**Exercise 1**

**Encapsulation**

This is the process of “hiding” a variable and making it so you cannot directly modify a variable. The variable must be accessed through a public method (get and set usually). This method is extremely useful in hiding what goes on behind the scenes from a user.

An example:

private int example;

public int getExample()

{

return example;

}

public void setExample(int newExample)

{

example = newExample;  
}

**Inheritance**

This allows you to have a subclass inherit properties and methods from a main class. The properties and methods in the base class can be used and called from within the derived class. The derived class can add methods of it’s own and variations on the initial classes by overriding them.

An example:

class Shoe  
{

String colour;

void printColour()  
{  
System.out.println(“Colour: “ + colour);  
}

}

class Trainers extends Shoe

{

String pattern;  
void trainerAttributes()  
{

System.out.println(“Colour: “ + colour);

System.out.println(“Pattern: “ + pattern);

}  
}

public class Example

{

public static void main(String args[])

{

Trainers t = new Trainers();

t.colour = “Red”

t.pattern = “striped”

t.trainerAttributes();

}

}

Output:

Colour: red

Pattern: striped

**Polymorphism**

This idea allows a method to have multiple implementations based on the objects/variables are passed to it. It is commonly used to have a private internal method with multiple user implementations on the front end. Two common ways of achieving this outcome are method overloading and method overriding.

Overloading example:

class OverloadExample

{

void example (int varOne)

{

System.out.println(“One: “ + varOne);

}

void example (int varOne, int varTwo)

{

System.out.println(“One: “ + varOne + “\nTwo: “ + varTwo);

}

}

class Overloading

{

public static void main (String args[])

{

OverloadExample a = new OverloadExample();

a.example(1);

a.example(2,3);

}

}

Output:

One: 1

One: 2

Two: 3

Overriding:

public class Base

{

public void example()

{

System.out.println ("Base method");

}

}

public class Derived extends Base

{

public void example()

{

System.out.println ("Derived method");

}

}

public class TestMethod

{

public static void main (String args [])

{

Base one = new Base();

Base two = new Derived();

one.example();

two.example();

}

}

Output:

Base method

Derived method

**Abstraction**

Abstract classes are used to prevent the main class from calling directly. You must inherit from a class for the function to be called in main. Abstract classes cannot be instantiated in main. Abstract classes may contain abstract methods (a method without a body) which require an implementation in all derived classes.

An example of a good use of an abstract class would be to do some form of calculation. i.e. if you declared integer variables “one” and “two” in the derived class and had a method to Add(int inputone, int inputtwo) in the abstract class, you may call this within the derived classes but NOT in the main class. You can add a pass through if required by using DerivedAdd(int one, int two) where the implementation is simply Add(one,two).