AP® Computer Science A Exam

SECTION I: Multiple-Choice Questions

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

At a Glance
Total Time 1 hour 30 minutes **Number of Questions** 40 **Percent of Total Score** 50% **Writing Instrument** Pencil required

Instructions

Section I of this examination 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.

Chicago is a
(A) state
(B) city
(C) country
(D) continent
(E) county

Sample Answer

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 the multiple-choice questions.

About Guessing

Many candidates wonder whether or not to guess the answers to questions about which they are not certain. Multiple-choice scores are based on the number of questions answered correctly. Points are not deducted for incorrect answers, and no points are awarded for unanswered questions. Because points are not deducted for incorrect answers, you are encouraged to answer all multiple-choice questions. On any questions you do not know the answer to, you should eliminate as many choices as you can, and then select the best answer among the remaining choices.

Java Quick Reference

Class Constructors and Methods	Explanation
String Class	
`String(String str)`	Constructs a new `String` object that represents the same sequence of characters as `str`
`int length()`	Returns the number of characters in a `String` object
`String substring(int from, int to)`	Returns the substring beginning at index `from` and ending at index `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
`boolean equals(String other)`	Returns `true` if `this` is equal to `other`; returns `false` otherwise
`int compareTo(String other)`	Returns a value <0 if `this` is less than `other`; returns zero if `this` is equal to `other`; returns a value >0 if `this` is greater than `other`
Integer Class	
`Integer(int value)`	Constructs a new `Integer` object that represents the specified `int` value
`Integer.MIN_VALUE`	The minimum value represented by an `int` or `Integer`

`Integer.MAX_VALUE`	The maximum value represented by an `int` or `Integer`
`int intValue()`	Returns the value of this `Integer` as an `int`
Double Class	
`Double(double value)`	Constructs a new `Double` object that represents the specified `double` value
`double doubleValue()`	Returns the value of this `Double` as a `double`
Math Class	
`static int abs(int x)`	Returns the absolute value of an `int` value
`static double abs(double x)`	Returns the absolute value of a `double` value
`static double pow(double base, double exponent)`	Returns the value of the first parameter raised to the power of the second parameter
`static double sqrt(double x)`	Returns the positive square root of a `double` value
`static double random()`	Returns a `double` value greater than or equal to `0.0` and less than `1.0`
ArrayList Class	
`int size()`	Returns the number of elements in the list
`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 adds 1 to size
`E get(int index)`	Returns the element at position `index` in the list

`E set(int index, E obj)`	Replaces the element at position `index` with `obj`; returns the element formerly at position `index`
`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 subtracts 1 from size; returns the element formerly at position `index`
Object Class	
`boolean equals(Object other)`	
`String toString()`	

COMPUTER SCIENCE A

SECTION I

Time—1 hour and 30 minutes

Number of Questions—40

Percent of total exam grade—50%

Directions: Determine the answer to each of the following questions or incomplete statements, using the available space for any necessary scratchwork. 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 Java Quick Reference 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 the method calls are not `null` and that methods are called only when their preconditions are satisfied.

1. Which of the following will print a number less than 5?

I. `System.out.println (24 / 5 % 3 * 2);`
II. `System.out.println (12 / 3 * 2 + 1);`
III. `System.out.println (1 + 4 % 3 * 2);`

(A) I only
(B) II only
(C) I and II only
(D) I and III only
(E) I, II, and III

2. What output is generated by the following line of code:

```
System.out.println("Simon says, \n\\\"insert phrase
here\"//");
```

(A) `Simon says, \n \\ insert phrase here//`
(B) `Simon says, \n \\ “insert phrase here”//`
(C) `Simon says, \n “insert phrase here”`
(D)
```
Simon says,
    \“insert phrase here”/
```
(E)
```
Simon says,
    \“insert phrase here”//
```

3. Consider the following class with the numbers added for reference:

```
public class Tree
{
     private String name;
     private double height;
     private int rateOfGrowth;

     Tree(String n, double h, int r)
     {
          name = n;
          height = h;
          rateOfGrowth = r;
     }
     Tree(double h, int r)
     {
          height = h;
          rateOfGrowth = r;
     }
     Tree(String n)
     {
          name = n;
     }
     public String toString()
     {
          return name + " can grow up to " + height + "
          feet high, at a rate of " + rateOfGrowth + "
          inches per year";
     }
}
```

Which of the following code excerpts in a client program would cause an error?

(A)
```
Tree elm = new Tree("Elm", 60, 36);
System.out.println(elm.toString());
```

(B)
```
Tree riverBirch = new Tree("River Birch", 40.0,
 (int)13.0);
```

```
    System.out.println(riverBirch.toString());
(C) Tree redMaple = new Tree(60.0, 18);
    System.out.println(redMaple.toString());
(D) Tree redwood = new Tree("Redwood", 300, 24.0);
    System.out.println(redwood.toString());
(E) Tree sequoia = new Tree("Sequoia");
    System.out.println(sequoia.toString());
```

Questions 4–5 refer to the class `SalesRep`.

```
public class SalesRep
{
     private int idNum;
     private String Name;
     private int ytdSales;

     SalesRep(int i, String n, int ytd)
     {
          idNum = i;
          name = n;
          ytdSales = ytd;
     }

     public int getYtdSales() {return ytdSales;}
}
```

4. A client method, `computeBonus`, will return a `salesRep bonus` computed by multiplying his `ytdSales` by a percentage.

/** Precondition: `SalesRep s` has `ytdSales` >= 0
* @param `s` a `SalesRep`
* @param `percentage` represents what percent of the `ytdSales` represents the `bonus`
* @return amount of `bonus` for the `SalesRep (ytdSales * bonus)`
*/

```
public static double computeBonus(SalesRep s, double
percentage)
{   /* missing code */   }
```

Which replacement for /* missing code */ is correct?

(A) `return ytdSales() * percentage;`
(B) `return getYtdSales() * percentage;`
(C) `return s.ytdSales() * percentage;`
(D) `return s.getYtdSales() * percentage;`
(E) `return s.getYtdSales() * s.percentage;`

5. An `ArrayList` was created to store a `SalesRep` object for every salesperson in the XYZ company. Below is the declaration for that `ArrayList`.

```
ArrayList<SalesRep> list1 = new ArrayList<SalesRep>();
```

The company decided to pay each `SalesRep` a bonus since the company had a very profitable year. Management wishes to project the total of that payout, but to do so, must first calculate the total sales for the company by adding together the `ytdSales` from each `SalesRep`. Which code excerpt will compute the `ytdSales` for the company?

(A)
```
double sum = 0;
   for (ArrayList r : list1)
   {  sum += r.get.getYtdSales();  }
```
(B)
```
double sum = 0;
   for (SalesRep r.get : list1)
   {  sum += r.getYtdSales();  }
```
(C)
```
double sum = 0;
   for (SalesRep r : list1)
   {   sum += r.getYtdSales();  }
```

(D)
```
double sum = 0;
for (int i = 0; i < list1.size(); i++)
{   sum += SalesRep.get(i).getYtdSales();}
```
(E)
```
double sum = 0;
for (int i = 0; i <= list1.size(); i++)
{   sum += list1.get(i).getYtdSales();}
```

6. Which of the following will print after the following code is executed?

```
String str = new String("superstar");
System.out.print(str.substring (1, 3) + " ");
str.substring(1);
System.out.print(str.substring (1, 3) + " ");
str.substring(1);
System.out.print(str.substring (1, 3) + " ");
```

(A) `su up pe`
(B) `up pe er`
(C) `sup upe per`
(D) `up up up`
(E) A `StringIndexOutOfBoundsException` will occur

7. A chess game must take turns allowing black and white players to move on the board. The player is indicated by the following variable where `true` indicates black and `false` indicates white:

```
boolean isBlack;
```

At the end of each player's turn, the variable `isBlack` must alternate between `true` and `false` to indicate the next player's turn. Consider the following code examples, then determine those that would accomplish this purpose.

I. `isBlack = !isBlack;`

II.
```
if (!isBlack)
    isBlack = true
else
    isBlack = false;
```
III.
```
if (isBlack)
    isBlack = false;
```

(A) I only
(B) II only
(C) I and II only
(D) I and III only
(E) I, II, and III

8. The following variables are declared, but their contents are unknown.

```
String str1 = new String("/*unknown value*/");
String str2 = new String("/*unknown value*/");
```

Consider the following decision statements and determine the statement that would fail to compare whether the value of `str1` is the same as the value of `str2`.

(A)
```
if (str1 == str2)
```
(B)
```
if (str1.equals(str2))
```
(C)
```
if (str1.compareTo(str2) == 0)
```
(D)
```
if
    (str1.substring(0).equals(str2.substring(0,str2.length
    ())))
```
(E)
```
if (str1.length() == str2.length() &&
    str1.indexOf(str2) == 0)
```

9. Given the following boolean expression: `!(p || (q || !r))`

Which of the following conditions would result in an evaluation as `true`?

(A) `p = true  q = true  r = true`
(B) `p = true  q = false  r = true`
(C) `p = false  q = true  r = false`
(D) `p = false  q = false  r = false`
(E) `p = false  q = false  r = true`

10. Which boolean expression and values demonstrate a short-circuit evaluation?

(A) `p || !(q && r)  p = true  q = true  r = true`
(B) `p || (q && !r)  p = false  q = true  r = false`
(C) `p || !(q || r)  p = false  q = true  r = true`
(D) `p && q && r  p = true  q = true  r = true`
(E) `p && !(q && r)  p = true  q = true  r = true`

11. Assume that `p` and `q` are boolean variables and have been properly initialized.

```
! (!p || q) || ! (p || !q)
```

Which of the following best describe the result of evaluating the expression above?

(A) `true` always
(B) `false` always
(C) `true` only if `p` is `true`
(D) `true` only if `q` is `true`
(E) `true` only if `p` and `q` have opposite truth values

12. What output is generated by the following code excerpt:

```
int count = 0;
String star = "*";
for (int i = 1; i < 11; i++)
     for (int j = 10; j > 1; j -= 2)
     {
          star += "**";
          count++;
     }
System.out.print("\n" + count + " " + star.length());
```

(A) 50 100
(B) 50 101
(C) 51 100
(D) 51 101
(E) 90 181

13. What output is generated by the following code excerpt:

```
for (int i = 1; i <= 5; i++)
{
     for (int j = 1; j < i; j++)
     {
          System.out.print("- ");
     }
     for (int j = i; j <= 5; j++)
     {
          System.out.print("* ");
     }
     System.out.println();
}
```

(A)
```
- - - - *
- - - * *
- - * * *
- * * * *
* * * * *
```
(B)
```
* * * * *
```

```
    - * * * *
    - - * * *
    - - - * *
    - - - - *
(C) * - - - -
    * * - - -
    * * * - -
    * * * * -
    * * * * *
(D) * * * * *
    * * * * -
    * * * - -
    * * - - -
    * - - - -
(E) - - - - -
    - - - - *
    - - - * *
    - - * * *
    - * * * *
```

14. Consider the following code segments and determine those that would produce the same output.

I.
```
int sum = 0;
for (int i = 1; i < 3; i++)
{
    sum += 2 * i + 1;
}
System.out.println(sum);
```

II.
```
int sum = 0;
for (int i = 1; i <= 5; i++)
{
    if (i % 2 == 1)
        sum += i;
}
```

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

III.
```
int i = 5;
int sum = i;
while (i > 1)
{
    i -= 2;
    sum += i;
}
System.out.println(sum);
```

(A) I and II only
(B) I and III only
(C) II and III only
(D) I, II and III
(E) All three outputs are different.

15. Consider the following code segment:

```
/**
* @param number is initialized with a positive integer value
* @return the sum of odd integers between 1 and number
*/

public static int sumOdds(int number)
{
     int sum = 0;
     for (/* missing code */)
     {
          sum += k;
     }
     return sum;
}
```

Which of the following replacements for /* missing code */ will satisfy the conditions of the method?

I. `int k = 1; k <= number; k++`

II. `int k = 1; k <= number; k += 2`
III. `int k = number; k >= 1; k -= 2`

(A) I only
(B) II only
(C) III only
(D) I and III only
(E) I, II, and III

16. Using the following method to find the index of the largest value in an array. Choose the replacement(s) for /* some code */ that will accomplish the task as described.

/** Precondition: `arr` is initialized with integer values and is not empty
* `Integer.MIN_VALUE` is a static constant containing the value –2147483648
* @param `arr` the array to be processed
* @return the location of the largest value in the array; if
* the largest value is stored in more than one element, return the
* first location within the array where the element is located
*/

Example: `int arr[] = {5, -3, 2, 5};` The method should return `0`.

```
public static int findMaximumIndex(int[] arr)
{
        /* some code */
}
```

I `int max = Integer.MIN_VALUE;`
`int loc = -1;`

```
int i = 0;
while (i < arr.length)
{
     if (arr[i] > max)
     {
          max = arr[i];
          loc = i;
     }
     i++;
}
return loc;
```

II
```
int max = arr[arr.length – 1];
int loc = arr.length – 1;
int i = arr.length – 1;
while (i >= 0)
{
     if (arr[i] > max)
     {
          max = arr[i];
          loc = i;
     }
     i––;
}
return loc;
```

III
```
int i = 0;
int loc = 0;
int max = arr[loc];
while (i < arr.length)
{
     if (arr[i] > max)
     {
          max = arr[i];
          loc = i;
     }
     i++;
}
return loc;
```

(A) I only
(B) II only
(C) III only
(D) I and III only
(E) I, II, and III

17. The values of the static fields in the `Integer` class named `MIN_VALUE` and `MAX_VALUE` are constants containing the values `-2147483648` and `2147483647` respectively. Determine the statement(s) below that will throw a compiler error.

(A) `int min = Integer.MIN_VALUE;`
(B) `int min = Integer.MIN_VALUE - 1;`
(C) `int num = Integer.MIN_VALUE + Integer.MAX_VALUE;`
(D) `int max = 2147483648;`
(E) All of the statements above

18. What is the output after the following code is executed:

```
int num = 5;
System.out.print("" + half(num) + num);

public static double half(int n)
{
     return n/2;
}
```

(A) `2.05`
(B) `2.5`
(C) `7.0`
(D) `7.5`
(E) Nothing will print, as an error will be thrown.

19. Consider the following class declaration that is intended to represent a rectangle.

```
public class MyRectangle
{
     private int width;
     private int height;
     private int perimeter;

     MyRectangle(int w, int h)
     {
          width = w;
          height = h;
          int perimeter = 2 * (width + height);
     }

     public double getPerimeter()
     {
          return perimeter;
     }
}
```

What is the output after the following code is executed:

```
MyRectangle rect = new MyRectangle(2, 3);
System.out.print(rect.getPerimeter());
```

(A) `0`
(B) `0.0`
(C) `10`
(D) `10.0`
(E) None of the above; an error will be thrown.

Questions 20–22 refer to the following incomplete class declaration that is intended to represent a car.

```
public class Car
{
     private String model;
     private int numDoors;
     private boolean isFourWheelDrive;
     private int mpg;
```

```
/**
* Constructs a car
*/
Car()
{

}

/**
* Constructs a car
* @param model
* @param numDoors
* @param isFourWheelDrive
* @param mpg
*/
Car(String model, int numDoors, boolean
isFourWheelDrive, int mpg)
{

      /* Implementation not shown */
}

/**
* Compute the miles per gallon (milesDriven/gallons)
* and stores the result in mpg, rounded to the nearest gallon
* >= 0.5 would round up, < 0.5 would round down
* @param milesDriven
* @param gallons
*/
public void setMpg(int milesDriven, double gallons)
{
      /* Implementation not shown */
}

/**
* Updates the model of the car
```

```
* @param model
*/
public void setModel(String model)
{
     model = model;
}

/**
*
* @ return model
*/
public String getModel()
{
     return model;
}
/**
* Updates the number of doors on the car
* @param numDoors
*/
public void setnumDoors(int numDoors)
{
     /* Implementation not shown */
}

/**
* Updates isFourWheelDrive true if the car has four-wheel drive, false if not
* @param isFourWheelDrive
*/
public void setIsFourWheelDrive (boolean
isFourWheelDrive)
{
     /* Implementation not shown */
}

/**
```

```
     * Returns the values stored in the object
     */
     public String toString()
     {
          return "Model: " + model + " is 4-wheel drive: "
          + isFourWheelDrive;
     }
}
```

20. The programmer wishes to add an additional constructor. Which of the following would be invalid as a constructor?

(A) `Car(String model)`
(B) `Car(String model, int mpg, boolean isFourWheelDrive)`
(C) `Car(String model, int numDoors)`
(D) `Car(String model, int mpg, boolean isFourWheelDrive, int numDoors)`
(E) `Car(String model, int numDoors, boolean isFourWheelDrive, int milesDriven, double gallons)`

21. The following code is in a client program.

```
Car fordTruck = new Car();

fordTruck.setModel("Tacoma");

if (fordTruck.getModel().equals("Tacoma"))
     fordTruck.setIsFourWheelDrive(true);

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

What will be output by the program?

(A) `Model: Tacoma is 4-wheel drive: true`
(B) `Model: Tacoma is 4-wheel drive: false`
(C) `Model: null is 4-wheel drive: true`
(D) `Model: null is 4-wheel drive: false`

(E) A `NullPointerException` will be thrown

22. Which of the following can replace /* Implementation not shown */ in the `setMpg` method?

(A) `mpg = milesDriven / gallons;`
(B) `mpg = milesDriven / gallons + 0.5;`
(C) `mpg = milesDriven / (int) gallons + 0.5;`
(D) `mpg = (int) (milesDriven / gallons) + 0.5;`
(E) `mpg = (int) (milesDriven / gallons + 0.5);`

Questions 23–25 refer to the following class declarations.

```
public class Bee
{
     private int lifeSpan;
     private String name;

     Bee(String n, int life)
     {
          lifeSpan = life;
          name = n;
     }

     public String getName()
     {
          return name;
     }

     public String toString()
     {
          return " The " + name + "s live " + lifeSpan + "
          months.";
     }
}

public class Queen extends Bee
{
```

```
    private int eggsPerDay;

    Queen(String name, int months, int eggs)
    {
        super(name, months);
        eggsPerDay = eggs;
    }

    public String toString()
    {
        return " The queen" + " lays " + eggsPerDay + "
        eggs.";
    }
}
```

23. What is output after the following lines of code are executed in a client program?

```
Bee bee1 = new Queen("honey bee", 6, 2000);
System.out.println(bee1.toString());
```

(A) `The queen lays 2000 eggs.`
(B) `The honey bee lays 2000 eggs.`
(C) `The honey bees live 6 months.`
(D) `The honey bees live 6 months. The queen lays 2000 eggs.`
(E) An error will be thrown because of mismatched data types.

24. The programmer wishes to include an accessor method for the field called `name`. In which class should this method be included?

(A) Only in the `Bee` class.
(B) Only in the `Queen` class.
(C) It could be included in either the `Bee` class or the `Queen` class.
(D) It should be included in both classes.
(E) The method should not be included in either class because the data is private.

25. Only queen bees lay eggs that hatch into productive bees for the colony. The other two types of bees in a colony are drone (male) and worker (female) bees. What information would be most helpful to determine the best design for a class implementation for drones and worker bees?

(A) Whether the drones and workers have distinctly different lifespans than queens
(B) Whether there are many more drones and workers than queens
(C) Whether there are more drones than workers (or the reverse)
(D) What data needs to be included to accurately represent the state and behavior of drones and workers
(E) All the above

26. A program passes an array to a method in the same class to create multiples of 5. Determine the output of the following code.

```
double arr[];
arr = new double[5];
multOf5(arr);
for (int i = 0; i < arr.length; i++)
     System.out.print(arr[i] + " ");

public static void multOf5(double a[])
{
     for (int i = 0; i < a.length; i++)
          a[i] = i * 5;
     }
```

(A) `0.0 5.0 10.0 15.0 20.0`
(B) `0.0 0.0 0.0 0.0 0.0 0.0`
(C) `0.0 5.0 10.0 15.0 20.0 25.0`
(D) An error will occur because of a type mismatch.
(E) An error will occur because the array was not initialized.

27. Consider the following method:

```
public static int[] op(int[][] matrix, int m)
{
     int[] result = new int[matrix.length];
     for (int j = 0; j < matrix.length; j++)
     {
          result[j] = matrix[m][j] - matrix[j][m];
     }
     return result;
}
```

The following code segment appears in the same class:

```
int mat[][] = {{1, 2, 3, 4}, {1, 3, 5, 7}, {2, 4, 6, 8},
{4, 3, 2, 1}};
int[] arr = op(mat, 3);
```

Which of the following represents the contents of `arr` as a result of the code segment?

(A) `{0, -4, -6, 0}`
(B) `{0,  4,  6, 0}`
(C) `{8, 10, 10, 2}`
(D) `{2,  4,  6, 8}`
(E) `{3,  5,  6, 2}`

28. A programmer wishes to declare and initialize an `ArrayList` with random integers between 1 and 100. Choose the code that can replace /* missing code */ to accomplish the task.

```
ArrayList<Integer> list2= new ArrayList<Integer>();
for (int i = 0; i < 1000; i++)
{

     /* missing code */
}
```

(A) `list2.add((Math.random() * 100 + 1));`
(B) `list2.add((int)(Math.random() * 99 + 0.5));`
(C) `list2.add((int)(Math.random() * 100 + 0.5));`
(D) `list2.add((int)(Math.random() * 100 + 1));`
(E) `list2.add((Integer)(Math.random() * 100 + 0.5));`

29. What is printed after the following code is executed?

```
ArrayList<String> list1 = new ArrayList<String>();
list1.add("A");
list1.add("B");
list1.add("C");
list1.add("D");
list1.add("E");

for (int k = 0; k < list1.size(); k += 2)
{
     list1.remove(k);
}
for (int k = 1; k <= 3; k++)
{
     list1.add(1, "*");
}
for (String word : list1)
{
     System.out.print(word + " ");
}
```

(A) `B * * * C`
(B) `B * * * D`
(C) `B * * * E`
(D) `B * C * E`
(E) `B * * * C E`

30. What is the result of calling `mystery(5)`?

```
public static int mystery(int n)
{
```

```
        if (n == 1)
             return 1;
        else if (n == 2)
             return 2;
        else
             return n + mystery(n – 1) + mystery(n – 2);
   }
```

(A) 13
(B) 15
(C) 17
(D) 19
(E) 23

31. What is the result of calling `mystery("PLANT");`

```
   public static void mystery(String s)
   {
        int i = 1;
        if (s.length() > 1)
        {
             String temp = s.substring(s.length() – i);
             System.out.println(temp);
             i++;
             mystery(temp);
        }
   }
```

(A) T
NT
ANT
LANT
PLANT

(B) LANT
ANT
NT
T

(C) LANT
ANT
NT
(D) T
(E) PLANT

Questions 32–33 refers to the following class declarations, which are intended to represent performances of plays.

```
public class Performance
{
     private String name;
     private String season;
     private int year;

     Performance(String n, String s, int y)
     {
          name = n;
          season = s;
          year = y;
     }
     public String toString()
     {
          return name + " will be performed in " + season
          + " of " + year;
     }
}

public class Play
{
     private Performance performance;
     private String mainCharacter;
     private String starringActor;

     Play(String n, String s, int y, String m, String
     star)
     {
          Performance p = new Performance(n, s, y);
          performance = p;
```

```
        mainCharacter = m;
        starringActor = star;
    }

    public String getMainCharacter()
    {
        return mainCharacter;
    }

    public String getStarringActor()
    {
        return starringActor;
    }
    public String toString()
    {
        return performance + " with " + starringActor +
        " as " + mainCharacter;
    }
}
```

32. What is the output after the following lines of code are executed in a client program?

```
Play p1 = new Play("Beauty and the Beast", "Winter", 2023,
  "Belle", "Kira");
Play p2 = new Play("Peter Pan", "Spring", 2023, "Peter",
  "Charlie");
Play p3 = new Play("Goldilocks and the Three Bears",
  "Summer", 2023, "Goldilocks", "Sophia");
Play[] schedule = {p1, p2, p3};
System.out.println(schedule[1]);
```

(A) `Charlie as Peter`

(B) `Performance@7c53a9eb with Charlie as Peter`

(C) `Peter Pan will be performed in Spring of 2023`

(D) `Peter Pan will be performed in Spring of 2023 with Charlie as Peter`

(E) An error will be thrown.

33. Code is to be added to the same client program to build the following `ArrayList` of actors.

```
ArrayList<String> actors = new ArrayList<String>();
```

The `starringActor`(s) found in the schedule array (in the previous problem) will be stored in the `actors ArrayList`. Which of the following will properly add every the `starringActor` to the `ArrayList`?

(A)
```
for (Play p : schedule)
     actors.add(p.starringActor);
```
(B)
```
for (Play p : schedule)
     actors.add(p.getStarringActor());
```
(C)
```
for (Performance p : schedule)
     actors.add(p.getStarringActor());
```
(D)
```
for (int j = 0; j < schedule.length; j++)
     actors.add(schedule[j].starringActor);
```
(E)
```
for (int j = 0; j < schedule.length; j++)
     actors.add(schedule.get[j].getStarringActor());
```

34. Consider the following classes:

```
public class Flower
{
     private int height;

     public Flower()      /* constructor without parameters */
     {
          height = 0;
     }
     public String toString()
     {
          return “height = ” + height;
     }
}
```

```
public class Daffodil extends Flower
{

     /* accessor and mutator methods not shown */
}
```

Which of the following declarations is valid?

I. `Flower lily = new Flower();`
II. `Flower daffy = new Daffodil();`
III. `Daffodil daffo = new Daffodil();`

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) I, II and III

35. A binary search is used to find a target value in an array of 4000 elements, sorted in ascending order. Assuming a target value is in the array, what is the maximum number of searches that will occur to locate the target value?

(A) 11
(B) 12
(C) 15
(D) 40
(E) 2000

Questions 36–37 refer to the following sort method.

```
public static void sort(int arr[])
{
     int i;
     int num;
```

```
        int j;
        for (i = 1; i < arr.length; i++)          /* outer loop
        */
        {
             num = arr[i];
             j = i - 1;
             while (j >= 0 && arr[j] > num)      /* inner loop */
             {
                  arr[j + 1] = arr[j];
                  j = j - 1;
             }
             arr[j + 1] = num;
        }
   }
```

36. If `sort` is called with an array of n elements, what is the maximum number of times the loop indicated by /* outer loop */ will be executed?

(A) n
(B) n / 2
(C) n – 1
(D) n + 1
(E) 2^n

37. If `sort` is called with an array of n elements, on any given pass through /* outer loop */, what is the least number of times the loop indicated by /* inner loop */ will be executed?

(A) 0
(B) 1
(C) n / 2
(D) n – 2
(E) n – 1

38. What changes to `mat` are implemented by the following code excerpt?

```
int for (int a = 0; a < mat.length; a++)
{
     temp = mat[a][0];
     for (int b = 1; b < mat[0].length; b++)
     {
          mat[a][b – 1] = mat[a][b];
     }
     mat[a][mat[0].length – 1] = temp;
}
```

(A) Every two columns are flipped. If there are an odd number of columns, there is no change to the last column.
(B) Every two rows are flipped. If there are an odd number of rows, there is no change to the last row.
(C) All columns are shifted left, and elements from the first column are moved to the last column.
(D) All rows are shifted upward, and elements from the first row are moved to the bottom row.
(E) An `outOfBounds` error is thrown.

Questions 39–40 refer to the following class declarations.

```
public class Sport
{
     private String season;
     private int year;

     Sport(String s, int y)
     {
          season = s;
          year = y;
     }

     public String toString()
     {
```

```
            return season + " " + year;
        }
}
public class Baseball extends Sport
{
        private int numPositions;

        Baseball(String s, int y)
        {
            super(s, y);
            numPositions = 10;
        }
}
```

39. The programmer wishes to store the following objects into an `ArrayList`.

```
Baseball b1 = new Baseball("Spring", 2022);
Baseball b2 = new Baseball("Summer", 2022);
Baseball b3 = new Baseball("Summer", 2023);
Baseball b4 = new Baseball("Summer", 2024);
Sport s1 = new Sport("Winter", 100);
```

Which of the following is a valid declaration for the `ArrayList`?

(A) `ArrayList<Sport> sports = new ArrayList<Sport>();`
(B) `ArrayList<Sport> sports = new ArrayList<Baseball>();`
(C) `ArrayList<Baseball> sports = new ArrayList<Sport>();`
(D) `ArrayList<Baseball> sports = new ArrayList<Baseball>();`
(E) These objects cannot be combined into a single `ArrayList`.

40. Which of the code excerpts will remove all objects from the `ArrayList` with a `season` designated as "`Summer`"?

I.
```
for (int i = 0; i < sports.size(); i++)
{
    if (sports.get(i).getSeason().equals("Summer"))
        sports.remove(i);
```

```
  }
```

II.
```
int i = 0;
for (Sport a: sports)
{
if (a.getSeason().equals("Summer"))
    a.remove(i);
i++;
}
```

III.
```
for (int i = sports.size() - 1; i >= 0; i--)
{
    if (sports.get(i).getSeason().equals("Summer"))
    sports.remove(i);
}
```

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) I and III only

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.

Section II

COMPUTER SCIENCE A

SECTION II

Time—1 hour and 30 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 Java Quick Reference 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 will not receive full credit.

FREE-RESPONSE QUESTIONS

1. This question involves writing methods for a `PigLatin` class. The `PigLatin` class stores a phrase in an instance variable and has three methods. The first will detect whether the parameter sent is a vowel, the second returns a word converted into pig latin, and the third will convert a phrase into pig latin.

```
public class PigLatin
{
     private String phrase;

     /** Constructs a new PigLatin object */
     public PigLatin(String phrase)
     {  this.phrase = phrase;  }

     /** Returns true if the letter sent is a vowel: a, e, i, o, or u
     * If the letter is not a vowel returns false: "y" is not
     considered a vowel
     * Preconditions: letter will not be null and will be exactly
     one character long,
     *                letter will not be a space; letter will be
     lowercase
     * Postconditions: letter is not modified,
     *                only true or false, as explained above, is
     returned
     */
     public boolean isLetterAVowel(String letter)
     {  /* to be implemented in part (a) */ }

     /** Returns a word converted to pig latin
     * Precondition: word.length > 0; all letters will be
     lowercase
     *             all words with word.length == 1 will be a
     single vowel
     * Postcondition: the parameter word is not modified
     */
     public String convertWord(String word)
     { /* to be implemented in part (b) */ }

     /** Returns the instance field phrase converted to pig latin
     * Returns the empty string if the parameter word is empty
     or null
```

```
     * Precondition: word may be null or any length
     *               the last character of the phrase will be a letter, not a space
     *               all characters will be lowercase
     * Postcondition: phrase is not modified, returns phrase converted to pig latin
     */

public String convertPhrase()
     { /* to be implemented in part (c) */ }
}
```

(a) Write the `PigLatin` method `isLetterAVowel`, which will return `true` if the parameter letter is a vowel (a, e, i, o, or u), and otherwise return `false`. The parameter letter will not be `null` or the empty string and will be exactly one character long. The value of the parameter will be a letter in lowercase.

Code segments	Value returned
`isLetterAVowel("a");`	`true`
`isLetterAVowel("e");`	`true`
`isLetterAVowel("u");`	`true`
`isLetterAVowel("b");`	`false`
`isLetterAVowel("y");`	`false`

Class information for this question

```
public class PigLatin

private String phrase;
public PigLatin(String phrase)
```

```
public boolean isLetterAVowel(String letter)
public String convertWord(String word)
public String convertPhrase()
```

The `PigLatin` class includes the method `isLetterAVowel`.

Complete method `isLetterAVowel` below.

```
/** Returns true if the letter sent is a vowel: a, e, i, o, or u
* If the letter is not a vowel returns false: "y" is not considered a vowel
* Preconditions: letter will not be null and will be exactly one character long,
*                letter will not be a space; letter will be lowercase
* Postconditions: letter is not modified
*                only true or false, as explained above, is returned
*/
public boolean isLetterAVowel(String letter)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(b) Write the `PigLatin` method `convertWord`, which will return a word converted to pig latin.

There will be three rules used to convert each word to pig latin:

- If `word` starts with a vowel, add the word “way” at the end of the word. “apple” would become “appleway”, “a” would become “away”
- If `word` starts with a consonant and a vowel, move the consonant to the end of the word and add “ay”. “dog” would become “ogday”, “house” would become “ousehay”
- If `word` starts with two consonants, move both consonants to the end of the word and add “ay”. “charge” would become “argechay”, “sled” would become “edslay”

You must use `isLetterAVowel` appropriately to receive full credit.

Code segments	Value returned
`convertWord(“ant”);`	`“antway”`
`convertWord(“cat”);`	`“atcay”`
`convertWord(“clone”);`	`“oneclay”`

Class information for this question

```
public class PigLatin

private String phrase;
public PigLatin(String phrase)
public boolean isLetterAVowel(String letter)
public String convertWord(String word)
public String convertPhrase()
```

The `PigLatin` class includes the method `convertWord`.

Complete method `convertWord` below.

```
/** Returns a word converted to pig latin
* Precondition: word.length > 0; all letters will be lowercase
*               all words with word.length == 1 will be a single vowel
* Postcondition: the parameter word is not modified
*                returns word converted to
*                pig latin
*/
public String convertWord(String word)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(c) Write the `PigLatin` method `convertPhrase`, which will return the instance field `phrase` converted to pig latin. If `phrase` is empty or `null`, it will return the empty string. All characters in `phrase` will be lowercase letters.

You must use `convertWord` appropriately to receive full credit.

Code segments	Value returned
`convertPhrase("the cat is sleepy");`	`"ethay atcay isway eepyslay"`
`convertPhrase("where are you");`	`"erewhay areway ouyay"`
`convertPhrase(null);`	`" "`
`convertPhrase("");`	`" "`

Class information for this question

```
public class PigLatin

private String phrase;
public PigLatin(String phrase)
public boolean isLetterAVowel(String letter)
public String convertWord(String word)
public String convertPhrase()
```

The `PigLatin` class includes the method `convertPhrase`.

Complete method `convertPhrase` below.

```
/** Returns the instance field phrase converted to pig latin
 * Returns the empty string if the parameter word is empty or
null
 * Precondition: word may be null or any length
 *               the last character of the phrase will be a letter, not
a space
 *               all characters will be lowercase
 * Postcondition: phrase is not modified, returns phrase
converted to pig latin
 */
public String convertPhrase()
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

This question involves writing a subclass for the `FrequentFlyerMember` class. This class holds information about frequent flyer members, including their names, account numbers, lifetime miles, and status levels.

2. The `FrequentFlyerMember` class definition is shown below.

```
public class FrequentFlyerMember
{
     private int acctNumber;
     private String flyerName;
     private int lifetimeMiles;
     private int statusLevel;

     FrequentFlyerMember(int n, String name, int miles)
     {
          acctNumber = n;
          flyerName = name;
          lifetimeMiles = miles;
          statusLevel = 1;
     }

     //* Adds miles to lifetimeMiles */
     public void addMiles(int miles)
     {
          lifetimeMiles += miles;
     }

     //* Changes the status level */
     public void setStatusLevel(int level)
     {
          statusLevel = level;
     }

     //* Prints selected status information */
     public String getStatusInfo()
     {
          return acctNumber + " " + flyerName + " level "
          + statusLevel;
```

```
    }
//* other methods may exist but are not shown */
}
```

You will write the class `PremierMember`, which is a subclass of `FrequentFlyerMember`.

A `PremierMember` has a `premierClubMembership` field represented by a boolean data type. It should be set to `true`. This flyer is also entitled to two `freeBags` which should be represented by an `int` field. The member will also have another String field named `otherFrequentFlyerMember` to store the name of any other frequent flyer memberships the flyer may be entitled to. These fields should be initialized in the constructor.

Information about the flyer's number, name, lifetime miles, and status level should be maintained and managed in the `FrequentFlyerMember` class.

Statement	Class Specifications or Print

`FrequentFlyerMember smith1 = new FrequentFlyerMember(14256, "Luke Smith", 1000);`	Class Specification:
	
	`smith1` is a `FrequentFlyerMember`
	`number: 14256`
	`name: Luke Smith`
	`miles: 1000`
	`statusLevel: 1`

`smith1.addMiles(3025);`	Class Specification change: `smith1`

	miles:3025
smith1.setStatusLevel(2);	Class Specification change: smith1 statusLevel: 2
System.out.println(smith1.getStatusInfo());	Printed: 14256 Luke Smith level 2
PremierMember jones1 = new PremierMember (97531, “Marcie Jones”, “British Airways”, 9000);	Class Specification: jones1 is a PremierMember number: 97531 name: Marcie Jones miles: 9000 statusLevel: 1
jones1.addMiles(10000);	Class Specification change: jones1 miles: 19000
jones1.setStatusLevel(5);	Class Specification change: jones1 statusLevel: 5
System.out.println(jones1.getStatusInfo());	Printed: 97531 Marcie Jones level 5 also a

	`member of British Airways`

Write the complete `PremierMember` class. Your implementation must meet all specifications and conform to the examples shown in the preceding table.

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

3. A `factorpair` is constructed from two (possibly non-distinct) numbers whose product is a given number. The `Factors` class constructs and stores all the given `FactorPair` objects of a number into an `ArrayList` with no duplicate pairs. The constructor uses a method: `buildArrayList` to create the `FactorPair` objects and create the `ArrayList`. The class utilizes a method `findMostPairs` to compare the number of `FactorPair` objects between two different numbers and return the number with the most `FactorPair` objects. If both numbers have the same number of `FactorPair` objects, the method returns `–1`. The `toString` method will print all the `FactorPair` objects within an `ArrayList`. You will write three methods of the `Factors` class: `buildArrayList`, `findMostPairs`, and `toString`.

```
public class FactorPair
{

    /** factor1 and factor2 represent two factors of a number
    */
```

```
    private int factor1;
    private int factor2;

    FactorPair(int f1, int f2)
    {
        factor1 = f1;
        factor2 = f2;
    }

    /* returns the first factor of a pair */
    public int getFactor1()
    {  return factor1;  }

    /* returns the second factor of a pair */
    public int getFactor2()
    {  return factor2;  }
}

public class Factors
{
    private int number;
    private ArrayList<FactorPair> pairs = new
    ArrayList<FactorPair>();

    Factors(int n)
    {
        number = n;
        pairs = buildArrayList(n);
    }

    /** Builds an ArrayList of all FactorPair objects of number
    /* Precondition: n > 0
    * Postcondition: the ArrayList will contain all FactorPair
    objects for number
    *             the ArrayList will not contain duplicate
    FactorPair objects
    *             return the ArrayList of FactorPair objects
    */
    public ArrayList<FactorPair> buildArrayList(int n)
```

```
        { /* to be implemented in part (a) }

    /* Given two numbers as parameters, the method will return the
        * number with the most FactorPair objects
        * Precondition: n1 > 0, n2 > 0
        * Postcondition: The numbers are not modified
        *   return the number with the most FactorPair objects; if
        tied, –1 will be returned
        */
        public int findMostPairs(Factors f)
            { /*  to be implemented in part (b) }

            /** Returns a string containing all the FactorPair
            objects in the ArrayList */

        public String toString()
            { /* to be implemented in part (c) */ }

        /* other methods may be implemented but not shown */
    }
```

(a) Write the `Factor` method `buildArrayList`, which will construct the `ArrayList` pairs for the number provided as the parameter.

Code segments

```
buildArrayList(24);
```

`FactorPairs` generated and added to the `ArrayList pairs`

pairs:	
factor1: 1	factor2: 24
factor1: 2	factor2: 12
Factor1: 3	factor2: 8
factor1: 4	factor2: 6

`buildArrayList(45);`

pairs:	
`factor1: 1`	`factor2: 45`
`factor1: 3`	`factor2: 15`
`factor1: 5`	`factor2: 9`

`buildArrayList(17);`

pairs:	
`factor1: 1`	`factor2: 17`

`buildArrayList(20);`

pairs:	
`factor1: 1`	`factor2: 20`
`factor1: 2`	`factor2: 10`
`factor1: 4`	`factor2: 5`

Class information for this question

```
public class Factors

private int number;
private ArrayList<FactorPair> pairs = new
ArrayList<FactorPair>();
Factors(int n)
public ArrayList<FactorPair> buildArrayList(int
n)
public int findMostPairs(Factors f) {
public String toString() {

public class FactorPair

private int factor1;
private int factor2;
FactorPair(int f1, int f2)
public int getFactor1()
public int getFactor2()
```

The Factor class includes the method `buildArrayList`.

Complete method `buildArrayList` below.

```
/** Returns an ArrayList of all FactorPair objects of number
/* Precondition: n > 0
* Postcondition: the ArrayList will contain all FactorPair objects of n
*                the ArrayList will not contain duplicate FactorPair objects */
public ArrayList<FactorPair> buildArrayList(int n)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(b) The Factor class includes the method `findMostPairs`.

Given two numbers as parameters, the method will evaluate the number of `FactorPair` objects for each number and return the number that has the most `FactorPair` objects. If the number of `FactorPair` objects is the same for both numbers, `-1` will be returned.

You must use `buildArrayList` appropriately to receive full credit.

Code segments	Value returned (examples of the contents of the

	`ArrayList` are shown in part (a))

`Factors f1 = new Factors(20);`
 	
`Factors f2 = new Factors(24);`
 	
`System.out.println(f1.findMostPairs(f2));`
	24

`Factors f1 = new Factors(20);`
 	
`Factors f2 = new Factors(45);`
 	
`System.out.println(f1.findMostPairs(f2));`
	-1

`Factors f1 = new Factors(17);`
 	
`Factors f2 = new Factors(45);`
 	
`System.out.println(f1.findMostPairs(f2));`
	45

Class information for this question

```
public class Factors

private int number;
private ArrayList<FactorPair> pairs = new
ArrayList<FactorPair>();
Factors(int n)
public ArrayList<FactorPair> buildArrayList(int
n)
public int findMostPairs(Factors f) {
public String toString() {

public class FactorPair
private int factor1;
private int factor2;
FactorPair(int f1, int f2)
```

```
    public int getFactor1()
    public int getFactor2()
```

Complete method `findMostPairs` below.

```
/** Returns the number with the most factored pairs; if tied, –1
will be returned
* Precondition: n1 > 0, n2 > 0
* Postcondition: The numbers are not modified
*/
public int findMostPairs(Factors f)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(c) The `Factor` class includes the method `toString`. This method will return a string containing containing the factors of each `FactorPair` object in the `ArrayList` using the format shown below.

Code segments	Value returned

`Factors f1 = new Factors(24);`
 	
`System.out.println(f1.toString());`
	`(1 24) (2 12) (3 8) (4 6)`

`Factors f2 = new Factors(17);`
 	
`System.out.println(f2.toString());`
	`(1 17)`

`Factors f3 = new Factors(20);`
 	
`System.out.println(f3.toString());`
	`(1 20) (2 10) (4 5)`

Class information for this question

```
public class Factors

private int number;
private ArrayList<FactorPair> pairs = new
ArrayList<FactorPair>();
Factors(int n)
public ArrayList<FactorPair> buildArrayList(int
n)
public int findMostPairs(Factors f) {
public String toString() {

public class FactorPair

private int factor1;
private int factor2;
FactorPair(int f1, int f2)
public int getFactor1()
public int getFactor2()
```

Complete method `toString` below.

```
/* Returns a string containing all the FactorPair objects
in the ArrayList pairs */
public String toString()
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the

top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

4. The `Array2DMultiples` class contains two static methods.

The first method, `buildMatrix`, is to construct and return a two-dimensional array. The parameters to the method will be a one-dimensional array and an `int` field `cols`. The length of the one-dimensional array will determine the number of rows in the two-dimensional array, while the `cols` field will determine the number of columns.

Each element in the parameter array will be used to determine the multiples that will populate the two-dimensional array. The `buildMatrix` method should work for any one-dimensional array that is provided, as well as any number of columns specified. For example, given the following parameters: `int [] arr = {7, 6, 2, 6}` and `cols = 5`, the two-dimensional array generated would have 4 rows and 5 columns. The first row would have multiples of 7, the second row would have multiples of 6, the third row would have multiples of 2, and the fourth row would have multiples of 6:

Multiples of 7	7	14	21	28	35
Multiples of 6	6	12	18	24	30
Multiples of 2	2	4	6	8	10
Multiples of 6	6	12	18	24	30

The second method, `eliminateDuplicateRows` will create a new array by removing any duplicate rows from the array. The above array would be changed to look like this:

Multiples of 7	7	14	21	28	35
Multiples of 6	6	12	18	24	30
Multiples of 2	2	4	6	8	10

```
/** Builds a two-dimensional array using the length of arr as the number of rows and
* cols as the number of columns
* @Precondition: arr.length > 0, cols > 0
* @Postcondition: return the two-dimensional array
*/
public static int[][] buildMatrix(int [] arr, int cols)
{ /* to be implemented in part (a) }

/** Create a new array by removing all duplicate rows from arrWithDups
* @Postcondition: arrWithDups is not modified
*                 returns the new two-dimensional array with no duplicate rows
*/
public static int [][] eliminateDuplicateRows(int [][] arrWithDups)
{ /* to be implemented in part (b) }
```

(a) The `Array2DMultiples` class includes the method buildMatrix.

Code segments

```
int [] arr = {5, 2, 3, 5};
int[][] arr2d = buildMatrix(arr, 6);
```

Two-dimensional Matrix generated and returned

5	10	15	20	25	30
2	4	6	8	10	12
3	6	9	12	15	18
5	10	15	20	25	30

Complete method `buildMatrix` below.

```
/** Builds a two-dimensional array using the length of arr as the number of rows and
 * cols as the number of columns
 * @Precondition: arr.length > 0, cols > 0
 * @Postcondition: return the two-dimensional array
 */
public static int[][] buildMatrix(int [] arr, int cols)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

(b) The `Array2DMultiples` method includes the method `eliminateDuplicateRows`.

Examine the two-dimensional array parameter. Eliminate any duplicate rows and return a smaller two-dimensional array.
arrWithDups:

5	10	15	20	25	30
2	4	6	8	10	12
3	6	9	12	15	18
5	10	15	20	25	30

After the call: `eliminateDuplicateRows(arrWithDups)`, the array that is returned, will not have any duplicate rows.

5	10	15	20	25	30
2	4	6	8	10	12
3	6	9	12	15	18

Complete method `eliminateDuplicateRows` below.

```
/** Create a new array by removing all duplicate rows from
arrWithDups
* @Postcondition: arrWithDups is not modified
*                 returns the new two-dimensional array with no
duplicate rows
*/
     public static int [][] eliminateDuplicateRows(int []
     [] arrWithDups)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

STOP

END OF EXAM

Practice Test 2: Answers and Explanations

PRACTICE TEST 2 ANSWER KEY

1. D
2. E
3. D
4. D
5. C
6. D
7. C
8. A
9. E
10. A
11. E
12. B
13. B
14. C
15. B
16. C
17. D
18. A
19. B
20. D
21. E
22. E
23. A
24. A
25. D
26. A
27. A
28. D
29. E
30. E
31. D
32. D
33. B
34. E
35. B
36. C
37. A
38. C
39. A
40. C

PRACTICE TEST 2 EXPLANATIONS

Section I: Multiple-Choice Questions

1. **D**

   The question involves integer division, modulus division, and order of operations. One important thing to remember is integer division is performed if both sides of the division operator (/) are integers, i.e., no rounding will occur. Another is that division, multiplication, and modulus division (%) are all performed left to right.

   Statement (I) will perform integer division first: 24 / 5 = 4. Next, 4 % 3 = 1. Finally, 1 * 2 = 2. This solution is less than 5 so (I) must be part of the answer. Eliminate (B).

   Statement (II) will also perform integer division first: 12 / 3 = 4. Next, 4 * 2 = 8. Finally, 8 + 1 = 9. This solution is not less than 5, so (II) must not be part of the answer. Eliminate (C) and (E).

   Statement (III) will perform mod first: 4 % 3 = 1, Next, 1 * 2 = 2. Finally, 1 + 2 = 3. This solution is also less than 5, so (III) must be part of the solution. Eliminate (A).

   Since (I) and (III) yield solutions less than 5, the correct choice is (D).

2. **E**

   The code "`\n`" is an escape character or line break in Java, causing the printing to begin a new line. This eliminates (A), (B), and (C) since those choices print `\n` instead of skipping to a new line. Additionally, the backslash character (`\`) is used to indicate when a special character such as a " or `\` is to be printed: `\`" will

print a double-quote (`"`) and `\\` will print the backslash. The forward slash (`/`) does not need any special character to print, thus the `//` will print two forward slashes as seen in (E). Choice (D) is not correct as it only has one forward slash. Choice (E) is correct.

3. **D**

This question involves matching data types of the arguments with parameters expected by the constructors. Choice (A) uses the first constructor on lines 7–12. An integer such as 60 may be used in a field defined as double. Choice (B) again uses the same constructor, this time by casting 13.0 to an int field. Choice (C) uses the constructor on lines 13–17. One might think the `toString()` method may cause a `nullExceptionPointer` because the name was not initialized. That exception would not occur when printing the field, only if trying to use the field, such as in a decision statement looking for a value other than `null`. Choice (E) uses the constructor on lines 18–21. The `height` and `rateOfGrowth` fields will contain 0.0 and 0, respectively. Choice (D) will cause an error because there is no constructor expecting a double datatype as the third parameter. The answer is (D).

4. **D**

The client method header for `computeBonus` has two parameters:
`computeBonus(SalesRep s, double percentage)`
The instance field `ytdSales` within the `SalesRep` object is private, so the data must be referenced using an accessor: `objectName.methodname()`. Thus, to obtain `ytdSales` for the object named `s`, we must use `s.getYtdSales()`, eliminating choices (A), (B), and (C). The field `percentage` is a parameter provided to the method, not a public instance field of the object, so there is no

need to use `s.percentage` as represented in choice (E). Thus, (D) is the correct answer.

5. **C**

Choice (A) is incorrect because it uses `ArrayList` as the data type in the enhanced-`for` loop, rather than `SalesRep` (the type of objects in the `ArrayList`). Choice (B) is incorrect because `r.get()` would not be a valid identifier for each entry of the `ArrayList`. Choice (D) is incorrect because it uses the data type with `.get(index)` (it should use the name of the `ArrayList` with `.get (index)`). Choice (E) accesses the `ArrayList` and method correctly; however, the loop goes out of bounds because the last index should be one less than the size of the `ArrayList`. Choice (C) uses an enhanced-`for` loop (which will not go out of bounds) and accesses the `ArrayList` and method correctly. The answer is (C).

6. **D**

When using the substring method, the first character of a string is at index 0. The parameters of the substring command are the first index and ending index, but the command stops just before the ending index is reached, so `substring(1, 3)` will print characters at indexes 1 and 2 (`"up"`). This eliminates Choices (A) and (C). The command `str.substring(1)` accesses the first character to the end of the string (`"uperstar"`), but it is NOT assigned back into `str`. Since strings are immutable, `str` is not modified (on the third or fifth lines)—"up" will be printed three times. Choice (E) is incorrect, since the code never reaches an out of bounds condition by using indexes 1 and 3. The answer is (D).

7. **C**

Statement (I) will successfully alternate the value of `isBlack`. The right side of the assignment statement will change the value to the opposite value every time the code is executed. Eliminate (B). Statement (II) specifically checks whether the value of `isBlack` is false. If so, it will change the value to `true`. The `else` will only be executed if `isBlack` is `true`, in which case it will change the value to `false`, so (II) is successful at alternating values, as well. Eliminate (A) and (D). Statement (III) only changes the value of `isBlack` if it is `true`: there is no provision to change the value from `false` to `true`, so Statement (III) is incorrect. Eliminate (E). Since only (I) and (II) would successfully alternate the values, the answer is (C).

8. **A**

Choice (A) would simply compare the object references of the two String variables, not the actual contents of the fields. Both choices (B) and (C) are perfectly straightforward methods to compare the contents of String variables. Choice (D) utilizes the substring method two different ways to examine the entire string for both `str1` and `str2` with the `.equals()` method being used to compare the contents of the strings. Choice (E) handles the comparison by making sure the strings are the exact same length and that `str2` is located in `str1` at index `0` (the very beginning of the string). Thus, (A) is the correct answer.

9. **E**

DeMorgan's Law can simplify this problem.
The original statement is
`!(p || (q || !r))`
After DeMorgan's law
`!p && !(q || !r)`
After DeMorgan's a second time

`!p && !q && r`
Since all the operators are &&, every part of the compound statement must evaluate to true.
The values must be: `p = false`, `q = false`, and `r = true`. The correct answer is (E).

10. **A**

Short circuit evaluation is used when the first part of an `&&` condition is `false` (since both sides of an `&&` condition must be `true`, a `false` in the first part causes the whole expression to be `false`). Both (D) and (E) start with an `&&` operator. Since neither has a `false` value in p, short circuit cannot be used on these statements.

Alternatively, short circuit can be used if the first part of an `||` condition is `true` (since only one side of an `||` condition needs to be `true`, a `true` in the first part causes the whole expression to be `true`). Since `p` is has a `false` value in (B) and (C), a short circuit cannot be used on these statements.

Choice (A) has a `true` value as the first part of an expression with the `||` operator, making the whole statement evaluate to `true`. The answer is (A).

11. **E**

Start with DeMorgan's Law.
The original statement is
`!(!p || q) || !(p || !q)`
After DeMorgan's Law
`(p && !q) || (!p && q)`
Examining the conditions on either side of the `||` operator, the truth values of `p` and `q` are combined with an && operator and

are direct opposites. Thus, if the two values are direct opposites of each other on either side of the || operator, the entire expression would be true. The correct choice is (E).

12. **B**

The outer loop will execute 10 times, where `i` will range from 1 to 10 (< 11) incrementing by one each time. The inner loop will execute 5 times, where `j` will range from 10 to 2, decreasing by 2 each time and stopping after `j` is no longer greater than 1. The variable count will be incremented by 1 each time the inner loop executes, so (10 * 5) = 50 times. The variable star starts with one "*" but adds two more ("**") each time the inner loop executes. (1 + 50 * 2) = 101. The final length of star will be 101. Thus, (B) is the correct answer.

13. **B**

The outer loop will execute 5 times (i is initialized to 1, the loop will continue until `i` is no longer <= 5). Within the outer loop are two loops, each using `print` statements so the output will continue across a single line. After the two loops execute, a `println` statement will advance the print to the next line.

The first inner loop initializes `j` to 1 and continues while it is less than `i`. So, on the first row, this loop doesn't execute at all because `j` and `i` are both 1 and `j` is not less than `i`. The "- " is never written, thus eliminating choices (A) and (E).

The second loop initializes `j` as the same value as `i`, continuing, up to 5, thus generating 5 "* ", eliminating choice (C).

Each time the outer loop executes, one additional "- " is written in the first loop, and one fewer "* " is written in the second loop.

Choice (D) writes the correct number of each symbol, but in reverse order. The "- " is written in the first inner loop. Thus, choice (B) is correct.

14. **C**

Three trace tables will make short work of this problem.
In (I), the variable `i` will have the values 1 and 2; the loop will stop when `i` increments to 3.

`i`	`2 * i + 1;`	`sum`
1	2 * 1 + 1 = 3	3
2	2 * 2 + 1 = 5	8

In (II), the variable `i` will have the values 1–5, the loop will stop when `i` increments to 6.

`i`	`if (i % 2 == 1)`	`sum`
1	`1 % 2 == 1`, so add `i` to `sum`	1
2	`2 % 2 == 0`	1
3	`3 % 2 == 1`, so add `i` to `sum`	4
4	`4 % 2 == 0`	4
5	`5 % 2 == 1`, so add `i` to `sum`	9

In (III), the variable `i` will start at 5; then the loop will change `i` to 3, then 1. The loop will not be re-entered when `i` is 1.

`i` 5	Initial values, before the loop begins	sum 5
3	`sum += i`	8
1		9

Using the trace tables above, (C) is the answer because (II) and (III) both print 9, while (I) prints 8.

15. **B**

Segment (I) starts the loop with `k` equal to 1, increments `k` by 1, then adds `k` to `sum`. The variable `k` is added whether it is even or odd, thus this option is unacceptable. Eliminate (A), (D), and (E).

Segment (II) starts the loop with `k` equal to 1, increments `k` by 2, ensuring only odd numbers will be added. The loop continues while `k` is less than or equal to `number`. If number is odd, the last value of `k` be equal to `number`, while if `k` is even, the last value of `k` to be added to the sum will be the previous odd `number`. Only odd integers up to number will be added. This segment works properly. Eliminate (C).

Four choices have been eliminated, so there is no need to go further. However, to see why (III) does not work, note that it starts the loop with `k` equal to `number` and decrements `k` by 2. If `number` is odd, this option will work as `k` will always be odd. However, if `number` is even, even numbers will be added, which does not meet the criteria of the method.

Since only (II) works properly, the correct choice is (B).

16. **C**

Sample (I) initializes `loc` to `-1`. If every element in the array contained `Integer.MIN _ VALUE`, (I) would never update `loc` to a location within the array. Therefore, (I) is not correct. Eliminate (A), (D), and (E).

Sample (II) traverses the array backwards, recording the index of the last entry as `loc`. Using the example data provided and directions in the Java doc, the location of the first matching element in the array should be returned. When there is a duplicate the entry closest to the end will be returned. Since (II) is incorrect, eliminate (B).

Only one choice remains, so there is no need to continue. However, to see why (III) is correct, note that it takes the first element in the array as `max` and initializes `loc` to `0`, assuming the first element in the array is currently the largest. The array is traversed, looking for a larger element, updating `max` and `loc` if one is found. This strategy works perfectly and is the only possible solution to the problem. Thus, (C) is correct.

17. **D**

Choice (A) works, as the static fields of the `Integer` class, `Integer.MIN _ VALUE` and `Integer.MAX _ VALUE`, represent the range of numbers that can be represented in 4 bytes of data (the size of an `int` field). Since (A) does not cause an error, (E) is also incorrect. The smallest number an `int` field can hold is the value held in `Integer.MIN _ VALUE`: thus, when 1 is subtracted, an overflow error occurs, resulting in `Integer.MAX _ VALUE`. The result is unexpected, but no error is thrown by the compiler or at run time. Choice (B) can be eliminated. In (C), the fields are added, producing `-1`, not causing any error at all. Choice (D) will

cause a compiler error because the value being assigned is 1 larger than the limit of what an `int` data field can store.

18. **A**

The method `half` divides the parameter and returns the result of dividing the parameter by 2. Since neither `n` nor 2 is a double data type, only integer division occurs. Thus, when `n` is `5, n / 2` will be `2`, but `2.0` is returned because the return type was specified as double. The contents of `num` remain the same `(5)`. The `print` statement starts with the empty string (“”), which will cause the results of `half(num)` and `num` to be concatenated together as strings. Thus, the output is `2.05` (where `2.0` and `5` have been concatenated together as a string). Choice (A) is correct.

19. **B**

The `getPerimeter()` method returns a double, so (A) and (C) can be eliminated. The `MyRectangle` constructor creates a local variable `perimeter`, which has the same name as the instance variable. When this occurs and the keyword `this` has not been used, the local variable is accessed. Thus, the local variable `perimeter` is assigned the value of `10`, but the instance field only contains a default value of `0`. When the `getPerimeter()` method returns the value of `perimeter` as a double, it returns `0.0` because the instance field `perimeter` has never been assigned any other value. Choice (D) is eliminated. Also of note: no error is thrown because an `int` can be returned when a `double` is expected because there is no loss of precision, so (E) is not correct. The answer is (B).

20. **D**

Choice (D) uses the same number and the same type of parameters in the same order as the existing constructor: `String, int, boolean, int`. This is not allowed in Java because it would be impossible to know which constructor to invoke. All the other choices would be acceptable because they have different numbers of parameters. The answer is (D).

21. **E**

The `Car()` constructor has no parameters, so all primitive fields will be initialized to `0` or `false` appropriately, but the `String` field will be initialized to `null`. The `setModel` method does not initialize the instance field model. When the parameter has the same name as the instance field, the keyword `this` must be used, otherwise the parameter is simply being assigned to itself. When the `if` statement attempts to compare the contents of model to "`Tacoma`", the model contains `null` so a `NullPointerException` will be thrown. Choice (E) is correct.

22. **E**

The field `mpg` is supposed to be rounded to the nearest integer. Choices (A), (B), and (C) produce real numbers without casting to `int` so they can be eliminated. The division in (D) produces a double but casts the double to an integer field too early. When `0.5` is added, the calculation has once again produced a double. Choice (E) performs regular division, then correctly adds `0.5` before casting to an integer. Casting to integer will truncate the decimal portion of the number. This algorithm will round the answer correctly. The answer is (E).

23. **A**

The superclass object `Bee` is referencing the subclass object `Queen`. When the object `bee1` invokes the method `toString()`, dynamic binding ensures the correct method is called. Since `bee1` is a `Queen`, the `toString()` method in the `Queen` class is called. The `toString()` method returns "`The queen lays 2000 eggs.`" Choice (A) is correct.

24. **A**

The `name` field has private access in the `Bee` class. The accessor must be in the `Bee` class because it is not visible to the `Queen` class. If access is needed within the `Queen` class, the accessor from the `Bee` class could be called with `super.getName()`. Choice (A) is correct.

25. **D**

Lifespan is part of the state of a `Bee`. There is no need for an additional class based on only that piece of information, so (A) can be eliminated, as well as (E). Choices (B) and (C) can be eliminated because the number of drones, workers, or queens may be relevant to the design, but more information would be needed to support a design decision. State and behavior are the cornerstones of class design. Detailing what is unique about workers and drones and what is like other bees will dictate the class where fields should reside and behavior is defined. Choice (D) is the correct answer.

26. **A**

An array is passed by reference to a method, so any changes that are enacted upon the array in the method are actually changing the array itself. The array is initialized with 5 elements and each element is assigned the index multiplied by 5, so (A) is

correct. Choice (B) is incorrect because it assumes the array was unchanged by the method and allows for 6 elements rather than 5. Choice (C) is incorrect because it again allows for 6 elements, although they would reflect the index multiplied by 5. Choice (D) is incorrect: there is no type mismatch because an integer can be assigned to a double data type (there is no loss of precision). Choice (E) is incorrect because the array is defined using the primitive `int`, so it was initialized with 0s until it later is assigned the multiples of 5. The answer is (A).

27. **A**

It is helpful to label the rows and columns with their indexes to assist with the math operations.

		Columns			
		0	1	2	3
ROWS	0	1	2	3	4
	1	1	3	5	7
	2	2	4	6	8
	3	4	3	2	1

A single loop controlled by `j` runs from 0 to less than the number of rows (4) The calculations for building the one-dimensional array are as follows:

```
result[0] = matrix[3][0] – matrix[0][3] 4 – 4 =
0
```

```
result[1] = matrix[3][1] – matrix[1][3] 3 – 7 =
–4
result[2] = matrix[3][2] – matrix[2][3] 2 – 8 =
–6
result[3] = matrix[3][3] – matrix[3][3] 1 – 1 =
0
```

Choice (A) is correct.

28. **D**

`Math.random() * 100` generates real numbers between 0 and 100. It includes 0, and goes up to, but does not include, `100`. `Math.random() * 100 + 1` will generate real numbers from 1, up to, but not including, 101. Choice (A) generates numbers of type `double`, which cannot be added to an Integer `ArrayList`. Choice (B) will generate numbers of type `int` ranging from 0 to 99 (`int` will truncate the decimal portion, not round it). Choice (C) will generate numbers between 0 and 100. Choice (E) will generate numbers between 1 and 100 but attempts to cast them directly to class Integer rather than primitive type `int`, which will not work. Choice (D), which is the correct answer, will generate numbers between 1 and 100. The numbers are cast to `(int)`, which can then be added to the Integer `ArrayList`. The answer is (D).

29. **E**

The `ArrayList` is initialized to `A B C D E`. The first loop removes the entry at the following indexes, modifying the `ArrayList` as shown in the table below. It's important to remember that the `ArrayList` is shrinking in size: the loop will only carry out twice.

Code	`list1`

`remove(0)`	`B C D E`
`remove(2)`	`B C E`

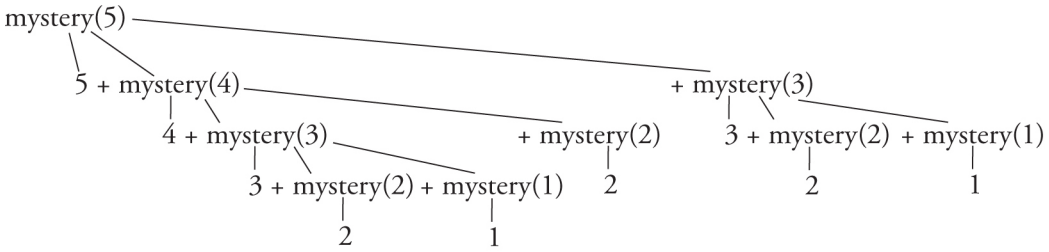
The second loop inserts an "*" three times, each time at index 1.

Code	`list1`
`add (1, "*")`	`B * C E`
`add (1, "*")`	`B * * C E`
`add (1, "*")`	`B * * * C E`

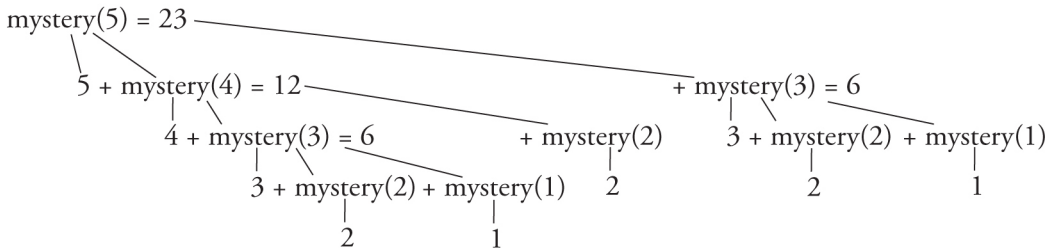
The answer is (E).

30. **E**

First, diagram the calls to the method, stopping each branch at its base case.



After diagramming the calls, add upwards.



31. **D**

After the call `mystery("PLANT"), i` is 1,

```
temp = s.substring(s.length() - i);
     s.substring(5 - 1);
```

`s.substring(4);` simply gives "T"

The length of `temp` on the next call is only 1 so nothing more is printed, and `mystery` is no longer called. Choice (D) is the correct answer.

32. **D**

The `ArrayList schedule`, contains objects that are `Play` datatypes. When `schedule[1]` is going to be printed, at run-time, Java looks to see whether the `object.toString()` method has been overridden. In other words, it looks to see whether there is a `toString()` method in the `Play` class. This method exists and does the following:

```
return performance + " with " + starringActor + " as " +
mainCharacter;
```

The word `" with "` will be printed and does not exist in (A) and (C), so these choices can be eliminated. The first part of the `return` statement, `returns performance` is an object of type `Performance`. Again, Java looks to see whether there is a `toString()` method in the `Performance` class. This method also exists and does the following:

```
return name + " will be performed in " + season + " of " +
year;
```

Since there is a `toString()` method defined, Java will use it instead of printing the memory location of the object, eliminating

(B). No errors will be thrown, eliminating (E). The following will be printed:

```
Peter Pan will be performed in Spring of 2023 with Charlie as Peter
```

Therefore, the correct answer is (D).

33. **B**

Choice (A) is incorrect because it tries to access the field `starringActor`, which is not visible because it is specified as a private variable in `Performance`. An accessor method is needed to use it. Choice (B) correctly uses an accessor method to return `starringActor` so that it may be added to the `actors ArrayList`. Choice (C) is incorrect because it uses the wrong data type in the enhanced–`for` loop: it needs to use `Play` instead of `Performance`. Choice (D) also tries to access `starringActor` directly but cannot because the field is private. Choice (E) is incorrect because it attempts to use `ArrayList` notation `(get[j])` with an array. Choice (B) is the correct answer.

34. **E**

Option (I) creates a `Flower` object using the `Flower` constructor without parameters. Since `Daffodil` is a subclass of `Flower`, it inherits all the attributes and methods in the `Flower` class. Both (II) and (III) use the `Flower` constructor without parameters. Options (I), (II), and (III) all work. Choice (E) is the answer.

35. **B**

The number of searches needed to find a target value within a sorted array is 2^n, where the value is just over the number of

elements in the array. In this case, $2^n = 4000$. $2^{11} = 2048$, so (A) is not large enough. Choice (B) leads us to $2^{12} = 4096$, which will provide enough searches to find a target in 4000 elements. Choices (C), (D), and (E) are too large. Choice (B) is the correct answer.

36. **C**

The index used to control the outer loop runs from 1 to just under the length of the array. If an array had four elements, the indexes would be $0, 1, 2, 3$. The loop would run from 1 to 3, which would be one less than the length of the array. The correct answer is (C), n – 1.

37. **A**

The inner loop is controlled by:

```
while (j >= 0 && arr[j] > num)
```

It is helpful to look at conditions in which the loop will stop executing or not execute at all. With the `&&` condition, both expressions have to be true, so look closer at the second part to determine when it might not be true. The logic surrounding the loop starts by assigning the second value in the array to `num`, then setting `i` to the previous location, resulting in two consecutive numbers in the array being compared. If those numbers happen to be the same OR in ascending order, the loop will not execute at all. Thus, the correct answer is (A), 0 times.

38. **C**

The expression `mat.length` returns the number of rows in a 2D array, while `mat [0].length` returns the number of columns in the first row. This code is selecting each row and moving columns within that row. There are no bounds errors with the loops, so (E) is incorrect. The value in the first column is assigned to `temp`, while the loop travels to the end of the row shifting all elements to the left:

```
mat[a][b - 1] = mat[a][b];
```

When the row is completed, the value in `temp` is assigned to the last column:

```
mat[a][mat[0].length - 1] = temp;
```

Choice (C) is the correct answer.

39. **A**

The `ArrayList` needs to store `Baseball` objects and `Sport` objects. Since `Baseball` extends `Sport`, the `ArrayList` should be declared with the datatype `Sport`. Referring to polymorphic ideas, `Baseball` IS-A `Sport`, not the other way around. This eliminates (B), (C), (D), and (E). Choice (A) is the correct answer.

40. **C**

Option (I) will remove some but not all the elements containing "`Summer`" due to the changes indexing as elements are removed within the loop.

b1: Spring 2022	index: 0

b2: Summer 2022	index: 1
b3: Summer 2022	index: 2
b4: Summer 2022	index: 3
s1: Winter 2024	index: 4

The first pass through the loop, the object at index 0 is examined. The season is not "`Summer`": the element is not removed.

The second pass through the loop, the object at index 1 is examined. The season is "`Summer`": the element is removed, and the indexes are updated.

b1: Spring 2022	index: 0
b3: Summer 2022	index: 2 1
b4: Summer 2022	index: 3 2
s1:	index: 4 3

Winter 2024	

The third pass through the loop, the object at index 2 is examined. But notice, `b4` is being examined (`b3` has been skipped because the indexes were updated after the remove). The object `b3` will not be removed even though the season is "`Summer`" because it has been skipped. Therefore, (I) will not work. Eliminate (A), (D), and (E).

An enhanced-`for` loop creates a local variable that does not reference an element in an array. It is assigned a value from the `ArrayList` and is used for read-only purposes. Thus, no elements in the `ArrayList` can be used with this method and Option (II) will not work. Eliminate (B).

Only one choice remains, so there is no need to continue. However, to see why (III) does work, note that it is similar to (I) but traverses the `ArrayList` backwards. Thus, when an element is removed from the `ArrayList`, the indexes update on elements that have already been processed. No elements containing "`Summer`" will be affected. Option (III) is the only option that will work properly, making (C) the correct choice.

Section II: Free-Response Questions

1. `PigLatin`–Canonical solution

(a)
```
public boolean isLetterAVowel(String letter)
{
     if (letter.equals("a") || letter.equals("e") ||
     letter.equals("i") || letter.equals("o") ||
     letter.equals("u"))
          return true;
     return false;
}
```

(b)
```
public String convertWord(String word)
{
     String w = word;
     String firstLet = w.substring(0,1);
     boolean firstIsVowel = isLetterAVowel(firstLet);
     if (firstIsVowel)
          return word + “way”;
     String secondLet = w.substring(1,2);
     boolean secondIsVowel = isLetterAVowel(secondLet);
     if (secondIsVowel)
          return word.substring(1) + firstLet + “ay”;
     return word.substring(2) + firstLet + secondLet +
     “ay”;
}
```

(c)
```
public String convertPhrase()
{
     if (phrase == null || phrase.length() == 0)
          return “”;
     String p = phrase;
     String newPigLatinPhrase = “”;
     String word = “”;
     int j = 0;
     boolean isLastWord = false;
     int k = p.indexOf(“ ”);
     while (k != -1)
     {
          word = convertWord(p.substring(0, k));
          newPigLatinPhrase += word + “ ”;
          p = p.substring(k + 1);
          k = p.indexOf(“ ”);
     }
     if (p.length() > 0)
     {
          word = convertWord(p);
     }
     newPigLatinPhrase += word + “ ”;
     return newPigLatinPhrase;
```

`PigLatin` Rubric

Part (**a**)

+2

+1	Must use the correct comparison for string `.equals()`
+1	Correctly returns `true` or `false`

Part (**b**)

+4

+1	Correctly finds first character and second character.
+1	Checks whether the length of the word is 1 and whether it starts with a vowel; returns `vowel + “way”`, otherwise returns `word + “ay”`
+1	Correctly converts `word` to `PigLatin` following the three rules
+1	Returns `word` converted to `pigLatin`, does not change the parameter: `word`

Part (**c**)

+3

+1	Checks for `null` or empty string. MUST check for `null` first. Returns empty string.
+1	Correctly finds a word and calls `convertWord(word)` for each word in the phrase, including the last word
+1	Concatenates all converted words that have been returned into a new pig latin phrase

2. `PremierMember`—Canonical solution

```
public class PremierMember extends FrequentFlyerMember
{
     boolean premierClubMembership;
     int freeBags;
     String otherFreqFlyerMember;

     PremierMember(int num, String name, int miles, String
     otherMember)
     {
          super(num, name, miles);
          premierClubMembership = true;
          freeBags = 2;
          otherFreqFlyerMember = otherMember;
     }

     public String getStatusInfo()
     {
          return super.getStatusInfo() + " also a member
          of " + otherFreqFlyerMember;
     }
}
```

`PremierMember` Rubric

+9

+1	Correct class declaration: must use `extends FrequentFlyerMember`
+1	Must have 3 private instance variables for `premierClubMembership, freeBags, otherFreqFlyerMember`
+1	Instance fields and assignments from `FrequentFlyerMember` are not repeated
+1	Header for constructor matches class name and uses 4 parameters as shown (parameters may be in a different order)
+1	First instruction in constructor must be `super(num, name, miles);` (matching the

	parameters in the constructor's parameters)
+1	Three other assignment statements in constructor as shown for `premierClubMembership, freeBags, otherFreqFlyerMember`
+1	Method header: `public String getStatusInfo()`
+1	Uses `super.getStatusInfo()` from the parent class
+1	Concatenates: `" also a member of " + otherFreqFlyerMember;`

3. `FactorPair`—Canonical solution

(a)
```
public ArrayList<FactorPair> buildArrayList(int n)
{
     ArrayList<FactorPair> tempPairs = new
     ArrayList<FactorPair>();
     for (int i = 1; i < (n / 2); i++)
     {
          if (n % i == 0)
          {
               if (i <= (n / i))
               {
                    FactorPair temp = new FactorPair(i, n /
                    i);
                    tempPairs.add(temp);
               }
          }
     }
     return tempPairs;
}
```

(b)
```
public int findMostPairs(Factors f)
}
     if (this.pairs.size() > f.pairs.size())
```

```
            return this.number;
        else if (f.pairs.size() > this.pairs.size())
            return f.number;
        else
            return -1;
    }
```

(c)
```
public String toString()
    }
        String s = "";
        for (FactorPair a : pairs)
        {
            s += "(" + a.getFactor1() + " " + a.getFactor2()
            + ") ";
        }
        return s;
    }
```

`FactorPair` Rubric

Part (a)

+5

+1	Correct declaration of `ArrayList` of type `FactorPair`
+1	Loop to find factors (1 through `n`, or 1 through `n / 2`) examines every number for factors without any bounds errors
+1	Finds factors correctly—looks for evenly divisible factors that are less than or equal to `n / i` (need the = to find factors of perfect square numbers, such as 5 in the number 25)
+1	Builds `FactorPair` object and adds it to the `ArrayList`
+1	No duplicate factors are added

Part (**b**)

+2

+1	Correctly evaluates the size of the two `ArrayLists` to find the object with the greater number of `FactorPair` objects
+1	Properly returns the number that had the most `FactorPair` objects or –1 if both numbers had the same number of `FactorPair` objects. Must use `objectname.number`.

Part (**c**)

+2

+1	Properly traverses the entire `ArrayList` with no bounds errors
+1	Properly accesses the `FactorPair` objects using the `getFactor1` and `getFactor2` methods

4. `Array2DMultiples`—Canonical solution

(a)
```
public static int[][] buildMatrix(int [] arr, int cols)
{
     int [][] mat = new int[arr.length][cols];
     for (int r = 0; r < arr.length; r++)
     {
          int hold = arr[r];
          for (int c = 0; c < cols; c++)
               mat[r][c] = hold * (c + 1);
     }
     return mat;
}
```

(b)
```
public static int [][] eliminateDuplicateRows(int [][]
arrWithDups)
{
     int [][] arr = arrWithDups;
     int holdRow = 0;
     int r = 1;
     while (holdRow < arr.length - 1)
     {
          while (r < arr.length)
          {
               if (arr[holdRow][0] == arr[r][0])
               {
                    int [][] smallerArr = new int
                    [arr.length - 1][arr[0].length];
                    for (int row = 0; row < r; row ++)
                         for (int col = 0; col <
                         arr[0].length; col++)
                         smallerArr[row][col] = arr[row]
                         [col];

                    for (int row = r + 1; row < arr.length;
                    row ++)
                         for (int col = 0; col <
                         arr[0].length; col++)
                         smallerArr[row - 1][col] =
                         arr[row][col];
                    arr = smallerArr;
               }
               else
                    r++;
          }
          holdRow ++;
          r = holdRow + 1;
     }
     return arr;
}
```

`Array2DMultiples` Rubric

Part (a)

+3

+1	Correct declaration of two-dimensional array using `arr.length` as the number of `rows`, `cols` as the number of columns
+1	Loops for rows and columns are correctly executed without any bounds errors
+1	Elements in the array are correctly assigned multiples of the first entry in the row

Part (**b**)

+6

+1	Parameter `arrWithDups` is not modified
+1	A row is held; a loop is used to examine each subsequent row for duplicates without any bounds errors
+1	Duplicate row is identified
+1	Smaller 2D array is declared
+1	New 2D array is built correctly without duplicates
+1	New 2D array is returned

HOW TO SCORE PRACTICE TEST 2

Section I: Multiple-Choice

_____ × 1.875 = _____

Number Correct (out of 40) → Weighted Section I Score (Do not round)

Section II: Free-Response

Question 1: _____ (out of 9) × 2.0833 = _____ (Do not round)

Question 2: _____ (out of 9) × 2.0833 = _____ (Do not round)

Question 3: _____ (out of 9) × 2.0833 = _____ (Do not round)

Question 4: _____ (out of 9) × 2.0833 = _____ (Do not round)

Sum = _____

Weighted Section II Score (Do not round)

AP Score Conversion Chart Computer Science A

Composite Score Range	AP Score
107–150	5
90–106	4
73–89	3
56–72	2
0–55	1

Composite Score

_____ + _____ = _____

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