CS141 programming II (java)

Assignment 1

Chapter 9 –Interface, polymorphism

Chapter 10 - Inheritance

Assignment 1 worth 5 points, Due on Sunday 16/10/2016. Will deduct 1 point for each day late. Copying each other work will result in giving 0 to both.

Q1. Explain why and give examples.   "In OO programming Polymorphism is only possible when using Interface or Inheritance" . (5 points)

ANSWER:

For a method to be polymorphic it has to behave differently depending on the object that calls it. The Java compiler bind the method call dynamically at run time and determine the correct method to execute.

The mechanism to implement polymorphism is in using Inheritance or Interface

In Interface more than one class can implement the same Interface. The same method name and signature that is specified in the interface must be implemented in the classes that implements that interface.

Example both BankAccount and Coint classes implement the Measurable Interface that has a method called getMeasure()

```
BankAccount a1 = new BankAccount();

Coin  c1 = new Coin();
```

Now both objects can call getMeasure() method which will be a polymorphic method since it will behave differently depending on the object that called it.

```
a1.getMeasure();     // will return balance
```

c1.getMeasure();   // will return coin value

In Inheritance polymorphism is also possible when a child class (subclass) implements the same method that was implemented in its parent (superclass).

Example: a superclass BankAccount class has a method called withdraw() that is implemented.

But it's child CheckingAccount class re-implement the same method withdraw()

BankAccount p1 = new BankAccount();

CheckingAccount c1 = new CheckingAccount();

Now both object can call withdraw() method which will be a polymorphic method since it will behave differently depending on the object that called it.

p1.withdraw();   // will run the parent method

c1.withdraw();   // will run the child method

Q2. Can we create an object of type Interface? Explain your answer. (5 points)

ANSWER:

No.

Interface is not a class, it is like a contract that specify the methods that must be implemented by the classes that will use that interface.

The Interface has no constructor or instance variables. It only has public abstract methods that are defined but not implemented.

Q3. Complete the following code, and run to demonstrate the use of Interface. (10 points)

The main class tester is given to you, complete the code in the Interface and the two Classes.

Notice when coding Interface using Eclipse choose New Interface, NOT New Class

Here is a sample Run:

```
Return 0.0
Return 2000.5
Return 1.0
Return 0.25
```

```
public class chapt9Tester {                  // the tester is complete,
just run it after you finish the rest of the code

      public static void main(String[] args) {

            // create two bankaccount objects a1 using constructor
without parameters, and object a2 using constructor with parameter
sending it the amount 2000.50

            BankAccount a1 = new BankAccount();

            BankAccount a2 = new BankAccount(2000.50);

            // create two coin objects  c1 using constructor without
parameters and c2 using constructor and sending 0.25  and "Quarter"

            Coin c1 = new Coin();

            Coin c2 = new Coin(0.25, "Quarter");


            // now create an array of 4 elements of type Measurable and
load it with the 4 objects that you just created

            Measurable marr[] = { a1, a2, c1, c2 };

            for ( int i=0; i< marr.length; i++) {

                  double r = marr[i].getMeasure();    // getMeasure()
will behave differently depending on the object that called it

                  System.out.println("Return " + r);

            }


                  // notice that a variable of type Measure marr  can
refer to BankAccount object or Coin object without casting.
                  // however it can only call the methods that are
declared in the Measure interface
                  // so a call like marr[0].withdraw() is not allowed

      }

}
```

```
// This is the Interface complete the missing code, (if you use
Eclipse: click on File -> New -> Interface )

Public _______________ Measurable {

         double getMeasure();

}

// BankAccount class will use the interface

public class BankAccount ___________________ Measurable{

      private double balance;

      public BankAccount(double amt) {                  //constructor
with parameter
            balance = amt;
      }

      public BankAccount() {              //constructor without
parameter, set balance to 0
            ____________________
      }

      public void deposit(double amt){
            balance = balance + amt;
      }

      public void withdraw(double amt){
            balance = balance - amt;
      }

      @Override
      public double getMeasure() {

            return balance;
      }

}



// Coin class will use the same interface



public class Coin ________________   ____________________ {

      private double value = 1;
      private String name = "Dollar";

      public Coin(double v, String n){          // constructor with two
parameters
            value = v;
            name = n;
      }

      public ________________ {              // constructor without
parameters
            value = 1;
            ______________________;
      }

      // implement the interface method getMeasure here

      public _________  ____________________ {
            return value;
      }


}
```

ANSWER:

```
public interface Measurable {

         double getMeasure();

}

public class BankAccount implements Measurable{

      private double balance;

      public BankAccount(double amt) {                  //constructor
with parameter
            balance = amt;
      }

      public BankAccount() {               //constructor without
parameter
            balance = 0;
      }

      public void deposit(double amt){
            balance = balance + amt;
      }

      public void withdraw(double amt){
            balance = balance - amt;
      }

      @Override
      public double getMeasure() {

            return balance;
      }
}


public class Coin implements Measurable {

	private double value = 1;
	private String name = "Dollar";

	public Coin(double v, String n){		// constructor with two 
paramters
		value = v;
		name = n;
	}
	public Coin(){			// constructor without paramters
		value = 1;
		name = "Dollar";
	}

	@Override
	public double getMeasure() {
		return value;
	}


}
```

Q4. What is the output of the following code: (10 points)

```
public class InheritanceTester {

	public static void main(String[] args) {
		MySubClass mysub = new MySubClass();
		System.out.println(mysub.myMethod());
	}

}
```

```
public class MySubClass extends MySuperClass{

	public MySubClass()
	{
		System.out.println("Subclass constructor has been 
called..");
	}

}
```

```
public class MySuperClass {

	public MySuperClass()
	{
		System.out.println("Superclass constructor has been called..");
	}

	public String myMethod()
	{
		return "Superclass method has been called";
	}
}
```

Answer:

```
Superclass constructor has been called..
Subclass constructor has been called..
Superclass method has been called
```

Q5. **Consider the following problem description and write java code: (20 points)**

```
        Employee
       /        \
Executive        SoftwareEngineer
                        \
                         SoftwareManager
```

A company need to store its employees information. Each employee has an id, name, date of birth, phone number and salary. Using the diagram, write your java classes that shows the inheritance between these classes, and consider the following:

1. Class employee has a method work() which returns the statement "`I am an employee`"
2. Class employee overrides toString() method to return the employee information. Or if you don't what to override toString() you can write your own method to return the employee info. (both ways are accepted)
3. Each subclass should overrides the method work() to return its work. You need to call work() of the superclass. For example:
   Method work() in executive class should returns:
   `I am an employee.. I work as an executive.`

   `So in each subclass work() method first call it's parent work() method by using super. Which will return "I am an employee .." and then add the child own details. in this example "I work as an executive"`

4. Implement the main class Company which constructs objects from the 3 subclasses, and test all of their methods.

   Sample Run will result in:

```
I am an employee.. I work as an executive.
My ID is:  1234567890
My Name is:   John Smith
My Date of Birth is:  1-1-1981
My phone is:  555-555-12345
My Salary is:  10,000


I am an employee.. I work as a Software Engineer.
My ID is:  0987654321
My Name is:   Jane Doe
My Date of Birth is:  1-1-1988
My phone is:  555-555-12345
My Salary is:  11,000


I am an employee.. I work as a Software Manager.
My ID is:  9988776655
My Name is:   Joe Green
My Date of Birth is:  1-1-1978
My phone is:  555-555-12345
My Salary is:  11,100
```

Answer:
This is my version of simplified answer for this program. I think some answer like this can be accepted from students, but it should include the "grand son child" SoftwareManager.

```
public class Ch10Company {

	public static void main(String[] args) {

		Executive exc = new Executive(123456789, "John Smith", 
12000.00);

		SoftwareEng eng = new SoftwareEng(987654321, "Jane Doe", 
10000.00);

		exc.work();
		System.out.println( exc.getInfo() );

		eng.work();
		System.out.println( eng.getInfo() );
	}
}

public class Employee {

	private int ID;
	private String name;
	private double salary;

	public Employee(int id, String n, double s)
	{
		ID = id;
		name = n;
		salary = s;
	}

	public String getInfo()
	{
		return  ( ID + " " + name + " " + salary );
	}
	public void work()
	{
		System.out.print("I am an Employee ...");
	}
}

public class Executive extends Employee {

	public Executive(int i, String n, double s)
	{
		super(i, n, s);
	}

	public void work()
	{
		super.work();
		System.out.println("I work as an Executive. ");
	}

}

public class SoftwareEng extends Employee {

	public SoftwareEng(int i, String n, double s)
	{
		super(i, n, s);
	}

	public void work()
	{
		super.work();
		System.out.println("I work as a Software Engineer. ");
	}

}
```