AP® Computer Science A
2007 Free-Response Questions

The College Board: Connecting Students to College Success

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 5,000 schools, colleges, universities, and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

© 2007 The College Board. All rights reserved. College Board, Advanced Placement Program, AP, AP Central, SAT, and the acorn logo are registered trademarks of the College Board. PSAT/NMSQT is a registered trademark of the College Board and National Merit Scholarship Corporation.
Permission to use copyrighted College Board materials may be requested online at:
www.collegeboard.com/inquiry/cbpermit.html.

Visit the College Board on the Web: www.collegeboard.com.
AP Central is the official online home for the AP Program: apcentral.collegeboard.com.

COMPUTER SCIENCE A
SECTION II

Time—1 hour and 45 minutes
Number of questions—4
Percent of total grade—50

Directions: 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 positive integer is called a "self-divisor" if every decimal digit of the number is a divisor of the number, that is, the number is evenly divisible by each and every one of its digits. For example, the number 128 is a self-divisor because it is evenly divisible by 1, 2, and 8. However, 26 is not a self-divisor because it is not evenly divisible by the digit 6. Note that 0 is not considered to be a divisor of any number, so any number containing a 0 digit is NOT a self-divisor. There are infinitely many self-divisors.

```
public class SelfDivisor
{
  /** @param number  the number to be tested
   *          Precondition: number > 0
   *  @return true  if every decimal digit of number is a divisor of number;
   *           false otherwise
   */
  public static boolean isSelfDivisor(int number)
  {  /* to be implemented in part (a) */  }


  /** @param start  starting point for values to be checked
   *          Precondition: start > 0
   *  @param num  the size of the array to be returned
   *          Precondition: num > 0
   *  @return  an array containing the first num integers ≥ start that are self-divisors
   */
  public static int[] firstNumSelfDivisors(int start, int num)
  {  /* to be implemented in part (b) */  }


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

© 2007 The College Board. All rights reserved.
Visit apcentral.collegeboard.com (for AP professionals) and www.collegeboard.com/apstudents (for students and parents).

GO ON TO THE NEXT PAGE.

(a) Write method `isSelfDivisor`, which takes a positive integer as its parameter. This method returns `true` if the number is a self-divisor; otherwise, it returns `false`.

Complete method `isSelfDivisor` below.

```
/** @param number  the number to be tested
 *         Precondition: number > 0
 *  @return true  if every decimal digit of number is a divisor of number;
 *          false otherwise
 */
public static boolean isSelfDivisor(int number)
```

(b) Write method `firstNumSelfDivisors`, which takes two positive integers as parameters, representing a start value and a number of values. Method `firstNumSelfDivisors` returns an array of size `num` that contains the first `num` self-divisors that are greater than or equal to `start`.

For example, the call `firstNumSelfDivisors(10, 3)` should return an array containing the values 11, 12, and 15, because the first three self-divisors that are greater than or equal to 10 are 11, 12, and 15.

In writing `firstNumSelfDivisors`, assume that `isSelfDivisor` works as specified, regardless of what you wrote in part (a).

Complete method `firstNumSelfDivisors` below.

```
/** @param start  starting point for values to be checked
 *         Precondition: start > 0
 *  @param num  the size of the array to be returned
 *         Precondition: num > 0
 *  @return  an array containing the first num integers ≥ start  that are self-divisors
 */
public static int[] firstNumSelfDivisors(int start, int num)
```

© 2007 The College Board. All rights reserved.
Visit apcentral.collegeboard.com (for AP professionals) and www.collegeboard.com/apstudents (for students and parents).

GO ON TO THE NEXT PAGE.

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

   A `PounceFish` is a type of fish that looks for prey and then "pounces" on it. A `PounceFish` can see only a limited distance in its forward direction. If the `PounceFish` sees another fish, it rushes forward and eats the nearest one that it sees, ending up in the location where its prey was originally located. If the `PounceFish` does not see another fish, it acts as a `Fish`.

   The `PounceFish` class is shown below.

```
public class PounceFish extends Fish
{
  private int range;   // the distance that a PounceFish can see; range > 0


  /** Looks ahead range locations in current direction
   *  @return the nearest fish in that direction within range (if any);
   *          null if no such fish is found
   */
  private Fish findFish()
  {  /* to be implemented in part (a) */  }


  /** Acts for one step in the simulation
   */
  public void act()
  {  /* to be implemented in part (b) */  }


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

© 2007 The College Board. All rights reserved.
Visit apcentral.collegeboard.com (for AP professionals) and www.collegeboard.com/apstudents (for students and parents).

GO ON TO THE NEXT PAGE.

The following diagrams show an example environment containing a `PounceFish` (represented by `P`) and other fish (represented by `F1`, `F2`, etc.). The direction of the `PounceFish` is indicated by the character `">"` showing that, in this example, the `PounceFish` is facing east. If the `PounceFish` can see 2 or more locations ahead in its forward direction, it will see fish `F3` as shown in the first diagram and will move to that location to eat it, causing `F3` to die as shown in the second diagram.

Environment before the `PounceFish` acts

North

		0	1	2	3	4	5	
	0			F1				
West	1	F2	P>		F3	F4		**East**
	2		F5					
	3							

South

Environment after the `PounceFish` acts

North

		0	1	2	3	4	5	
	0			F1				
West	1	F2			P>	F4		**East**
	2		F5					
	3							

South

If the `PounceFish` in the first diagram above could see only 1 location ahead, it would not see any prey and therefore would act as an ordinary fish.

© 2007 The College Board. All rights reserved.
Visit apcentral.collegeboard.com (for AP professionals) and www.collegeboard.com/apstudents (for students and parents).

GO ON TO THE NEXT PAGE.

(a) Write the `PounceFish` method `findFish`. If any fish are located within `range` locations in the direction that the `PounceFish` is currently facing, the method returns the nearest of these. Otherwise, the method returns `null`.

Complete method `findFish` below.

```
/** Looks ahead range locations in current direction
 *  @return the nearest fish in that direction within range (if any);
 *          null if no such fish is found
 */
private Fish findFish()
```

(b) Override the `act` method for the `PounceFish` class. A `PounceFish` attempts to find a fish that it can eat. If it finds such a fish, the `PounceFish` eats it (causing it to die) and moves to its location. If the `PounceFish` does not find a fish that it can eat, it acts as an ordinary fish.

In writing `act`, assume that `findFish` works as specified, regardless of what you wrote in part (a).

Complete method `act` below.

```
/** Acts for one step in the simulation
 */
public void act()
{
  if ( ! isInEnv() )
    return;

  // Write your code below.
```

© 2007 The College Board. All rights reserved.
Visit apcentral.collegeboard.com (for AP professionals) and www.collegeboard.com/apstudents (for students and parents).

GO ON TO THE NEXT PAGE.

3. Consider a system for processing student test scores. The following class will be used as part of this system and contains a student's name and the student's answers for a multiple-choice test. The answers are represented as strings of length one with an omitted answer being represented by a string containing a single question mark (`"?"`). These answers are stored in an `ArrayList` in which the position of the answer corresponds to the question number on the test (question numbers start at 0). A student's score on the test is computed by comparing the student's answers with the corresponding answers in the answer key for the test. One point is awarded for each correct answer and ¼ of a point is deducted for each incorrect answer. Omitted answers (indicated by `"?"`) do not change the student's score.

```
public class StudentAnswerSheet
{
  private ArrayList<String> answers;   // the list of the student's answers


  /** @param key  the list of correct answers, represented as strings of length one
   *            Precondition: key.size() is equal to the number of answers in this answer sheet
   *  @return this student's test score
   */
  public double getScore(ArrayList<String> key)
  {  /* to be implemented in part (a) */  }


  /** @return the name of the student
   */
  public String getName()
  {  /* implementation not shown */  }

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

© 2007 The College Board. All rights reserved.
Visit apcentral.collegeboard.com (for AP professionals) and www.collegeboard.com/apstudents (for students and parents).

GO ON TO THE NEXT PAGE.

The following table shows an example of an answer key, a student's answers, and the corresponding point values that would be awarded for the student's answers. In this example, there are six correct answers, three incorrect answers, and one omitted answer. The student's score is ((6 * 1) - (3 * 0.25)) = 5.25 .

Question number	0	1	2	3	4	5	6	7	8	9
`key`	"A"	"C"	"D"	"E"	"B"	"C"	"E"	"B"	"B"	"C"
`answers`	"A"	"B"	"D"	"E"	"A"	"C"	"?"	"B"	"D"	"C"
Points awarded	1	−0.25	1	1	−0.25	1	0	1	−0.25	1

(a) Write the `StudentAnswerSheet` method `getScore`. The parameter passed to method `getScore` is an `ArrayList` of strings representing the correct answer key for the test being scored. The method computes and returns a `double` that represents the score for the student's test answers when compared with the answer key. One point is awarded for each correct answer and ¼ of a point is deducted for each incorrect answer. Omitted answers (indicated by `"?"`) do not change the student's score.

Complete method `getScore` below.

```
/** @param key  the list of correct answers, represented as strings of length one
 *             Precondition: key.size() is equal to the number of answers in this answer sheet
 *  @return this student's test score
 */
public double getScore(ArrayList<String> key)
```

© 2007 The College Board. All rights reserved.
Visit apcentral.collegeboard.com (for AP professionals) and www.collegeboard.com/apstudents (for students and parents).

GO ON TO THE NEXT PAGE.

(b) Consider the following class that represents the test results of a group of students that took a multiple-choice test.

```
public class TestResults
{
  private ArrayList<StudentAnswerSheet> sheets;

  /** Precondition: sheets.size() > 0;
   *                all answer sheets in sheets have the same number of answers
   *  @param key  the list of correct answers represented as strings of length one
   *              Precondition: key.size() is equal to the number of answers
   *                            in each of the answer sheets in sheets
   *  @return  the name of the student with the highest score
   */
  public String highestScoringStudent(ArrayList<String> key)
  {  /* to be implemented in part (b) */  }

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

Write the `TestResults` method `highestScoringStudent`, which returns the name of the student who received the highest score on the test represented by the parameter `key`. If there is more than one student with the highest score, the name of any one of these highest-scoring students may be returned. You may assume that the size of each answer sheet represented in the `ArrayList sheets` is equal to the size of the `ArrayList key`.

In writing `highestScoringStudent`, assume that `getScore` works as specified, regardless of what you wrote in part (a).

Complete method `highestScoringStudent` below.

```
  /** Precondition: sheets.size() > 0;
   *                all answer sheets in sheets have the same number of answers
   *  @param key  the list of correct answers represented as strings of length one
   *              Precondition: key.size() is equal to the number of answers
   *                            in each of the answer sheets in sheets
   *  @return  the name of the student with the highest score
   */
  public String highestScoringStudent(ArrayList<String> key)
```

© 2007 The College Board. All rights reserved.
Visit apcentral.collegeboard.com (for AP professionals) and www.collegeboard.com/apstudents (for students and parents).

GO ON TO THE NEXT PAGE.

4. In this question, you will complete methods in classes that can be used to represent a multi-player game. You will be able to implement these methods without knowing the specific game or the players' strategies.

   The `GameState` interface describes the current state of the game. Different implementations of the interface can be used to play different games. For example, the state of a checkers game would include the positions of all the pieces on the board and which player should make the next move.

   The `GameState` interface specifies these methods. The `Player` class will be described in part (a).

```
public interface GameState
{
  /** @return true if the game is in an ending state;
   *          false otherwise
   */
  boolean isGameOver();


  /** Precondition: isGameOver() returns true
   *  @return the player that won the game or null if there was no winner
   */
  Player getWinner();


  /** Precondition: isGameOver() returns false
   *  @return the player who is to make the next move
   */
  Player getCurrentPlayer();


  /** @return a list of valid moves for the current player;
   *          the size of the returned list is 0 if there are no valid moves.
   */
  ArrayList<String> getCurrentMoves();


  /** Updates game state to reflect the effect of  the specified move.
   *  @param move a description of the move to be made
   */
  void makeMove(String move);


  /** @return a string representing the current GameState
   */
  String toString();

}
```

The `makeMove` method makes the move specified, updating the state of the game being played. Its parameter is a `String` that describes the move. The format of the string depends on the game. In tic-tac-toe, for example, the move might be something like `"X-1-1"`, indicating an X is put in the position (1, 1).

© 2007 The College Board. All rights reserved.
Visit apcentral.collegeboard.com (for AP professionals) and www.collegeboard.com/apstudents (for students and parents).

(a) The `Player` class provides a method for selecting the next move. By extending this class, different playing strategies can be modeled.

```
public class Player
{
  private String name;     // name of this player

  public Player(String aName)
  {  name = aName;  }

  public String getName()
  {  return name;  }

  /**  This implementation chooses the first valid move.
   *   Override this method in subclasses to define players with other strategies.
   *   @param state  the current state of the game; its current player is this player.
   *   @return  a string representing the move chosen;
   *            "no move"  if no valid moves for the current player.
   */
  public String getNextMove(GameState state)
  {   /*  implementation not shown  */   }
}
```

The method `getNextMove` returns the next move to be made as a string, using the same format as that used by `makeMove` in `GameState`. Depending on how the `getNextMove` method is implemented, a player can exhibit different game-playing strategies.

Write the complete class declaration for a `RandomPlayer` class that is a subclass of `Player`. The class should have a constructor whose `String` parameter is the player's name. It should override the `getNextMove` method to randomly select one of the valid moves in the given game state. If there are no valid moves available for the player, the string `"no move"` should be returned.

© 2007 The College Board. All rights reserved.
Visit apcentral.collegeboard.com (for AP professionals) and www.collegeboard.com/apstudents (for students and parents).

GO ON TO THE NEXT PAGE.

(b) The `GameDriver` class is used to manage the state of the game during game play. The `GameDriver` class can be written without knowing details about the game being played.

```
public class GameDriver
{
  private GameState state;   // the current state of the game

  public GameDriver(GameState initial)
  {  state = initial;  }


  /** Plays an entire game, as described in the problem description
   */
  public void play()
  {  /* to be implemented in part (b) */ }

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

Write the `GameDriver` method `play`. This method should first print the initial state of the game. It should then repeatedly determine the current player and that player's next move, print both the player's name and the chosen move, and make the move. When the game is over, it should stop making moves and print either the name of the winner and the word `"wins"` or the message `"Game ends in a draw"` if there is no winner. You may assume that the `GameState makeMove` method has been implemented so that it will properly handle any move description returned by the `Player getNextMove` method, including the string `"no move"`.

Complete method `play` below.

```
  /** Plays an entire game, as described in the problem description
   */
  public void play()
```

STOP

END OF EXAM

© 2007 The College Board. All rights reserved.
Visit apcentral.collegeboard.com (for AP professionals) and www.collegeboard.com/apstudents (for students and parents).