Quiz #2 Review

Java methods can only return primitive types.

ANS: F
Java methods can also return objects such as `String`.

Formal parameters are those that appear in the method call and actual parameters are those that appear in the method header.

ANS: F

The question has the two definitions reversed. Formal parameters are those that appear in the method header, actual parameters are the parameters in the method call (those being passed to the method).

All Java classes must contain a main method, which is the first method executed when the Java class is called.

ANS: F

Only the driver program requires a main method. The driver program is the one that is first executed in any Java program, but it may call upon other classes as needed, and these other classes do not need main methods.

Java methods can return more than one item if they are modified with the reserved word `continue`, as in `public continue int foo() {...}`

ANS: F

All Java methods return a single item, whether it is a primitive data type an object, or void. The reserved word `continue` is used to exit the remainder of a loop and test the condition again.

The following method header definition will result in a syntax error:

```
public void aMethod();
```

ANS: T

The reason for the syntax error is because it ends with a ; symbol. It instead needs to be followed by {} with 0 or more instructions inside of the brackets. An abstract method will end with a ; but this header does not define an abstract method.

A method defined in a class can access the class's instance data without needing to pass them as parameters or declare them as local variables.

ANS: T

The instance data are globally available to all of the class's methods and therefore the methods do not need to receive them as parameters or declare them locally. If variables of the same name as instance data were declared locally inside a method then the instance data would be "hidden" in that method because the references would be to the local variables.

Defining formal parameters requires including each parameter's type.

ANS: T

In order for the compiler to check to see if a method call is correct, the compiler needs to know the types for the parameters being passed. Therefore, all formal parameters (those defined in the method header) must include their type. This is one element that makes Java a s*trongly typed* language.

Every class definition that is used to create objects must include a constructor.

ANS: F

Java allows classes to be defined without constructors. However, there is a default constructor that is used in such a case.

While multiple objects of the same class can exist, in a given program there can only be one version of each class.

ANS: T

A class is an abstraction; that is, it exists as a definition, but not as a physical instance. Physical instances are created when an object is instantiated using `new`. Therefore, there can be many objects of type `String`, but only one `String` class.

An object should be encapsulated in order to guard its data and methods from inappropriate access.

ANS: T
Encapsulation is the concept that objects should be protected from accidental (or purposeful) misuse.

Accessors and mutators provide mechanisms for controlled access to a well-encapsulated class.

ANS: T

Accessors provide read access to variables that otherwise would be inaccessible. Mutators provide write access to otherwise inaccessible variables.

A GUI control sets up an event, but it is the programmer who writes the code for the event handler, which executes when an event occurs.

ANS: T

Because an Image cannot directly be added to a container, it must be displayed using an ImageView object.

ANS: T

In Java, selection statements consist only of `if` and `if-else` statements.

ANS: F
This list omits `switch` statements.

In Java, the symbol "=" and the symbol "==" are used synonymously (interchangeably).

ANS: F

"=" is used for assignment statements while "==" is used to test equality.

The statement `{}` is a legal block.

ANS: T

A block consists of `{`, followed by zero or more Java statements, followed by `}`. So it is acceptable to have no statements between the brackets. Situations where this is necessary occur in Java, particularly when implementing methods of abstract classes, something you will study later.

The statement:
`if (a >= b) a++; else b--;`
will do the same thing as the statement:
`if (a < b) b--; else a++;.`

ANS: T
We can reverse the `if` clause and `else clause` if we reverse the condition. The opposite condition of `(a >= b)` is `(a < b)` so this works out logically. Note that if we used the condition `(a <= b)` then the resulting statement would not do the same thing as the original if `(a >= b).`

An `if` statement may or may not have an `else` clause, but an `else` clause must be part of an `if` statement.

ANS: T

Java allows for either `if` or `if-else` statements. But `else` is only used as part of an `if` statement.

In order to compare `int, float and double` variables, you can use `<, >, ==, !=, <=, >=,` but to compare `char` and `String` variables, you must use `compareTo(), equals()` and `equalsIgnoreCase()`.

ANS: F

You can also directly compare `char` variables using `<, >, ==, !=, <=, >=,` but you must use `compareTo(), equals()` and `equalsIgnoreCase()` for any `String` comparisons.

Assume that `boolean done = false, int x = 10,` and `int y = 11.` Then the expression `(!done && x <= y)` is `true`.

ANS: T

Since `done` is `false`, `!done` is `true`. Since `10 < 11`, `x <= y` is `true`. Therefore, the entire expression is `true`.

Assume that `boolean done = false, int x = 10, int y = 11, String s = "Help"` and `String t = "Goodbye"`. Then the expression `(s.concat(t).length() < y)` is `true`.

ANS: F

Concatenating `s` and `t` gives a `String` that is 11 characters long and `11 < 11` is `false`.

Assume that `boolean done = false, int x = 10, int y = 11, String s = "Help"` and `String t = "Goodbye"`. Then the expression `(done || s.compareTo(t) < 0)` is `true`.

ANS: F

Both `done` is `false` and `s.compareTo(t) < 0` is `false` since `s` does not come before `t` alphabetically, so the entire expression is `false`.

A `switch` statement must have a default clause.

ANS: F
The default clause is optional.

Each `switch case` statement must terminate with a `break` statement.

ANS: F

They often do but if the `break` statement is not present, the flow of control continues into the next `case`.

Control in a `switch` statement jumps to the first matching `case`.

ANS: T

The `switch` expression is evaluated and control jumps to the first matching `case`, then continues from there.

The following `for` loop is an infinite loop::

```
for(int j = 0; j < 1000;) i++;
```

ANS: T

This loop initializes `j` to 0 and compares it to `1000`, but does not alter `j` after each loop iteration. In reality, the loop will terminate with a run-time error eventually once `i` becomes too large to store in memory, but logically, this is an infinite loop.

It is possible to convert any type of loop (`while`, `do`, `for`) into any of the other two Java loops.

ANS: T

All loop statements have equivalent expressive power.

The following loop is syntactically valid:

```
for(int j = 0; j < 1000; j++) j--;
```

ANS: T

The Java compiler does not care that you are incrementing `j` in the loop but decrementing `j` in the loop body. Logically, this loop makes no sense because `j` will continuously be incremented and decremented so that it never reaches `1000`, but there is nothing wrong with the loop syntactically.

In Java, it is possible to create an infinite loop out of `while` and `do` loops but not `for` loops.

ANS: F

It is true that `while` and `do` loops can be infinite loops, but it is also true that Java `for` loops can be infinite loops. This is not true in some other programming languages where `for` loops have a set starting and ending point, but Java `for` loops are far more flexible than most other language's `for` loops.

A conditional operator is virtually the same as a `switch` statement.

ANS: F

The conditional operator is more like an `if-else` statement.

A `for` statement is normally used when you do *not* know how many times the loop should be executed.

ANS: F

A `for` statement is normally used when you *do* know how many times the loop should be executed.

A loop can be used in a GUI to draw concentric circles.

ANS: T

The code below presumably used to create Box objects is syntactically correct.

```
public class Box
   {
     double length;
     double width;
     double height;

   Box(double l, double w, double h)
   {
    length = l;
    width = w;
    height = h;
   }

   double volume()
   {
    return  length * width * height;
   }
}
```

ANS: T

Write the statement to instantiate a Box object, blueBox, with a length of 6, height of 2, and width of 4.

```
public class Box
   {
     double length;
     double width;
     double height;

   Box(double l, double w, double h)
   {
    length = l;
    width = w;
    height = h;
   }

   double volume()
   {
    return  length * width * height;
   }
  }
```

ANS: `Box blueBox = new Box(6,4,2);`

The following set of statements will add 1 to x if x is positive and subtract 1 from x if x is negative but leave x alone if x is 0.

```
if (x > 0) x++;
else x--;
```

ANS: F

x-- is done if x is not positive, thus if x is 0, x becomes -1 which is the wrong answer.

Assume that `count` is `0`, `total` is `20`, and `max` is `1`.

When the following code is executed, the condition short circuits and the assignment statement is not executed.

```
if (count != 0 || total / count > max)
      max = total / count;
```

ANS: F

The above statement would be true if the `||` was replaced with &&.

A truth table shows, for the various `true` or `false` values of `boolean` variables and what the result of a `boolean` condition is. Fill in the following truth table. Assume that `a`, `b` and `c` are `boolean` variables.

a	b	c	a && (!b || c)
false	false	false	
false	false	true	
false	true	false	
false	true	true	
true	false	false	
true	false	true	
true	true	false	
true	true	true	

ANS:

a	b	c	a && (!b || c)
false	false	false	false
false	false	true	false
false	true	false	false
false	true	true	false
true	false	false	true
true	false	true	true
true	true	false	false
true	true	true	true

Find the error in the following code and fix the code so that this statement is a legal `if-else` statement.

```
if (x < 0); x++;
  else x--;
```

ANS:

```
if (x < 0) x++;
  else x--;
```

Rewrite the following `if-else` statement using a conditional operator.

```
if (x > y)
  z = x;
  else z = y;
```

ANS:

```
z = (x > y) ? x : y;
```

A `do` loop should be used in a situation where you want to do data verification by asking the user for a value, and only if the user has entered an improper value does your code repeat and try again.

ANS: T

The `do` loop is used if you want to execute the loop body at least one time. This is useful for data verification (asking the user for a value, and only if the user has entered an improper value does your code repeat and try again).

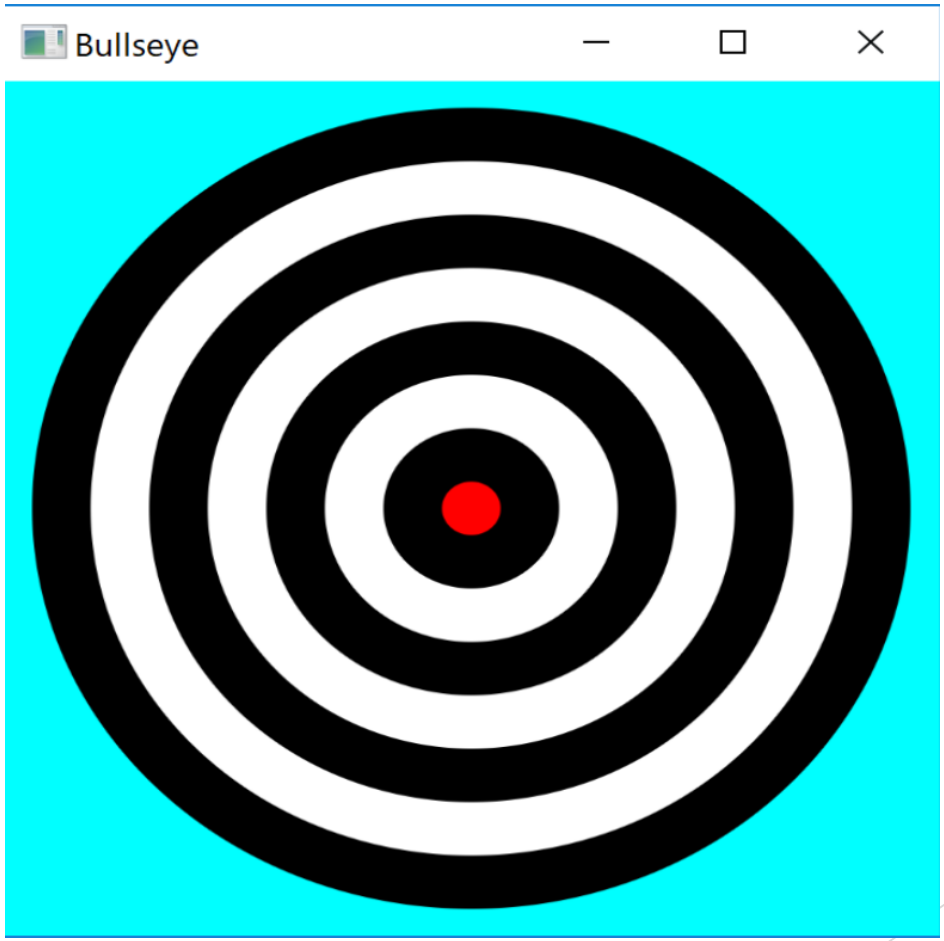
The following JavaFX statement will rotate an `Ellipse` named `lipse` 30 degrees in the counterclockwise direction.

```
lipse.setRotate(30);
```

ANS: F

This statement will rotate `lipse` 30 degrees in the clockwise direction.

Find the errors in the Bullseye program that displays a target using concentric black and white circles and a red center.
Assume the appropriate packages have been imported.



```
public class Bullseye
{
    public void start(Stage primaryStage)
    {
    Group root = new Group();
    Color ringColor = Color.BLACK;
    Circle ring = null;
    int radius = 150;
    for (int count = 1; count <= 8; count--)
        {
        ring = new Circle(160, 160, radius);
        ring.setFill(ringColor);
        root.getChildren().add(ring);
        if (ringColor.equals(Color.BLACK))
            ringColor = Color.WHITE;
        else
            ringColor = Color.BLACK;
        radius = radius + 20;
        }
    ring.setFill(Color.RED);
    Scene scene = new Scene(root, 320, 320, CYAN);
    }
    public static void main(String[] args)
    {
        launch(args);
    }
```

```
public class Bullseye extends Application
{
    public void start(Stage primaryStage)
    {
    Group root = new Group();
    Color ringColor = Color.BLACK;
    Circle ring = null;
    int radius = 150;
    for (int count = 1; count <= 8; count++)
        {
        ring = new Circle(160, 160, radius);
        ring.setFill(ringColor);
        root.getChildren().add(ring);
        if (ringColor.equals(Color.BLACK))
            ringColor = Color.WHITE;
        else
            ringColor = Color.BLACK;
        radius = radius - 20;
        }
    ring.setFill(Color.RED);
    Scene scene = new Scene(root, 320, 320, Color.CYAN);
    primaryStage.setTitle("Bullseye");
    primaryStage.setScene(scene);
    primaryStage.show();
    }
    public static void main(String[] args)
    {
        launch(args);
    }
```