MULTIPLE-CHOICE QUESTIONS ON STANDARD CLASSES

1. Here is a program segment to find the quantity `base`$^{\texttt{exp}}$. Both `base` and `exp` are entered at the keyboard.

```
System.out.println("Enter base and exponent: ");
double base = IO.readDouble();   //read user input
double exp = IO.readDouble();    //read user input
/* code to find power, which equals base^exp */
System.out.print(base + " raised to the power " + exp);
System.out.println(" equals " + power);
```

Which is a correct replacement for
/* *code to find* `power`, *which equals* `base`$^{\texttt{exp}}$ */?

```
  I double power;
    Math m = new Math();
    power = m.pow(base, exp);

 II double power;
    power = Math.pow(base, exp);

III int power;
    power = Math.pow(base, exp);
```

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

2. Consider the `squareRoot` method defined below:

```
/** @param d a real number such that d >= 0
 *  Postcondition: Returns a Double whose value is the square
 *                 root of the value represented by d.
 */
public Double squareRoot(Double d)
{
    /* implementation code */
}
```

Which `/*` *implementation code* `*/` satisfies the postcondition?

```
  I double x = d.doubleValue();
    x = Math.sqrt(x);
    return new Double(x);

 II return new Double(Math.sqrt(d.doubleValue()));

III return (Double) Math.sqrt(d.doubleValue());
```

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

3. Here are some examples of negative numbers rounded to the nearest integer.

Negative real number	Rounded to nearest integer
−3.5	−4
−8.97	−9
−5.0	−5
−2.487	−2
−0.2	0

Refer to the declaration

```
double d = -4.67;
```

Which of the following correctly rounds `d` to the nearest integer?

(A) `int rounded = Math.abs(d);`

(B) `int rounded = (int) (Math.random() * d);`

(C) `int rounded = (int) (d - 0.5);`

(D) `int rounded = (int) (d + 0.5);`

(E) `int rounded = Math.abs((int) (d - 0.5));`

4. A program is to simulate plant life under harsh conditions. In the program, plants die randomly according to some probability. Here is part of a `Plant` class defined in the program.

```
public class Plant
{
    /** probability that plant dies, a real number between 0 and 1 */
    private double probDeath;

    public Plant(double plantProbDeath, < other parameters >)
    {
        probDeath = plantProbDeath;
        < initialization of other instance variables >
    }

    /** Plant lives or dies. */
    public void liveOrDie()
    {
        /* statement to generate random number */
        if (/* test to determine if plant dies */)
            < code to implement plant's death >
        else
            < code to make plant continue living >
    }

    //Other variables and methods are not shown.
}
```

Which of the following are correct replacements for
(1) `/*` *statement to generate random number* `*/` and
(2) `/*` *test to determine if plant dies* `*/`?

(A) (1) `double x = Math.random();`
    (2) `x == probDeath`

(B) (1) `double x = (int) (Math.random());`
    (2) `x > probDeath`

(C) (1) `double x = Math.random();`
    (2) `x < probDeath`

(D) (1) `int x = (int) (Math.random() * 100);`
    (2) `x < (int) probDeath`

(E) (1) `int x = (int) (Math.random() * 100) + 1;`
    (2) `x == (int) probDeath`

5. A program simulates fifty slips of paper, numbered 1 through 50, placed in a bowl for a raffle drawing. Which of the following statements stores in `winner` a random integer from 1 to 50?

(A) `int winner = (int) (Math.random() * 50) + 1;`
(B) `int winner = (int) (Math.random() * 50);`
(C) `int winner = (int) (Math.random() * 51);`
(D) `int winner = (int) (Math.random() * 51) + 1;`
(E) `int winner = (int) (1 + Math.random() * 49);`

6. Consider the code segment

```
Integer i = new Integer(20);
/* more code */
```

Which of the following replacements for `/* more code */` correctly sets `i` to have an integer value of `25`?

```
I   i = new Integer(25);

II  i.intValue() = 25;

III Integer j = new Integer(25);
    i = j;
```

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

7. Consider these declarations:

```
Integer intOb = new Integer(3);
Object ob = new Integer(4);
Double doubOb = new Double(3.0);
```

Which of the following will *not* cause an error?

(A) `if ((Integer) ob.compareTo(intOb) < 0) ...`
(B) `if (ob.compareTo(intOb) < 0) ...`
(C) `if (intOb.compareTo(doubOb) < 0) ...`
(D) `if (intOb.compareTo(ob) < 0) ...`
(E) `if (intOb.compareTo((Integer) ob) < 0) ...`

8. Refer to these declarations:

```
Integer k = new Integer(8);
Integer m = new Integer(4);
```

Which test will *not* generate an error?

```
I   if (k.intValue() == m.intValue())...

II  if ((k.intValue()).equals(m.intValue()))...

III if ((k.toString()).equals(m.toString()))...
```

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

9. Consider the code fragment

```
Object intObj = new Integer(9);
System.out.println((String) intObj);
```

What will be output as a result of running the fragment?
(A) No output. A `ClassCastException` will be thrown.
(B) No output. An `ArithmeticException` will be thrown.
(C) `9`
(D) `"9"`
(E) `nine`

10. Consider these declarations:

```
String s1 = "crab";
String s2 = new String("crab");
String s3 = s1;
```

Which expression involving these strings evaluates to `true`?

I `s1 == s2`

II `s1.equals(s2)`

III `s3.equals(s2)`

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

11. Suppose that `strA = "TOMATO"`, `strB = "tomato"`, and `strC = "tom"`. Given that `"A"` comes before `"a"` in dictionary order, which is true?
(A) `strA.compareTo(strB) < 0 && strB.compareTo(strC) < 0`
(B) `strB.compareTo(strA) < 0 || strC.compareTo(strA) < 0`
(C) `strC.compareTo(strA) < 0 && strA.compareTo(strB) < 0`
(D) `!(strA.equals(strB)) && strC.compareTo(strB) < 0`
(E) `!(strA.equals(strB)) && strC.compareTo(strA) < 0`

12. This question refers to the following declaration:

```
String line = "Some more silly stuff on strings!";
//the words are separated by a single space
```

What string will `str` refer to after execution of the following?

```
int x = line.indexOf("m");
String str = line.substring(10, 15) + line.substring(25, 25 + x);
```

(A) `"sillyst"`
(B) `"sillystr"`
(C) `"silly st"`
(D) `"silly str"`
(E) `"sillystrin"`

13. A program has a `String` variable `fullName` that stores a first name, followed by a space, followed by a last name. There are no spaces in either the first or last names. Here are some examples of `fullName` values: `"Anthony Coppola"`, `"Jimmy Carroll"`, and `"Tom DeWire"`. Consider this code segment that extracts the last name from a `fullName` variable, and stores it in `lastName` with no surrounding blanks:

```
int k = fullName.indexOf(" ");    //find index of blank
String lastName = /* expression */
```

Which is a correct replacement for `/*` *expression* `*/`?

```
  I fullName.substring(k);
 II fullName.substring(k + 1);
III fullName.substring(k + 1, fullName.length());
```

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

14. One of the rules for converting English to Pig Latin states: If a word begins with a consonant, move the consonant to the end of the word and add “ay”. Thus “dog” becomes “ogday,” and “crisp” becomes “rispcay”. Suppose `s` is a `String` containing an English word that begins with a consonant. Which of the following creates the correct corresponding word in Pig Latin? Assume the declarations

```
String ayString = "ay";
String pigString;
```

(A)
```
pigString = s.substring(0, s.length()) + s.substring(0,1)
          + ayString;
```
(B)
```
pigString = s.substring(1, s.length()) + s.substring(0,0)
          + ayString;
```
(C)
```
pigString = s.substring(0, s.length()-1) + s.substring(0,1)
          + ayString;
```
(D)
```
pigString = s.substring(1, s.length()-1) + s.substring(0,0)
          + ayString;
```
(E)
```
pigString = s.substring(1, s.length()) + s.substring(0,1)
          + ayString;
```

15. This question refers to the `getString` method shown below:

```
public static String getString(String s1, String s2)
{
    int index = s1.indexOf(s2);
    return s1.substring(index, index + s2.length());
}
```

Which is true about `getString`? It may return a string that

I Is equal to `s2`.
II Has no characters in common with `s2`.
III Is equal to `s1`.

(A) I and III only
(B) II and III only
(C) I and II only
(D) I, II, and III
(E) None is true.

16. Consider this method:

```
public static String doSomething(String s)
{
    final String BLANK = " ";   //BLANK contains a single space
    String str = "";            //empty string
    String temp;
    for (int i = 0; i < s.length(); i++)
    {
        temp = s.substring(i, i + 1);
        if (!(temp.equals(BLANK)))
            str += temp;
    }
    return str;
}
```

Which of the following is the most precise description of what `doSomething` does?
(A) It returns `s` unchanged.
(B) It returns `s` with all its blanks removed.
(C) It returns a `String` that is equivalent to `s` with all its blanks removed.
(D) It returns a `String` that is an exact copy of `s`.
(E) It returns a `String` that contains `s.length()` blanks.

Questions 17 and 18 refer to the classes `Position` and `PositionTest` below.

```
public class Position
{
    /** row and col are both >= 0 except in the default
     *  constructor where they are initialized to -1.
     */
    private int row, col;

    public Position()         //constructor
    {
       row = -1;
       col = -1;
    }

    public Position(int r, int c)       //constructor
    {
        row = r;
        col = c;
    }

    /** @return row of Position */
    public int getRow()
    { return row; }

    /** @return column of Position */
    public int getCol()
    { return col; }

    /** @return Position north of (up from) this position */
    public Position north()
    { return new Position(row - 1, col); }

    //Similar methods south, east, and west
            ...

    /** Compares this Position to another Position object.
     *  @param p a Position object
     *  @return -1 (less than), 0 (equals), or 1 (greater than)
     */
    public int compareTo(Position p)
    {
        if (this.getRow() < p.getRow() || this.getRow() == p.getRow()
             && this.getCol() < p.getCol())
                return -1;
        if (this.getRow() > p.getRow() || this.getRow() == p.getRow()
             && this.getCol() > p.getCol())
                return 1;
        return 0;           //row and col both equal
    }

    /** @return string form of Position */
    public String toString()
    { return "(" + row + "," + col + ")"; }
}
```

```
public class PositionTest
{
    public static void main(String[] args)
    {
        Position p1 = new Position(2, 3);
        Position p2 = new Position(4, 1);
        Position p3 = new Position(2, 3);

        //tests to compare positions
            ...
    }
}
```

17. Which is true about the value of `p1.compareTo(p2)`?
    (A) It equals `true`.
    (B) It equals `false`.
    (C) It equals `0`.
    (D) It equals `1`.
    (E) It equals `-1`.

18. Which boolean expression about `p1` and `p3` is true?

    I `p1 == p3`

    II `p1.equals(p3)`

    III `p1.compareTo(p3) == 0`

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

Questions 19 and 20 deal with the problem of swapping two integer values. Three methods are proposed to solve the problem, using primitive `int` types, `Integer` objects, and `IntPair` objects, where `IntPair` is defined as follows:

```
public class IntPair
{
    private int firstValue;
    private int secondValue;

    public IntPair(int first, int second)
    {
        firstValue = first;
        secondValue = second;
    }

    public int getFirst()
    { return firstValue; }

    public int getSecond()
    { return secondValue; }

    public void setFirst(int a)
    { firstValue = a; }

    public void setSecond(int b)
    { secondValue = b;}
}
```

19. Here are three different swap methods, each intended for use in a client program.

```
  I public static void swap(int a, int b)
    {
        int temp = a;
        a = b;
        b = temp;
    }

 II public static void swap(Integer obj_a, Integer obj_b)
    {
        Integer temp = new Integer(obj_a.intValue());
        obj_a = obj_b;
        obj_b = temp;
    }

III public static void swap(IntPair pair)
    {
        int temp = pair.getFirst();
        pair.setFirst(pair.getSecond());
        pair.setSecond(temp);
    }
```

When correctly used in a client program with appropriate parameters, which method will swap two integers, as intended?

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

20. Consider the following program that uses the `IntPair` class:

```
public class TestSwap
{
    public static void swap(IntPair pair)
    {
        int temp = pair.getFirst();
        pair.setFirst(pair.getSecond());
        pair.setSecond(temp);
    }

    public static void main(String[] args)
    {
        int x = 8, y = 6;
        /* code to swap x and y */
    }
}
```

Which is a correct replacement for `/*` ***code to swap*** `x` ***and*** `y */`?

I
```
IntPair iPair = new IntPair(x, y);
swap(x, y);
x = iPair.getFirst();
y = iPair.getSecond();
```

II
```
IntPair iPair = new IntPair(x, y);
swap(iPair);
x = iPair.getFirst();
y = iPair.getSecond();
```

III
```
IntPair iPair = new IntPair(x, y);
swap(iPair);
x = iPair.setFirst();
y = iPair.setSecond();
```

(A) I only
(B) II only
(C) III only
(D) II and III only
(E) None is correct.

Refer to the `Name` class below for Questions 21 and 22.

```
public class Name
{
    private String firstName;
    private String lastName;

    public Name(String first, String last)  //constructor
    {
        firstName = first;
        lastName = last;
    }

    public String toString()
    { return firstName + " " + lastName; }

    public boolean equals(Object obj)
    {
        Name n = (Name) obj;
        return n.firstName.equals(firstName) &&
                n.lastName.equals(lastName);
    }

    public int hashCode()
    { /* implementation not shown */ }

    public int compareTo(Name n)
    {
        /* more code */
    }
}
```

21. The `compareTo` method implements the standard name-ordering algorithm where last names take precedence over first names. Lexicographic or dictionary ordering of `Strings` is used. For example, the name Scott Dentes comes before Nick Elser, and Adam Cooper comes before Sara Cooper.

    Which of the following is a correct replacement for `/*` ***more code*** `*/`?

    I
    ```
    int lastComp = lastName.compareTo(n.lastName);
    if (lastComp != 0)
        return lastComp;
    else
        return firstName.compareTo(n.firstName);
    ```

    II
    ```
    if (lastName.equals(n.lastName))
        return firstName.compareTo(n.firstName);
    else
        return 0;
    ```

    III
    ```
    if (!(lastName.equals(n.lastName)))
        return firstName.compareTo(n.firstName);
    else
        return lastName.compareTo(n.lastName);
    ```

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

22. Which statement about the `Name` class is *false*?
    (A) `Name` objects are immutable.
    (B) It is possible for the methods in `Name` to throw a `NullPointerException`.
    (C) If `n1` and `n2` are `Name` objects in a client class, then the expressions `n1.equals(n2)` and `n1.compareTo(n2) == 0` must have the same value.
    (D) The `compareTo` method throws a run-time exception if the parameter is null.
    (E) Since the `Name` class has a `compareTo` method, it *must* provide an implementation for an `equals` method.