

Write a class called LargestDimension that implements the interface below to return the largest dimension of Rectangle objects. (note: this is similar to the assignment with the PerimeterMeasurer class)

```
public interface Measurer
{
    double measurer(Object obj);
}
```

```
public class LargestDimension implements Measurer
{
    double measurer(Object obj)
    {
        Rectangle r = (Rectangle) obj;
        if(r.getWidth() > r.getHeight())
        {
            return r.getWidth();
        }
        return r.getHeight();
    }
}
```

The class you implemented above is sometimes called a callback class - what are some of the advantages of a callback class?

You can use a callback class to adapt any type of class - even system class that we cannot change.

Also, callback classes allow us to implement an interface in multiple ways. If we implement the interface directly in a single class, we can only implement it once. For instance, if we implement the method **measure** in **Sphere**, it could return either surface area or volume, but not both.

Consider the Java code below & determine what the final output is

```
int x=0;
for(int a=5; a>=0; a--)
{
    for(int b=0; b<a; b++)
    {
        x++;
    }
}
System.out.println("x: " + x);
```

```
x=0
a=5, b=0 to 4, x++ 5 times, x=5
a=4, b=0 to 3, x++ 4 times, x=9
a=3, b=0 to 2, x++ 3 times, x=12
a=2, b=0 to 1, x++ 2 times, x=14
a=1, b=0 to 0, x++ 1 time, x=15
a=0, b=done, x++ 0 times, x=15
a=done
final result, x = 15
```

List and briefly define the 6 stages of the software life cycle. Where applicable, indicate the outputs (aka deliverables) of the stages.

Analysis - Deciding what the software should do. Output is a requirements document

Design - Deciding how to build the system, i.e. what classes and methods are needed. Output is a description of classes & methods and diagrams indicating their relationships to each other.

Implementation - Writing code & compiling code. Output is a working, commented program.

Testing - Verifying that the program works as intended. Output is a listing of test results.

Deployment - Getting the program to the users.

Maintenance - Correcting problems in the software that were missed, or not present earlier. Output is an improved version of the software.

Consider the following code & determine the output:

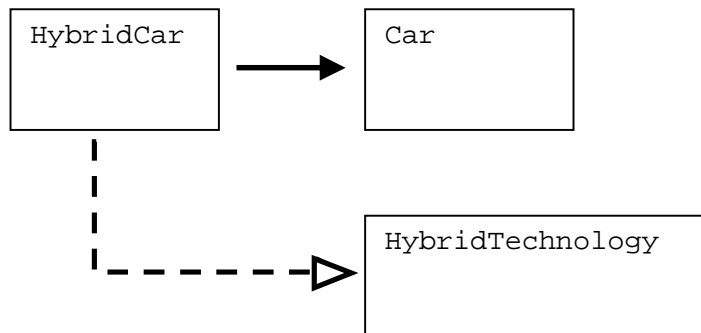
```
public class Person
{
    private String name;

    public Person(String n)
    {
        name = n;
    }
    public void changeName(String n)
    {
        name = n;
    }
    public String getName()
    {
        return name;
    }
}
. . .

Person p1 = new Person("smith");
Person p2 = p1;
p1.changeName("jones");
System.out.println( p2.getName() );
```

jones - the reason is that both p1 & p2 refer to the same object (notice that we only used **new** once, only one object of type **Person** was built)

Consider a class called HybridCar that inherits from a class called Car & also implements an interface called HybridTechnology. Draw out the UML diagram that illustrates the relationships between HybridCar, Car and HybridTechnology.



Referring to the example above, indicate which classes, if any, are super classes & which classes, if any, are subclasses.

Hybrid car is the subclass of Car  
Car is the superclass of Car

What type of programming uses classes called listeners? Given an example of software that uses this style of programming & explain your answer.

Event based programming makes strong use of listener classes. Media player software uses this style of programming - the software listens for certain events (such as clicking pause, play, exit, etc) while continuing to perform a function (in this case, playing music)

When we print out objects from the Rectangle class & BankAccount class, we get something like the following:

```
"java.awt.Rectangle[x=5,y=10,width=20,height=30]"
```

```
"BankAccount@d24606bf"
```

What is the name of the method that is responsible for printing out the object's information?

```
toString
```

Which class is overriding the given method?

```
Rectangle
```

Where does the other class inherit the method from?

```
Object
```

Given two classes, CheckingAccount & SavingsAccount which both inherit from a class BankAccount, place an --> by the statements that will print:

```
CheckingAccount ca = new CheckingAccount(100.0);
SavingsAccount sa = new SavingsAccount (200.0);
BankAccount ba = new BankAccount (300.0);
Object o1 = (Object)ca;
Object o2 = (BankAccount)o1;
```

```
--> if(ca instanceof CheckingAccount)
    System.out.println("ca instanceof CheckingAccount");

if(sa instanceof CheckingAccount)
    System.out.println("sa instanceof BankAccount");

--> if(ba instanceof BankAccount)
    System.out.println("ba instanceof BankAccount");

--> if(ba instanceof Object)
    System.out.println("ba instanceof Object");

--> if(o1 instanceof Object)
    System.out.println("o1 instanceof Object");

--> if(o2 instanceof CheckingAccount)
    System.out.println("o2 instanceof CheckingAccount");
```

Consider the following class

```
public class RegularPerson
{
    private String name;
    public RegularPerson (String n)
    {
        name = n;
    }
    public String getName()
    {
        return name;
    }
}
```

Make a new class called KnightedPerson that inherits from regular person. Assume the following bit of code will be used to test your class:

```
KnightedPerson k = new KnightedPerson("Jim", "Moorhead");
System.out.println( k.getName() );
System.out.println( "expected: Sir Jim of Moorhead" );
```

Make certain that you include documentation comments for your class & all of its methods/constructors.

```
/** models a Person who has been knighted
 *
 */
public class KnightedPerson extends Person
{
    String kingdom;

    /** Inits the KnightedPerson to have the given name & kingdom
     * @param n then name of the person
     * @param k the name of the kigndom
     */
    public KnightedPerson(String n, String k)
    {
        super(n);
        kingdom = k;
    }
    /**
     * Displays the info in the form of Sir <name> of <kingdom>
     */
    public void getName()
    {
        String result = "Sir " + super.getName() + " of " + kingdom;
        return result;
    }
}
```

Describe a similarity and a difference between the waterfall & spiral software development models:

They are similar in that they both move through the software lifecycle stages in the same order (analysis, design, implementation, testing, and deployment)

They are different in that the waterfall model visits each stage once, while the spiral model cycles back to analysis after the deployment of a prototype.

Waterfall is the less realistic of the two (i.e. staying at design until it's perfect before moving to implementation)

What is the difference between the "uses" & aggregation relationships? Which of the two is considered stronger & why?

"Uses" occurs if a given class needs another class for any reason - aggregation occurs if the given class uses another class as an instance field.

Let's say class X uses class Y & aggregates class Z. Aggregation is the stronger of the two because X 'remembers' Z between method calls, while Y might only be used as a parameter to a method, or as a local variable.

When developing classes for a problem, nouns usually indicate , while verbs or actions indicate .

In Java a given class may implement at most a single interface.

True

A shallow copy of a given object creates complete copies of its subobjects.

True

The operator instanceof works on both classes & interfaces.

False