Nested Class

```
class Outer_Demo
{
  private class Inner_Demo
   {
     public void print()
     {
       System.out.println("This is an inner class");
     }
  }
  void display_Inner()
  {
    Inner_Demo inner = new Inner_Demo();
    inner.print();
  }
}

public class NestedClass
{
  public static void main(String args[]) {
    Outer_Demo outer = new Outer_Demo();
    outer.display_Inner();
  }
}
```

```
class Outer_Demo
{
  int num;
private  class Inner_Demo
   {
     public void print()
     {
       System.out.println("This is an inner class");
     }
  }
  void display_Inner()
  {
    Inner_Demo inner = new Inner_Demo();
    inner.print();
  }
}

public class NestedClass1
{
  public static void main(String args[]) {
   Outer_Demo O=new  Outer_Demo();
   Outer_Demo.Inner_Demo  inner = O.new Inner_Demo();
     //inner.display_Inner();
```

```
   inner.print();
  }
}
```

Constructor Chaining

```
class Temp
{
public Temp() //2
{
this(5);
System.out.println("The Default constructor");
}
public Temp(int x) //3
{
this(5, 15);
System.out.println(x);
}
public Temp(int x, int y) //4
{
System.out.println(x * y);
}
}
class ConstructorChaining
{
public static void main(String args[])
{
new Temp(); //1
}
}
```

Copy Constructor

```
class Data
{
   int data1;
   int data2;
public Data(int d1, int d2) // int  parameter
   {
      data1 = d1;
      data2 = d2;
   }
public  Data(Data obj)   // constr parameter
   {
      data1 = obj.data1 + 100;
      data2 = obj.data2 + 100;
   }
   void showData()
   {
      System.out.println("Data1 :"+data1);
```

```
      System.out.println("Data2 :"+data2);
   }
}

public class CopyConstructor
{
   public static void main(String[] args)
    {
      Data d1  = new Data(20,40);
      d1.showData();

      Data d2 = new Data(d1);
      d2.showData();
   }
}
```

Private Constructor

```
class MyClass
{
   private static MyClass object = null;

   private MyClass()
   {
      //private constructor
   }

   public MyClass getObject()
   {
      if(object == null)
      {
         object = new MyClass();   //Creating object using private constructor
      }

      return object;
   }
}

public class PrivateConstructor
{
  public static void main(String args[])
   {
   }
}

/*

the use of private constructor?
```

Private constructors are used to restrict the instantiation of a class. When a class needs to prevent other classes from creating it's objects then private constructors are suitable for that. Objects to the class which has only private constructors can be created within the class. A very good use of private constructor is in singleton pattern. This ensures only one instance of a class exist at any point of time. Here is an example of singleton pattern using private constructor.

*/

Static Class

```
class OuterClass
{
private static String msg = "Static Class";
private String msg1 = "Non Static and Static Class";

public static class NestedStaticClass
{
	public void printMessage() {

		System.out.println(msg);
	}
}
	public class InnerClass
	 {
	public void display()
	 {
		System.out.println(msg);
		System.out.println(msg1);
	}
	}
}
class StaticClass
{
	public static void main(String args[])
	{
	OuterClass.NestedStaticClass printer = new OuterClass.NestedStaticClass();
	printer.printMessage();

	OuterClass outer = new OuterClass();
	OuterClass.InnerClass inner = outer.new InnerClass();
	inner.display();
	}
}


/*
1) Nested static class doesn't need reference of Outer class,
 but Non-static nested class or Inner class requires Outer class reference.
```

2) Non-static nested class can access both static and non-static members of Outer class.
A static class cannot access non-static members of the Outer class.
It can access only static members of Outer class.
*/

Singleton Demo

```
class ST
{
  private static ST  s = new ST( );

    public static ST getInstance( )
  {
    return s;
  }

   protected static void demoMethod( )
  {
    System.out.println("demoMethod for singleton");
  }
}

public class SingletonDemo
{

  public static void main(String[] args)
  {
    ST tmp = ST.getInstance( );
    tmp.demoMethod( );
  }
}
```

This Keyword

```
class person
{
private String name;

public void setName(String name)
{
this.name=name;
}
public String getName()
{
return name;
}
}
class this1
{
```

```
public static void main(String args[])
{
person p1=new person();
p1.setName("Raju");
System.out.println("Name= " +p1.getName());
}
}
```

Abstract Method

```
abstract class Sum
 {
   public abstract int Two(int n1, int n2);
   public abstract int Three(int n1, int n2, int n3);
 }

class AbstractMethod extends Sum
{
   public int Two(int num1, int num2)
   {
          return num1+num2;
   }
   public int Three(int num1, int num2, int num3)
  {
          return num1+num2+num3;
   }
   public static void main(String args[])
   {
          Sum obj = new AbstractMethod();
          System.out.println(obj.Two(3, 7));
          System.out.println(obj.Three(4, 3, 19));
   }
}
```

ATM Program

```
import java.util.*;

class ATM
{
static int p;
public ATM()
{ System.out.print("Welcome Ur Name .... "); }

public ATM(int p)
{  this.p=p; }


public ATM(int Amt,char T)
{
```

```
if(T=='W')
{
this.p=this.p-Amt;
System.out.println("You have Withdrawn : " + Amt + " Rs/-");
}
else if(T=='D')
{
this.p=this.p+Amt;
System.out.println("You have Deposited: " + Amt + " Rs/-");
}
}
public ATM(char T)
{
System.out.println("Balance Amount is : " + this.p);
}
}

class AtmConstructor
{
static ATM d;
static int n,P=0;

static void options()
{
System.out.println("\n\t Select One Option : \n\t 1. Deposit \n\t 2. Withdraw  \n\t 3. Balance \n\t 4.
Quit");
Scanner sc=new Scanner(System.in);
n=sc.nextInt();

if(n==1)
{
System.out.print("\nEnter Amount to Deposit  :  ");
P=sc.nextInt();
d=new ATM(P,'D');
options();
}
else if(n==2)
{
System.out.print("\nEnter Amount to Withdraw : ");
P=sc.nextInt();
d=new ATM(P,'W');
options();
}

else if(n==3)
{
d=new ATM('B');
options();
```

```
}

else if(n==4)
{
System.exit(0);
}
}

public static void main(String args[])
{
ATM d;
Scanner sc=new Scanner(System.in);

d=new ATM();

System.out.print("\nEnter Principle Amount  : ");
P=sc.nextInt();
d=new ATM(P);

options();
}
}
```
