Exercise 1

Define the following key Object Orientate Programming (OOP) principles with examples:

* Encapsulation
* Inheritance
* Polymorphism
* Abstraction
* **Encapsulation**

Encapsulation allows you to encapsulate things like a function or variable so that they are not affected by using the same function or variable in another class.

To avoid the clash you can use the syntax “private” which means variables and functions cannot be accessed by another class.

**Example**

public class encapExample{

private string name;

private string idNum;

private int age;

public string getName() {

return name;

}

public int getAge() {

return age;

}

public string getidNum() {

return idNum;

}

public void setName( string newName) {

name = newName;

}

public void setAge( int newAge) {

age = newAge’

}

public void setidNum( string newId) {

idNum = newId;

}

The public “set…()” and “get…()” methods are the access points of the class named encapsExample. If you want to be able to access these methods from another class you must use these access points which are also known as getters and setters.

**Example**

Public class RunExample {

Public static void main(string args[]) {

encapExample encap = new encapExample();

encap.setName(“James”);

encap.setAge(20);

encap.setidNum(4244ms);

system.out.print(“Name : “ + encap.getName() + “Age : “ + encap.getAge());

}

}

* **Inheritance**

Inheritance is when a sub class “*Inherits*” the properties and attributes from a super-class. This technique will usually be used for re-usability and maintainability.

To use the inheritance method in Java you will see the keyword “extends”

**Example**

public class employee {

private string Department;

private int salary;

private byte age;

}

public class manager extends employee {

}

* Polymorphism

***“Poly” means many and “morphos” means forms.*** Polymorphism allows a word to be interpreted correctly in different situations based on its context and there are two different types of polymorphism: Dynamic and Static.

**Dynamic (run time polymorphism / Method Overriding)**

Dynamic polymorphism happens when you have created a sub-class that has inherited attributes that have already been defined from its super-class. In certain cases you may find that some of the pre-defined attributes from the super-class are not suitable for your sub-class objects so you have to override the parent members.

**Example**

public class animal {

public void makeNoise( )

{

system.out.println(“animal sound”);

}

}

public class pig extends animal {

public void makeNoise( )

{

system.out.println(“Oink Oink”);

}

}

public class horse extends animal {

public void makeNoise( )

{

system.out.println(“Neigh”);

}

}

Here we can see inheritance with altered data to explain the polymorphism stage.

Depending on what animal is chosen will determine which makeNoise( ) method will be printed. After “public class

**Example**

public class example {

private static void main(string[] args) {

Animal a1 = new Pig( );

a1.makeNoise( ); // this will print Oink Oink based on

overridden inheritance

Animal a2 = new Horse( );

A2.makeNoise( ); // this will print Neigh based on

} overridden inheritance

}

**Static (compile time polymorphism / Method Overloading)**

Static polymorphism is when you use a different method of calling upon a command. You can see from the example three different print methods with different arguments.

In the second half of the declaration after “*public class static* “ you can see the first obj.print is an int but in the first half the print line says string.

**Example**

class Overloadsample {

public void print(String s){

System.out.println("First Method String"+ s);

}

public void print (int i){

System.out.println("Second Method int"+ i);

}

public void print (String s, int i){

System.out.println("Third Method string and int"+ s + "--" + i);

}

}

public class Static {

public static void main(String[] args) {

Overloadsample obj = new Overloadsample();

obj.print(10);

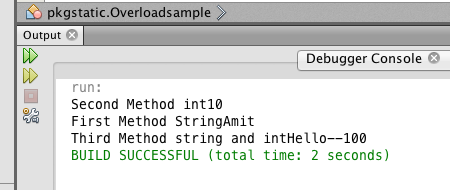
obj.print("Amit");

obj.print("Hello", 100);

}

}

Lets see what will happen after it is compiled.



We can see that the first half of the declaration demanded a string however in the second half the first argument was obj.print(10); and java knows to look for the int print line and use that argument first. It also correctly figures out the string print line and the string & int print line and compiles correctly.

The word “public” in the declaration means anybody can access the class “static” in the declaration means the information can be share by all objects in the class.

This is polymorphism because it is a different way of

* Abstraction

Abstraction is the action of dealing with the idea rather than the event. for example, when you turn on a light you flick a light switch and the light is on. What you don’t see is a circuit being complete and the current running through the circuit from positive round to negative to turn on that light. In object oriented programming, abstraction hides the details from the user but provides the function for him. (Hides the circuit but provides the light switch.) Another way to put it, the user has information on what the object does but not how it does it.

**Abstract** **Class**

A class, which contains the keyword “abstract” in its declaration, is known as an “Abstract Class”

* A class must be declared abstract from it contains at least one abstract method.
* To use an abstract class, you must inherit it from another class, and then you must provide the getters to abstract methods in it.