# ICS4U

# Assignment 4: Object-Oriented Programming Principles

**Part 1: Definitions with Examples from last week's learning**

Create your own definitions / descriptions for each of the following terms. They are chunked into groups so that a short paragraph can give a description for all the terms within a group by explaining how they are related. Also, give examples of where these ideas occurred in the Robot Programming Tasks.

- Group 1: Classes, Objects.
- Group 2: Attributes/Properties, Constructors, Methods/Behaviours.
- Group 3: Modifiers (public, private, protected, static, final, etc.)
- Group 4: Encapsulation, Accessor/Mutator Methods (get and set methods).
- Group 5: Extending classes, Inheritance, Overriding Methods, Polymorphism.

** If you are having difficulty creating a good description for Polymorphism, complete the activities in Part 3 of these instructions (listed below) and then come back to this Part to finish.

➔ In the Assignment 4 Google Doc in the Google Classroom, put all of your definitions / descriptions / examples for the above topics.

**Go to the next page for Part 2.**

**Part 2: Using Objects as parameters … do they "pass by value" or "pass by reference"**

First, read this site that explains why Java is always "pass by value":

- https://www.geeksforgeeks.org/g-fact-31-java-is-strictly-pass-by-value/

ps. The Output to the Exercise is "i = 10, j = 20"

Technically, even though Java is "Pass by value", any operations that change the properties of an object that was passed as a parameter will be preserved after the method call. You can see this in the third code sample where the output is "10".

Consider the following two classes, one of which has a main method:

```
public class Balloon {

   private String color;

   public Balloon(String c){
      this.color=c;
   }

   public String getColor() {
      return color;
   }

   public void setColor(String color) {
      this.color = color;
   }

}
```

```
public class Test {

   public static void main(String[] args) {

      Balloon red = new Balloon("Red");
      Balloon blue = new Balloon("Blue");

      swap(red, blue);
      System.out.println("red color = " + red.getColor());
      System.out.println("blue color = " + blue.getColor());

      scrambleColors(blue);
      System.out.println("blue color = " + blue.getColor());

   }

   private static void scrambleColors(Balloon balloon) {
      balloon.setColor("Yellow");
      balloon = new Balloon("Green");
      balloon.setColor("Purple");
   }

   public static void swap(Object o1, Object o2){
      Object temp = o1;
      o1=o2;
      o2=temp;
   }

}
```

What is the output of the Test Class's main method? DO NOT RUN THESE IN AN I.D.E. like Eclipse. "Trace" through them manually to determine the output of the three System.out.println commands.

➔ In the Assignment 4 Google Doc in the Google Classroom, give the output for the main method, and provide a brief explanation for each line.

**Part 3: More Practice with Polymorphism**

Polymorphism can be complex to trace, but here's the rule:

- When a method is called on an Object, we start in the Class used to create the Object to see if the method has been defined in that Class (whether it's a new method or an overridden method).
- If it exists in that Class, run that code.
- If it does not exist in that Class, but the Class is a Subclass of a Superclass, go "up" the chain of Superclasses one at a time to see where it was most recently defined, and run that code.

** The above sequence happens EVERY time a method is called using either the object name or "`this.`" UNLESS we use "`super.`" to explicitly start the process in the superclass.

Let's take a look at some code to see how it works. All four Classes are provided in the Dropbox link on the assignment. You may copy them into an Eclipse project to test for yourself.

```
public class ClassA {
   public String method1() {
      return "A1";
   }
   public String method2() {
      return "A2";
   }
}
```

```
public class ClassB extends ClassA {
   public String method1(int x) {
      return "B1(" + x + ")";
   }
   public String method2() {
      return "B2";
   }
   public String method3A() {
      return "B3A, " + this.method2() + ", " + this.method1();
   }
   public String method3B() {
      return "B3B, " + super.method2() + ", " + this.method1();
   }
}
```

```
public class ClassC extends ClassB {
   public String method1(int x) {
      return "C1";
   }
   public String method2() {
      return "C2";
   }
}
```

```
public class UnderstandingPolymorphism {
   public static void main(String[] args) {
      ClassA objA = new ClassA();
      System.out.println("objA.method1(): " + objA.method1());
      System.out.println("objA.method2(): " + objA.method2());
      ClassB objB = new ClassB();
      System.out.println("objB.method1(10): " + objB.method1(10));
      System.out.println("objB.method2(): " + objB.method2());
      System.out.println("objB.method3A(): " + objB.method3A());
      System.out.println("objB.method3B(): " + objB.method3B());
      ClassC objC = new ClassC();
      System.out.println("objC.method2(): " + objC.method2());
      System.out.println("objC.method3A(): " + objC.method3A());
      System.out.println("objC.method3B(): " + objC.method3B());
   }
}
```

The output for the UnderstandingPolymorphism main method is on the next page, along with explanations.

| Output of Program | Explanation for that line of output |
|---|---|
| objA.method1(): A1 | ClassA has defined a method1(), so it will run that method. |
| objA.method2(): A2 | ClassA has defined a method2(), so it will run that method. |
| objB.method1(10): B1(10) | ClassB has overloaded method1. So we now have two methods with the same name of "method1". What distinguishes them in the parameters. Since we are passing in an integer parameter of 10, we use the new method1(int x) in ClassB. |
| objB.method2(): B2 | ClassB has overridden method2() from ClassA, so when an object created from ClassB calls method2() it runs the new method and output "B2". |
| objB.method3A(): B3A, B2, A1 | ClassB has defined a new method3A(), so it will run that method.<br>The method also calls the overridden method2 in ClassB since this.method2() is called (see explanation on line above), and then it will try to call this.method1(). It starts in ClassB looking for a method1(), but since it does not exist in ClassB, it goes "up" into ClassA where it finds method1() and calls it. |
| objB.method3B(): B3B, A2, A1 | ClassB has defined a new method3B(), so it will run that method.<br>The method also calls the original method2 in ClassA since super.method2() is called (the word super forces it "up" into ClassA), and then it will try to call method1(). Similar to the line above, method1() in ClassA will be called since it does not exist in ClassB. |
| objC.method2(): C2 | ClassC has overridden method2() from ClassB, so when an object created from ClassC calls method2() it runs the new method and outputs "C2". |
| objC.method3A(): B3A, C2, A1 | ClassC has NOT overridden method3A() from ClassB. When it is called by a program, we start in ClassC where the method3A() is not found, and then go "up" into ClassB where it is found, so that method starts running. This is why the first thing that is output is "B3A".<br><br>Then, the method in ClassB is trying to call "this.method2()". What is **REALLY IMPORTANT** here is that "this." always starts with the Class used to create the Object, which in this case was ClassC. So, we start in ClassC where method2() exists. This is why the next thing that is output is "C2".<br><br>Finally, the method in Class B is trying to call "this.method1()". Again, we start in Class C where it is not found. So we go "up" to ClassB and it is not found. So we go "up" again to ClassA where it is finally found and run. This is why the next thing that is output is "A1". |
| objC.method3B(): B3B, A2, A1 | ClassC has NOT overridden method3B() from ClassB, so the method in ClassB will be called. This is why the first thing output is "B3B".<br><br>Then, the method in ClassB is calling "super.method2()". What is **REALLY IMPORTANT** here is that "super." forces the program to go "up" into the superclass FOR THE CLASS THAT DEFINED THE METHOD and not the superclass for the object that was created. This is why we go "up" into ClassA and call method2() to output "A2" next.<br><br>Finally, like the above line, since we are calling "this.method1()" we need to use the method in ClassA since it is the only class where the method is defined. |

The last two lines on the above output at the most crucial to understand. ClassB's method3A() and method3B() have one minor difference: the call to "this.method2()" vs. "super.method2()". This minor difference causes different and possibly unexpected results based on the object used to call the method (ie. the difference in the output for the 5th and 8th lines even though the same method is called).

To practice with Polymorphism, trace through the following code WITHOUT USING AN I.D.E. LIKE ECLIPSE.

```
public class PolymorphismChallenge {
    public static void main(String[] args) {
        new Lisa().talk("Sax :)");
        Bart simpson = new Bart("D'oh");
        simpson.talk();
        Lisa lisa = new Lisa();
        lisa.talk();
        ((Bart) simpson).prank();
    }
}
```

```
public class Simpson {
    public void talk() {
        System.out.println("Simpson!");
    }
    public void prank(String prank) {
        System.out.println(prank);
    }
}
```

```
public class Bart extends Simpson {
    String prank;
    public Bart(String prank) {
        this.prank = prank;
    }
    public void talk() {
        System.out.println("Eat my shorts!");
    }
    public void prank() {
        super.prank(prank);
        System.out.println("Knock Homer down");
    }
}
```

```
public class Lisa extends Simpson {
    public void talk(String toMe) {
        System.out.println("I love Sax!");
    }
}
```

➔ In the Assignment 4 Google Doc in the Google Classroom, give the output for the main method and explain each line of your output.