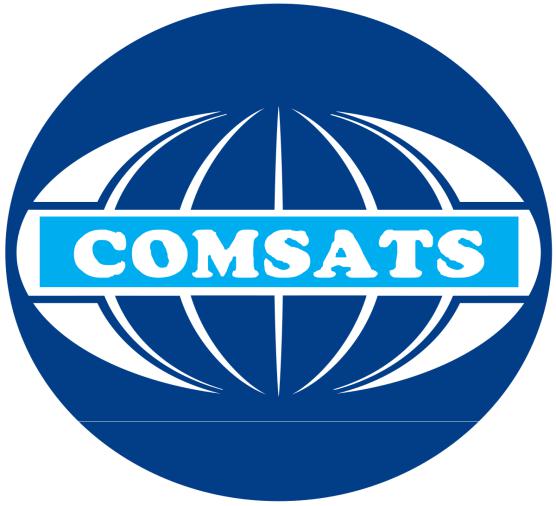
**Object Oriented Programming Theory**

****

**Assignment 3rd**

**Submitted to: Sir-Muhammad Ali-Khan**

**Submitted by: Nasratullah (FA18-BCS-401)**

**Submission Date: 30 May ,2021.**

**Section: B**

**Q1.** Explain with the help of code fragments the following:

1. Types of Inner classes (four types)
2. Final methods
3. Type casting
4. Aggregation

## Question 1 Ans: Inner Classes in Java

Inner classes in Java are the classes that are defined inside the scope of another class. Inner classes are also called nested classes. This helps in easy documentation and better maintenance of the code. Types of nested class.

* Non- Static nested class (inner class)

1. Member inner class
2. Anonymous inner class
3. Local inner class

* Static nested class

**Member Inner class:**

* A Class that is declared inside a class but outside a method is known as member inner class.

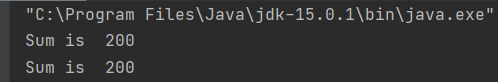
**Invocation of Member Inner class:**

* From within the class
* From outside the class

Example:

|  |
| --- |
| package Terminal;  public class OuterExample {  // member variable  int a = 50;  private int b = 80;  static int *c* = 70;  // member inner class  class InnerExample {  public void doSum (){  System.*out*.println("Sum is "+ (a+b+*c*));  }  }  public static void main(String[] args) {  // Outer Object  OuterExample outer = new OuterExample();  // Inner object we pass ref of outer  OuterExample.InnerExample ob = outer.new InnerExample();  ob.doSum();  // You can also above above object in one steps  OuterExample.InnerExample ob1 = new OuterExample().new InnerExample();  ob1.doSum();  } } |

Output:



**Anonymous inner class:**

* A Class that have no name is known as anonymous inner class.

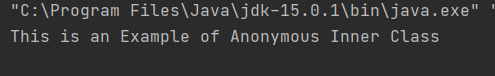
**Anonymous class can be created by:**

* Class (may be abstract class also)
* Interface

**Example:**

package Terminal;  
abstract class AnonymousClassExample {  
 public abstract void display();  
}  
public class OuterClass {  
  
 public static void main(String[] args) {  
 AnonymousClassExample AnClass = new AnonymousClassExample() {  
 @Override  
 public void display() {  
 System.*out*.println("This is an Example of Anonymous Inner Class");  
 }  
 };  
 AnClass.display();  
 }  
}

**Output:**

****

**Local Inner Class**

* A Class that is Created inside a method is known as Local Inner Class.
* If you want to invoke the methods of local inner class, you must instantiate this class inside the method.

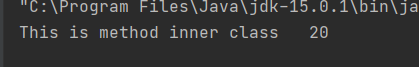
**Rules for Local Inner Class:**

* Local variable can’t be private, public or protected.
* Local inner class cannot be invoked from outside the method.
* Local inner class cannot access non-final local variable.

**Example:**

|  |
| --- |
| package Terminal;  public class MethodLocalInnerClass {  int num = 20;  // Instance Method of the Outer Class  public void Demo(){  // Metthod-Local inner class  class InnerClass{  public void display(){  System.*out*.println("This is method inner class "+ num);  }  } // end of inner class  // Accessing inner class( Instantiation)  InnerClass InClass= new InnerClass();  InClass.display();  }  public static void main(String[] args) {  MethodLocalInnerClass obj = new MethodLocalInnerClass();  obj.Demo();    } } |

**Output:**

****

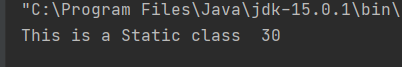
**Static nested Class**

* A static class that is created inside a class is known as static nested class.
* It can access static data members of outer class including private.
* Static nested class cannot access non-static (instance) data member or method.

**Example:**

|  |
| --- |
| package Terminal;   public class OuterClass {  static int *data* = 30;  static class StaticInnerClass{  public void display(){  System.*out*.println("This is a Static class " + *data*);  }  }   public static void main(String[] args) {  OuterClass.StaticInnerClass Sic = new OuterClass.StaticInnerClass();  Sic.display(); // You can also Instantiate like this. // StaticInnerClass Ob1 = new StaticInnerClass(); // Ob1.display();    }  } |

Output:



**Purpose of nested Classes**

* Nested Classes represent a special type of relationship that is it can access all the members (data members and methods) of outer class including private.
* Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one place only.
* Code Optimization: It requires less code to write.

**Question 2: Final Methods.**

* We cannot Override final method in subclass. We define a method final when we are concerned that a subclass may accidentally redefine the method( override).
* **Syntax:**

Final void disp ()

{

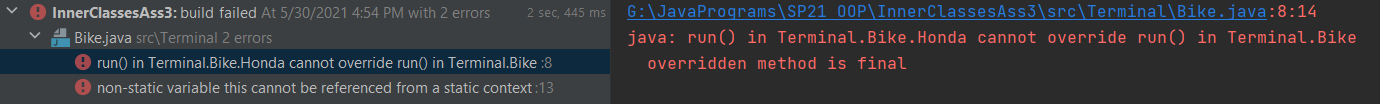
System.out.println(“Final Method ”);

}

**Example:**

|  |
| --- |
| package Terminal;  public class Bike {  final void run (){  System.*out*.println("Bike is Running");  }  class Honda extends Bike{  void run (){  System.*out*.println("Bike is running safely with 100 kmph");  }  }  public static void main(String[] args) {  Honda honda = new Honda();  honda.run();    }  } |

**Output:**

****

**The final method is inherited but you cannot Override it.**

**Example:**

|  |
| --- |
| package Terminal;  class Bike{  final void run() {  System.*out*.println("Bike is Running...");  } }  public class Honda2 extends Bike {  public static void main(String[] args) {  new Honda2().run();  } } |

**Output:**

****

**Question 3: Type Casting in Java.**

* Assigning a value of one type to a variable of another type is known as Type Casting. Ex: int x= 10; byte y = (byte) x;
* There are Two types of type casting.

1. Widening Casting (Implicit)
2. Narrowing Casting (Explicit)

**Widening Casting (Implicit)**

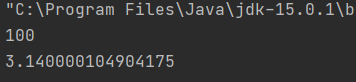
* In widening casting we converted the data or value into broad data **Ex:**

Byte 🡪 Short🡪int🡪long🡪float🡪double

**Example:**

|  |
| --- |
| package Terminal; public class Casting {  public static void main(String[] args) {  // conversion of int to lang  int a = 100;  long b = a;  // conversion of float to double .  float f = 3.14f;  double d = f;   System.*out*.println(b);  System.*out*.println(d);   } } |

**Output:**

****

**Narrowing Casting (Explicit)**

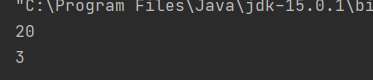
* In Narrowing casting we converted the data or value into Narrow data **Ex:**

double 🡪 float🡪long🡪int🡪short🡪byte

**Example:**

|  |
| --- |
| package Terminal; public class Casting {  public static void main(String[] args) {  int a = 20;  byte b = (byte)a;  float f = 3.14f;  int c = (int)f;   System.*out*.println(b);  System.*out*.println(c);   } } |

**Output:**



**Question no 4: Aggregation in java**

Aggregation is the same as association. Like association, this relationship develops while one object uses the other. Additionally, the object develop a part-whole relationship and the lifetime of part does not depend on life time of whole.

Aggregation is also a HAS-A relationship but it contains a PART-WHOLE relationship, where PART and WHOLE can live separately. For example, the relationship between a Library and Books is aggregation because Book are a PART of WHOLE Library, but if we remove Books from Library, the Library still exist but its just an empty Library. Similarly, Books can also exist in some other place. This is also called weak Association.

**Example:**

**// Employee class**

|  |
| --- |
| package Terminal; // Aggregatin ( weak association example) public class Employee {  int id;  String name;  Address addr;   public Employee(int id, String name, Address addr) {  this.id = id;  this.name = name;  this.addr = addr;  }  public String toString (){  return (id+ " "+ name +" " +addr.HouseNo + " "+ addr.City+ " " + addr.Country );  } // you can also print by using Methods. // public void printInfo (){ // System.out.println(id+ " "+ name +" " +addr.HouseNo + " "+ addr.City+ " " + addr.Country); // } } |

**// Address class**

package Terminal;  
  
public class Address {  
 int HouseNo;  
 String City;  
 String Country;  
  
 public Address(int houseNo, String city, String country) {  
 this.HouseNo = houseNo;  
 this.City = city;  
 this.Country = country;  
 }  
}

/**/ Test class**

package Terminal;  
public class TestClass {  
 public static void main(String[] args) {  
 // Both object are not defend on each other  
 Address Adobj = new Address(5, "Peshawar","pakistan");  
  
 Employee emp1= new Employee(10,"Ali",Adobj);  
 System.*out*.println(emp1);  
 // emp1.printInfo();  
 }  
}

**// Output**

