OBJECT ORIENTED PROGRAMMING (PCCCS503)

Question-5 marks

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

Answer:

Encapsulation in Java is a process of wrapping data and code acting on the data (methods) together into a single unit.
It is a way to achieve data hiding in Java because other class will not be able to access the data through the private data members. The data can be accessed via getter and setters. Example given below:

```
public class Student{
    private String name;
    public String getName(){
    return name;
    }
    public void setName(String name){
    this.name=name
    }
}
class Test{
    public static void main(String[] args){
    Student s=new Student();
    s.setName("vijay");
    System.out.println(s.getName());
  }
}
```

Difference between oop programming and procedural programming:

Procedural Oriented Programming	***Object-Oriented Programming***
● In procedural programming, the program is divided into small parts called functions.	In object-oriented programming, the program is divided into small parts called ***objects***.
● Procedural programming follows a top-down approach.	Object-oriented programming follows a ***bottom-up approach***.
● Adding new data and functions is not easy. .	Adding new data and function is easy.
● There is no access specifier in procedural programming.	OOP has access specifiers like private, public, protected, etc.

- Write a program to access static variable and static method to explain static keyword property.

Answer:

Static variable: The static variable can be used to refer to the common property of all objects (which is not unique for each object. The static variable gets memory only once in the class area at the time of class loading.

Static method: A static method belongs to the class rather than the object of a class. A static method can be invoked without the need for creating an instance of a class. A static method can access static data member and can change the value of it.

Examples:

```
class Counter2{
 //Java Program to illustrate the use of static variable which
static int count=0;//will get memory only once and retain its value

Counter2(){
        count++;
        System.out.println(count);
}

public static void main(String args[]){

        Counter2 c1=new Counter2();
        Counter2 c2=new Counter2();
        Counter2 c3=new Counter2();
}
}
```

```
 //Java Program to demonstrate the use of a static method.
class Student{
    int rollno;
    String name;
    static String college = "ITS";
    //static method to change the value of static variable
    static void change(){
    college = "BBDIT";
    }
    //constructor to initialize the variable
    Student(int r, String n){
    rollno = r;
    name = n;
    }
    //method to display values
    void display(){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to create and display the values of object
public class TestStaticMethod{
   public static void main(String args[]){
   Student.change();//calling change method
   //creating objects
   Student s1 = new Student(111,"Karan");
   Student s2 = new Student(222,"Aryan");
   Student s3 = new Student(333,"Sonoo");
   //calling display method
   s1.display();
   s2.display();
   s3.display();
   }
}
```

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

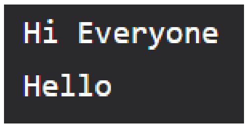
Answer:
The advantages of using packages are thus- Since the package creates a new namespace there won't be any name conflicts with names in other packages, using packages, it is easier to provide access control Protected members are accessible by package members and their subclasses.Default member is accessible by classes only in the same package., It is also easier to locate the related classes.

```
package data;

// Class to which the above package belongs
public class Demo {

   // Member functions of the class- 'Demo'
   // Method 1 - To show()
   public void show()
   {

      // Print message
      System.out.println("Hi Everyone");
   }

   // Method 2 - To show()
```

```
    public void view()
    {
        // Print message
        System.out.println("Hello");
    }
}

// Code to show how the package is imported

import data.*;

// Class to which the package belongs
class ncj {

    // main driver method
    public static void main(String arg[])
    {

        // Creating an object of Demo class
        Demo d = new Demo();

        // Calling the functions show() and view()
        // using the object of Demo class
        d.show();
        d.view();
    }
}
```

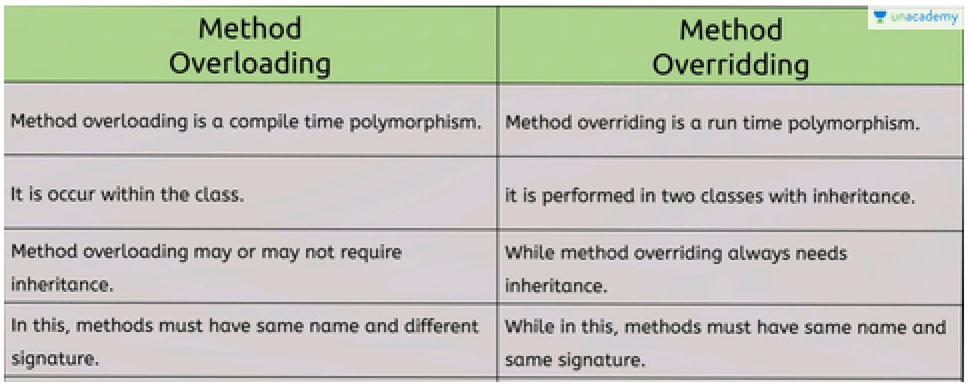
To generate the output, do javac Demo.java , this Give us a class file that is non-runnable so we do need a command further to make it an executable run file- java ncj

Output:



```
Hi Everyone
Hello
```

- 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.



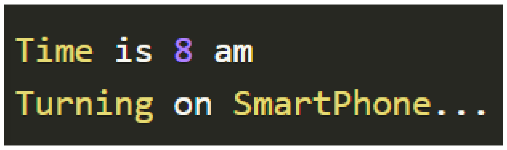
Method Overloading	Method Overridding
Method overloading is a compile time polymorphism.	Method overriding is a run time polymorphism.
It is occur within the class.	it is performed in two classes with inheritance.
Method overloading may or may not require inheritance.	While method overriding always needs inheritance.
In this, methods must have same name and different signature.	While in this, methods must have same name and same signature.

The binding which can be resolved at compile time by the compiler is known as **static or early binding.** The binding of all the static, private, and final methods is done at compile-time.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.

In Dynamic binding compiler doesn't decide the method to be called. Overriding is a perfect example of **dynamic binding**. In overriding both parent and child classes have the same method.

Dynamic method dispatch is also known as run time polymorphism. It is the process through which a call to an overridden method is resolved at runtime. It is related to upcasting- It is a technique in which a superclass reference variable refers to the object of the subclass. Example :

```
class Phone{
   public void showTime(){
      System.out.println("Time is 8 am");
   }
   public void on(){
      System.out.println("Turning on Phone...");
   }
}

class SmartPhone extends Phone{
   public void music(){
      System.out.println("Playing music...");
   }
   public void on(){
      System.out.println("Turning on SmartPhone...");
   }
}
public class CWH {
   public static void main(String[] args) {

      Phone obj = new SmartPhone();
      obj.showTime();
      obj.on();

   }
}
```

Output: 

```
Time is 8 am
Turning on SmartPhone...
```

- 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.

Answer:
An interface in Java is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction

```
 //Example showcasing use of interface
interface Drawable{
  void draw();
  }
  //Implementation: by second user
  class Rectangle implements Drawable{
  public void draw(){System.out.println("drawing rectangle");}
  }
  class Circle implements Drawable{
  public void draw(){System.out.println("drawing circle");}
  }
  //Using interface: by third user
  class TestInterface1{
  public static void main(String args[]){
  Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDrawable()
  d.draw();
  }}
```

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.

2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.

- Describe a mechanism where both Inheritance and Polymorphism is used. Explain with an example.
- Explain the use of final keyword using a example program.

Answer:
The final keyword in java is used to restrict the user. If you make any variable as final, you cannot change the value of final variable(It will be constant). If a method is declared as final it can't be overridden, if a class is declared as final it can't be extended.

Question 1 marks

1. Which of this keyword must be used to inherit a class? - extends
2. Which of the following is true about inheritance in Java?

- Private methods are final. (True)
- Protected members are accessible within a package and inherited classes outside the package.(True)
- Protected methods are final.(False)
- We cannot override private methods.(True)

3. Identify the correct restriction on static methods.

- They must access only static data
- They can only call other static methods.
- They cannot refer to this or super.

4. Super refers to –parent class-------------
5. -----static final------------keyword is used to declare class constant.
6. Method overloading shows ………compile time………………….polymorphism.

Predict output of following

```
1.Public class Test{
        Public static void main(String argos[]){
                String str1 = "one";
                String str2 = "two";
                System.out.println(str1.concat(str2));
        }
}
```

o/p : onetwo

```
2. class A
   {
      int i;
      void display()
      {
         System.out.println(i);
      }
   }
   class B extends A
   {
      int j;
      void display()
      {
         System.out.println(j);
      }
   }
   class inheritance_demo
   {
      public static void main(String args[])
      {
         B obj = new B();
         obj.i=1;
         obj.j=2;
         obj.display();
      }
  }
```

o/p: 2

```
3.  class average {
      public static void main(String args[])
      {
        double num[] = {5.5, 10.1, 11, 12.8, 5.5};
         double result;
         result = 0;
         for (int i = 0; i < 6; ++i)
            result = result + num[i];
            System.out.print(result/6);

      }
```

```
    }
```

o/p: compile error

4. class increment {

```
                 public static void main(String args[])
                 {
                     int g = 3;
                     System.out.print(++g * ++g);
                 }
              }
```

o/p: 20

5.   class selection_statements

```
  {
     public static void main(String args[])
     {
        int var1 = 4;
        int var2 = 6;
        if ((var2 = 1) == var1)
           System.out.print(var2);
        else
           System.out.print(++var2);
     }
  }
```

o/p: 2

6. class A

```
   {
      public int i;
      public int j;
      A()
      {
         i = 1;
         j = 2;
         }
   }
   class B extends A
   {
      int a;
      B()
      {
         super();
```

```
        }
    }
    class super_use
    {
        public static void main(String args[])
        {
            B obj = new B();
            System.out.println(obj.i + " " + obj.j)   }}
```

o/p: 1 2

7. **public class** Calculator

```
{
    int num = 100;
    public void calc(int num)  { num = num * 10;  }
    public void printNum()     { System.out.println(num); }

    public static void main(String[] args)
    {
        Calculator obj = new Calculator();
        obj.calc(2);
        obj.printNum();
    }
}
```

o/p: 100

8. class main_class

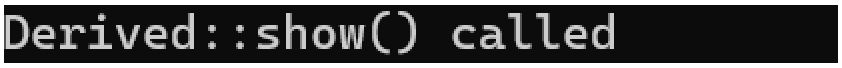
```
        {
          public static void main(String args[])
           {
               int x = 9;
               if (x == 9)
               {
                  int x = 15;
                   System.out.println(x);
               }}}
```

o/p: Error

9. class box

```
        {
           int width;
        int height;
        int length;
    }
    class mainclass
    {
        public static void main(String args[])
        {
            box obj = new box();
            obj.width = 10;
            obj.height = 20;
            obj.length = 10;
            int y = obj.width * obj.height * obj.length;
            System.out.print(y);
        }
    }
```

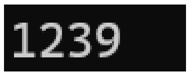
10.
```
class Base {
   public void show() {
      System.out.println("Base::show() called");
   }
}

class Derived extends Base {
   public void show() {
      System.out.println("Derived::show() called");
   }
}

public class Main {
   public static void main(String[] args) {
       Base b = new Derived();;
       b.show();
   }
}
```
OUTPUT:



```
Derived::show() called
```

11.
```
public class Test
  {
    static
    {  System.out.print("1");    }

    static
    {System.out.print("2");    }

    static
    {      System.out.print("3");   }

private int function()
    {return 4; }

    private static int myMethod()
    {     return 5; }


    public static void main(String[] args)
    {
        System.out.println((new Test()).function()+myMethod());
    }
}
```

OUTPUT:



```
1239
```