AP® Computer Science A Exam

SECTION I: Multiple-Choice Questions

2009

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

At a Glance

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

Instructions

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

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

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

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

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

About Guessing

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

Section I

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

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

Notes:

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

Unauthorized copying or reuse of any part of this page is illegal.

1. Consider the following code segment.

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

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

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

nauthorized copying or reuse of
ny part of this page is illegal.

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

Unauthorized copying or reuse of any part of this page is illegal.

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

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

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

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

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

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

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

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

nauthorized copying or reuse of
y part of this page is illegal.

4. Consider the following code segment.

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

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

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

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

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

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

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

---

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

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

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

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

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

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

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

Questions 6-7 refer to the following code segment.

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

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

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

(A) 0

(B) 1

(C) 2

(D) $n - 1$

(E) $n - 2$

---

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

(A) 2

(B) $n - 1$

(C) $n - 2$

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

(E) n^2

unauthorized copying or reuse of
any part of this page is illegal.

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

```
private int[] vals;

public int calcTotal()
{
  int total = 0;

  /* missing code */

  return total;
}
```

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

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

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

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

(A) I only

(B) II only

(C) III only

(D) I and III

(E) II and III

Unauthorized copying or reuse of any part of this page is illegal.

). Consider the following code segment.

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

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

(A) `abcdef`

(B) `aabbccddeeff`

(C) `abbccddeef`

(D) `abcbcdcdedef`

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

ıauthorized copying or reuse of
y part of this page is illegal.

10. Consider the following method.

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

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

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

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

(A) `5 20 5`

(B) `5 20 10`

(C) `5 25 5`

(D) `5 25 10`

(E) `30 25 10`

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAG

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

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

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

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

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

(C) `nameList.remove(j)`

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

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

Unauthorized copying or reuse of
any part of this page is illegal.

12. Consider the following incomplete method.

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

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

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

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

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

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

return n;
```


Unauthorized copying or reuse of any part of this page is illegal.

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

uthorized copying or reuse of part of this page is illegal.

13. Consider the following method.

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

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

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

(A) `16`

(B) `56`

(C) `123456`

(D) `654321`

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

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAG

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

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

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

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

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

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

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

(A) I only

(B) II only

(C) III only

(D) II and III only

(E) I, II, and III

uthorized copying or reuse of part of this page is illegal.

15. Consider the following method.

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

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

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

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

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

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

(A) I only

(B) II only

(C) III only

(D) I and III

(E) II and III

Unauthorized copying or reuse of any part of this page is illegal.

Consider the following class declaration.

```
public class SomeClass
{
  private int num;

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

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

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

The following code segment appears in another class.

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

one.increment(200);

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

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

(A) 100 100 100

(B) 300 100 100

(C) 300 100 300

(D) 300 300 100

(E) 300 300 300

thorized copying or reuse of
art of this page is illegal.

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

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

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

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

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

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

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

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

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

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

Unauthorized copying or reuse of any part of this page is illegal.

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

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

The result of evaluating the expression above is best described as

(A) always `true`

(B) always `false`

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

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

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

uthorized copying or reuse of part of this page is illegal.

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

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

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

(A) `r = d + 0.5;`

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

(C) `r = d * 5.0;`

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

(E) `r = d * 5.5;`

Unauthorized copying or reuse of any part of this page is illegal.

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

```
private int assignmentsCompleted;
private double testAverage;

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

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

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

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

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

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

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

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

(A) I only

(B) II only

(C) I and III only

(D) II and III only

(E) I, II, and III

uthorized copying or reuse of part of this page is illegal.

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

21. Consider the following code segment.

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

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

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

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

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

(A) aaaXXX555

(B) aaaXXX

(C) XXX555

(D) 555

(E) aaa

Unauthorized copying or reuse of any part of this page is illegal.

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

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

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

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

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

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

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

Which of the replacements will produce the desired behavior?

(A) I only

(B) II only

(C) I and II only

(D) I and III only

(E) I, II, and III

uthorized copying or reuse of part of this page is illegal.

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

Unauthorized copying or reuse of any part of this page is illegal.

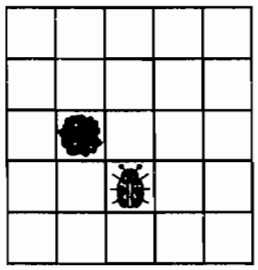
GO ON TO THE NEXT PAG

. Consider the following `TestBug` class declaration.

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

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

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



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

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

(B) Moves forward two locations and turns 180 degrees

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

(D) Moves forward one location and turns 180 degrees

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

uthorized copying or reuse of part of this page is illegal.

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

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

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

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


Unauthorized copying or reuse of any part of this page is illegal.

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

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

A) I only

B) II only

C) III only

D) I and II only

E) I, II, and III

uthorized copying or reuse of part of this page is illegal.

26. Consider the following code segment.

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

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

(A) 0 2 1 0 2

(B) 0 2 0 2 0 2

(C) 0 2 1 0 2 1 0

(D) 0 2 0 2 0 2 0

(E) 0 1 2 1 2 1 2

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAG

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

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

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

  return isEven;
}
```

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

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

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

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

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

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

uthorized copying or reuse of part of this page is illegal.

28. Consider the following code segment.

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

boolean result = (x < y);

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

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

(A) Only when `x < y`

(B) Only when `x >= y`

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

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

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

---

29. Consider the following code segment.

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

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

(A) `0 1 2 3`

(B) `0 0 1 0 1 2`

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

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

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

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAG

30. Consider the following class declarations.

```
public class Base
{
  private int myVal;

  public Base()
  {  myVal = 0;  }

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


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

Which of the following statements will NOT compile?

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

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

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

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

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

---

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

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

is equivalent to which of the following?

(A) `true`

(B) `false`

(C) `a == b`

(D) `a != b`

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

nauthorized copying or reuse of
ny part of this page is illegal.

32. Consider the following code segment.

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

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

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

(A) 0

(B) 6

(C) 12

(D) 24

(E) 30

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE

3. Consider the following method.

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

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

  return s;
}
```

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

(A) 20

(B) 45

(C) 55

(D) 100

(E) 385

ıauthorized copying or reuse of
y part of this page is illegal.

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

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

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

  return pos;
}
```

For example, consider the following code segment.

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

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

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

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

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

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

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

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

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

Unauthorized copying or reuse of any part of this page is illegal.

5. Consider the following code segments.

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

  k = k + 3;
}
```

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

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

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

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

(A) I only

(B) II only

(C) I and II only

(D) II and III only

(E) I, II, and III

nauthorized copying or reuse of
y part of this page is illegal.

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

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

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


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

  changeIt(nums, value);

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

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

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

(A) 0 0 0 0 0 0

(B) 0 0 0 0 0 6

(C) 1 2 3 4 5 6

(D) 1 2 3 4 5 0

(E) `changeIt` will throw an exception.

Unauthorized copying or reuse of any part of this page is illegal.

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

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

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

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

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

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

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

seq = temp;
```

(A) II only

(B) III only

(C) I and II

(D) I and III

(E) II and III

authorized copying or reuse of
part of this page is illegal.

38. Consider the following code segment.

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

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

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

(A) Precedence rules make the if condition true.

(B) Associativity rules make the if condition true.

(C) Roundoff error makes the if condition true.

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

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

---

39. Consider the following recursive method.

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

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

  return dig;
}
```

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

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

(A) 20

(B) 102

(C) 210

(D) 1020

(E) 2101

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAG

0. Consider the following method.

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

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

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

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

(A) `today`

(B) `todayto`

(C) `todayoday`

(D) `todayodayay`

(E) `todayodaydayayy`

END OF SECTION I

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

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

authorized copying or reuse of
y part of this page is illegal.

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

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 isa Location with the same row and column as this location; false otherwise

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

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

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

Compass directions:

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

Turn angles:

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

info.gridworld.grid.Grid<E> interface

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

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

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

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

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

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

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

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

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

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

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

info.gridworld.actor.Actor class

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

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

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

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

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

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

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

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

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

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

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

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

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

`public Rock()`
constructs a black rock

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

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

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

`public Flower()`
constructs a pink flower

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

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

Appendix C — Testable Code for APCS A/AB

3ug.java

```
ackage info.gridworld.actor;

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

mport java.awt.Color;

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

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


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


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


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

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

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

oxBug.java

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

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

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

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

Critter.java

```
package info.gridworld.actor;

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

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


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

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

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

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

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

ChameleonCritter.java

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

import java.util.ArrayList;

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

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

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

Appendix E — Quick Reference A/AB

Location Class (implements Comparable)

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

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

Grid<E> Interface

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

Actor Class

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

Rock Class (extends Actor)

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

Flower Class (extends Actor)

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

Bug Class (extends Actor)

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

BoxBug Class (extends Bug)

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

Critter Class (extends Actor)

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

ChameleonCritter Class (extends Critter)

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

Appendix G — Index for Source Code

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

Bug.java

Bug() C1
Bug(Color bugColor) C1
act() C1
turn() C1
move() C2
canMove() C2

BoxBug.java

BoxBug(int length) C3
act() C3

Critter.java

act() C4
getActors() C4
processActors(ArrayList<Actor> actors) C5
getMoveLocations() C5
selectMoveLocation(ArrayList<Location> locs) C5
makeMove(Location loc) C6

ChameleonCritter.java

processActors(ArrayList<Actor> actors) C6
makeMove(Location loc) C6

AP® Computer Science A Exam

SECTION II: Free-Response Questions

2009

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

At a Glance

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

IMPORTANT Identification Information

PLEASE PRINT WITH PEN:

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

Instructions

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

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

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

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

COMPUTER SCIENCE A
SECTION II

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

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

tes:

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

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

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

Section II

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

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

The number cube is represented by the following class.

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

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

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

GO ON TO THE NEXT PAG

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

Complete method `getCubeTosses` below.

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

Section II

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

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

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

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

WRITE YOUR SOLUTION ON THE NEXT PAGE.

mplete method `getLongestRun` below.

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

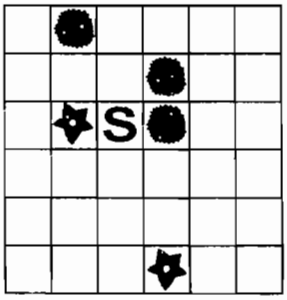
Section II

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

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

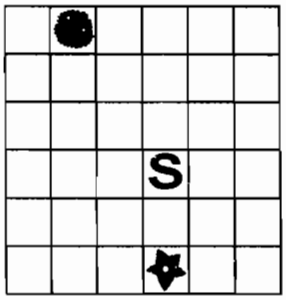
Consider the following scenario.

INITIAL WORLD



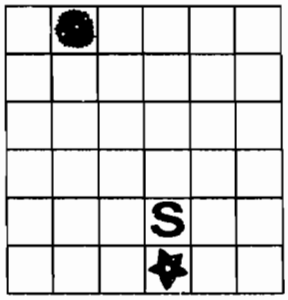
`StockpileCritter` is in location (2, 2), stockpile is empty

AFTER ONE ACT



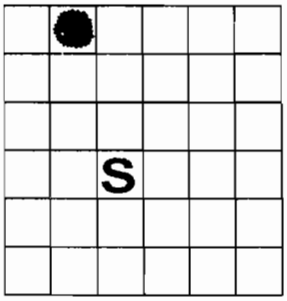
Gathered 3 actors, used 1 energy unit, 2 remaining in stockpile, moved to location (3, 3)

AFTER TWO ACTS



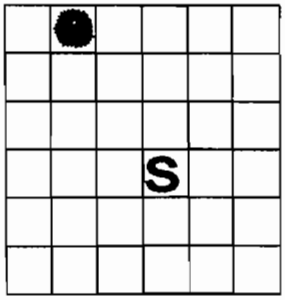
No actors gathered, used 1 energy unit, 1 remaining in stockpile, moved to location (4, 3)

AFTER THREE ACTS



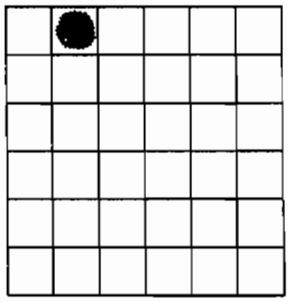
Gathered 1 actor, used 1 energy unit, 1 remaining in stockpile, moved to location (3, 2)

AFTER FOUR ACTS



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

AFTER FIVE ACTS



Stockpile empty, removed self from grid

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

ADDITIONAL WORK SPACE

Section II

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

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

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

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

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

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

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

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

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

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


GO ON TO THE NEXT PAGE

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

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

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

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

Complete method `getChargingCost` below.

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

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

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

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

WRITE YOUR SOLUTION ON THE NEXT PAGE.

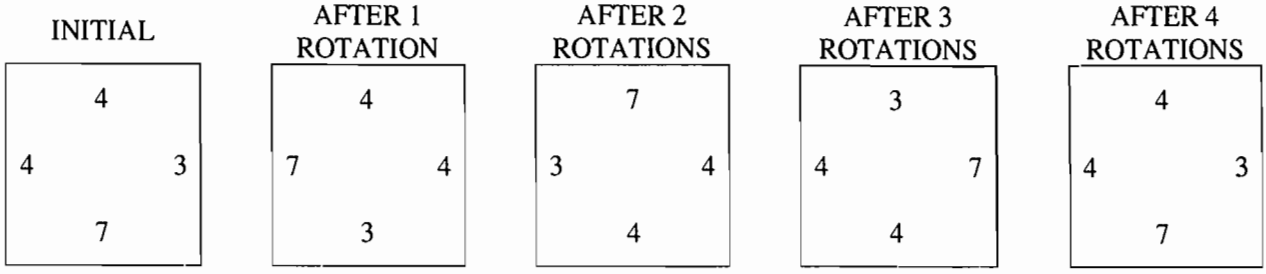
Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

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

Section II

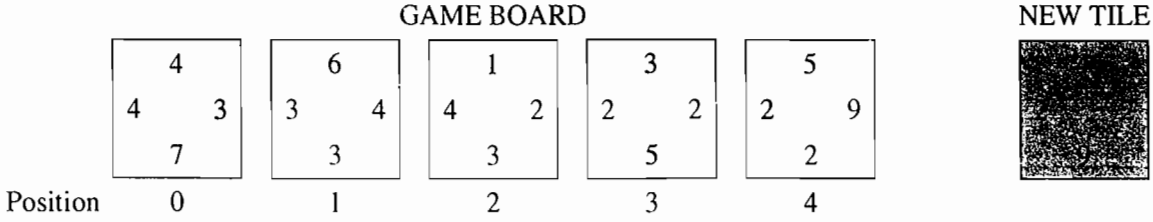
4. A game uses square tiles that have numbers on their sides. Each tile is labeled with a number on each of its four sides and may be rotated clockwise, as illustrated below.



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

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


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


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

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

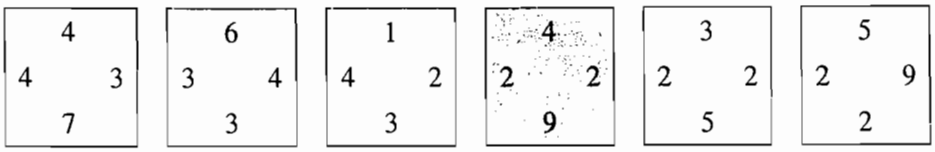
Tiles are placed on a game board so that the adjoining sides of adjacent tiles have the same number. The following figure illustrates an arrangement of tiles and shows a new tile that is to be placed on the game board.



GO ON TO THE NEXT PAGE

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

GAME BOARD AFTER INSERTING TILE



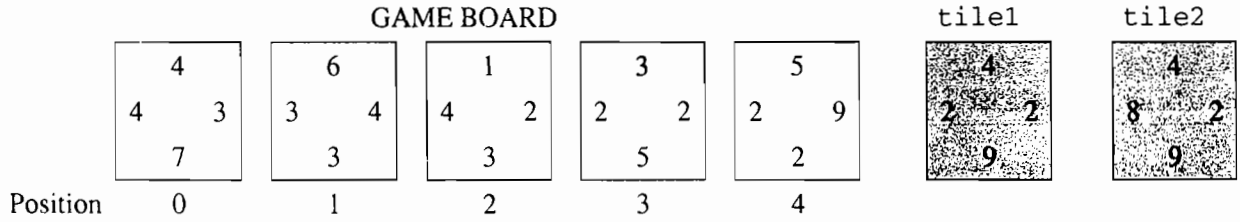
partial definition of the `TileGame` class is given below.

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

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

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

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

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

Section II

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

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



WRITE YOUR SOLUTION ON THE NEXT PAGE.

GO ON TO THE NEXT PAG

Complete method `getIndexForFit` below.

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

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

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

Complete method `insertTile` below.

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

STOP

END OF EXAM