OBJECT ORIENTED PROGRAMMING

- What is encapsulation?How does a class accomplish data hiding? Give an example. List out the differences between Procedural and Object oriented programming.

Ans : Encapsulation is a method of wrapping the data(variables) and the methods together as a single unit. On achieving encapsulation variables of a class can only be accessed through methods of their current class.

Example:

```
import java.util.*;
class Student{
  // for "private" variables we have to use 'getters' and 'setters'
  private int roll;
  public void setRoll(int roll){
    this.roll = roll;
  }
  public int getRoll(){
    return this.roll;
  }
}
class TestEncap{
  public static void main(String[] args) {
    Student s1 = new Student();
    s1.setRoll(83);
    System.out.println("Student :: Roll : " + s1.getRoll() );
  }
}
```

OUTPUT:

```
PS D:\BPPIMT\Sem_5\OOP Theory\CA3> javac encap.java
PS D:\BPPIMT\Sem_5\OOP Theory\CA3> java TestEncap
Student :: Roll : 83
```

Points	Procedural programming	OOP
Definition	Programming paradigm based on procedure call where procedures contain a series of computational steps to be carried out	Programming paradigm based on the concept of “objects”,here, procedures can be attached to objects
Security	Less secure	Data hiding possible due to abstraction; hence more secure
Approach	Top-down	Bottom-up
Orientation	Structure/ procedure oriented	Object oriented

Access modifiers	No access modifiers	Public, private, protected
Inheritance	Doesn’t support	Exists
Code reusability	Not possible	Possible due to inheritance
Overloading	Not possible	Function(method) and operator overloading possible
Importance	Gives importance to functions over data	Gives importance to data over functions
Program division	A program is divided into divisions referred to as “functions”	Program divided into objects
languages	C, Fortran, Pascal	Java, Python, C++

Write a program to access static variables and static methods to explain static keyword property.

Solution:

CODE:

```
import java.util.*;
class StaticDemo{
  static int roll = 83;
  static void method1(){
    System.out.println("Python tow Java r theke easy!!");
    // 'roll' couldn't have been accessed if it wasn't static ( as we had to create
//an object of class 'StaticDemo' first
    System.out.println("Sample roll :: " + roll);
  }
  public static void main(String[] args) {
    method1();
  }
}
```

OUTPUT:

```
PS D:\BPPIMT\Sem_5\OOP Theory\CA3> javac static.java
PS D:\BPPIMT\Sem_5\OOP Theory\CA3> java StaticDemo
Python tow Java r theke easy!!
Sample roll :: 83
```

- What are the benefits of organizing classes into package?How can you create your own package and add classes to that? Explain with help of an example

Benefits of organizing data into packages :

1. Preventing naming conflicts : classes having same name can be located in 2 different packages, thus preventing “duplicate class” error

2. Maintenance : Java packages are used to categorize classes and interfaces so that hey can be easily maintained
3. Access protection : 'protected' and 'default' have package level access control. Protected members are accessible by package members and their subclasses.Default member is accessible by classes only in the same package.

Example:

CODE:

Package2.java:

```
import pack1.*;
class Sample{
  public static void main(String[] args) {
    pack1.HelloWorld obj = new pack1.HelloWorld();
    obj.insideClass();
  }
}
```

HelloWorld.java

```
package pack1;
import java.util.*;
public class HelloWorld{
  public void insideClass(){
    System.out.println("Hello World !!");
  }
}
```

OUTPUT:

```
PS D:\BPPIMT\Sem_5\OOP Theory\CA3> javac -d. HelloWorld.java
PS D:\BPPIMT\Sem_5\OOP Theory\CA3> javac package2.java
PS D:\BPPIMT\Sem_5\OOP Theory\CA3> java Sample
Hello World !!
```

- What are the differences between method overloading and method overriding? What is dynamic binding and early binding.Explain dynamic method dispatch with a suitable example.

Solution.

Point	Method Overloading	Method Overriding
Polymorphism	Compile-time polymorphism	Runtime polymorphism
Uses	Increases readability of program	Grants a specific implementation of inherited method
Scope	It occurs within the class	Performed in two classes with inheritance relationships
Inheritance	May/ may not require inheritance	Inheritance is necessary
Method specifications	Same name, different	Same name, same signature

	signatures	
Return type	May or may not be same, we have to just change parameters	Must be same or a co-variant
Binding	Static binding	Dynamic binding
Performance	Better performance	Poor performance
Access	Private, final methods can be overloaded	Private, final methods cannot be overridden
Argument list	Should be different	Should be same

Overloading in inherited class : https://www.geeksforgeeks.org/does-overloading-work-with-inheritance/
Static binding : Static Binding or Early Binding in Java refers to a process where the compiler determines the type of object and resolves the method during the compile-time. Generally, the compiler binds the overloaded methods using static binding.
Dynamic binding : When type of the object is determined at run-time, it is known as dynamic binding or late binding . Generally , overridden methods are binded using dynamic binding. Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at runtime.
CODE:

```
class Number{
  public int returnNumber(int i){
    System.out.print(" Number  (int) : ");
    return i;
  }
}
class DerivedNumber extends Number{
  public double returnNumber(double d){
    System.out.print(" Number  (double) : ");
    return d;
  }
}
class DynaDispatchTrial{
  public static void main(String[] args) {
    DerivedNumber obj = new DerivedNumber();
    System.out.println(obj.returnNumber(29));
    System.out.println(obj.returnNumber(2.9));
  }
}
```

OUTPUT:

```
PS D:\BPPIMT\Sem_5\OOP Theory\CA3> javac binding.java
PS D:\BPPIMT\Sem_5\OOP Theory\CA3> java DynaDispatchTrial
 Number  (int) : 29
 Number  (double) : 2.9
```

- What is interface?Explain with help of an example how java gets befitted by using interface. Illustrate abstract class with an example. Differentiate abstract class with interface.

Solution.