

## **QUIZ 2 SOLUTIONS**

**Professor :** Dave Houtman

**Office:** T323

**Office Hrs:** Friday 10:15 – 11:30

**Email:** [houtmad@algonquincollege.com](mailto:houtmad@algonquincollege.com)

```
public class MySuperClass {  
    private int x;  
    public int setX(int x){this.x = x;}  
    public int getTwiceAsMuch{return 2*x;}  
}
```

```
public class MySubClass extends MySuperClass {  
    // ... assume subclass members and constructors are here  
}
```

2. 4. **True** (a) or **False** (b): In the example above, MySubClass inherits all of MySuperClass's members, which it may then override
3. 5. Which one of the following is used in UML diagrams to signal that a class or member is abstract?
- a) underline      b) *italics*      c) **boldface**  
d) CAPITALIZE      e) none of the above
4. 11. **True** (a) or **False** (b): you cannot inherit from a final class



5. 2. The JVM resolves constructor overloading by comparing each constructor's
- a) return type
  - b) signature**
  - c) methods
  - d) parameters
  - e) none of the above
6. 3. Failure to use the `@Override` annotation before a method header may result in
- a) method overloading**
  - b) failure to override the actual method**
  - c) compile-time errors
  - d) infinite loops
  - e) none of the above
7. 8. **True (a)** or **False (b)**: Subclasses do not inherit constructors.
8. 6. **True(a)** or **False(b)**: Using the code at the top of the page, any object instantiated from `MySubclass` will not have access to the `toString()` method, only the methods extended from `MySuperClass`, i.e. `setX()` and `getTwiceAsMuch()` – but not `toString()`.



- 9 10 Using the code at the top of the page, if you instantiate a new object based on `MySubClass`:
- a) Only the `MySubClass` constructor is called
  - b) The `MySubClass` constructor is called first, and then the JVM calls its own `MySuperClass` constructor
  - ☒ c) The JVM generates and calls its own default `MySuperClass` constructor, and then calls the `MySubClass` constructor
  - d) No constructors are called during the instantiation of an object unless `this()` is specifically used
  - e) None of the above
- 10 7. Using the code at the top of the previous page, if `setX()` was made an abstract method, which one of the following statements would *not* be true?
- a) You must override `setX()` in `MySubClass`
  - ☒ b) all of `MySuperClass`'s methods become abstract
  - c) You cannot instantiate an object from `MySuperClass`
  - d) If `MySubClass` was also made abstract, then it would not need to override `setX()`
  - e) none of the above
- 11 9. **True** (a) or **False** (b): In general, an object instantiated from a superclass will have at least as many or more members than an object instantiated from one of its subclasses



12. In Java, single inheritance ensures that each subclass has one parent.

13. abstraction is the OOP term for reducing something to its essential characteristics

14. The instanceof keyword allows you to check if one object is an instance of another

15. dynamic binding allows the JVM to load an object and its members at runtime.

16. polymorphism means that a variable of a superclass type can be used to store or pass a subclass object.



**40.** The following code is intended to read in a student's number and then asks for their first and last name. It calls on the method `makeUpperCase()` to convert the last name to upper case characters. Unfortunately, the code is flawed in a number of ways. Place a number next to each of the errors in the code below, and explain why the code is in error in the box at right next to the number you used in the code.

Note:

- (1) you may assume that the overall construction of the code shown at left is essentially correct even if its implementation is unnecessarily bloated and flawed in many critical areas. Your job is not to critique the construction of the program, but simply to find *specific* compilation and run-time/logic errors. Remember: ONLY INDICATE ERRORS THAT ACTUALLY EXIST; DO NOT FLAG AS ERRORS THINGS THAT YOU DISAGREE WITH.
- (2) If you find an error that seems to occur more than once, chances are that that code is in fact correct, and the real error lies elsewhere, at another location, and only once.
- (3) Even if an error *does* occur more than once, you only get credit for the first occurrence of that error, not for repeat errors of the same kind.
- (4) You can assume that only characters and numbers are ever used read into the program by the user; the code does not need to check for improper input
- (5) You may assume that the methods associated with the String objects used in this example are real and correct and do not generate compile-time errors as written.



```
public final class Shape {  
    private double x;  
    public void Shape(double d){x = d;}  
    public double getX(){return x;}  
    public int setX(double d){x = d;}  
    public double getMaxWidth();  
}
```

1. Cannot inherit from a `final` class. (Note: this does not mean that the class *must* be `abstract`; that's a whole other issue related to not being able to instantiate from an object. Abstraction is about code reuse. You will not flag an error with or without the word `abstract` at this location—but you will with the word `final`.)
2. Constructors cannot have a specific return type
3. Should be `void`, since there is no `return` statement. Alternately, return an `int`. Note : a setter/mutator *may* return a value to indicate if it was successful or not.
4. Missing `{ }` at the end of the line to indicate a method body. Alternately, you could use `abstract` here, but you need to explain *why* it is necessary. In other words, adding `{ }` solves the problem nicely; adding `abstract` has other issues which need to be explained. *Why* should you use `abstract`?



```

public class Triangle extends Shape {
    public double y, z;
    public Triangle(double x, double y, double z){
        this.x = getX();
        this.y = y;
        this.z = z;
    }
    public double getMaxWidth(){
        if ((y > z) && (z > x)) return y;
        else ((z > x) && (x > y)) return z;
        else return y;
    }
}

```

(Note: Using `public` here does not cause compile-time errors. If you said it should be `private`, you must explain what potential run-time errors `private` solves)

5. The purpose is to initialize `x`, which is declared in the superclass. Therefore, `super(x)` or `setX(x)` are acceptable.

6. Can't refer to `x` directly, since it is `private` in the superclass. Use `getX()`.

7. Should be `else if`

8. Since the purpose of the function is to get the longest side of a triangle, it must return `x`, `y`, or `z`. Thus this should be `return x;` or more appropriately, `return getX()`



```

public class Rectangle extend Shape {
    private double y;
    public Rectangle(double x, double y){
        super.x = x;
        this.y = y;
    }
    public double getMaxWidth(){
        // Use pythagoras to return hypothenus of triangle
        return Math.sqrt(y*y + getX() * getX());
    }
}

```

9. Again, `setX(x)` or `super(x)` are acceptable here (but not if duplicated above)



```

public class FitTester() {
    public static main(String[] args) {
        Shape shape1 = new Triangle(3.0, 4.0, 5.0);
        Shape shape2 = new Rectangle(3.0, 4.0);
        if (compareShapes(shape1, shape2))
            System.out.println("Shape1 fits inside Shape2");
        else
            System.out.println("Shape2 fits inside Shape1");
    }

    public static boolean compareShapes(Shape sh1, Shape sh2){
        return (sh1.getMaxWidth() > sh2.getMaxWidth())
    }
}

```

10. Missing `void`

11. Missing `'`

12. Logic is reversed; if Shape1 fits inside Shape2, the width of sh1 needs to be less than the width of sh2. So change `>` to `<`, or use `!compareShapes` in the `if` above

13. Missing `;` at the end



# 15F – CST8284 Quiz 2 Marks Distribution

