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

Materials included in this Released Exam may not reflect the current AP Course Description and exam in this subject, and teachers are advised to take this into account as they use these materials to support their instruction of students. For up-to-date information about this AP course and exam, please download the official AP Course Description from the AP Central® Web site at apcentral.collegeboard.com.
Chapter I: The AP® Process

What Is the Purpose of the AP® Computer Science A Exam?

Who Develops the Exam?

How Is the Exam Developed?

Section I—Multiple Choice

Section II—Free Response

Question Types

Multiple Choice

Free Response

Scoring the Exam

Who Scores the AP Computer Science A Exam?

Ensuring Accuracy

How the Scoring Guidelines Are Created

Training Readers to Apply the Scoring Guidelines

Maintaining the Scoring Guidelines

Preparing Students for the Exam

Essential Features of Student Responses

Teaching Free-Response Writing

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

What Is the Purpose of the AP Computer Science A Exam?

The AP Computer Science A Exam is designed to allow students to demonstrate that their knowledge, understanding, and skills are equivalent to those gained by students who have successfully completed an introductory one-year-level course in computer science. The AP Computer Science A Exam includes material that is covered in a one-semester computer science course in which the Java programming language is used to illustrate an object-oriented approach to problem solving and algorithm development.

The multiple-choice section of the exam is designed to test the breadth of the curriculum in terms of knowledge, principles, and conceptual understanding. The free-response section requires students to solve problems involving more extended reasoning. In both sections, students must demonstrate their ability to design, write, and document programs and subprograms. Allowing scores on the AP Computer Science A Exam allow students to begin their college careers with credit in introductory computer science courses 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.

**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 that students of similar ability levels taking a different form of the exam will receive the same scores, is controlled more 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, thereby 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 using 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 current one, allowing comparisons to be made between the scores of the earlier group of students and those of the current group. This information, along with other data, is used to establish AP scores that reflect the competence demanded 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 overall course objectives and exam specifications. After each exam administration, the questions in each section are analyzed both individually and collectively, and the findings are used 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 members.
to teach either a high school AP Computer Science A course or an equivalent course at a college or university. Care is taken to obtain a broad and balanced group of readers. Among the factors considered before appointing someone to the role are school locale and setting (urban, rural, and so on), as well as the potential Reader's gender, nicity, and years of teaching experience. University and high school computer science teachers who are interested in applying to be a Reader at a future AP Reading can complete and submit an online application via AP Central (central.collegeboard.com/readers) or request more information by e-mailing apreader@ets.org.

In June 2009, approximately 130 computer science teachers and professors gathered at the Duke Energy Center in Cincinnati, Ohio, to participate in the scoring session for the AP Computer Science A Exam. Some of the most experienced members of this group were invited to serve as Question Leaders and Table Leaders, and they arrived at the site early to help prepare for the scoring session. The training Readers were divided into groups, with each group advised and supervised by a Table Leader. Under the direction of the Chief Reader, Question Leaders and Table Leaders assisted in establishing scoring guidelines, selecting student responses that exemplified the guidelines, and preparing for Reader training. All of the free-response questions on the 2009 AP Computer Science A Exam were evaluated by the Readers at this single, central scoring site under the supervision of the Chief Reader.

Suring Accuracy

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

How 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 resulting 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 problem solving and algorithm development, and is meant to be the equivalent of a first-semester course in computer science. It 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 appropriate 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 understand 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 more to the exam than testing syntax. Student responses are evaluated as "good first drafts" of the solution to a problem in a world where so many students (as well as professiona
puter scientists) use tools such as IDEs to help in their grammar, it is foolish to expect students to worry about punctuation exactly right. As part of evaluating free-response questions, Readers are provided with General Grading Guidelines that describe how to penalize (or ignore) errors, such as missing braces, confusing = and ==, using a class name as an instance of that class. In grading questions, Readers focus more on whether a student response adequately addresses the major issues of question (for example, array access, looping, class manipulation, etc.) than more minor issues (is the code commented, are the braces aligned, are parentheses used, are of brackets, etc.). Additionally, the ability of students in college classes is assessed by programming projects as well as test questions. Many professors consider programming projects to be more important than test questions because they are more representative of how computer science is actually done. Since Computer Science A students are assessed with only questions, these questions may have more structure than those on a college test. For example, students on a college test may be asked simply to write code to determine if the values in an array are in increasing order. Instead of something so straightforward, a question on the Computer Science A Exam may pose the problem of writing code to determine if a student’s scores have improved over the course of a term, where the scores are kept in an array. This more complicated question can then assess students can do basic array manipulation and how they would 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
Giving a Practice Exam
Instructions for Administering the Exam
Blank Answer Sheet
The 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 “grade,” 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 _________.

9
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.
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.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
</table>

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 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |

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.

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

ETS USE ONLY

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

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>W</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT04</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OTHER

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>W</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DO NOT WRITE IN THIS AREA.
### T. YOUR MAILING ADDRESS

- Your grade report will be mailed to this address in July.

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

- 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 Q below.

#### STATE

<table>
<thead>
<tr>
<th>State Abbreviation</th>
<th>State Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>ALABAMA</td>
</tr>
<tr>
<td>AK</td>
<td>ALASKA</td>
</tr>
<tr>
<td>AZ</td>
<td>ARIZONA</td>
</tr>
<tr>
<td>AR</td>
<td>ARKANSAS</td>
</tr>
<tr>
<td>CA</td>
<td>CALIFORNIA</td>
</tr>
<tr>
<td>CO</td>
<td>COLORADO</td>
</tr>
<tr>
<td>CT</td>
<td>CONNECTICUT</td>
</tr>
<tr>
<td>DE</td>
<td>DELAWARE</td>
</tr>
<tr>
<td>DC</td>
<td>DISTRICT OF COLUMBIA</td>
</tr>
<tr>
<td>FL</td>
<td>FLORIDA</td>
</tr>
<tr>
<td>GA</td>
<td>GEORGIA</td>
</tr>
<tr>
<td>HI</td>
<td>HAWAII</td>
</tr>
<tr>
<td>ID</td>
<td>IDAHO</td>
</tr>
<tr>
<td>IL</td>
<td>ILLINOIS</td>
</tr>
<tr>
<td>IN</td>
<td>INDIANA</td>
</tr>
<tr>
<td>IA</td>
<td>IOWA</td>
</tr>
<tr>
<td>KS</td>
<td>KANSAS</td>
</tr>
<tr>
<td>KY</td>
<td>KENTUCKY</td>
</tr>
<tr>
<td>LA</td>
<td>LOUISIANA</td>
</tr>
<tr>
<td>ME</td>
<td>MAINE</td>
</tr>
<tr>
<td>MD</td>
<td>MARYLAND</td>
</tr>
<tr>
<td>MA</td>
<td>MASSACHUSETTS</td>
</tr>
<tr>
<td>MI</td>
<td>MICHIGAN</td>
</tr>
<tr>
<td>MN</td>
<td>MINNESOTA</td>
</tr>
<tr>
<td>MS</td>
<td>MISSISSIPPI</td>
</tr>
<tr>
<td>MO</td>
<td>MISSOURI</td>
</tr>
<tr>
<td>MT</td>
<td>MONTANA</td>
</tr>
<tr>
<td>NE</td>
<td>NEBRASKA</td>
</tr>
<tr>
<td>NH</td>
<td>NEW HAMPSHIRE</td>
</tr>
<tr>
<td>NJ</td>
<td>NEW JERSEY</td>
</tr>
<tr>
<td>NM</td>
<td>NEW MEXICO</td>
</tr>
<tr>
<td>NY</td>
<td>NEW YORK</td>
</tr>
<tr>
<td>NC</td>
<td>NORTH CAROLINA</td>
</tr>
<tr>
<td>ND</td>
<td>NORTH DAKOTA</td>
</tr>
<tr>
<td>OH</td>
<td>OHIO</td>
</tr>
<tr>
<td>OK</td>
<td>OKLAHOMA</td>
</tr>
<tr>
<td>OR</td>
<td>OREGON</td>
</tr>
<tr>
<td>PA</td>
<td>PENNSYLVANIA</td>
</tr>
<tr>
<td>RI</td>
<td>RHODE ISLAND</td>
</tr>
<tr>
<td>SC</td>
<td>SOUTH CAROLINA</td>
</tr>
<tr>
<td>SD</td>
<td>SOUTH DAKOTA</td>
</tr>
<tr>
<td>TN</td>
<td>TENNESSEE</td>
</tr>
<tr>
<td>TX</td>
<td>TEXAS</td>
</tr>
<tr>
<td>UT</td>
<td>UTAH</td>
</tr>
<tr>
<td>VT</td>
<td>VERMONT</td>
</tr>
<tr>
<td>VA</td>
<td>VIRGINIA</td>
</tr>
<tr>
<td>WA</td>
<td>WASHINGTON</td>
</tr>
<tr>
<td>WV</td>
<td>WEST VIRGINIA</td>
</tr>
<tr>
<td>WI</td>
<td>WISCONSIN</td>
</tr>
<tr>
<td>WY</td>
<td>WYOMING</td>
</tr>
</tbody>
</table>

#### ZIP OR POSTAL CODE

- Indicate a space in your address by leaving a blank box; do not grid that column. Only one box is to be filled in for each column.

#### U. COUNTRY CODE

<table>
<thead>
<tr>
<th>Country Code</th>
<th>Country Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>United States</td>
</tr>
<tr>
<td>1</td>
<td>Canada</td>
</tr>
<tr>
<td>2</td>
<td>Mexico</td>
</tr>
<tr>
<td>3</td>
<td>Japan</td>
</tr>
<tr>
<td>4</td>
<td>Australia</td>
</tr>
<tr>
<td>5</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>6</td>
<td>Germany</td>
</tr>
<tr>
<td>7</td>
<td>France</td>
</tr>
<tr>
<td>8</td>
<td>Italy</td>
</tr>
<tr>
<td>9</td>
<td>Other</td>
</tr>
</tbody>
</table>

### INTERNATIONAL TELEPHONE

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

### X. SCHOOL YOU ATTEND

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

#### School Name, City, and State

- **School Name**:
- **City**:
- **State**:

### Y. COLLEGE TO RECEIVE YOUR AP GRADE REPORT

- 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 and Address

- **College Name**:
- **City**:
- **State**:
The Exam

AP® Computer Science A Exam

Section I: Multiple-Choice Questions

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

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:  Chicago is a ___.
   (A) state
   (B) city
   (C) country
   (D) continent
   (E) village

   Sample Answer:  A

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.
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.

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

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

<table>
<thead>
<tr>
<th></th>
<th>First</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>15</td>
<td>27</td>
</tr>
<tr>
<td>(B)</td>
<td>15</td>
<td>28</td>
</tr>
<tr>
<td>(C)</td>
<td>16</td>
<td>27</td>
</tr>
<tr>
<td>(D)</td>
<td>16</td>
<td>28</td>
</tr>
<tr>
<td>(E)</td>
<td>16</td>
<td>29</td>
</tr>
</tbody>
</table>
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
Assume that an array of integer values has been declared as follows and has been initialized.

```java
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];`
Section 1

4. Consider the following code segment.

```java
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.
Questions 6-7 refer to the following code segment.

```java
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^3
Section 1

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

   ```java
   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++)`
   ```java
   {
       total += vals[pos];
   }
   ```

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

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

(A) I only
(B) II only
(C) III only
(D) I and III
(E) II and III
1. 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) aabbcdddeeff
   (C) abbcdddeeff
   (D) abbcddcdeedef
   (E) Nothing is printed because an IndexOutOfBoundsException is thrown.
10. Consider the following method.

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

   ```java
   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)`
12. Consider the following incomplete method.

```java
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`.

<table>
<thead>
<tr>
<th>Input Value n</th>
<th>Value Returned by someProcess(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>80</td>
</tr>
<tr>
<td>9</td>
<td>90</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>120</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>16</td>
<td>160</td>
</tr>
</tbody>
</table>

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.

```java
// 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.
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.

```java
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.

```java
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 II
(E) II and III
Consider the following class declaration.

```java
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.

```java
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.

```java
// 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++`
1. Assume that \( x \) and \( y \) are boolean variables and have been properly initialized.

\[ (x \land y) \lor \neg (x \land y) \]

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
Section I

19. Assume that the following variable declarations have been made:
   
   ```java
   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) \( r = d + 0.5 \times 5.0 \);
   
   (C) \( r = d \times 5.0 \);
   
   (D) \( r = d \times 5.0 + 0.5 \);
   
   (E) \( r = d \times 5.5 \);
Consider the following instance variables and method that appear in a class representing student information.

```java
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.

```java
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.

```java
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.

```java
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.

```java
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
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);
        }
    }

Which 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.

```java
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
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`.

```java
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?

### Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Expression</th>
<th>Loop Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td><code>false</code></td>
<td><code>if ((arr[k] % 2) == 0) isEven = true;</code></td>
</tr>
<tr>
<td>(B)</td>
<td><code>false</code></td>
<td><code>if ((arr[k] % 2) != 0) isEven = false; else isEven = true;</code></td>
</tr>
<tr>
<td>(C)</td>
<td><code>true</code></td>
<td><code>if ((arr[k] % 2) != 0) isEven = false;</code></td>
</tr>
<tr>
<td>(D)</td>
<td><code>true</code></td>
<td><code>if ((arr[k] % 2) != 0) isEven = false; else isEven = true;</code></td>
</tr>
<tr>
<td>(E)</td>
<td><code>true</code></td>
<td><code>if ((arr[k] % 2) == 0) isEven = false; else isEven = true;</code></td>
</tr>
</tbody>
</table>
28. Consider the following code segment.

```java
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.

```java
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 3 3 3 3

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

Unauthorized copying or reuse of any part of this page is illegal.
30. Consider the following class declarations.

```java
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.

```java
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
3. Consider the following method.

```java
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`.

```java
// 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.

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

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

III. 
```java
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.

```java
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.println(value);
}
```

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

(A) 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).

```java
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.

```java
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.

```java
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?

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

(A) 20
(B) 102
(C) 210
(D) 1020
(E) 2101
Consider the following method.

```java
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?

```java
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
Appendix B .................................................... Testable API
Appendix C .................................................... Testable Code for APCS A/AB
Appendix E .................................................... Quick Reference A/AB
Appendix G .................................................... Index for Source Code
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

58
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
The `Rock` class (extends `Actor`) is defined as follows:

```java
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
```

The `Flower` class (extends `Actor`) is defined as follows:

```java
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

**bug.java**

```java
package info.gridworld.actor;

import info.gridworld.grid.Grid;
import info.gridworld.grid.Location;

import 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.
 */

public 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);
  }

} // end class Bug
```
/**
 * 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
}
**BoxBug.java**

```java
import 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.
 */

private int steps;
private int sideLength;

/**
 * Constructs a box bug that traces a square of a given side length
 * @param length the side length
 */

{   steps = 0;
    sideLength = length;
}

/**
 * Moves to the next location of the square.
 */

{   if (steps < sideLength && canMove())
   {   move();
       steps++;
   }
   else
   {   turn();
       turn();
       steps = 0;
   }
}
```
**Critter.java**

```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.
 */

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: 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 moveLocation(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)**

```java
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**

```java
int numRows()
int numCols()
boolean isValid(Location loc)
void put(Location loc, E obj)
void remove(Location loc)
get(Location loc)
```

```java
List<Location> getOccupiedLocations()
List<Location> getValidAdjacentLocations(Location loc)
List<Location> getEmptyAdjacentLocations(Location loc)
List<Location> getOccupiedAdjacentLocations(Location loc)
List<E> getNeighbors(Location loc)
```

**Actor Class**

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

- Bug()
- Bug(Color bugColor)
- act()
- turn()
- move()
- canMove()

**BoxBug.java**

- BoxBug(int length)
- act()

**Critter.java**

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

**ChameleonCritter.java**

- processActors(ArrayList<Actor> actors)
- makeMove(Location loc)
AP® Computer Science A Exam

SECTION II: Free-Response Questions

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

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

Instructions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:
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.
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.

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

The number cube is represented by the following class.

```java
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.
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
 * @return an array of `numTosses` values
 */
public static int[] getCubeTosses(NumberCube cube, int numTosses)
(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.

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td></td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

WRITE YOUR SOLUTION ON THE NEXT PAGE.
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

/ *
  @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.
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.

<table>
<thead>
<tr>
<th>Hour</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td>2</td>
<td>160</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
</tr>
<tr>
<td>4</td>
<td>80</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>120</td>
</tr>
<tr>
<td>8</td>
<td>150</td>
</tr>
<tr>
<td>9</td>
<td>150</td>
</tr>
<tr>
<td>10</td>
<td>150</td>
</tr>
<tr>
<td>11</td>
<td>200</td>
</tr>
<tr>
<td>12</td>
<td>40</td>
</tr>
<tr>
<td>13</td>
<td>240</td>
</tr>
<tr>
<td>14</td>
<td>220</td>
</tr>
<tr>
<td>15</td>
<td>220</td>
</tr>
<tr>
<td>16</td>
<td>200</td>
</tr>
<tr>
<td>17</td>
<td>200</td>
</tr>
<tr>
<td>18</td>
<td>180</td>
</tr>
<tr>
<td>19</td>
<td>180</td>
</tr>
<tr>
<td>20</td>
<td>140</td>
</tr>
<tr>
<td>21</td>
<td>100</td>
</tr>
<tr>
<td>22</td>
<td>80</td>
</tr>
<tr>
<td>23</td>
<td>60</td>
</tr>
</tbody>
</table>

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.

```java
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
    * @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
1) 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.

<table>
<thead>
<tr>
<th>Start Hour of Charge</th>
<th>Hours of Charge Time</th>
<th>Last Hour of Charge</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1</td>
<td>12</td>
<td>40</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>110</td>
</tr>
<tr>
<td>22</td>
<td>7</td>
<td>4 (the next day)</td>
<td>550</td>
</tr>
<tr>
<td>22</td>
<td>30</td>
<td>3 (two days later)</td>
<td>3,710</td>
</tr>
</tbody>
</table>

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

Complete method `getChargingCost` below.

```java
/**
 * Determines the total cost to charge the battery starting at the beginning of `startHour`.
 * @param startHour the hour at which the charge period begins
 * @param chargeTime the number of hours the battery needs to be charged
 * @precondition: 0 ≤ startHour ≤ 23
 * @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.

<table>
<thead>
<tr>
<th>Hours of Charge Time</th>
<th>Minimum Cost</th>
<th>Start Hour of Charge</th>
<th>Last Hour of Charge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>110</td>
<td>0 or 23</td>
<td>0 (the next day)</td>
</tr>
<tr>
<td>7</td>
<td>550</td>
<td>22</td>
<td>4 (the next day)</td>
</tr>
<tr>
<td>30</td>
<td>3,710</td>
<td>22</td>
<td>3 (two days later)</td>
</tr>
</tbody>
</table>

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.

```java
/**
 * 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.

<table>
<thead>
<tr>
<th>INITIAL</th>
<th>AFTER 1</th>
<th>AFTER 2</th>
<th>AFTER 3</th>
<th>AFTER 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 3 7</td>
<td>4 7 3</td>
<td>7 3 4</td>
<td>3 4 7</td>
<td>4 3 7</td>
</tr>
</tbody>
</table>

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

```java
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.

<table>
<thead>
<tr>
<th>GAME BOARD</th>
<th>NEW TILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 3 7 7</td>
<td>2 9</td>
</tr>
<tr>
<td>4 6 4</td>
<td></td>
</tr>
<tr>
<td>4 3 2 2</td>
<td></td>
</tr>
</tbody>
</table>

Position 0 1 2 3 4
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 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

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>9</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Position 0 1 2 3 4 5

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.
}
(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.

```
+---+---+---+---+---+
|   |   |   |   |   |
+---+---+---+---+---+
| 4 | 3 |   |   |   |
+---+---+---+---+---+
| 3 | 4 | 2 | 2 | 2 |
+---+---+---+---+---+
| 3 | 5 | 2 | 9 |   |
+---+---+---+---+---+
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.
Complete method `getIndexForFit` below.

```java
/** 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(NumberOfTile 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 `getIndexOfFit` works as specified, regardless of what you wrote in part (a).

Complete method `insertTile` below.

```java
/**
 * Places tile on the game board if it fits (checking all possible tile orientations if necessary).
 * @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