USING SUPER KEYWORD IN CONSTRUCTOR

```
class Loan{
   protected int tenure;
   protected float interestRate;

   Loan(int tenure, float interestRate){
      this.tenure = tenure;
      this.interestRate = interestRate;
   }
}

class HomeLoan extends Loan{
   HomeLoan(){
      super(5,8.5f);  //invoking super class constructor
   }
   public double calculateEMI(double principal){
      double simpleInterest = (principal * interestRate * tenure) / 100;
      double emi = (simpleInterest + principal) / tenure;
      int additionalTax = 200;
      return emi + additionalTax;
   }
}

class ExecuteLoan{
   public static void main (String[] args) {
      HomeLoan loan = new HomeLoan();     //Runtime polymorphism
      double hloan = loan.calculateEMI(2000000);
      System.out.println("Home loan emi per year..." + hloan);
```

OUTPUT:

```
Home loan emi per year...570200.0
```

```
class Loan {
public double calculateEMI(double principal) {
   double simpleInterest = (principal*8.5*5) / 100;
   double emi = (simpleInterest+principal)/5;
   return emi;
   }
}

   class HomeLoan extends Loan {
      public double calculateEMI(double principal) {
         int additionaltax = 200;
         double emi = super.calculateEMI(principal);      //calling super class method
         return emi + additionaltax;
      }
   }

   class ExecuteLoan {
      public static void main(String[] args) {
         Loan loan = null;
         loan =  new HomeLoan();               // Runtime polymorphism
          double hloan = loan.calculateEMI(2000000);
          System.out.println("Home loan emi per year..."+ hloan);
      }
   }
```

OUTPUT:

```
Home loan emi per year...570200.0
```

FINAL KEYWORD

```
class Demo {
	final int tenure = 0;
	double principal;
	float interestRate;
	String accountNumber;
	final double calculateEMI(){
		return 2000;
	}
}

 class Demo2 extends Demo{

 // Error as  final method is overriding
 double calculateEMI(){
		return 8000;
	}

 }

class FinalDemo{
	public static void main(String[] args) {
		Demo d = new Demo();
		d.tenure = 1;		//Error as tenure is final
		System.out.println(d.tenure);
		System.out.println(d.calculateEMI());
	}
}
```

OUTPUT:

```
Main.java:14: error: calculateEMI() in Demo2 cannot override calculateEMI() in Demo
  double calculateEMI(){
         ^
  overridden method is final
Main.java:23: error: cannot assign a value to final variable tenure
                d.tenure = 1;           //Error as tenure is final
                 ^
2 errors
```

```
final class Demo {
        int tenure = 0;
        double principal;
        float interestRate;
        String accountNumber;
   double calculateEMI(){
                return 2000;
        }
}

 class Dummy extends Demo{

 // Error as  class is final
 double calculateEMI(){
                return 8000;
        }

 }

class FinalDemo{
        public static void main(String[] args) {
                Demo d = new Demo();
                System.out.println(d.tenure);
                System.out.println(d.calculateEMI());
```

```
        }
}
```

OUTPUT

```
Main.java:11: error: cannot inherit from final Demo
  class Dummy extends Demo{
                      ^
1 error
```

STATIC MODIFIER

```
class Account{
   static int minbalance;  //class variable

   static{
      minbalance = 500;   // static block
   }

   public static int getMinimumBalance(){
      return minbalance;  //can't use instance variable in static method
                  //and block
   }

   public static void main (String[] args) {
      System.out.println("The value.." + getMinimumBalance());
   }
}
```

OUTPUT:

```
The value..500
```

VARIABLE ARGUMENTS

```
class Employee{
    private String employeeId;
    Employee(String employeeId){
      this.employeeId=employeeId;
    }
    public int reward(double...fixedDeposit){    //Variable argument
    double sum=0;
    int rewardPoint=0;
    for(double deposit:fixedDeposit){
      sum=sum+deposit;
    }
    if(sum>1000000){
      rewardPoint=20000;
    }
    else if(sum<1000000 && sum>=500000){
      rewardPoint=10000;
    }
    else{
      rewardPoint = 20000;
    }
    return rewardPoint;
    }
    public String getEmployeeId(){
      return employeeId;
    }
}

class Execute{
  public static void main(String[] args){
    Employee employee1=new Employee("E1001");
    int rewardPoint=employee1.reward(100000,200000,300000);
```

```
    Employee employee2=new Employee("E1002");
    int rewardPoint1=employee2.reward(100000,100000);
    System.out.println(employee1.getEmployeeId() +" has got a reward of "+rewardPoint);
    System.out.println(employee2.getEmployeeId() +" has got a reward of "+rewardPoint1);
  }
}
```

OUTPUT

```
E1001 has got a reward of 10000
E1002 has got a reward of 20000
```

ENUMARATED DATA TYPE

```
enum Day{
    SUNDAY(1),MONDAY(2),TUESDAY(3),WEDNESDAY(4),THURSDAY(5),FRIDAY(6),SATURDAY(7);
    private int value;
    private Day(int value){
      this.value=value;
    }
    public int getValue(){
      return this.value;
    }
  }
 class UserInterface{
    public static void main (String[] args) {
       //printing all constants of an enum
        for(Day day:Day.values())
            System.out.println("Day:"+day.name()+" Value:"+day.getValue());
    }
  }
```

OUTPUT

```
Day:SUNDAY Value:1
Day:MONDAY Value:2
Day:TUESDAY Value:3
Day:WEDNESDAY Value:4
Day:THURSDAY Value:5
Day:FRIDAY Value:6
Day:SATURDAY Value:7
```

ABSTRACT CLASS

```
abstract class Branch{
   public abstract boolean validatePhotoProof(String proof);
   public abstract boolean validateAddressProof(String proof);
   public void openAccount(String photoProof,String addressProof,int amount){
    if(amount>=1000){
     if(validateAddressProof(addressProof) && validatePhotoProof(photoProof)){
      System.out.println("Account opened");
     }
     else{
      System.out.println("cannot open account");
     }
    }
    else{
     System.out.println("cannot open account");
    }
   }
 }

class MumbaiBranch extends Branch{
   public boolean validatePhotoProof(String proof){
    if(proof.equalsIgnoreCase("pan card")){
     return true;
```

```
    }
     return false;
   }
   public boolean validateAddressProof(String proof){
     if(proof.equalsIgnoreCase("ration card")){
       return true;
     }
     return false;
   }
 }

class Execute{
  public static void main(String[] args){
    Branch mumbaiBranch=new MumbaiBranch();
    mumbaiBranch.openAccount("pan card","ration card",2000);
  }
}
```

OUTPUT:

```
Account opened
```