.awt.Color;

**
* A Bug is an actor that can move and turn. It drops flowers as it moves.
* The implementation of this class is testable on the AP CS A and AB Exams.
*/
ublic class Bug extends Actor

 /**
  * Constructs a red bug.
  */
 public Bug()
 {
   setColor(Color.RED);
 }


 /**
  * Constructs a bug of a given color.
  * @param bugColor the color for this bug
  */
 public Bug(Color bugColor)
 {
   setColor(bugColor);
 }


 /**
  * Moves if it can move, turns otherwise.
  */
 public void act()
 {
   if (canMove())
     move();
   else
     turn();
 }


 /**
  * Turns the bug 45 degrees to the right without changing its location.
  */
 public void turn()
 {
   setDirection(getDirection() + Location.HALF_RIGHT);
 }
```

```
  /**
   * Moves the bug forward, putting a flower into the location it previously occupied.
   */
  public void move()
  {
    Grid<Actor> gr = getGrid();
    if (gr == null)
      return;
    Location loc = getLocation();
    Location next = loc.getAdjacentLocation(getDirection());
    if (gr.isValid(next))
      moveTo(next);
    else
      removeSelfFromGrid();
    Flower flower = new Flower(getColor());
    flower.putSelfInGrid(gr, loc);
  }


  /**
   * Tests whether this bug can move forward into a location that is empty or contains a flower.
   * @return true if this bug can move.
   */
  public boolean canMove()
  {
    Grid<Actor> gr = getGrid();
    if (gr == null)
      return false;
    Location loc = getLocation();
    Location next = loc.getAdjacentLocation(getDirection());
    if (!gr.isValid(next))
      return false;
    Actor neighbor = gr.get(next);
    return (neighbor == null) || (neighbor instanceof Flower);
    // ok to move into empty location or onto flower
    // not ok to move onto any other actor
  }
}
```

oxBug.java

```
mport info.gridworld.actor.Bug;

**
* A BoxBug traces out a square "box" of a given size.
* The implementation of this class is testable on the AP CS A and AB Exams.
*/
ublic class BoxBug extends Bug
{
 private int steps;
 private int sideLength;

 /**
  * Constructs a box bug that traces a squarc of a given side length
  * @param length the side length
  */
public BoxBug(int length)
 {
   steps = 0;
   sideLength = length;
 }

   /**
    * Moves to the next location of the square.
    */
public void act()
 {
   if (steps < sideLength && canMove())
   {
     move();
     steps++;
   }
   else
   {
     turn();
     turn();
     steps = 0;
   }
 }
```

Critter.java

```
package info.gridworld.actor;

import info.gridworld.grid.Location;
import java.util.ArrayList;

/**
 * A Critter is an actor that moves through its world, processing
 * other actors in some way and then moving to a new location.
 * Define your own critters by extending this class and overriding any methods of this class except for act.
 * When you override these methods, be sure to preserve the postconditions.
 * The implementation of this class is testable on the AP CS A and AB Exams.
 */
public class Critter extends Actor
{
  /**
   * A critter acts by getting a list of other actors, processing that list, getting locations to move to,
   * selecting one of them, and moving to the selected location.
   */
  public void act()
  {
    if (getGrid() == null)
      return;
    ArrayList<Actor> actors = getActors();
    processActors(actors);
    ArrayList<Location> moveLocs = getMoveLocations();
    Location loc = selectMoveLocation(moveLocs);
    makeMove(loc);
  }


  /**
   * Gets the actors for processing. Implemented to return the actors that occupy neighboring grid locations.
   * Override this method in subclasses to look elsewhere for actors to process.
   * Postcondition: The state of all actors is unchanged.
   * @return a list of actors that this critter wishes to process.
   */
  public ArrayList<Actor> getActors()
  {
    return getGrid().getNeighbors(getLocation());
  }
```

```
/**
 * Processes the elements of actors. New actors may be added to empty locations.
 * Implemented to "eat" (i.e., remove) selected actors that are not rocks or critters.
 * Override this method in subclasses to process actors in a different way.
 * Postcondition: (1) The state of all actors in the grid other than this critter and the
 * elements of actors is unchanged. (2) The location of this critter is unchanged.
 * @param actors the actors to be processed
 */
public void processActors(ArrayList<Actor> actors)
{
  for (Actor a : actors)
  {
    if (!(a instanceof Rock) && !(a instanceof Critter))
      a.removeSelfFromGrid();
  }
}

/**
 * Gets a list of possible locations for the next move. These locations must be valid in the grid of this critter.
 * Implemented to return the empty neighboring locations. Override this method in subclasses to look
 * elsewhere for move locations.
 * Postcondition: The state of all actors is unchanged.
 * @return a list of possible locations for the next move
 */
public ArrayList<Location> getMoveLocations()
{
  return getGrid().getEmptyAdjacentLocations(getLocation());
}

/**
 * Selects the location for the next move. Implemented to randomly pick one of the possible locations,
 * or to return the current location if locs has size 0. Override this method in subclasses that
 * have another mechanism for selecting the next move location.
 * Postcondition: (1) The returned location is an element of locs, this critter's current location, or null.
 * (2) The state of all actors is unchanged.
 * @param locs the possible locations for the next move
 * @return the location that was selected for the next move.
 */
public Location selectMoveLocation(ArrayList<Location> locs)
{
  int n = locs.size();
  if (n == 0)
    return getLocation();
  int r = (int) (Math.random() * n);
  return locs.get(r);
}
```

```
  /**
   * Moves this critter to the given location loc, or removes this critter from its grid if loc is null.
   * An actor may be added to the old location. If there is a different actor at location loc, that actor is
   * removed from the grid. Override this method in subclasses that want to carry out other actions
   * (for example, turning this critter or adding an occupant in its previous location).
   * Postcondition: (1) getLocation() == loc.
   * (2) The state of all actors other than those at the old and new locations is unchanged.
   * @param loc the location to move to
   */
  public void makeMove(Location loc)
  {
    if (loc == null)
      removeSelfFromGrid();
    else
      moveTo(loc);
  }
}
```

ChameleonCritter.java

```
import info.gridworld.actor.Actor;
import info.gridworld.actor.Critter;
import info.gridworld.grid.Location;

import java.util.ArrayList;

/**
 * A ChameleonCritter takes on the color of neighboring actors as it moves through the grid.
 * The implementation of this class is testable on the AP CS A and AB Exams.
 */
public class ChameleonCritter extends Critter
{
  /**
   * Randomly selects a neighbor and changes this critter's color to be the same as that neighbor's.
   * If there are no neighbors, no action is taken.
   */
  public void processActors(ArrayList<Actor> actors)
  {
    int n = actors.size();
    if (n == 0)
      return;
    int r = (int) (Math.random()  * n);

    Actor other = actors.get(r);
    setColor(other.getColor());
  }

  /**
   * Turns towards the new location as it moves.
   */
  public void makeMove(Location loc)
  {
    setDirection(getLocation().getDirectionToward(loc));
    super.makeMove(loc);
  }
}
```

Appendix E — Quick Reference A/AB

Location Class (implements Comparable)

```
public Location(int r, int c)
public int getRow()
public int getCol()
public Location getAdjacentLocation(int direction)
public int getDirectionToward(Location target)
public boolean equals(Object other)
public int hashCode()
public int compareTo(Object other)
public String toString()

NORTH, EAST, SOUTH, WEST, NORTHEAST, SOUTHEAST, NORTHWEST, SOUTHWEST
LEFT, RIGHT, HALF_LEFT, HALF_RIGHT, FULL_CIRCLE, HALF_CIRCLE, AHEAD
```

Grid<E> Interface

```
int getNumRows()
int getNumCols()
boolean isValid(Location loc)
 put(Location loc, E obj)
 remove(Location loc)
 get(Location loc)
ArrayList<Location> getOccupiedLocations()
ArrayList<Location> getValidAdjacentLocations(Location loc)
ArrayList<Location> getEmptyAdjacentLocations(Location loc)
ArrayList<Location> getOccupiedAdjacentLocations(Location loc)
ArrayList<E> getNeighbors(Location loc)
```

Actor Class

```
ublic Actor()
ublic Color getColor()
ublic void setColor(Color newColor)
ublic int getDirection()
ublic void setDirection(int newDirection)
ublic Grid<Actor> getGrid()
ublic Location getLocation()
ublic void putSelfInGrid(Grid<Actor> gr, Location loc)
ublic void removeSelfFromGrid()
ublic void moveTo(Location newLocation)
ublic void act()
ublic String toString()
```

Rock Class (extends Actor)

```
public Rock()
public Rock(Color rockColor)
public void act()
```

Flower Class (extends Actor)

```
public Flower()
public Flower(Color initialColor)
public void act()
```

Bug Class (extends Actor)

```
public Bug()
public Bug(Color bugColor)
public void act()
public void turn()
public void move()
public boolean canMove()
```

BoxBug Class (extends Bug)

```
public BoxBug(int n)
public void act()
```

Critter Class (extends Actor)

```
public void act()
public ArrayList<Actor> getActors()
public void processActors(ArrayList<Actor> actors)
public ArrayList<Location> getMoveLocations()
public Location selectMoveLocation(ArrayList<Location> locs)
public void makeMove(Location loc)
```

ChameleonCritter Class (extends Critter)

```
public void processActors(ArrayList<Actor> actors)
public void makeMove(Location loc)
```

Appendix G — Index for Source Code

This appendix provides an index for the Java source code found in Appendix C.

Bug.java

BoxBug.java

Critter.java

ChameleonCritter.java

AP® Computer Science A Exam

SECTION II: Free-Response Questions

2009

DO NOT OPEN THIS BOOKLET UNTIL YOU ARE TOLD TO DO SO.

At a Glance

Total Time
1 hour, 45 minutes
Number of Questions
4
Percent of Total Grade
50%
Writing Instrument
Pencil
Electronic Device
None allowed
Weight
The questions are weighted equally.

IMPORTANT Identification Information

PLEASE PRINT WITH PEN:

1. First two letters of your last name ☐☐
   First letter of your first name ☐
2. Date of birth
   ☐☐ ☐☐ ☐☐☐☐
   Month Day Year
3. Six-digit school code
   ☐☐☐☐☐☐
4. Unless I check the box below, I grant the College Board the unlimited right to use, reproduce, and publish my free-response materials, both written and oral, for educational research and instructional purposes. My name and the name of my school will not be used in any way in connection with my free-response materials. I understand that I am free to mark "No" with no effect on my grade or its reporting.
   No, I do not grant the College Board these rights. ☐

Instructions

The questions for Section II are printed in the green insert and in this booklet. You may use the insert to organize your answers and for scratch work, but you must write your answers in this pink Section II booklet. No credit will be given for any work written in the insert. The orange Appendix booklet is provided for use in Section II.

Write your answer to each question in the space provided in the Section II booklet. Some questions require you to write program segments, and these must be written in Java. Show all your work. Credit for partial solutions will be given. Write clearly and legibly. Cross out any errors you make. Erased or crossed-out work will not be graded.

Assume classes listed in the Quick Reference found in the Appendix are included in any program that uses a program segment you write. If other classes are to be included, that information will be specified in individual questions. Unless otherwise noted, assume that all methods are invoked only when their preconditions are satisfied. A Quick Reference to required Java classes is included in the Appendix.

Manage your time carefully. Do not spend too much time on any one question. You may proceed freely from one question to the next. You may review your responses if you finish before the end of the exam is announced.

COMPUTER SCIENCE A
SECTION II

Time—1 hour and 45 minutes

Number of questions—4

Percent of total grade—50

rections: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE RITTEN IN JAVA.

tes:

Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.

Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.

In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

Section II

1. A statistician is studying sequences of numbers obtained by repeatedly tossing a six-sided number cube. On each side of the number cube is a single number in the range of 1 to 6, inclusive, and no number is repeated on the cube. The statistician is particularly interested in runs of numbers. A run occurs when two or more consecutive tosses of the cube produce the same value. For example, in the following sequence of cube tosses, there are runs starting at positions 1, 6, 12, and 14.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Result	1	5	5	4	3	1	2	2	2	2	6	1	3	3	5	5	5	5

The number cube is represented by the following class.

```
public class NumberCube
{
  /** @return an integer value between 1 and 6, inclusive
   */
  public int toss()
  {  /* implementation not shown */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

You will implement a method that collects the results of several tosses of a number cube and another method that calculates the longest run found in a sequence of tosses.

GO ON TO THE NEXT PAG

Write the method `getCubeTosses` that takes a number cube and a number of tosses as parameters. The method should return an array of the values produced by tossing the number cube the given number of times.

Complete method `getCubeTosses` below.

```
/** Returns an array of the values obtained by tossing a number cube numTosses times.
 *  @param cube a NumberCube
 *  @param numTosses the number of tosses to be recorded
 *         Precondition: numTosses > 0
 *  @return an array of numTosses values
 */
public static int[] getCubeTosses(NumberCube cube, int numTosses)
```

Section II

(b) Write the method `getLongestRun` that takes as its parameter an array of integer values representing a series of number cube tosses. The method returns the starting index in the array of a run of maximum size. A run is defined as the repeated occurrence of the same value in two or more consecutive positions in the array.

For example, the following array contains two runs of length 4, one starting at index 6 and another starting at index 14. The method may return either of those starting indexes.

If there are no runs of any value, the method returns `-1`.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Result	1	5	5	4	3	1	2	2	2	2	6	1	3	3	5	5	5	5

WRITE YOUR SOLUTION ON THE NEXT PAGE.

mplete method `getLongestRun` below.

```
* Returns the starting index of a longest run of two or more consecutive repeated values
  in the array values.
  @param values an array of integer values representing a series of number cube tosses
         Precondition: values.length > 0
  @return the starting index of a run of maximum size;
          -1 if there is no run
/
blic static int getLongestRun(int[] values)
```

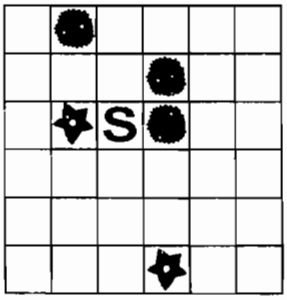
Section II

2. This question involves reasoning about the code from the GridWorld case study. A copy of the code is provided as part of this exam.

A `StockpileCritter` is a `Critter` that uses other actors as a source of energy. Each actor represents one unit of energy. The `StockpileCritter` behaves like a `Critter` except in the way that it interacts with other actors. Each time the `StockpileCritter` acts, it gathers all neighboring actors by removing them from the grid and keeps track of them in a stockpile. The `StockpileCritter` then attempts to reduce its stockpile by one unit of energy. If the stockpile is empty, the `StockpileCritter` runs out of energy and removes itself from the grid.

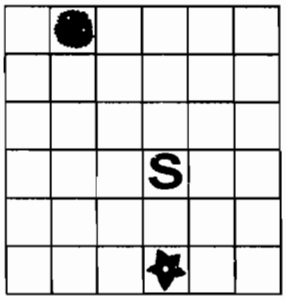
Consider the following scenario.

INITIAL WORLD



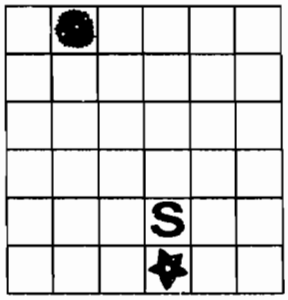
`StockpileCritter` is in location (2, 2), stockpile is empty

AFTER ONE ACT



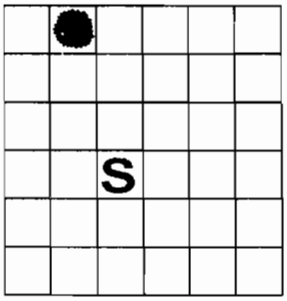
Gathered 3 actors, used 1 energy unit, 2 remaining in stockpile, moved to location (3, 3)

AFTER TWO ACTS



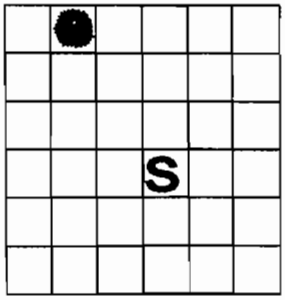
No actors gathered, used 1 energy unit, 1 remaining in stockpile, moved to location (4, 3)

AFTER THREE ACTS



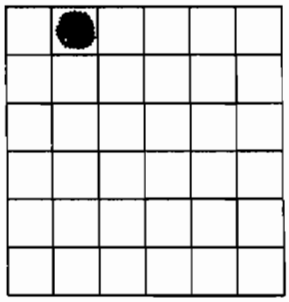
Gathered 1 actor, used 1 energy unit, 1 remaining in stockpile, moved to location (3, 2)

AFTER FOUR ACTS



No actors gathered, used 1 energy unit, 0 remaining in stockpile, moved to location (3, 3)

AFTER FIVE ACTS



Stockpile empty, removed self from grid

Write the complete `StockpileCritter` class, including all instance variables and required methods. Do NOT override the `act` method. Remember that your design must not violate the postconditions of the methods of the `Critter` class and that updating an object's instance variable changes the state of that object.

ADDITIONAL WORK SPACE

Section II

3. An electric car that runs on batteries must be periodically recharged for a certain number of hours. The battery technology in the car requires that the charge time not be interrupted.

   The cost for charging is based on the hour(s) during which the charging occurs. A rate table lists the 24 one-hour periods, numbered from 0 to 23, and the corresponding hourly cost for each period. The same rate table is used for each day. Each hourly cost is a positive integer. A sample rate table is given below.

Hour	Cost
0	50
1	60
2	160
3	60
4	80
5	100
6	100
7	120

Hour	Cost
8	150
9	150
10	150
11	200
12	40
13	240
14	220
15	220

Hour	Cost
16	200
17	200
18	180
19	180
20	140
21	100
22	80
23	60

The class `BatteryCharger` below uses a rate table to determine the most economic time to charge the battery. You will write two of the methods for the `BatteryCharger` class.

```
public class BatteryCharger
{
  /** rateTable has 24 entries representing the charging costs for hours 0 through 23. */
  private int[] rateTable;

  /** Determines the total cost to charge the battery starting at the beginning of startHour.
   *  @param startHour the hour at which the charge period begins
   *         Precondition: 0 ≤ startHour ≤ 23
   *  @param chargeTime the number of hours the battery needs to be charged
   *         Precondition: chargeTime > 0
   *  @return the total cost to charge the battery
   */
  private int getChargingCost(int startHour, int chargeTime)
  {  /* to be implemented in part (a) */  }

  /** Determines start time to charge the battery at the lowest cost for the given charge time.
   *  @param chargeTime the number of hours the battery needs to be charged
   *         Precondition: chargeTime > 0
   *  @return an optimal start time, with 0 ≤ returned value ≤ 23
   */
  public int getChargeStartTime(int chargeTime)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```


GO ON TO THE NEXT PAGE

a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour the hour at which the charge period begins
 *         Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
```

(b) Write the `BatteryCharger` method `getChargeStartTime` that returns the start time that will allow the battery to be charged at minimal cost. If there is more than one possible start time that produces the minimal cost, any of those start times can be returned.

For example, using the rate table given at the beginning of the question, the following table shows the resulting minimal costs and optimal starting hour of several possible charges.

Hours of Charge Time	Minimum Cost	Start Hour of Charge	Last Hour of Charge
1	40	12	12
2	110	0 **or** 23	1 **or** 0 (the next day)
7	550	22	4 (the next day)
30	3,710	22	3 (two days later)

WRITE YOUR SOLUTION ON THE NEXT PAGE.

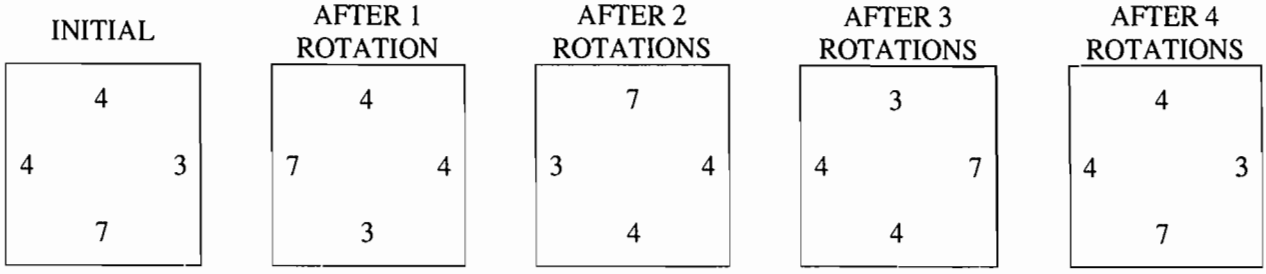
Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
```

Section II

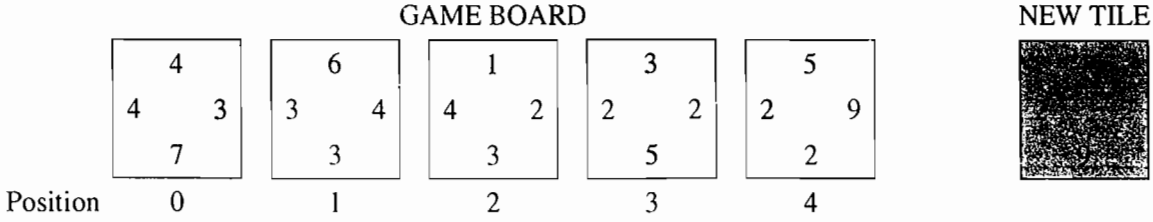
4. A game uses square tiles that have numbers on their sides. Each tile is labeled with a number on each of its four sides and may be rotated clockwise, as illustrated below.



The tiles are represented by the `NumberTile` class, as given below.

```
public class NumberTile
{
  /** Rotates the tile 90 degrees clockwise
   */
  public void rotate()
  {  /* implementation not shown */  }


  /** @return value at left edge of tile
   */
  public int getLeft()
  {  /* implementation not shown */  }


  /** @return value at right edge of tile
   */
  public int getRight()
  {  /* implementation not shown */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

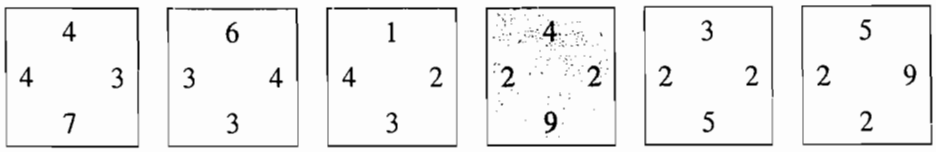
Tiles are placed on a game board so that the adjoining sides of adjacent tiles have the same number. The following figure illustrates an arrangement of tiles and shows a new tile that is to be placed on the game board.



GO ON TO THE NEXT PAGE

its original orientation, the new tile can be inserted between the tiles at positions 2 and 3 or between the tiles positions 3 and 4. If the new tile is rotated once, it can be inserted before the tile at position 0 (the first tile) or er the tile at position 4 (the last tile). Assume that the new tile, in its original orientation, is inserted between : tiles at positions 2 and 3. As a result of the insertion, the tiles at positions 3 and 4 are moved one location to : right, and the new tile is inserted at position 3, as shown below.

GAME BOARD AFTER INSERTING TILE



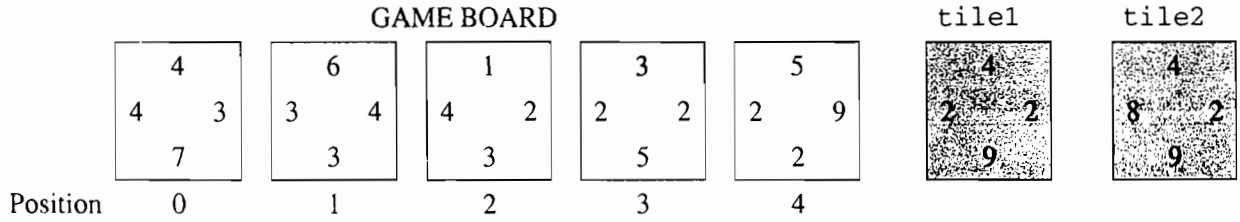
partial definition of the `TileGame` class is given below.

```
public class TileGame
{
  /** represents the game board; guaranteed never to be null */
  private ArrayList<NumberTile> board;

  public TileGame()
  { board = new ArrayList<NumberTile>(); }

  /** Determines where to insert tile, in its current orientation, into game board
   *  @param tile the tile to be placed on the game board
   *  @return the position of tile where tile is to be inserted:
   *            0 if the board is empty;
   *           -1 if tile does not fit in front, at end, or between any existing tiles;
   *           otherwise, 0 ≤ position returned ≤ board.size()
   */
  private int getIndexForFit(NumberTile tile)
  {  /* to be implemented in part (a) */  }

  /** Places tile on the game board if it fits (checking all possible tile orientations if necessary).
   *  If there are no tiles on the game board, the tile is placed at position 0.
   *  The tile should be placed at most 1 time.
   *  Precondition: board is not null
   *  @param tile the tile to be placed on the game board
   *  @return true if tile is placed successfully; false otherwise
   *  Postcondition: the orientations of the other tiles on the board are not changed
   *  Postcondition: the order of the other tiles on the board relative to each other is not changed
   */
  public boolean insertTile(NumberTile tile)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

Section II

(a) Write the `TileGame` method `getIndexForFit` that determines where a given tile, in its current orientation, fits on the game board. A tile can be inserted at either end of a game board or between two existing tiles if the side(s) of the new tile match the adjacent side(s) of the tile(s) currently on the game board. If there are no tiles on the game board, the position for the insert is 0. The method returns the position that the new tile will occupy on the game board after it has been inserted. If there are multiple possible positions for the tile, the method will return any one of them. If the given tile does not fit anywhere on the game board, the method returns `-1`.

For example, the following diagram shows a game board and two potential tiles to be placed. The call `getIndexForFit(tile1)` can return either 3 or 4 because `tile1` can be inserted between the tiles at positions 2 and 3, or between the tiles at positions 3 and 4. The call `getIndexForFit(tile2)` returns `-1` because `tile2`, in its current orientation, does not fit anywhere on the game board.



WRITE YOUR SOLUTION ON THE NEXT PAGE.

GO ON TO THE NEXT PAG

Complete method `getIndexForFit` below.

```
/** Determines where to insert tile, in its current orientation, into game board
 *  @param tile the tile to be placed on the game board
 *  @return the position of tile where tile is to be inserted:
 *              0 if the board is empty;
 *             -1 if tile does not fit in front, at end, or between any existing tiles;
 *             otherwise, 0 ≤ position returned ≤ board.size()
 */
private int getIndexForFit(NumberTile tile)
```

(b) Write the `TileGame` method `insertTile` that attempts to insert the given tile on the game board. The method returns `true` if the tile is inserted successfully and `false` only if the tile cannot be placed on the board in any orientation.

Assume that `getIndexForFit` works as specified, regardless of what you wrote in part (a).

Complete method `insertTile` below.

```
/** Places tile on the game board if it fits (checking all possible tile orientations if necessary).
 *  If there are no tiles on the game board, the tile is placed at position 0.
 *  The tile should be placed at most 1 time.
 *  Precondition: board is not null
 *  @param tile the tile to be placed on the game board
 *  @return true if tile is placed successfully; false otherwise
 *  Postcondition: the orientations of the other tiles on the board are not changed
 *  Postcondition: the order of the other tiles on the board relative to each other is not changed
 */
public boolean insertTile(NumberTile tile)
```

STOP

END OF EXAM

Chapter III: Answers to the 2009 AP Computer Science A Exam

Section I: Multiple Choice
- Section I Answer Key and Percent Answering Correctly
- Analyzing Your Students' Performance on the Multiple-Choice Section
- Diagnostic Guide for the 2009 AP Computer Science A Exam

Section II: Free Response
- Comments from the Chief Reader
- Scoring Guidelines, Sample Student Responses, and Commentary

Section I: Multiple Choice

Listed below are the correct answers to the multiple-choice questions, the percent of AP students who answered each question correctly by AP score, and the total percent answering correctly.

Section I Answer Key and Percent Answering Correctly

Item No.	Correct Answer	Percent Correct by Score					Total Percent Correct
		5	4	3	2	1	
1	A	98	97	96	96	79	92
2	A	98	95	93	89	73	89
3	D	99	96	90	84	50	81
4	C	97	93	88	81	56	81
5	A	95	86	76	73	53	76
6	A	93	80	69	60	30	65
7	D	88	65	42	28	16	50
8	D	99	96	91	85	63	85
9	C	69	47	39	31	20	42
10	D	92	85	77	74	49	74
11	A	96	83	65	47	27	64
12	B	96	93	91	88	57	82
13	C	73	35	17	8	13	33
14	D	86	64	44	37	23	52
15	B	83	69	63	58	44	63
16	C	72	49	40	36	36	48
17	B	77	48	32	19	10	39
18	A	92	78	67	61	42	68
19	D	95	74	55	41	19	57
20	E	96	85	69	58	30	67

Item No.	Correct Answer	Percent Correct by Score					Total Percent Correct
		5	4	3	2	1	
21	B	83	66	56	47	31	56
22	E	88	71	56	44	20	56
23	C	93	83	75	66	50	73
24	A	96	88	78	71	44	74
25	C	47	23	19	20	20	27
26	D	91	75	62	54	29	61
27	C	90	61	34	23	15	47
28	B	88	68	52	44	25	56
29	D	84	71	52	37	18	53
30	E	93	79	61	45	21	60
31	C	88	68	61	57	46	64
32	B	86	62	47	37	25	52
33	C	80	58	37	27	15	45
34	A	67	41	28	22	14	36
35	E	92	76	61	54	31	63
36	C	37	20	13	14	19	22
37	E	63	29	12	7	6	26
38	C	44	25	19	18	16	25
39	E	41	19	12	8	5	18
40	D	52	25	15	11	8	24

Analyzing Your Students' Performance on the Multiple-Choice Section

If you give your students the 2009 exam for practice, you may want to analyze the results to find overall strengths and weaknesses in their understanding of AP Computer Science A. The following diagnostic worksheet will help you do this. You are permitted to photocopy and distribute it to your students for completion.

1. In each section, students should insert a check mark for each correct answer.
2. Add together the total number of correct answers for each section.
3. To compare the student's number of correct answers for each section with the average number correct for that section, copy the number of correct answers to the "Number Correct" table at the end of the Diagnostic Guide.

In addition, under each item the percent of AP students who answered correctly is shown, so students can analyze their performance on individual items. This information will be helpful in deciding how students should plan their study time. Please note that one item may appear in several different categories, as questions can cross over different topics.

)iagnostic Guide for the 2009 AP Computer Science A Exam

'rogramming Fundamentals (Average number correct = 17.43)

Question #	1	2	3	4	6	7	8	9	10	11	12	13	14	15	17
Correct/Incorrect															
Percent of Students Answering Correctly	*92*	*89*	*81*	*81*	*65*	*50*	*85*	*42*	*74*	*64*	*82*	*33*	*52*	*63*	*39*

Question #	18	19	20	21	23	26	27	29	32	33	34	35	36	37	40
Correct/Incorrect															
Percent of Students Answering Correctly	*68*	*57*	*67*	*56*	*73*	*61*	*47*	*53*	*52*	*45*	*36*	*63*	*22*	*26*	*24*

)ata Structures (Average number correct = 3.97)

Question #	3	4	11	17	27	34	36	37
Correct/Incorrect								
Percent of Students Answering Correctly	*81*	*81*	*64*	*39*	*47*	*36*	*22*	*26*

ogic (Average number correct = 2.54)

Question #	18	20	28	31
Correct/Incorrect				
Percent of Students Answering Correctly	*68*	*67*	*56*	*64*

lgorithms/Problem Solving (Average number correct = 8.83)

Question #	1	2	3	6	7	8	12	14	15	17	22	25	27	28
Correct/Incorrect														
Percent of Students Answering Correctly	*92*	*89*	*81*	*65*	*50*	*85*	*82*	*52*	*63*	*39*	*56*	*27*	*47*	*56*

bject-Oriented Programming (Average number correct = 4.14)

Question #	5	16	22	23	24	25	30
Correct/Incorrect							
Percent of Students Answering Correctly	*76*	*48*	*56*	*73*	*74*	*27*	*60*

Diagnostic Guide for the 2009 AP Computer Science A Exam *(continued)*

Recursion (Average number correct = 0.74)

Question #	13	39	40
Correct/Incorrect			
Percent of Students Answering Correctly	*33*	*18*	*24*

Software Engineering (Average number correct = 0.25)

Question #	38
Correct/Incorrect	
Percent of Students Answering Correctly	*25*

Case Study (Average number correct = 2.86)

Question #	21	22	23	24	25
Correct/Incorrect					
Percent of Students Answering Correctly	*56*	*56*	*73*	*74*	*27*

Number Correct

	Programming Fundamentals	Data Structures	Logic	Algorithms/ Problem Solving	Object-Oriented Programming	Recursion	Software Engineering	Case Stu
Number of Questions	30	8	4	14	7	3	1	5
Average Number Correct	17.43 (58.1%)	3.97 (49.6%)	2.54 (63.5%)	8.83 (63.1%)	4.14 (59.1%)	0.74 (24.7%)	0.25 (25.0%)	2.86 (57.2%)
My Number Correct								

Section II: Free Response

Comments from the Chief Reader

Jody Paul
Department of Computer Science
Metropolitan State College of Denver
Denver, Colorado

The free-response section of the 2009 AP Computer Science A Exam is comparable in content, structure, and difficulty to exams from the preceding several years. It consists of 4 multi-part questions that afforded performance-based assessment of fundamental computer science skills and concepts from the curriculum defined in the *AP Computer Science A Course Description*. The free-response questions were scored according to rigorous standards and procedures, and the resultant scores comprised 50 percent of the composite score for the exam.

All questions required students to read and analyze problem descriptions and to design and generate programs using the Java programming language. Question 2 required students to design and implement a complete Java class derived from one defined in the *GridWorld AP Computer Science Case Study*. All others required writing the bodies of Java methods for which signatures were provided.

Question 1 focused on the array data structure, its construction and traversal, the application of basic algorithms, and method invocation for a specified object. Question 2 involved reasoning about code from the GridWorld case study, emphasizing object-oriented concepts. Question 3 focused on array traversal, abstraction, and algorithms for accumulation and finding a minimum. Question 4 focused on object-oriented programming, algorithm development and list processing, and required significant problem analysis and algorithm design before writing code. The dominant concepts assessed included the following:

- interpretation of textual problem statements (a problem-solving skill)
- object-oriented design and programming
- adhering to specified interfaces and constraints
- Java-based array construction, traversal and element access
- manipulation and element access of Java's `ArrayList` objects
- translation of algorithms into Java programming language code
- implementation of common algorithmic tasks (such as finding maximum or minimum values)
- numerical iteration using Java's `for` or `while` loop constructs
- correct use of the Java programming language

The most prevalent errors noticed during scoring included:

- failure to meet required solution specifications (such as violations of well-specified constraints or failure to return the specified result)
- incorrect use of nested `if/else` constructs
- incorrect array construction, traversal and element access
- difficulty working with boundary conditions of array and `ArrayList` data structures
- confusion of array and `ArrayList` indices with their elements
- loop boundary errors

A cautionary reminder: this set of questions, as any set of such questions, represents only a subset of the testable curriculum that may be assessed in the future. For example, although Question 2 involved extending the `Critter` class from the GridWorld case study, that case study includes a rich set of other classes and concepts upon which future questions may be based.

Scoring Guidelines, Sample Student Responses, and Commentary

The answers presented on the following pages are actual student responses to the free-response questions on the 2009 AP Computer Science A Exam. The students gave permission to have their work reproduced at the time they took the exam. These responses were read and scored by the Table Leaders and Readers assigned to each particular question during the AP Reading in June 2009. The actual scores that these student responses earned, as well as a brief explanation of why, are included.

2009 General Scoring Guidelines

Apply the question-specific rubric first. The question-specific rubric *always* takes precedence.

Refer to the *general error categorization* below only for cases not already covered by the question-specific rubric.

Points can only be deducted if the error occurs in a part which has earned credit via the question-specific rubric.

Any error included below is **penalized only once** in a question, even if it occurs on different parts of that question.

Special handling of minor errors (½ point): If a specific minor error (½ point) is the **only instance, one of two,** or occurs **two or more times,** then it **must be penalized as shown.** If a specific minor error occurs **exactly once** when the same concept is **correct two or more times,** then it is **not penalized** (regarded as an oversight).

Nonpenalized Errors

spelling/case discrepancies*

local variable not declared if other variables are declared in some part

use keyword as identifier

`[]` vs. `()` vs. `<>`

`=` instead of `==` (and vice versa)

`length/size` confusion for array, `String`, and `ArrayList`, with or without `()`

`private` qualifier on local variable

extraneous code with no side-effect; *e.g., precondition check*

common mathematical symbols for operators (× • ÷ ≤ ≥ < > ≠)

missing { } where indentation clearly conveys intent and { } used elsewhere

missing `;` where indentation clearly conveys intent and `;` used elsewhere

default constructor called without parens; *e.g.,* `new Fish;`

missing `( )` on parameter-less method call

missing `( )` around `if/while` conditions

missing `public` on class or constructor header

extraneous `[]` when referencing entire array

extraneous size in array declaration, *e.g.,* `int[size] nums = new int[size];`

Minor Errors (1/2 point)

confused identifier *(e.g.,* `len` *for* `length` *or* `left()` *for* `getLeft()` *)*

no local variables declared

missing `new` in constructor call

modifying a constant `(final)`

use `equals` or `compareTo` method on primitives, *e.g.,* `int x; ...x.equals(val)`

array/collection access confusion `([] get)`

assignment dyslexia, *e.g.,* `x + 3 = y;` *for* `y = x + 3;`

`super(method())` instead of `super.method()`

formal parameter syntax (with type) in method call, *e.g.,* `a = method(int x)`

missing { } where indentation clearly conveys intent and { } not used elsewhere

missing `;` where indentation clearly conveys intent and `;` not used elsewhere

missing `public` from method header when required

`"false"/"true"` or 0/1 for boolean values

`"null"` for `null`

Major Errors (1 point)

extraneous code which causes side-effect; *e.g., information written to output*

use interface or class name instead of variable identifier; *e.g.,* `Simulation.step()` *instead of* `sim.step()`

`aMethod(obj)` instead of `obj.aMethod()`

use private data or method when not accessible

destruction of data structure *(e.g., by using root reference to a* `TreeNode` *for traversal of the tree)*

use class name in place of `super` either in constructor or in method call

`void` method (or constructor) returns a value

Note: Spelling and case discrepancies for identifiers fall under the "non-penalized" category* ***only if* *the correction can be* ***unambiguously*** *inferred from context; for example, "*`Queu`*" instead of "*`Queue.`*" Note, however, that if a solution declares "*`Actor actor,`*" then uses "*`Actor.move()`*" instead of "*`actor.move(),`*" the context does not allow assuming the object versus the class.*

Question 1—Overview

This question focused on the array data structure, its construction and traversal, the application of basic algorithms, and method invocation for a specified object. Students were provided with the framework of a helper class, `NumberCube`, that represented a conventional six-sided die (a cube with the numbers 1 to 6 on its sides). They were asked to implement two static methods of unspecified classes. In part (a) students were required to implement the `getCubeTosses` method that returns an array of values obtained by invoking the `toss` method of a `NumberCube` object. This could be accomplished by creating an integer array of the specified length, then filling it with values obtained by invoking `toss` on the supplied `NumberCube` object. In part (b) students were required to implement the `getLongestRun` method that identifies and returns the starting index of the longest sequence of two or more consecutively repeated values in an array. This involved traversing a supplied array of integer values to locate such sequences.

Most of the student errors on this question involved incorrect array construction and access, flawed loop-based iteration, and failures to address the question's specifications. A significant number of students were unable to construct an array properly, and there appeared to be confusion of array indices with array elements. There was a proliferation of typical loop boundary errors as well. Solutions included both single and nested loop algorithms, with the nested loop attempts usually implemented incorrectly. Many students also failed to address the question's specifics and did unnecessary work, for example, by generating a random number instead of invoking the required `toss` method, or returning the length of the maximum-length run rather than the required index. Identifying the maximum-length run appeared to be the most difficult task, and solutions often failed when the maximum-length run included the end of the array.

This question appears to have been of average difficulty and was comparable in difficulty to similar questions on previous exams. The mean score was 4.70 out of a possible 9 points. Scores were generally in the 5 to 9 range, but 10 percent of students received scores of 0 or submitted blank papers. Disregarding scores of 0 and blank papers, the mean was 5.88.

Scoring Guidelines for Question 1

Number Cube

Part (a) `getCubeTosses` **4 points**

+1 constructs array
- +1/2 constructs an array of type `int` **or** size `numTosses`
- +1/2 constructs an array of type `int` **and** size `numTosses`

+2 1/2 processes tosses
- +1 repeats execution of statements `numTosses` times
- +1 tosses cube in context of iteration
- +1/2 collects results of tosses

+1/2 returns array of generated results

Part (b) `getLongestRun` **5 points**

+1 iterates over `values`
- +1/2 accesses element of `values` in context of iteration
- +1/2 accesses all elements of `values`, no out-of-bounds access potential

+1 determines existence of run of consecutive elements
- +1/2 comparison involving an element of `values`
- +1/2 comparison of consecutive elements of `values`

+1 always determines length of at least one run of consecutive elements

+1 identifies maximum length run based on all runs

+1 return value
- +1/2 returns starting index of identified maximum length run
- +1/2 returns -1 if no run identified

Sample Student Responses for Question 1

Student Response 1 (Score: 8)

(a) Write the method `getCubeTosses` that takes a number cube and a number of tosses as parameters. The method should return an array of the values produced by tossing the number cube the given number of times.

Complete method `getCubeTosses` below.

```
/** Returns an array of the values obtained by tossing a number cube numTosses times.
 *  @param cube a NumberCube
 *  @param numTosses the number of tosses to be recorded
 *         Precondition: numTosses > 0
 *  @return an array of numTosses values
 */
public static int[] getCubeTosses(NumberCube cube, int numTosses)
{
    int[] count = new int[numTosses];
    for (int i=0; i<numTosses; i++)
    {
        count[i] = cube.toss();
    }
    return count;
}
```

Student Response 1 *(continued)*

Complete method `getLongestRun` below.

```
/** Returns the starting index of a longest run of two or more consecutive repeated values
 *  in the array values.
 *  @param values an array of integer values representing a series of number cube tosses
 *         Precondition: values.length > 0
 *  @return the starting index of a run of maximum size;
 *          -1 if there is no run
 */
public static int getLongestRun(int[] values)
{   boolean inLong = false;
    int longest = 1, temp = 1;
    int pos = -1, maxPos = -1;
    for(int i=0; i< values.length-1; i++)
    {
        if(values[i] == values[i+1])
        {
            if(inLong)
            {
                temp++;
            }
            else
            {
                pos = i;
                inLong = true;
            }
        }
        else
        {
            inLong = false;
            if(temp > longest)
            {   longest = temp;
                maxPos = pos;
            }
            temp = 1;
            pos = -1;
        }
    return maxPos;
    }
}
```

ɔommentary

art (a) is a canonical solution that earned all 4 points.

Part (b) is a good solution that earned 4 out of 5 possible points. It iterates over all elements of `values` because ıe loop test of `i < values.length-1` keeps `values[i+1]` from going out-of-bounds. It also compares onsecutive elements of `values`. The variables `temp` and `longest` are one less than the actual run lengths, ut they are consistent, so it does not cause a problem.

The check for the maximum-length run is in the `else` clause, so the maximum run isn't identified until `i` dvances beyond the current run and `values[i] != values[i+1]`. Therefore, the longest run isn't identified if occurs at the end of `values`. The variable `pos` is set to `i` (the next position that could begin a run) when `alues[] != values[i+1]`. This position is then assigned to `maxPos` when a new longest run is identified nd `maxPos` is subsequently returned. Finally, the solution code returns -1 if there is no run because `maxPos` is ıitialized to -1 and is not changed when there is no run.

Student Response 2 (Score: 6)

(a) Write the method `getCubeTosses` that takes a number cube and a number of tosses as parameters. The method should return an array of the values produced by tossing the number cube the given number of times.

Complete method `getCubeTosses` below.

```
/** Returns an array of the values obtained by tossing a number cube numTosses times.
 *  @param cube a NumberCube
 *  @param numTosses the number of tosses to be recorded
 *         Precondition: numTosses > 0
 *  @return an array of numTosses values
 */
public static int[] getCubeTosses(NumberCube cube, int numTosses)
{
    int[] arr = new int[numTosses];

    for(int i=0; i<numTosses; i++)
    {
        arr[i] = toss();
    }
    return arr;
}
```

Complete method `getLongestRun` below.

```
/** Returns the starting index of a longest run of two or more consecutive repeated values
 *  in the array values.
 *  @param values an array of integer values representing a series of number cube tosses
 *         Precondition: values.length > 0
 *  @return the starting index of a run of maximum size;
 *          -1 if there is no run
 */
public static int getLongestRun(int[] values)
{
   int count = 0;
   int count2 = 1;
   for(int i=0; i<values.length-1; i++)
   {
      if (values[i] == values[i+1])
      { count2++;
        for(int j=i+2; j<values.length; i++)
        {
            if(values[j] == values[i]
               count2++;
        }
        if (count2 > count)
        {
           count = count2;
           count2 = 0;
        }
   }
   if (count > 0)
      return count;
   return -1;
}
```

Commentary

Part (a) is a good solution that earned 3 out of 4 possible points. The `toss` method was not invoked properly on the `cube` parameter, but the remainder of the solution is canonical.

Part (b) earned 2½ points out of 5 posssible points. It iterates over all elements of `values` because the outer loop test of `i < values.length-1` keeps `values[i+1]` from going out-of-bounds. It does not correctly find one run length because the inner `for` loop counts all pairs where `values[j] == values[i]`, not just the ones in a single consecutive run. Also, `count2`, the run length counter, is reset prematurely when a new maximum length run is identified. This can result in the longest run being missed. The method returns a count instead of an arra index but correctly returns -1 when there are no runs.

This solution scored 5½ points, which was rounded to 6 points.

lent Response 3 (Score: 3)

Write the method `getCubeTosses` that takes a number cube and a number of tosses as parameters. The method should return an array of the values produced by tossing the number cube the given number of times.

Complete method `getCubeTosses` below.

```
/** Returns an array of the values obtained by tossing a number cube numTosses times.
 *  @param cube a NumberCube
 *  @param numTosses the number of tosses to be recorded
 *         Precondition: numTosses > 0
 *  @return an array of numTosses values
 */
public static int[] getCubeTosses(NumberCube cube, int numTosses)
{
   int[] cubeTosses = new Array;
   for(int i=0; i < numTosses; i++)
   {
      cubeTosses[] += cube.toss(i)
   }
   return      cubeTosses[];
}
```

Student Response 3 *(continued)*

Complete method `getLongestRun` below.

```
/** Returns the starting index of a longest run of two or more consecutive repeated values
 *  in the array values.
 *  @param values an array of integer values representing a series of number cube tosses
 *         Precondition: values.length > 0
 *  @return the starting index of a run of maximum size;
 *          -1 if there is no run
 */
public static int getLongestRun(int[] values)
{
int consecutiveNums = 0;
for(int i=0; i< values.length();i++)
  {
    if (values[i+1] == values[i])
      consecutiveNums++;
  }
   return consecutiveNums - i;

   if (consecutiveNums == 0)
   return -1;
}
```

Commentary

Part (a) earned 1½ points out of 4 possible points. The declaration of the new array contains the keyword `new` but none of the other "constructs array" elements are present. It does repeat the execution of statements `numTosses` times, but it does not correctly toss the cube or properly collect the results of the attempted tosses. The "returns generated results" ½ point was earned because the `[]` on `return cubeTosses[];` is a non-penalized error under the general scoring guidelines.

Part (b) earned 1½ of the possible 5 points. It does not iterate over all elements of values because `values[i+1]` is out-of-bounds when `i` is `values.length-1`. It does compare consecutive elements of `values`. However, the loop counts all equal pairs, not just equal pairs in a single run, because `consecutiveNums` is not reset at the end of each run. So the "length of one run" point was not earned. There is no attempt to determine the maximum length run, so this point and the "returns starting index of identified maximum length run" ½ point were not earned. The statements after the `return consecutiveNums - i;` are dead code, so the "return -1" ½ point was not earned.

ıestion 2—Overview

; question involved reasoning about the code from the GridWorld case study, emphasizing object-oriented concepts. lents demonstrated their understanding of the case study and its interacting classes by extending the `Critter` class erive a `StockpileCritter` class with partially modified behavior. This question tested numerous concepts: :ting a class, inheriting from an existing class, overriding appropriate methods, and maintaining the overridden hods' postconditions. Students were specifically instructed not to override the `act` method, and they were explicitly ioned to abide by the postconditions of all methods.

tudents were asked to change the behavior of the `StockpileCritter` in the following ways: during each `act` `StockpileCritter` (1) gathers up all neighboring actors, removes them from the grid, and adds them to its kpile, and (2) reduces the stockpile by one, if possible, or, if not possible, removes itself from the grid. In all other ways, `StockpileCritter` should behave as a `Critter` would behave. Students were cautioned to abide by the :conditions of all methods and additionally were told that they could not override the `act` method.

he postconditions for `getActors`, `getMoveLocations`, and `selectMoveLocation` do not allow any : changes. The postconditions for `processActors` only allow state changes for this critter and the actors in the ımeter `actors`. Additionally, the postconditions for `processActors` state that the location of this critter is hanged. The `makeMove` method has two postconditions: at the end of `makeMove`, `getLocation` must equal parameter `loc` and only this critter and the actor in location `loc` can change state.

Most students chose to override the `processActors` method, since this method is responsible for the way in which :ritter processes its actors. The requirements for this method were to stockpile the actors in some way, removing them ı the grid in the process. Most students chose to count the number of actors in the parameter `actors` and then ove each of the actors from the grid, although some students stored the actors in a second array list and then removed ı actor from the grid. Additionally, some students chose to decrement the stockpile after gathering the actors.

he `makeMove` method was chosen by the vast majority of students as the second (and last) method to override. osing only `processActors` and `makeMove` to override resulted in a loss of 1 point for violating postconditions not overriding the required methods to correctly code the `StockpileCritter`. If students chose to call ıoveSelfFromGrid() in `processActors`, they violated the postcondition that states that the location of this er is unchanged. If they removed this `StockpileCritter` based on the status of the stockpile in `makeMove`, violated the postcondition `getLocation() == loc`.

he `selectMoveLocation` method was rarely chosen by the students to override. To correctly code the ›ckpileCritter class, the `selectMoveLocation` method had to be overridden. This method is responsible :hoosing the next location and then returning it to the `act` method. The `act` method then passes that information o `makeMove` by way of the parameter `loc`. If `loc == null`, then the `makeMove` method will remove the ›ckpileCritter from the grid and at the conclusion of the method, the postconditions will be satisfied.

he most common student errors were ignoring required solution specifications through either overriding the `act` hod or violating well-specified postconditions. For example, a common erroneous approach involved decrementing the kpile in the `makeMove` method, which violated one of that method's postconditions. There was also apparent 'usion over the responsibilities of the various case-study methods. Other common errors included declaration of local ıbles instead of instance variables, incomplete overriding of methods, and calculations that were off by one when rmining stockpile depletion.

ery few solutions were awarded 9 points. Nearly every solution violated a postcondition of `makeMove` or ›cessActors. Once that point was not awarded, the rest of the students' code was scored to see if they created a ›ckpileCritter that would behave as outlined in the problem.

his question had the fewest scores of 0 and blank papers by far (less than 13 percent combined), which is extraordinary ormance, especially for a case-study question. The mean score was 4.64 out of a possible 9 points. Students generally well, though very few solutions earned a perfect score of 9 because nearly every solution violated a necessary condition. Disregarding scores of 0 and blank papers, the mean was 5.32.

Scoring Guidelines for Question 2

Stockpile Critter (GridWorld)

+1 class header
- +1/2 properly formed class header for `StockpileCritter`
- +1/2 extends `Critter` class

+1 1/2 stockpile state
- +1/2 declares instance variable capable of maintaining state
- +1/2 private visibility
- +1/2 initialization of state appropriate to usage of variable

+1 overrides methods and maintains all necessary post-conditions
(No points awarded if overrides `act` *method)*

+1 `processActors` overridden *(No points awarded if overrides* `act` *method)*

+1 stockpile state maintenance
- +1/2 accumulates based on number of actors passed to `processActors`
- +1/2 decrements appropriately each `act`

+1 1/2 removes neighboring actors from grid
- +1/2 removes **at least one** neighboring actor from grid
- +1 removes **all** neighboring actors from grid

+2 self-removal
- +1/2 checks status of stockpile by using state variable in a relational expression
- +1/2 ever removes self from grid
- +1 removes self from grid when and only when stockpile state indicates empty

ımple Student Responses for Question 2

udent Response 1 (Score: 8)

```
public class StockpileCritter extends Critter
{
  private int energy;
  public StockpileCritter()
  {
    energy = 0;
  }
  public void processActors(ArrayList<Actor> actors)
  {
    for (Actor a : actors)
    {
      energy++;
      a.removeSelfFromGrid();
    }
  }
  public void makeMove(Location loc)
  {
    if (energy == 0)
      removeSelfFromGrid();
    else
      super.makeMove(loc);
    energy--;
  }
}
```

Commentary

The student correctly declares a class `StockpileCritter` that extends `Critter`. There is a private instance variable, `energy`, that is correctly initialized. The student fails to override the method `selectMoveLocation` but does override the method `processActors` correctly. Within `processActors`, the student correctly adds the number of neighboring actors to the stockpile and correctly removes all these neighbors from the grid. When overriding `makeMove`, the student checks the status of the stockpile and correctly calls `removeSelfFromGrid` in all appropriate cases. There is also a correct decrement to the stockpile.

Student Response 2 (Score: 7)

```
public class StockpileCritter extends Critter {
    int stockPile = 0;
    public void processActors (ArrayList<Actor> actors){
        for (Actor a: actors)
        {
            if (!(a instanceof Rock) && !(a instanceof Critter))
                a.removeSelfFromGrid();
                StockPile ++;
        }
    }

    Public void makeMove (Location loc)
    {
        if (loc == null || stockPile <= 0)
            removeSelfFromGrid();
        else
            stockPile --;
            super.makeMove(loc);
    }
}
```

dent Response 2 *(continued)*

ommentary

ıe student correctly declares a class `StockpileCritter` that extends `Critter`. There is an instance variable, `:ockpile`, that is correctly initialized; however, this variable does not have `private` scope. The student fails override the method `selectMoveLocation` but does override the method `processActors` correctly. 'ithin `processActors`, the student correctly adds the number of neighboring actors to the stockpile and ırrectly removes one of these neighbors from the grid. However, because of the conditional guard, all of the neighbors ight not be removed from the grid. When overriding `makeMove`, the student checks the status of the stockpile and ırrectly calls `removeSelfFromGrid` in all appropriate cases. There is also a correct decrement to the stockpile. ıis solution scored 6½ points, which was rounded to 7 points.

Student Response 3 (Score: 3)

```
public class StockpileCritter extends Critter
{ public int stockpile=0;
  public StockpileCritter( )
  {
   if( getOccupiedAdjacentLocations() = null )
  {return  removeSelfFromGrid; }

  else
  { getOccupiedAdjacentLocations().removeSelfFromGrid() ;
    stockpile++; }
 if( stockpile == 0)
 {removeSelfFrom Grid;}
   }
}
```

Commentary

The student correctly declares a class `StockpileCritter` that extends `Critter`. There is an instance variable `stockpile`, that is correctly initialized, but it does not have `private` scope. The student does not override either of the methods `processActors` or `selectMoveLocation`. There is no accumulation to the state variable based on the parameter passed to `processActors`, and the student never correctly removes any neighboring actors from the grid. The student checks the status of the stockpile and correctly calls `removeSelfFromGrid` in one case. However, there is never a decrement to the stockpile variable. Because there is no decrement, there is no way to ensure that `removeSelfFromGrid` is called correctly in all cases.

uestion 3—Overview

is question focused on array traversal, abstraction, and algorithms for accumulation and finding a minimum. ıdents were provided with the framework of the `BatteryCharger` class that included a private array instance ıriable with exactly 24 `int` elements, and they were asked to implement two instance methods. The first method, `tChargingCost`, required calculation of a total charging cost given a start time (`startHour`) and a number hours (`chargeTime`). This could be accomplished by accessing elements of the instance array, beginning with the ment at index `startHour`, and traversing in a circular manner (for example, by using the modulus operator), :umulating the values from the array, and returning the sum. The second method, `getChargeStartTime`, required dents to return the start time that would allow the battery to be charged at minimal cost. This was best accomplished by 'oking the `getChargingCost` method from part (a) for each of the 24 potential start times, comparing the results determine which achieve the minimum charging cost, and returning that start time.

Although most students appeared to understand the concept of accumulating, many had difficulty with properly ınding the accumulation process. Most students did not take advantage of the modulus operator (`%`), instead using ernative error-prone approaches. There was also some apparent confusion with 0-based array indexing. A significant mber of solutions to part (b) did not use the most straightforward approach of invoking `getChargingCost`. Other nmon errors included improperly initialized variables, incorrect approaches to finding the minimum, and returning the nimum charging cost rather than the start hour of that minimum charging cost.

Students performed best on this question out of all the questions on the exam, with more than 40 percent earning scores 7, 8, or 9. The mean score was 4.75 out of a possible 9 points, even though 25 percent earned scores of 0 or submitted nk papers. Disregarding scores of 0 and blank papers, the mean was a very strong 6.35. A somewhat bimodal distribu- ı indicates that this question was straightforward for those who were well prepared, yet quite difficult for those who :e not.

Scoring Guidelines for Question 3

Battery Charger

Part (a) `getChargingCost` **5 points**

+1 1/2 accesses array elements
- **+1/2** accesses any element of `rateTable`
- **+1/2** accesses an element of `rateTable` using an index derived from `startHour`
- **+1/2** accesses multiple elements of `rateTable` with no out-of-bounds access potential

+2 1/2 accumulates values
- **+1/2** declares and initializes an accumulator
- **+1/2** accumulates values from elements of `rateTable`
- **+1/2** selects values from `rateTable` using an index derived from `startHour` and `chargeTime`
- **+1** determines correct sum of values from `rateTable` based on `startHour` and `chargeTime`

+1 value returned
- **+1/2** returns any non-constant (derived) value
- **+1/2** returns accumulated value

Part (b) `getChargeStartTime` **4 points**

+1/2 invokes `getChargingCost` or replicates functionality with no errors

+1 determines charging cost
- **+1/2** considers **all** potential start times; must include at least 0 ... 23
- **+1/2** determines charging cost for potential start times

Note: No penalty here for parameter passed to `getChargingCost` *that violates its preconditions (e.g., 24)*

+1 compares charging costs for two different start times

+1 determines minimum charging cost based on potential start times

Note: Penalty here for using result of call to `getChargingCost` *that violates its preconditions (e.g., 2*

+1/2 returns start time for minimum charging cost

ımple Student Responses for Question 3
ıdent Response 1 (Score: 9)

) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour the hour at which the charge period begins
 *         Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
{
  int cost = 0;
  for(int i = 0; i< chargeTime; i++)
  {
    int index = (startHour + i) % 24;
    cost += rateTable[index];
  }
  return cost;
}
```

Student Response 1 *(continued)*

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime  the number of hours the battery needs to be charged
 *                     Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
{
    int startTime = 0;
    int startCost = getChargingCost(0, chargeTime);
    for (int i = 1; i < 24; i++)
    {
        int currentCost = getChargingCost(i, chargeTime);
        if (currentCost < startCost)
        {
            startTime = i;
            startCost = currentCost;
        }
    }
    return startTime;
}
```

Commentary

In part (a) the student correctly accesses an element from `rateTable` using an index derived from `startHour`. The loop body demonstrates accessing multiple elements of `rateTable` with no out-of-bounds potential. The student declares and initializes an accumulator and accumulates values from `rateTable` using an index derived from `startHour` and `chargeTime`. A correct sum of the values from `rateTable` is determined and this accumulated value is returned. Part (a) earned all 5 points.

In part (b) the student correctly invokes `getChargingCost`. For each potential start time (0 through 23), the charging cost is determined. The student compares two charging costs and correctly determines the minimum charging cost. The student correctly initializes and returns the start time for the minimum charging cost. Part (b) earned all 4 points.

tudent Response 2 (Score: 6)

(a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour  the hour at which the charge period begins
 *                    Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *                    Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
{
    int total = 0;
    for(int i = startHour; i < startHour + chargeTime; i++)
    {
      if(i ≤ 23)
          total += rateTable[i];

      else
          total += rateTable[i-23];

    }

    return total;

}
```

Student Response 2 *(continued)*

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime  the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
{
    int position;
    for(int i=1 ; int i<=23; i++)
    {
        if( getChargingCost(i, chargeTime) < getChargingCost(i-1, chargeTime)
            position = i;
        else
            position = i-1;
    }
    return position;
}
```

Commentary

In part (a) the student correctly accesses an element from `rateTable` using an index derived from `startHour` and so earned the first two ½ points. Although the student attempts to handle the wrap-around case (`rateTable[23]` to `rateTable[0]`) by subtracting 23, when `startHour + chargeTime` is large (greater than 46), an index out-of-bounds error will occur. The student did not earn this ½ point. The student declares and initializes an accumulator and accumulates values from `rateTable` using an index derived from [illegible] and `chargeTime` and so earned the three ½ points. Since the correct sum will not be determined [illegible] a wrap-around is necessary, the student did not earn 1 point. A derived accumulated value is [illegible] the student earned the last two ½ points. Part (a) earned 3 ½ points.

[illegible] student correctly invokes `getChargingCost` and so earned the first ½ point. The student [illegible] times 0 through 23, and for each potential start time that is considered, the charging cost is [illegible] student earned the next two ½ points. The student compares two charging costs and so earned [illegible] point. The minimum charging cost is not determined so the student did not earn 1 point. Since the start time for the minimum charging cost is not returned, the last ½ point was not earned. Part (b) earned 2 ½ points.

ıdent Response 3 (Score: 3)

) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour  the hour at which the charge period begins
 *                    Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *                    Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
```

```
int start = startHour;
start = rateTable[start];
int cTime = chargeTime;
cTime = rateTable[cTime];
charge = start - 24;
return // values of rateTable[] inclusive
       // from cTime to charge, sum of all
       // values is charging cost.
```

Student Response 3 *(continued)*

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
{
    int cTIME = chargeTime;
    int big;          int temp=0;
    for(int k=0; k<23; k++)
    {
temp= getChargingCost(k, cTIME)
    if(temp<big)
    {
        temp=big;
    }
    return temp;
```

Commentary

In part (a) the student accesses an element from `rateTable` using an index derived from `startHour` and so earned the first two ½ points. The access to `rateTable[cTime]` will cause an out-of-bounds error when `chargeTime` is greater than 23 so the student did not earn the next ½ point. Since there is no accumulator, the 2½ "accumulates values" points were not earned. Since no value is returned, the student did not earn either of the last two ½ points. Part (a) earned 1 point.

In part (b) the student correctly invokes `getChargingCost` and so earned the first ½ point. The student only considers start times 0 through 22 and did not earn the second ½ point. The student does determine a charging cost for each potential start time considered and so earned the next ½ point. The student does compare charging costs of two different start times and earned 1 point. Since the local variable `big` is not initialized, the minimum charge time will not be correctly determined so the student did not earn this 1 point. The start time for the minimum charging cost is not returned so the last ½ point was not earned. Part (b) earned 2 points.

Question 4—Overview

This question focused on object-oriented programming, algorithm development, and list processing. It required significant problem analysis and algorithm design. Students were expected to analyze the problem, design algorithms, and then implement them using the well-specified public interface of a `NumberTile` class. The question involved writing two related methods for a given `TileGame` class. The method `getIndexForFit` determines where a given `NumberTile` object can be added into the `ArrayList` instance variable `board`. The method `insertTile` modifies the state of a `TileGame` object by adding a given `NumberTile` object to `board`. In designing their solutions, students were required to consider the full set of potential cases and to work within the constraints of the public methods provided for `NumberTile`. This question assessed the ability to decompose a stated problem into computational constituents, use algorithmic thinking, rely on abstraction, and demonstrate facility with an `ArrayList` object.

Many students seemed unprepared to address this level of problem-solving and algorithm development in a free-response question. Incorrect program logic was pervasive, often evidenced by incorrect use of nested `if-else` constructs and behaviors that resulted in changes of an object's state that were prohibited by the specification. Incorrect use of object-oriented method invocation was common. There was also substantial confusion about working with lists, specifically accessing elements of an `ArrayList` and manipulating `ArrayList` objects.

This question had the highest number of scores of 0 and blank papers (almost 28 percent). This may indicate students' relative discomfort with more complex problem descriptions and greater emphasis on problem analysis, or it may be because the question required algorithm design rather than simply writing code for a given algorithm. (It was also the last question on the exam, so students could have been tired.) Otherwise, scores were evenly distributed, with a mean score of out of a possible 9 points. Disregarding scores of 0 and blank papers, the mean was 5.54.

Scoring Guidelines for Question 4

Tile Game

Part (a) `getIndexForFit` **6 points**

+1 empty board
- +1/2 checks for zero-sized board
- +1/2 returns 0 if empty board detected

+1 accesses tiles from board
- +1/2 accesses any tile from board
- +1/2 accesses all tiles of board (as appropriate) with no out-of-bounds access potential

+1 uses tile values
- +1/2 accesses left or right value of any tile
- +1/2 compares left (right) value of parameter with right (left) value of any tile from board

+2 determines tile fit
- +1/2 only right value of parameter compared with left value of initial tile of board
- +1/2 only left value of parameter compared with right value of final tile of board
- +1 compares appropriate values of parameter and interior tiles of board

+2 resultt
- +1/2 returns located index if tile fits in board
- +1/2 returns -1 if tile does not fit in board

Part (b) `insertTile` **3 points**

+1/2 invokes `getIndexForFit` or replicates functionality with no errors

+1 1/2 tile orientation
- +1/2 invokes `rotate` on parameter
- +1/2 performs all necessary rotations
- +1/2 invokes `getIndexForFit` for each necessary orientation

+1/2 adds tile correctly and only if `getIndexForFit` returns value other than -1

+1/2 returns `true` if `getIndexForFit` returns value other than -1; `false` otherwise

ample Student Responses for Question 4

tudent Response 1 (Score: 9)

Complete method `getIndexForFit` below.

```
/** Determines where to insert tile, in its current orientation, into game board
 *  @param tile the tile to be placed on the game board
 *  @return the position of tile where tile is to be inserted:
 *          0 if the board is empty;
 *         -1 if tile does not fit in front, at end, or between any existing tiles;
 *          otherwise, 0 ≤ position returned ≤ board.size()
 */
private int getIndexForFit(NumberTile tile)
{
    if(board.size()==0)
        return 0;
    if(tile.getRight()== board.get(0).getLeft())
        return 0;
    if(tile.getLeft()== board.get(board.size()-1).getRight())
        return board.size();

    if(board.size()>1)
    {
        for(int i=1; i<board.size(); i++)
        {
            if(board.get(i).getLeft()== tile.getRight())
                if(board.get(i-1).getRight()== tile.getLeft())
                    return i;
        }
    }
    return -1;
}
```

(b) Write the `TileGame` method `insertTile` that attempts to insert the given tile on the game board. The method returns `true` if the tile is inserted successfully and `false` only if the tile cannot be placed on the board in any orientation.

Assume that `getIndexForFit` works as specified, regardless of what you wrote in part (a).

Complete method `insertTile` below.

```
/** Places tile on the game board if it fits (checking all possible tile orientations if necessary).
 *  If there are no tiles on the game board, the tile is placed at position 0.
 *  The tile should be placed at most 1 time.
 *  Precondition: board is not null
 *  @param tile the tile to be placed on the game board
 *  @return true if tile is placed successfully; false otherwise
 *  Postcondition: the orientations of the other tiles on the board are not changed
 *  Postcondition: the order of the other tiles on the board relative to each other is not changed
 */
public boolean insertTile(NumberTile tile)
{
    int insertN = getIndexForFit(tile);
    if(insertN==-1)
    {
        tile.rotate();
        insertN= getIndexForFit(tile);
    }
    if(insertN==-1)
    {
        tile.rotate();
        insertN = getIndexForFit(tile);
    }
    if(insertN==-1)
    {
        tile.rotate();
        insertN= getIndexForFit(tile);
    }
    if(insertN==-1)
        return false;
    board.add(insertN, tile);
    return true;
}
```

Commentary

Part (a) earned all 6 points. The student determines if the board is empty and then proceeds to systematically check to see if the tile will fit into the board in its current orientation. Points were awarded for accessing the board and tile properly as well as checking to see if the tile can be placed at the beginning, middle, or end of the list. The expected return is the location of the tile to be placed in the board, 0 if the board is empty, or –1 if the tile cannot be placed.

Part (b) earned all 3 points. A proper call to the method `getIndexForFit(tile)` is made in conjunction with calls to `tile.rotate()`. This student chooses to perform the tests separately using `if` statements. If a placement for the tile is found, the tile is added to the board and the method will return `true`; otherwise the method will return `false`.

Student Response 2 (Score: 7)

Complete method `getIndexForFit` below.

```
/** Determines where to insert tile, in its current orientation, into game board
 *  @param tile the tile to be placed on the game board
 *  @return the position of tile where tile is to be inserted:
 *              0 if the board is empty;
 *             -1 if tile does not fit in front, at end, or between any existing tiles;
 *             otherwise, 0 ≤ position returned ≤ board.size()
 */
private int getIndexForFit(NumberTile tile)
{
 int r = tile.getRight();
 int left = tile.getLeft();
 if(r != left)
 {
  if(board.get(0).getLeft() == r)
  {
   return 0;
  }
  else if(board.get(board.size() - 1).getRight() == left)
  {
   return board.size();
  }
  else
  {
   return -1;
  }
 }
 else
 {
   for(int k = 0; k < board.size()-2; k++)
   {
    if(board.get(k).getRight == left)
    {
     return k + 1;
    }
   }
  }
  return -1;
}
```

(b) Write the `TileGame` method `insertTile` that attempts to insert the given tile on the game board. The method returns `true` if the tile is inserted successfully and `false` only if the tile cannot be placed on the board in any orientation.

Assume that `getIndexForFit` works as specified, regardless of what you wrote in part (a).

Complete method `insertTile` below.

```
/** Places tile on the game board if it fits (checking all possible tile orientations if necessary).
 *  If there are no tiles on the game board, the tile is placed at position 0.
 *  The tile should be placed at most 1 time.
 *  Precondition: board is not null
 *  @param tile the tile to be placed on the game board
 *  @return true if tile is placed successfully; false otherwise
 *  Postcondition: the orientations of the other tiles on the board are not changed
 *  Postcondition: the order of the other tiles on the board relative to each other is not changed
 */
public boolean insertTile(NumberTile tile)
{
  for(int k = 0; k < 4 && tile.getIndexForFit()< 0; k ++)
  {
   tile.rotate();
   if(k == 3 && tile.getIndexForFit() < 0)
      return false;
  }
  board.add(tile.getIndexForFit, tile);
  return true;
}
```

Commentary

Part (a) lost the first two ½ points for not checking to see if the board is empty. This student takes a different approach from the canonical solution. The student determines if the left and right sides of the tile are equal and then proceeds to compare the right side of the board tile to the inserted tile. The student earned the ½ point for properly obtaining a tile from the board but lost the next ½ point for not searching the entire board for possible tile placement as indicated by the upper loop boundary of `(board.size()-2)`. The student earned the remaining points for this section by making the proper comparisons of the tile in its current orientation. Checks are made to see if the tile can be placed at the beginning, middle, or end of the list. The expected return is the index of the tile to be placed in the board, 0 if the board is empty, or -1 if the tile cannot be placed in the board. Part (a) earned 4½ points.

Part (b) earned 2½ points out of 3 possible points. The student lost one ½ point due to an improper call to the method `getIndexForFit(tile)`.

Student Response 3 (Score: 2)

Complete method `getIndexForFit` below.

```
/** Determines where to insert tile, in its current orientation, into game board
 *  @param tile the tile to be placed on the game board
 *  @return the position of tile where tile is to be inserted:
 *            0 if the board is empty;
 *           -1 if tile does not fit in front, at end, or between any existing tiles;
 *           otherwise, 0 ≤ position returned ≤ board.size()
 */
private int getIndexForFit(NumberTile tile)
{  if ( board.size() == 0)
     return 0;
  for (int i = 0; i < board.size(); i++)
  {
    if ( tile.right() == board.get(i).left() &&
         tile.left() == board.get(i+1).right() )
           return i+1;
    else if (tile.right == board.get(board.size-i
                                     .left)
        return board.size - i;
    else if (tile.left == board.get( ).right)
        return 0;
    else if (tile != board.get(i))
        return -1;
  }
}
```

(b) Write the `TileGame` method `insertTile` that attempts to insert the given tile on the game board. The method returns `true` if the tile is inserted successfully and `false` only if the tile cannot be placed on the board in any orientation.

Assume that `getIndexForFit` works as specified, regardless of what you wrote in part (a).

Complete method `insertTile` below.

```
/** Places tile on the game board if it fits (checking all possible tile orientations if necessary).
 *  If there are no tiles on the game board, the tile is placed at position 0.
 *  The tile should be placed at most 1 time.
 *  Precondition: board is not null
 *  @param tile the tile to be placed on the game board
 *  @return true if tile is placed successfully; false otherwise
 *  Postcondition: the orientations of the other tiles on the board are not changed
 *  Postcondition: the order of the other tiles on the board relative to each other is not changed
 */
public boolean insertTile(NumberTile tile)
{
   if(board != null)
   {
      int result = getIndexForFit();
      if (result == 0)
        return false;
      else if (result > 0)
      {  return true;
         board.add(result, tile)
      }
      else if (result < 0)
         return false
   }
}
```

Commentary

Part (a) earned the first two ½ points for determining if the board is empty. The student earned two additional ½ points by accessing the board properly and attempting to compare sides of the tile to the interior tiles of the board. Points were not awarded for the checks of the end tiles due to reversed logic. Part (a) earned 2 points.

Part (b) earned no points. The student has an improper call to `getIndexForFit(tile)` and the return of `true / false` is incorrect.

Chapter IV: Statistical Information

Table 4.1—Section II Scores

Table 4.2—Scoring Worksheet

Table 4.3—Score Distributions

Table 4.4—Section I Scores and AP Scores

How AP Scores Are Determined

College Comparability Studies

Reminders for All Score Report Recipients

Reporting AP Scores

Purpose of AP Scores

his chapter presents statistical information about overall tudent performance on the 2009 AP Computer Science A xam.

Table 4.1 shows and summarizes score distributions for each of the free-response questions. The scoring worksheet presented in Table 4.2 provides step-by-step instructions for calculating AP section and composite scores and converting composite scores to AP Exam scores. Table 4.3 includes distributions for the overall exam scores. The score distributions conditioned on multiple-choice performance presented in Table 4.4 are useful in estimating a student's AP Exam score given only the student's multiple-choice score.

College comparability studies, which are conducted to collect information for setting AP score cut-points, are briefly discussed in this chapter. In addition, the purpose and intended use of AP Exams are reiterated to promote appropriate interpretation and use of the AP Exam and exam results.

Table 4.1—Section II Scores

The following table shows the score distributions for AP students on each free-response question from the 2009 AP Computer Science A Exam.

Score	Question 1		Question 2		Question 3		Question 4	
	No. of Students	% at Score	No. of Students	% at Score	No. of Students	% at Score	No. of Students	% at Score
9	1,522	9.60	45	0.28	2,322	14.64	1,482	9.34
8	2,214	13.96	3,569	22.50	2,429	15.31	1,659	10.46
7	2,140	13.49	2,497	15.74	1,961	12.36	1,689	10.65
6	1,998	12.60	1,480	9.33	1,522	9.60	1,489	9.39
5	1,367	8.62	1,179	7.43	941	5.93	1,274	8.03
4	1,156	7.29	1,071	6.75	993	6.26	1,105	6.97
3	853	5.38	1,395	8.80	658	4.15	906	5.71
2	771	4.86	1,275	8.04	529	3.34	798	5.03
1	671	4.23	1,308	8.25	525	3.31	1,074	6.77
0	2,465	15.54	414	2.61	2,574	16.23	2,319	14.62
No Response	704	4.44	1,628	10.26	1,407	8.87	2,066	13.03
Total Students	15,861		15,861		15,861		15,861	
Mean	4.70		4.64		4.76		4.01	
Standard Deviation	3.13		2.89		3.40		3.27	
Mean as % of Maximum Score	52		52		53		45	

Table 4.2—Scoring Worksheet

Section I: Multiple Choice

[_____ − (1/4 × _____)] × 1.0000 = _____

Number Correct (out of 40) | Number Wrong | Weighted Section I Score (If less than zero, enter zero; do not round)

Section II: Free Response

Question 1 _____ × 1.1111 = _____ (out of 9) (Do not round)

Question 2 _____ × 1.1111 = _____ (out of 9) (Do not round)

Question 3 _____ × 1.1111 = _____ (out of 9) (Do not round)

Question 4 _____ × 1.1111 = _____ (out of 9) (Do not round)

Sum = _____

Weighted Section II Score (Do not round)

AP Score Conversion Chart Computer Science A

Composite Score Range	AP Score
60–80	5
44–59	4
33–43	3
25–32	2
0–24	1

Composite Score

_____ + _____ = _____

Weighted Section I Score | Weighted Section II Score | Composite Score (Round to nearest whole number)

Table 4.3—Score Distributions

lmost 62 percent of the AP students who took this exam earned a qualifying score of 3 or above.

	Exam Score	Number of Students	Percent at Score
Extremely well qualified	5	3,698	23.32
Well qualified	4	4,051	25.54
Qualified	3	2,081	13.12
Possibly qualified	2	1,289	8.13
No recommendation	1	4,742	29.90
Total Number of Students		15,861	
Mean Score		3.04	
Standard Deviation		1.57	

Table 4.4—Section I Scores and AP Scores

ɔr a given range of multiple-choice scores, this table shows the percentage of students receiving each AP score. If you ıve calculated the multiple-choice score (**Weighted Section I Score**) by using the formula shown in Table 4.2, you can ɜe this table to figure out the most likely score that the student would receive based only on that multiple-choice score.

	AP Score					
Multiple-Choice Score	**1**	**2**	**3**	**4**	**5**	**Total**
34 to 40	0.0%	0.0%	0.1%	0.9%	99.0%	7.5%
27 to 33	0.0%	0.2%	1.0%	29.1%	69.8%	20.4%
20 to 26	0.7%	3.1%	19.4%	70.0%	6.8%	23.9%
13 to 19	23.6%	26.1%	37.0%	13.4%	0.0%	21.1%
6 to 12	85.3%	11.5%	3.1%	0.1%	0.0%	15.9%
0 to 5	99.8%	0.2%	0.0%	0.0%	0.0%	11.2%
Total	29.9%	8.1%	13.1%	25.5%	23.3%	100.0%

How AP Scores Are Determined

As described in Chapter II, the AP Computer Science A Exam has two sections. Section I has 40 multiple-choice questions and a score range from a minimum possible score of 0 to a maximum possible score of 40 points. Section II has 4 free-response questions that each range from a minimum possible score of 0 to a maximum possible score of 9 points.

The scores on the different parts of the exam are combined to produce a composite score that ranges from a minimum possible score of 0 to a maximum possible score of 80 points. In calculating the composite scores, scores on different parts are multiplied by weights.

Composite scores are not released to students, schools, or colleges. Instead, the composite scores are converted to scores on an AP 5-point scale, and it is these scores that are reported. The process of calculating the composite score and converting it to an AP Exam score involves a number of steps which are shown in the Scoring Worksheet (Table 4.2) and described in detail here.

1. **The score on Section I is calculated.** In calculating the score for Section I, a fraction of the number of wrong answers is subtracted from the number of right answers. With this adjustment to the number of right answers, students are not likely to benefit from random guessing. The value of the fraction is 1/4 for the five-choice questions in the AP Computer Science A Exam. The maximum possible weighted score on Section I is 40 points, and it accounts for 50 percent of the maximum possible composite score.
2. **The score on Section II is calculated.** The 4 questions in Section II are weighted equally. The weighted scores on the questions in Section II are summed to give the total weighted score for Section II. The maximum possible weighted score on Section II is 40 points, and it accounts for 50 percent of the maximum possible composite score.
3. **AP Exam scores are calculated.** Composite scores are calculated by adding the weighted Section I and weighted Section II scores together. The AP Exam scores are calculated by comparing the composite scores to the four composite cut-scores selected during the score-setting process. A variety of information is available during the score-setting process to help determine the cut-scores corresponding to each AP score:
   - Statistical information based on test score equating
   - College/AP score comparability studies, if available
   - The Chief Reader's observations of students' free-response performance
   - The distribution of scores on different parts of the exam
   - AP score distributions from the past three years

See Table 4.3 for the score distributions for the 2009 AP Computer Science A Exam.

If you are interested in more detailed information about this process, please visit AP Central (apcentral.collegeboard.com). There you will also find information about how the AP Exams are developed, how validity and reliability studies are conducted, and other data on all AP subjects.

College Comparability Studies

The Advanced Placement Program has conducted college grade comparability studies in all AP subjects. These studies have compared the performance of AP students with that of college students in related courses who have taken the AP Exam at the end of their course. In general, AP cut-points are selected so that the lowest AP 5 is equivalent to the average A in college, the lowest AP 4 is equivalent to the average B, and the lowest AP 3 is equivalent to the average C (see below).

AP Score	Average College Grade
5	
	A
4	
	B
3	
	C
2	
	D
1	

Research studies conducted by colleges and universities and by the AP Program indicate that AP students generally receive higher grades in advanced courses than do students who have taken the regular first-year courses at the institution. Colleges and universities are encouraged to periodically undertake such studies to establish appropriate policy for accepting AP scores and ensure that admissions and placement standards remain valid. It is critical to verify that admissions and placement measures established for a previous class continue for future classes. Summaries of several studies are available at AP Central. Also on the College Board Web site is the free Admitted Class Evaluation Service™ (http://professionals.collegeboard.com/higher-ed/validity) that can predict how admitted college students will perform at a particular institution generally and how successful they can be in specific classes.

Reminders for All Score Report Recipients

AP Exams are designed to provide accurate assessments of achievement. However, any exam has limitations, especially when used for purposes other than those intended. Presented here are some suggestions for teachers to aid in the use and interpretation of AP scores:

- AP Exams in different subjects are developed and evaluated independently of each other. They are linked only by common purpose, format, and method of reporting results. Therefore, comparisons should not be made between scores on different AP Exams. An AP score in one subject may not have the same meaning as the same AP score in another subject, just as national and college standards vary from one discipline to another.
- Score reports are confidential. Everyone who has access to AP scores should be aware of the confidential nature of the scores and agree to maintain their security. In addition, school districts and states should not release data about high school performance without the school's permission.
- AP Exams are not designed as instruments for teacher or school evaluation. Many factors influence AP Exam performance in a particular course or school in any given year. Thus, differences in AP Exam performance should be carefully studied before being attributed to the teacher or school.
- Where evaluation of AP students, teachers, or courses is desired, local evaluation models should be developed. An important aspect of any evaluation model is the use of an appropriate method of comparison or frame of reference to account for yearly changes in student composition and ability, as well as local differences in resources, educational methods, and socioeconomic factors.
- The AP Instructional Planning Report is sent to schools automatically and can be a useful diagnostic tool in reviewing course results. This report identifies areas of strength and weakness for the students in each AP course. The information may also provide teachers with guidance for course emphasis and student evaluation.
- Many factors can influence exam results. AP Exam performance can be affected by the degree of agreement between a course and the course defined in the relevant AP Course Description, use of different instructional methods, differences in emphasis or preparation on particular parts of the exam, differences in curriculum, or differences in student background and preparation in comparison with the national group.

Reporting AP Scores

The results of AP Exams are disseminated in several ways to students, their secondary schools, and the colleges they select:

- College and student score reports contain a cumulative record of all scores earned by the student on AP Exams during the current or previous years. These reports are sent in July. (School score reports are sent shortly thereafter.)
- Group results for AP Exams are available to AP teachers in the AP Instructional Planning Report mentioned previously. This report provides useful information comparing local student performance with that of the total group of students taking an exam, as well as details on different subsections of the exam.

Several other reports produced by the AP Program provide summary information on AP Exams:

- State, National, and Canadian Reports show the distribution of scores obtained on each AP Exam for all students and for subsets of students broken down by gender and by ethnic group.
- The Program also produces a one-page summary of AP score distributions for all exams in a given year.

For information on any of the above, please call AP Services at 609 771-7300 or e-mail apexams@info.collegeboard.org.

Purpose of AP Scores

AP scores are intended to allow participating colleges and universities to award college credit, advanced placement, or both to qualified students. In general, an AP score of 3 or higher indicates sufficient mastery of course content to allow placement in the succeeding college course, or credit for and exemption from a college course comparable to the AP course. Students seeking credit through their AP scores should note that each college, not the AP Program or the College Board, determines the nature and extent of its policies for awarding advanced placement, credit, or both. Because policies regarding AP scores vary, students should consult the AP policy of individual colleges and universities. Students can find information in a college's catalog or Web site, or by using the AP Credit Policy search at www.collegeboard.com/ap/creditpolicy.