CLASSES AND OBJECTS IN JAVA

WHAT IS OOP?

- OOP stands for **Object-Oriented Programming**.
- Procedural programming is about writing procedures or methods that perform operations on the data.
- object-oriented programming is about creating objects that contain both data and methods.
- Everything in Java is associated with classes and objects, along with its attributes and methods.

WHAT ARE CLASSES AND OBJECTS?

CLASS	OBJECTS
Fruit	Apple Banana Mango

CLASS	OBJECTS
Car	Volvo Audi Toyota

So, a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the variables and methods from the class.

Comparison of OOPS with other programming styles with help of an Example

Let's understand with example how OOPs is different than other programming approaches.

Programming languages can be classified into 3 primary types

Unstructured Programming Languages: The most primitive of all programming languages having sequentially flow of control. Code is repeated through out the program.

Structured Programming Languages: Has non-sequentially flow of control. Use of functions allows for re-use of code.

Object Oriented Programming: Combines Data & Action Together.

Let's understand these 3 types with an example.

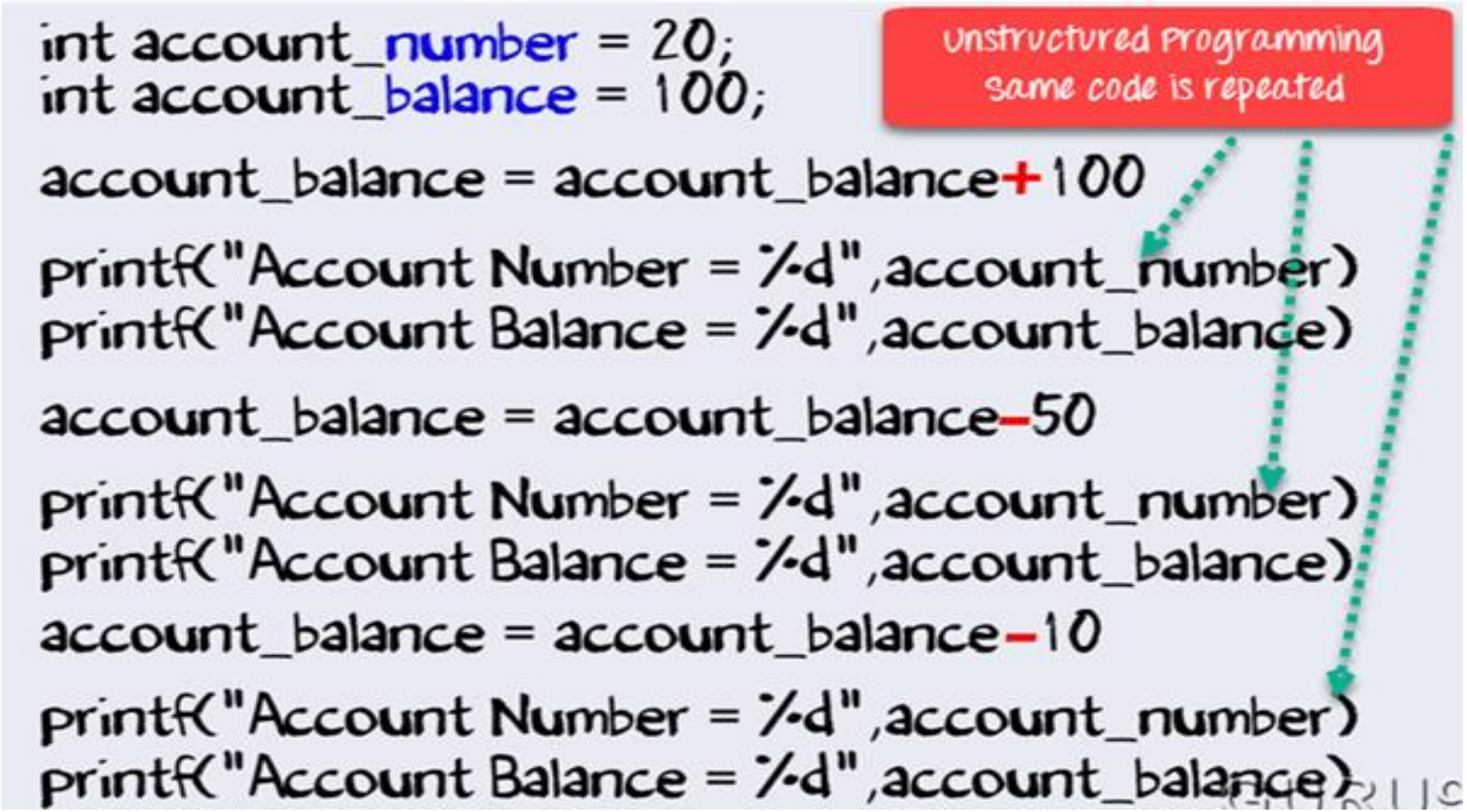
Suppose you want to create a Banking Software with functions like

1. Deposit
2. Withdraw
3. Show Balance

UNSTRUCTURED PROGRAMMING LANGUAGES

The earliest of all programming language were unstructured programming language. A very elementary code of banking application in unstructured Programming language will have two variables of one account number and another for account balance

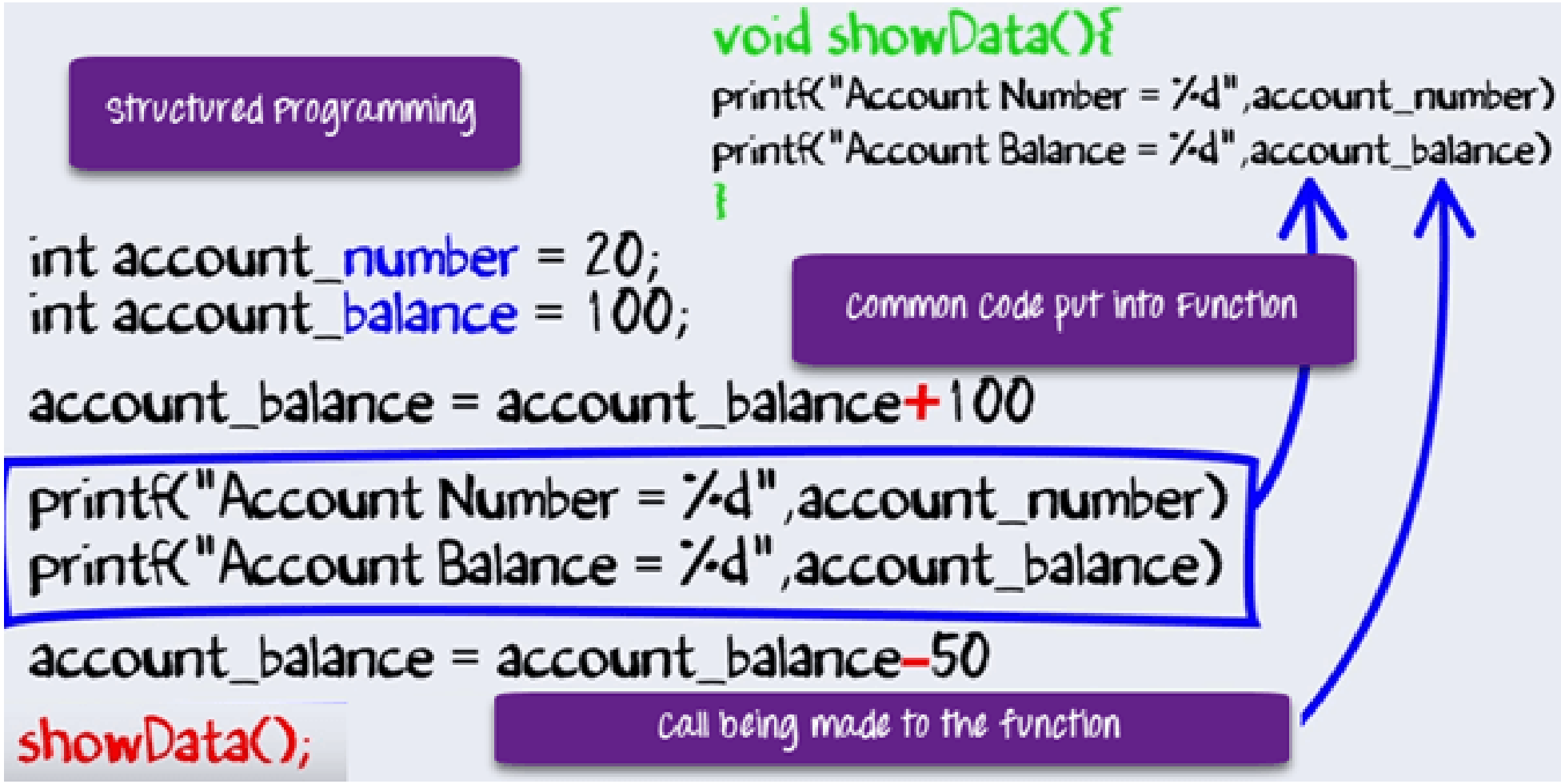
UNSTRUCTURED PROGRAMMING LANGUAGES



STRUCTURED PROGRAMMING

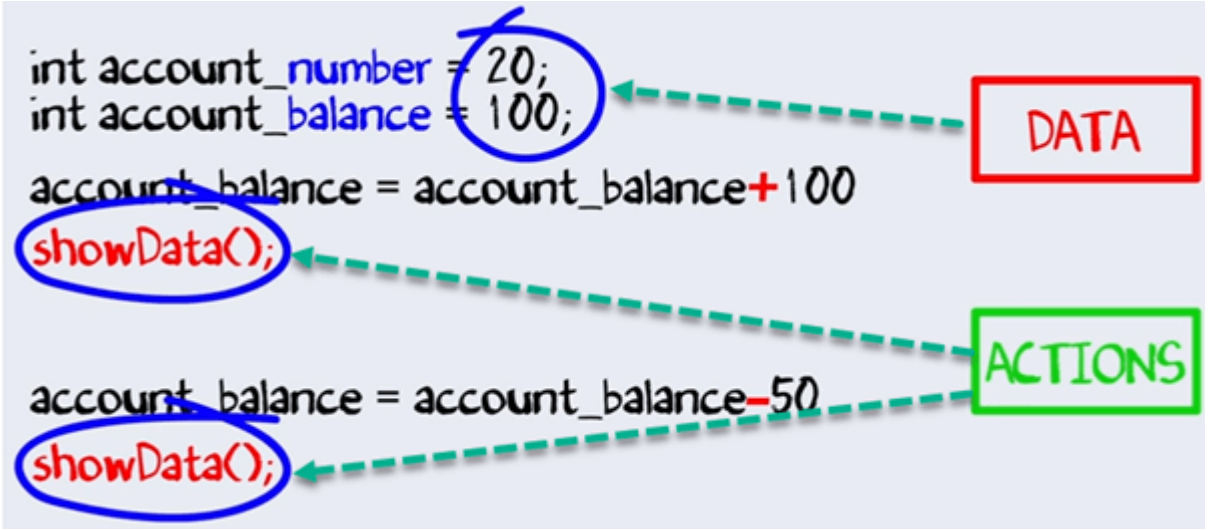
With the arrival of Structured programming repeated lines on the code were put into structures such as functions or methods. Whenever needed, a simple call to the function is made.

STRUCTURED PROGRAMMING



OBJECT-ORIENTED PROGRAMMING

- In our program, we are dealing with data or performing specific operations on the data.
- In fact, having data and performing certain operation on that data is very basic characteristic in any software program.
- Experts in Software Programming thought of combining the Data and Operations. Therefore, the birth of Object Oriented Programming which is commonly called OOPS.
- The same code in OOPS will have same data and some action performed on that data.



OBJECT-ORIENTED PROGRAMMING

```
class Account
{
    int account_number;
    int account_balance;
    public void showdata()
   {
    system.out.println("Account Number"+account_number)
    system.outprintln("Account Balance"+ account_balance)
   }
}
```

By combining data and action, we will get many advantages over structural programming viz,

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

JAVA CLASSES/OBJECTS

- Java is an object-oriented programming language.
- Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.
- A Class is like an object constructor, or a "blueprint" for creating objects.

```
class <class_name>
{
    field;
    method;
 }
```

```
ClassName ReferenceVariable = new ClassName();
```

JAVA CLASSES/OBJECTS

```
class MyClass
{
  int x = 5;
}
```

In Java, an object is created from a class. We have already created the class named MyClass, so now we can use this to create objects.

To create an object of MyClass, specify the class name, followed by the object name, and use the keyword new:

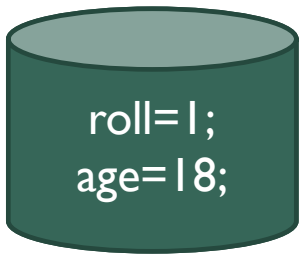
```
class MyClass
{
  int x = 5;
  public static void main(String[] args)
  {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
   }
}
```

Create two objects of `MyClass`:

```
class MyClass
{
    int x = 5;
    public static void main(String[] args)
    {
            MyClass myObj1 = new MyClass(); // Object 1
            MyClass myObj2 = new MyClass(); // Object 2
            System.out.println(myObj1.x);
            System.out.println(myObj2.x);
     }
}
```

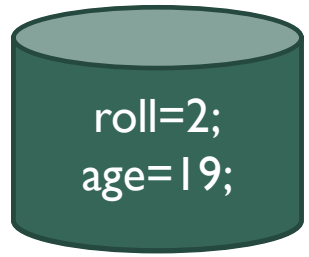
```
class Student
{
  int roll;
  int age;
}

Student obj1;
obj1=new Student();
Student obj2=new Student ();
```



obj1

obj1.roll=1;
obj1.age=18;



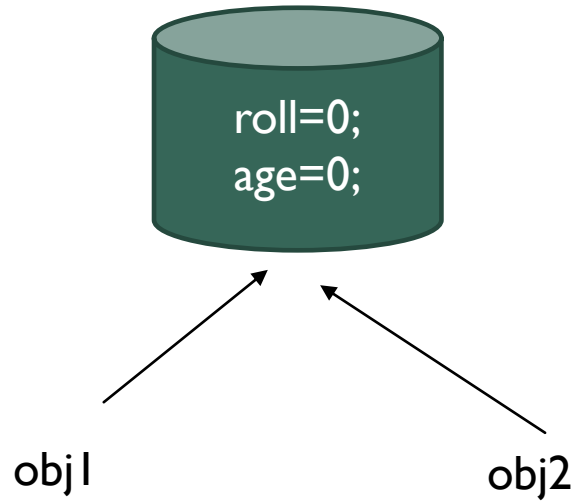
obj2

obj2.roll=2;
obj2.age=19;

```
class Student
{
  int roll;
  int age;
}

Student obj1=new Student ();

Student obj2=obj1;
```



Using Multiple Classes

You can also create an object of a class and access it in another class. This is often used for better organization of classes (one class has all the attributes and methods, while the other class holds the main() method (code to be executed)).

Remember that the name of the java file should match the class name. In this example, we have created two files in the same directory/folder:

MyClass.java

OtherClass.java

MyClass.java

```
class MyClass
{
  int x = 5;
}
```

OtherClass.java

```
class OtherClass
{
  public static void main(String[] args)
  {
      MyClass myObj = new MyClass();
      System.out.println(myObj.x);
  }
}
```

When both files have been compiled:

C:\Users\Your Name>javac MyClass.java
C:\Users\Your Name>javac OtherClass.java
Run the OtherClass.java file:

C:\Users\Your Name>java OtherClass

JAVA CLASS ATTRIBUTES

class attributes are variables within a class:

Another term for class attributes is **fields**.

```
class MyClass
{
  int x = 5;
  int y = 3;
}
```

You can access attributes by creating an object of the class, and by using the dot syntax (.):

```
class MyClass
{
  int x = 5;
  public static void main(String[] args)
  {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
  }
}
```

MODIFY ATTRIBUTES

```
class MyClass
{
  int x;
  public static void main(String[] args)
  {
    MyClass myObj = new MyClass();
    myObj.x = 40;
    System.out.println(myObj.x);
  }
}
```

```
public class MyClass
{
  final int x = 10;

  public static void main(String[] args)
  {
    MyClass myObj = new MyClass();
    myObj.x = 25;
    // will generate an error: cannot assign a
    value to a final variable

    System.out.println(myObj.x);
  }
}
```

The final keyword is useful when you want a variable to always store the same value, like PI (3.14159...).

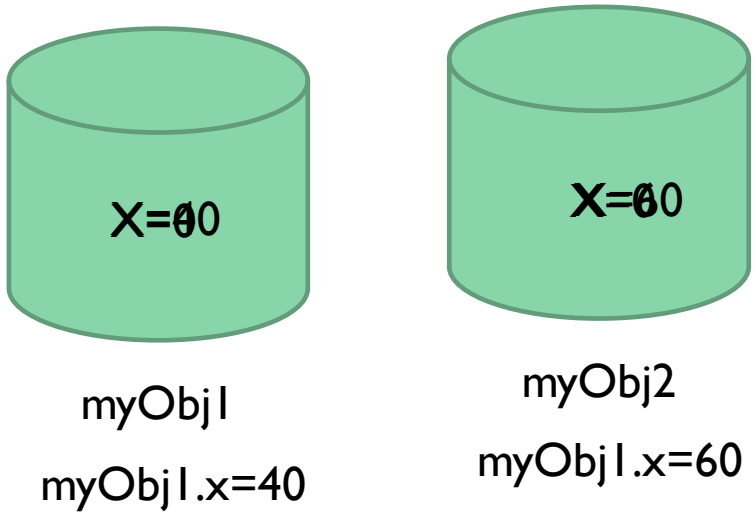
The final keyword is called a "modifier".

EXAMPLE

```
class MyClass
{
  int x;
  public static void main(String[] args)
  {
    MyClass myObj1 = new MyClass();
    myObj1.x = 40;
    System.out.println(myObj1.x);

    MyClass myObj2 = new MyClass();
    myObj2.x = 60;
    System.out.println(myObj2.x);

  }
}
```



JAVA CLASS METHODS

```
class MyClass
{
  static void myMethod()
 {
    System.out.println("Hello World!");
 }
  public static void main(String[] args)
 {
    myMethod();
  }
}

// Outputs "Hello World!"
```

STATIC VS. NON-STATIC

```
class MyClass
{
  // Static method
  static void myStaticMethod()
  {
    System.out.println("Static methods can be called
                        without creating objects");
  }

  // Public method
  public void myPublicMethod()
  {
    System.out.println("Public methods must be called
                        by creating objects");
  }

// Main method
public static void main(String[] args)
{
    myStaticMethod(); // Call the static method
    myPublicMethod(); //This would compile an error

    MyClass myObj = new MyClass(); // Create an object of
                                          MyClass
    myObj.myPublicMethod(); // Call the public method on the
                                       object
  }
}
```

USING MULTIPLE CLASSES

Car.java

```
class Car
{
  public void fullThrottle()
  {
    System.out.println("The car is going as fast as it can!");
  }

  public void speed(int maxSpeed)
  {
    System.out.println("Max speed is: " + maxSpeed);
  }
}
```

```
The car is going as fast as it can!
Max speed is: 200
```

OtherClass.java

```
class OtherClass
{
  public static void main(String[] args)
  {
    Car myCar = new Car();      // Create a myCar object

    myCar.fullThrottle();       // Call the fullThrottle() method

    myCar.speed(200);           // Call the speed() method
  }
}
```

When both files have been compiled:

```
C:\Users\Your Name>javac Car.java
C:\Users\Your Name>javac OtherClass.java
```

Run the OtherClass.java file:

```
C:\Users\Your Name>java OtherClass
```

```
class Test
{
    int a,b,c;
    void getData(int x, int y)
    {
        a=x;
        b=y;
    }
    void add()
    {
        c=a+b;
        System.out.println(“sum=“+c);
    }
}
```

```
class MainTest
{
    public static void main(String args[])
    {
        Test obj=new Test();
        obj.getData(10,20);
        obj.add();
    }
}
```

```
class Rectangle
{
    int length, width;
    void getData(int x, int y)
    {
        length=x;
        width=y;
    }
    int rectArea()
    {
      int area=length*width;
      return(area);
    }
}
```

```
class RectangleMain
{
    int area1,area2;
    Rectangle rect1=new Rectangle();
     Rectangle rect2=new Rectangle();

    rect1.length=15;
    rect1.width=10;

     area1=rect1.length*rect1.width;

      rect2.getData(20,12);
      area2=rect2.rectArea();

    System.out.println(“Area1=”+area1);
    System.out.println(“Area1=”+area1);

}
```

3 WAYS TO INITIALIZE OBJECT

There are 3 ways to initialize object in Java.

- By reference variable
- By method
- By constructor

OBJECT AND CLASS EXAMPLE: INITIALIZATION THROUGH REFERENCE

Initializing an object means storing data into the object. Let's see a simple example where we are going to initialize the object through a reference variable.

```
class Student
{
 int id;
 String name;
}
class TestStudent2
{
 public static void main(String args[])
 {
  Student s1=new Student();
  s1.id=101;
  s1.name="Sonoo";
  System.out.println(s1.id+" "+s1.name);//printing members with a white space
 }
}
```

OBJECT AND CLASS EXAMPLE: INITIALIZATION THROUGH METHOD

```
class Student
{
    int rollno;
    String name;
    void insertRecord(int r, String n)
    {
        rollno=r;
        name=n;
    }
    void displayInformation()
    {
        System.out.println(rollno+" "+name);
    }
}
```

```
class TestStudent4
{
    public static void main(String args[])
    {
        Student s1=new Student();
        Student s2=new Student();

        s1.insertRecord(111,"Karan");
        s2.insertRecord(222,"Aryan");

        s1.displayInformation();
        s2.displayInformation();
    }
}
```

```
class Account
{
    int acc_no;
    String name;
    float amount;

    //Method to initialize object
    void insert(int a,String n,float amt)
    {
        acc_no=a;
        name=n;
        amount=amt;
    }

    //deposit method
    void deposit(float amt)
    {
        amount=amount+amt;
        System.out.println(amt+" deposited");
    }
```

```
//withdraw method
void withdraw(float amt)
{
    if(amount<amt)
    {
        System.out.println("Insufficient Balance");
    }
    else
    {
        amount=amount-amt;
        System.out.println(amt+" withdrawn");
    }
}


//method to check the balance of the account
void checkBalance()
{
    System.out.println("Balance is: "+amount);
}
```

```
//method to display the values of an object
void display()
{
    System.out.println(acc_no+" "+name+" "+amount);}
}

//Creating a test class to deposit and withdraw amount
class TestAccount
{
    public static void main(String[] args)
    {
        Account a1=new Account();
        a1.insert(832345,"Ankit",1000);
        a1.display();
        a1.checkBalance();
        a1.deposit(40000);
        a1.checkBalance();
        a1.withdraw(15000);
        a1.checkBalance();
    }
}
```

ARRAY OF OBJECTS

- As we all know, the Java programming language is all about objects as it is an object-oriented programming language.

- If you want to store a single object in your program, then you can do so with the help of a variable of type object. But when you are dealing with numerous objects, then it is advisable to use an array of objects.

```
ClassName obj[]= new ClassName[array_length]
```

```
class Employee
{
  int empId;
  String name;

  public void setData(inteid, String n)
  {
     empId = eid;
     name = n;
  }
  public void showData()
  {
   System.out.print("EmpId = "+empId + "   " + " Employee Name = "+name);
   System.out.println();
 }
}
```

```
class Main
{
   public static void main(String args[])
   {
          Employee[] obj = new Employee[2] ;
          obj[0] = new Employee();
          obj[1] = new Employee();

          obj[0].setData(1,"Smith");
          obj[1].setData(2,"John");

          System.out.println("Employee Object 1:");
          obj[0].showData();
          System.out.println("Employee Object 2:");
          obj[1].showData();
   }
}
```