What is an Object

An object has **identity** (it acts as a single whole).

An object has **state** (it has various properties, which might change).

An object has **behavior** (it can do things and can have things done to it).

Software Objects

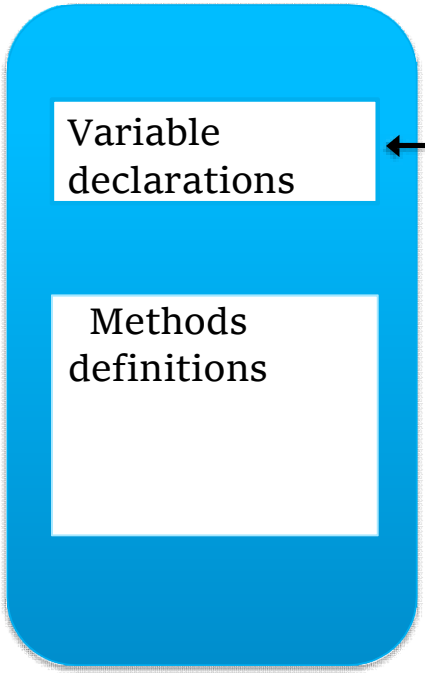
Software objects have **identity** because each is a separate chunk of memory

Software objects have **state.** Some of the memory that makes a software object is used for variables which contain values.

Software objects have **behavior.** Some of the memory that makes a software object is used to contain programs (called *methods*) that enable the object to "do things." The object does something when one of its method runs.

Classes

A class contains variable declarations and method definitions



Variable declarations

(variable describes the attributes)

Variable may be: instance variables or static variables or final variables

Method definitions

(methods handle the behavior)

Methods may be: instance methods or static methods

Try it

```
class First
 {
  int a=10;
  void disp()
   {
       System.out.println(" Value of a is :"+a);
   }
  public static void main(String args[])
   {
       System.out.println(a);
       disp();
   }
 }
```

Instantiation

- Once a class is defined, you can declare a variable (**object reference**) of type class

  Student stud1;
  Employee emp1;

- The **new** operator is used to create an object of that reference type

  Employee emp = new Employee();

  Creating an object is called **instantiation**.

Objects and References

- The **new** operator,

  Dynamically allocates memory for an object

  Creates the object on the heap

  Returns a reference to it

  The reference is then stored in the variable

Try it

```
class First
 {
  int a=10;
  void disp()
   {
        System.out.println(" Value of a is :"+a);
   }
  public static void main(String args[])
   {
        First fob=new First();
        System.out.println(fob.a);
        fob.disp();
   }
 }
```

Try it


```
class First
  {
    int a=0;
    void update(int value)
     {
       a=value;
     }
    void disp()
     {
           System.out.println("Value of a is :"+a);
     }
```

```
public static void main(String args[])
    {
          First fob=new First();
          System.out.println("Initial value is " +fob.a);
          fob.update(10);
          fob.disp();
    }
 }
```

Defining a Class in java

Define an Employee class with instance variables and instance methods

```
class Employee{
   int id;
   String name;
   int salary;

  void setId(int i)
   {
     id = i;
   }

 void setName(String n) {
     name = n;
  }

void setSalary(int s)
 {
    salary = s;
 }
void getEmployeeDetails( ) {
  System.out.println (name + " salary is  " + salary);
  }
}
```

return

A **return statement** causes the program control to transfer back to the caller of a method.

Every method in Java is declared with a return type and it is mandatory for all java methods.

A return type may be a **primitive type** like i**nt, float, double,** a **reference type** or **void type**(returns nothing).

```
String getName()
 {
   return name;
 }
```

return

```
class First
 {
  int a=100;
  int getValue()
   {
    return a;
   }
  public static void main(String args[])
   {
    First fob=new First();
    System.out.println(fob.getValue());
   }
 }
```

return


```
class Account
 {
   double balance;
   void addBalance(double value)
    {
          balance = balance+value;
    }
   double getBalance()
    {
     return balance;
    }
```

```
public static void main(String args[])
   {
     Account acc=new Account();
     acc.addBalance(1000);
     acc.addBalance(acc.getBalance()*2);
     System.out.println(acc.getBalance());
   }
}
```

return

Add the Function **double withdraw(**double **Amount)**

- **Minimum Balance** to be maintained is 5000. If the balance after withdrawal is more than Minimum Balance the Amount is retuned
- If the Amount to be withdrawn is greater than the Balance in the Account or violates the Minimum Balance after withdrawal, it will return -1

Member variables

A variable declared within a class(outside any method) is known as an **instance variable**.

A variable declared within a method is known as **local variable**.

Variables with method declarations are known as **parameters or arguments**.

A class variable can also be declared as static where as a local variable cannot be static.

Employee class - Example

```
class Employee
 {
    int id;
    String name;
    int salary;

    void setId(int no){
        id = no;
      }

    public static void main(String[] args)
      {
         Employee emp1 = new Employee();
         emp1.setId(101);
       }
 }
```

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)).

```
class MainClass
 {
  public static void main(String args[])
   {
        subClass sc=new subClass();
        sc.disp();
   }
 }
```

```
class subClass
 {
  String msg="Hello Welcome to Classes";
  void disp()
   {
        System.out.println(msg);
   }
 }
```

Using Multiple Classes

```
class subClass
 {
   String msg="Hello ";
   String myname;
   void disp(String name)
    {
         myname=name;
         System.out.println(msg + name);
    }
 }
```


```
class MainClass
 {
   public static void main(String args[])
    {
         subClass sc1=new subClass();
         subClass sc2=new subClass();
         subClass sc3=new subClass();
         sc1.disp("Sai Kiran");
         sc2.disp("Nandan");
         sc3.disp("Vamsi");
         System.out.println(sc1.myname +","+
         sc2.myname +","+sc3.myname);
    }
 }
```

Constructors

- While designing a class, the class designer can define within the class, a special method called ‘constructor’
- Constructor is automatically invoked whenever an object of the class is created
- Rules to define a constructor
  - A constructor has the same name as the class name
  - A constructor should not have a return type
  - A constructor can be defined with any access specifier (like private, public)
  - A class can contain more than one constructor, So it can be overloaded

Constructor - Example

```
class Constructor
 {
  Constructor()
   {
        System.out.println("This is Default
        Constructor");
   }
  Constructor(int a)
   {
        System.out.println("This is Constructor with
        One Argument "+a);
   }
 }
```

```
class Main
 {
  public static void main(String args[])
   {
        Constructor c1=new Constructor();
        Constructor c2=new Constructor(100);
   }
 }
```

What Happens???

```
class Number
 {
  int no=0;
  int addNo(int no)
   {
      no=no+no;
      return(no);
   }
 }
```

```
class Main
 {
  public static void main(String args[])
   {
      Number n1=new Number();
      int res=n1.addNo(10);
      System.out.println(res);
   }
 }
```

this

- Each class member function contains an implicit reference of its class type, named **this**
- this reference is created automatically by the compiler
- It contains the address of the object through which the function is invoked
- Use of this keyword
  - this can be used to refer instance variables when there is a clash with local variables or method arguments
  - this can be used to call overloaded constructors from another constructor of the same class

this

- this can be used to refer instance variables when there is a clash with local variables or method arguments

```
class Number
 {
  int no=0;
  int addNo(int no)
   {
      this.no=this.no+no;
      return(no);
   }
 }
```

```
class Main
 {
  public static void main(String args[])
   {
      Number n1=new Number();
      int res=n1.addNo(10);
      System.out.println(res);
   }
}
```

this

- this can be used to call overloaded constructors from another constructor of the same class

```
class Sample{
Sample(){
    this("Java"); // calls overloaded constructor
      System.out.println("Default constructor ");
   }
Sample(String str){
   System.out.println("One argument constructor "+ str);
   }
}
```

```
class Main
 {
   public static void main(String args[])
     {
          Sample a1=new Sample();
      }
 }
```

Static Class Members

- Static class members are the members of a class that do not belong to an instance of a class
- We can access static members directly by prefixing the members with the class name

  ClassName.staticVariable

  ClassName.staticMethod(...)

Static Class Members

Static variables:

- Shared among all objects of the class
- Only one copy exists for the entire class to use
- Stored within the class code, separately from instance variables that describe an individual object
- Public static final variables are global constants

Static variables

```
class Number
 {
   static int a=1;
   void add(int num)
    {
         a=a+num;
    }
 }
```


```
class Main
 {
   public static void main(String args[])
    {
         Number n1=new Number();
         n1.add(10);
         Number n2=new Number();
         n2.add(20);
         System.out.println(n1.a);
         System.out.println(n2.a);

    }
 }
```

Static Class Members

Static methods:

- Static methods can only access directly the static members and manipulate a class’s static variables
- Static methods cannot access non-static members(instance variables or instance methods) of the class
- Static method can't access this and super references

Quiz

```
class Sample{
  int i_val;
    public static void main(String[] xyz)
     {
        System.out.println("i_val is :"+this.i_val);
     }
    }
```

Quiz

```
class Sample{
   int i_val=10;
   Sample(int i_val){
      this.i_val=i_val;
      System.out.println("inside Sample
      i_val: "+this.i_val);
   }
   public static void main(String[] xyz)
   {
      Sample o = new Sample();
   }
   }
```

Static block

- A static block is a block of code enclosed in braces, preceded by the keyword static

Ex :

```
static {
  System.out.println("Within static block");
  }
```

- The statements within the static block are executed automatically when the class is loaded into JVM

Static block

- A class can have any number of static blocks and they can appear anywhere in the class
- They are executed in the order of their appearance in the class
- JVM combines all the static blocks in a class as single static block and executes them
- You can invoke static methods from the static block and they will be executed as and when the static block gets executed

Example

```
class StaticBlockExample {
  StaticBlockExample() {
    System.out.println("Within constructor");
    }
   static {
     System.out.println("Within 1st static block");
    }
   static void m1() {
      System.out.println("Within static m1 method");
     }
   static {
      System.out.println("Within 2nd static block");
      m1();
     }
public static void main(String [] args) {
System.out.println("Within main");
StaticBlockExample x = new StaticBlockExample();
}
static
  {
    System.out.println("Within 3rd static block");
  }
}
```

Do iT

Create a class Box that uses a parameterized method to initialize the dimensions of a box.(dimensions are width, height, depth of double type).

The class should have a method that can return volume.

Obtain an object and print the corresponding volume in main() function.

Do iT

Create a new class called “Calculator” which contains the following:

1. A static method called **powerInt(int num1,int num2)** that accepts two integers and returns num1 to the power of num2 (num1 power num2).

2. A static method called **powerDouble(double num1,int num2)** that accepts one double and one integer and returns num1 to the power of num2 (num1 power num2).

3. Call your method from another class without instantiating the class (i.e. call it like Calculator.powerInt(12,10) since your methods are defined to be static)

Hint: Use Math.pow(double,double) to calculate the power.

Do iT

Design a class that can be used by a health care professional to keep track of a patient's vital statistics. Here's what the class should do:

1. Construct a class called Patient

2. Store a String name for the patient

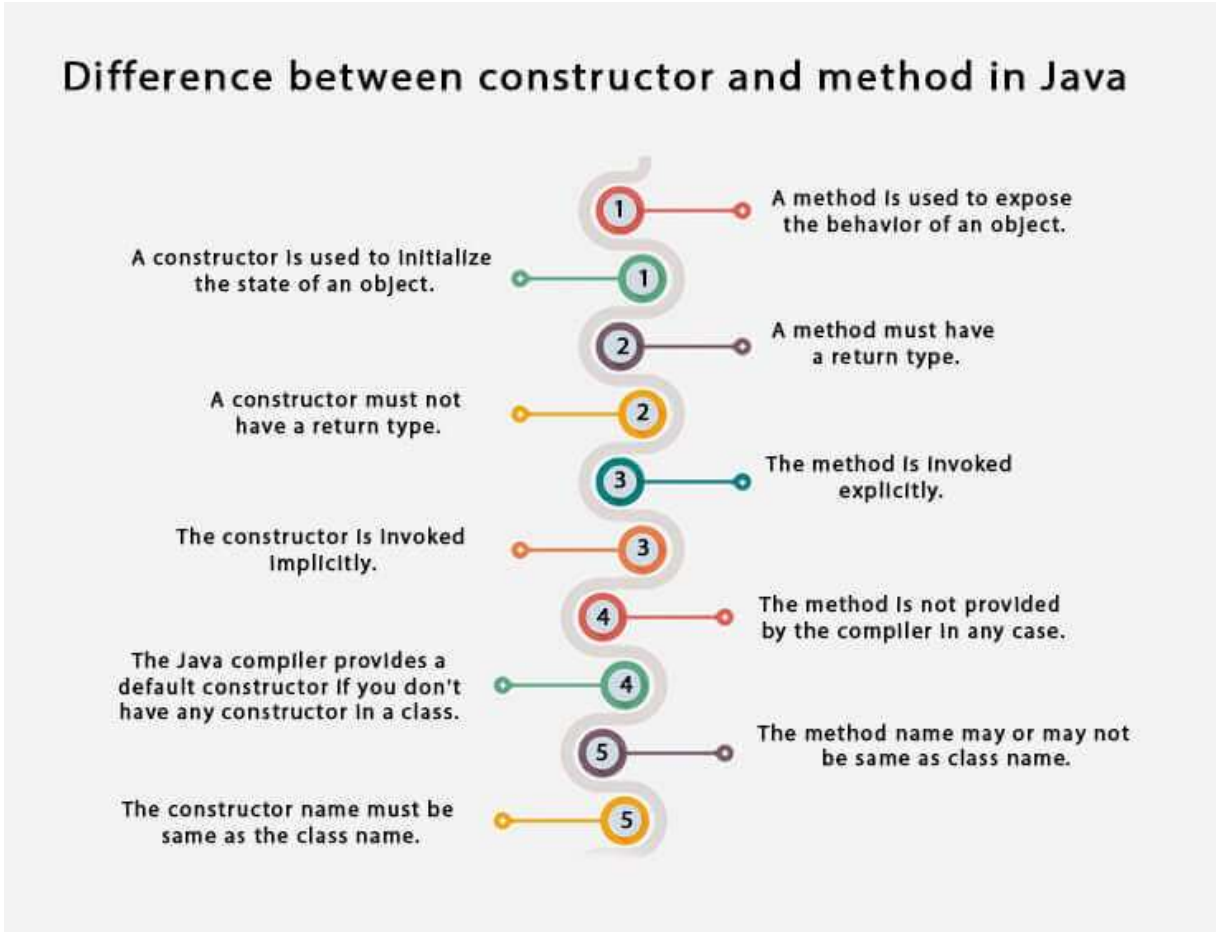
3. Store weight and height for patient as doubles

4. Construct a new patient using these values

5. Write a method called BMI which returns the patient's BMI as a double. BMI can be calculated as BMI = (Weight in Pounds / (Height in inches x Height in inches)) x 703

6. Next, construct a class called "Patients" and create a main method. Create a Patient object and assign some height and weight to that object. Display the BMI of that patient.

Constructors Vs Methods



Encapsulation

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.

In encapsulation, the variables of a class will be hidden from other classes and can be **accessed only through the methods** of their current class.

Therefore, it is also known as **data hiding**

Try it

```
class First
 {
   int a=10;
   void disp()
    {
        System.out.println(a);
    }
 }
```

```
class Main
 {
   public static void main(String args[])
    {
        First fob=new First();
        System.out.println(fob.a);
        fob.disp();
    }
 }
```

Try it

```
class First
 {
   int a=10;
   void disp()
    {
         System.out.println(a);
    }
   int geta()
    {
      return a;
    }
 }
```


```
class Main
 {
   public static void main(String args[])
    {
          First fob=new First();
          System.out.println(fob.geta());
          fob.disp();
    }
 }
```

Try it

```
class First
 {
   int a=10;
   void disp()
    {
         System.out.println(a);
    }
   int geta()
    {
      return a;
    }
 }
```


```
class Main
 {
   public static void main(String args[])
    {
          First fob=new First();
          System.out.println(fob.geta());
          fob.disp();
          System.out.println(fob.a);
   }
 }
```

Still Accessible

Access Specifiers

- Java provides access specifiers to control access to class members
- Access specifiers help implement Encapsulation by hiding implementation-level details in a class
- The **private** access specifier is generally used to encapsulate or hide the member data in the class
- The **public** access specifier is used to expose the member functions as interfaces to the outside world

Try it

```
class First
 {
   private int a=10;
   void disp()
    {
         System.out.println(a);
    }
   int geta()
    {
      return a;
    }
 }
```


```
class Main
 {
   public static void main(String args[])
    {
          First fob=new First();
          System.out.println(fob.geta());
          fob.disp();
          System.out.println(fob.a);
    }
 }
```

Now ???????

Quiz

```
class T {
  int t = 20;
  T() {
    t = 40;
  }
}
class Main {
   public static void main(String args[])
  {
      T t1 = new T();
      System.out.println(t1.t);
   }
}
```

Quiz

Is there any compiler error in the below Java program?

```
class Point {
    int m_x, m_y;
    public Point(int x, int y)
     {
        m_x = x;
        m_y = y;
      }
    public static void main(String args[])
    {
      Point p = new Point();
    }
}
```

Encapsulation

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.

In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class.

Therefore, it is also known as **data hiding**

Encapsulation

To achieve encapsulation in Java −

- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.

Software Objects

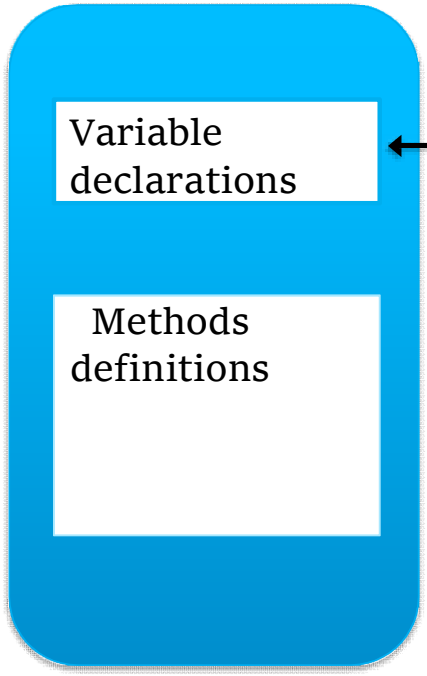
Software objects have **identity** because each is a separate chunk of memory

Software objects have **state.** Some of the memory that makes a software object is used for variables which contain values.

Software objects have **behavior.** Some of the memory that makes a software object is used to contain programs (called *methods*) that enable the object to "do things." The object does something when one of its method runs.

Classes

A class contains variable declarations and method definitions



Variable declarations

(variable describes the attributes)

Variable may be: instance variables or static variables or final variables

Method definitions

(methods handle the behavior)

Methods may be: instance methods or static methods

Try it

```
class First
 {
  int a=10;
  void disp()
   {
       System.out.println(" Value of a is :"+a);
   }
  public static void main(String args[])
   {
       System.out.println(a);
       disp();
   }
 }
```

Instantiation

- Once a class is defined, you can declare a variable (**object reference**) of type class

  Student stud1;
  Employee emp1;

- The **new** operator is used to create an object of that reference type

  Employee emp = new Employee();

  Creating an object is called **instantiation**.

Objects and References

- The **new** operator,

  Dynamically allocates memory for an object

  Creates the object on the heap

  Returns a reference to it

  The reference is then stored in the variable

Try it

```
class First
 {
  int a=10;
  void disp()
   {
        System.out.println(" Value of a is :"+a);
   }
  public static void main(String args[])
   {
        First fob=new First();
        System.out.println(fob.a);
        fob.disp();
   }
 }
```

Try it


```
class First
  {
    int a=0;
    void update(int value)
     {
       a=value;
     }
    void disp()
     {
           System.out.println("Value of a is :"+a);
     }
```

```
public static void main(String args[])
    {
          First fob=new First();
          System.out.println("Initial value is " +fob.a);
          fob.update(10);
          fob.disp();
    }
 }
```

Defining a Class in java

Define an Employee class with instance variables and instance methods

```
class Employee{
   int id;
   String name;
   int salary;

  void setId(int i)
   {
     id = i;
   }

 void setName(String n) {
     name = n;
  }

void setSalary(int s)
 {
    salary = s;
 }
void getEmployeeDetails( ) {
  System.out.println (name + " salary is  " + salary);
  }
}
```

return

A **return statement** causes the program control to transfer back to the caller of a method.

Every method in Java is declared with a return type and it is mandatory for all java methods.

A return type may be a **primitive type** like i**nt, float, double,** a **reference type** or **void type**(returns nothing).

```
String getName()
 {
   return name;
 }
```

return

```
class First
 {
  int a=100;
  int getValue()
   {
     return a;
   }
  public static void main(String args[])
   {
     First fob=new First();
     System.out.println(fob.getValue());
   }
 }
```

return


```
class Account
 {
   double balance;
   void addBalance(double value)
    {
          balance = balance+value;
    }
   double getBalance()
    {
     return balance;
    }
```

```
public static void main(String args[])
   {
     Account acc=new Account();
     acc.addBalance(1000);
     acc.addBalance(acc.getBalance()*2);
     System.out.println(acc.getBalance());
   }
}
```

return

Add the Function **double withdraw(**double **Amount)**

- **Minimum Balance** to be maintained is 5000. If the balance after withdrawal is more than Minimum Balance the Amount is retuned
- If the Amount to be withdrawn is greater than the Balance in the Account or violates the Minimum Balance after withdrawal, it will return -1

Member variables

A variable declared within a class(outside any method) is known as an **instance variable**.

A variable declared within a method is known as **local variable**.

Variables with method declarations are known as **parameters or arguments**.

A class variable can also be declared as static where as a local variable cannot be static.

Employee class - Example

```
class Employee
 {
    int id;
    String name;
    int salary;

    void setId(int no){
        id = no;
      }

    public static void main(String[] args)
      {
         Employee emp1 = new Employee();
         emp1.setId(101);
       }
 }
```

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)).

```
class MainClass
 {
  public static void main(String args[])
   {
        subClass sc=new subClass();
        sc.disp();
   }
 }
```

```
class subClass
 {
  String msg="Hello Welcome to Classes";
  void disp()
   {
        System.out.println(msg);
   }
 }
```

Using Multiple Classes

```
class subClass
 {
  String msg="Hello ";
  String myname;
  void disp(String name)
   {
        myname=name;
        System.out.println(msg + name);
   }
 }
```


```
class MainClass
 {
  public static void main(String args[])
   {
        subClass sc1=new subClass();
        subClass sc2=new subClass();
        subClass sc3=new subClass();
        sc1.disp("Sai Kiran");
        sc2.disp("Nandan");
        sc3.disp("Vamsi");
        System.out.println(sc1.myname +","+
        sc2.myname +","+sc3.myname);
   }
}
```

Constructors

- While designing a class, the class designer can define within the class, a special method called ‘constructor’
- Constructor is automatically invoked whenever an object of the class is created
- Rules to define a constructor
  - A constructor has the same name as the class name
  - A constructor should not have a return type
  - A constructor can be defined with any access specifier (like private, public)
  - A class can contain more than one constructor, So it can be overloaded

Constructor - Example

```
class Constructor
 {
  Constructor()
   {
        System.out.println("This is Default
        Constructor");
   }
  Constructor(int a)
   {
        System.out.println("This is Constructor with
        One Argument "+a);
   }
 }
```

```
class Main
 {
  public static void main(String args[])
   {
        Constructor c1=new Constructor();
        Constructor c2=new Constructor(100);
   }
 }
```

What Happens???

```
class Number
 {
  int no=0;
  int addNo(int no)
   {
      no=no+no;
      return(no);
   }
 }
```

```
class Main
 {
  public static void main(String args[])
   {
      Number n1=new Number();
      int res=n1.addNo(10);
      System.out.println(res);
   }
 }
```

this

- Each class member function contains an implicit reference of its class type, named **this**
- this reference is created automatically by the compiler
- It contains the address of the object through which the function is invoked
- Use of this keyword
  - this can be used to refer instance variables when there is a clash with local variables or method arguments
  - this can be used to call overloaded constructors from another constructor of the same class

this

- this can be used to refer instance variables when there is a clash with local variables or method arguments

```
class Number
 {
  int no=0;
  int addNo(int no)
   {
      this.no=this.no+no;
      return(no);
   }
 }
```

```
class Main
 {
  public static void main(String args[])
   {
      Number n1=new Number();
      int res=n1.addNo(10);
      System.out.println(res);
   }
}
```

this

- this can be used to call overloaded constructors from another constructor of the same class

```
class Sample{
Sample(){
    this("Java"); // calls overloaded constructor
      System.out.println("Default constructor ");
   }
Sample(String str){
   System.out.println("One argument constructor "+ str);
   }
}
```

```
class Main
 {
   public static void main(String args[])
     {
          Sample a1=new Sample();
      }
 }
```

Static Class Members

- Static class members are the members of a class that do not belong to an instance of a class
- We can access static members directly by prefixing the members with the class name

  ClassName.staticVariable

  ClassName.staticMethod(...)

Static Class Members

Static variables:

- Shared among all objects of the class
- Only one copy exists for the entire class to use
- Stored within the class code, separately from instance variables that describe an individual object
- Public static final variables are global constants

Static variables

```
class Number
 {
   static int a=1;
   void add(int num)
    {
         a=a+num;
    }
 }
```


```
class Main
 {
   public static void main(String args[])
    {
         Number n1=new Number();
         n1.add(10);
         Number n2=new Number();
         n2.add(20);
         System.out.println(n1.a);
         System.out.println(n2.a);

    }
 }
```

Static Class Members

Static methods:

- Static methods can only access directly the static members and manipulate a class’s static variables
- Static methods cannot access non-static members(instance variables or instance methods) of the class
- Static method can't access this and super references

Static method

```
class First
 {
  int a=10;
  void disp()
   {
       System.out.println(" Value of a is :"+a);
   }
  public static void main(String args[])
   {
       System.out.println(a);
       disp();
   }
 }
```

Quiz

```
class Sample{
  int i_val;
    public static void main(String[] xyz)
     {
        System.out.println("i_val is :"+this.i_val);
     }
    }
```

Quiz

```
class Sample{
   int i_val=10;
   Sample(int i_val){
      this.i_val=i_val;
      System.out.println("inside Sample
      i_val: "+this.i_val);
   }
   public static void main(String[] xyz)
   {
      Sample o = new Sample();
   }
   }
```

Static block

- A static block is a block of code enclosed in braces, preceded by the keyword static

Ex :

```
static {
  System.out.println("Within static block");
  }
```

- The statements within the static block are executed automatically when the class is loaded into JVM

Static block

- A class can have any number of static blocks and they can appear anywhere in the class
- They are executed in the order of their appearance in the class
- JVM combines all the static blocks in a class as single static block and executes them
- You can invoke static methods from the static block and they will be executed as and when the static block gets executed

Example

```
class StaticBlockExample {
  StaticBlockExample() {
    System.out.println("Within constructor");
    }
   static {
     System.out.println("Within 1st static block");
    }
   static void m1() {
      System.out.println("Within static m1 method");
     }
   static {
      System.out.println("Within 2nd static block");
      m1();
     }
public static void main(String [] args) {
System.out.println("Within main");
StaticBlockExample x = new StaticBlockExample();
}
static
  {
    System.out.println("Within 3rd static block");
  }
}
```

Output:

```
Within 1st static block
Within 2nd static block
Within static m1 method
Within 3rd static block
Within main
Within constructor
```

Quiz

```
class Sample{
    public static void main(String[] xyz){
        System.out.println("Inside main  method line1");
    }
    static {
        System.out.println("Inside class  line1");
     }
  }
```

Do iT

Create a class Box that uses a parameterized method to initialize the dimensions of a box.(dimensions are width, height, depth of double type).

The class should have a method that can return volume.

Obtain an object and print the corresponding volume in main() function.

Do iT

Create a new class called “Calculator” which contains the following:

1. A static method called **powerInt(int num1,int num2)** that accepts two integers and returns num1 to the power of num2 (num1 power num2).

2. A static method called **powerDouble(double num1,int num2)** that accepts one double and one integer and returns num1 to the power of num2 (num1 power num2).

3. Call your method from another class without instantiating the class (i.e. call it like Calculator.powerInt(12,10) since your methods are defined to be static)

Hint: Use Math.pow(double,double) to calculate the power.

Do iT

Design a class that can be used by a health care professional to keep track of a patient's vital statistics. Here's what the class should do:

1. Construct a class called Patient

2. Store a String name for the patient

3. Store weight and height for patient as doubles

4. Construct a new patient using these values

5. Write a method called BMI which returns the patient's BMI as a double. BMI can be calculated as BMI = (Weight in Pounds / (Height in inches x Height in inches)) x 703

6. Next, construct a class called "Patients" and create a main method. Create a Patient object and assign some height and weight to that object. Display the BMI of that patient.

String

Strings, which are widely used in Java programming, are a sequence of characters.
In Java programming language, strings are treated as objects.
The Java platform provides the String class to create and manipulate strings.

Creating Strings
The most direct way to create a string is to write −

String greeting = "Hello world!";
String(byte[] bytes)
String(byte[] bytes, Charset charset)

US-ASCII
ISO-8859-1
UTF-8
UTF-16BE
UTF-16LE
UTF-16

String

```
public class StringDemo {
   public static void main(String args[]) {
      String str1="Direct Assignment";

      char[] ch = { 'c','h','a','r','A','R','R','A','Y'};
      String str2=new String(ch);

      String str3=new String("Through Object");

      System.out.println(str1);
      System.out.println(str2);
      System.out.println(str3);
   }
}
```

String

- String s1 = “HELLO”;
- String s2 = “HELLO”;
- s1=s1.toLowerCase();

- String s1 = new String(“HELLO”);
- String s2 = new String(“HELLO”);

Change Case

- **toLowerCase():** Method converts all of the characters in a String to lower case
- **toUpperCase():** Method converts all of the characters in a String to upper case

public String toLowerCase()
public String toUpperCase()

Change Case

```
import java.util.Scanner;
class Test
 {
   public static void main(String args[])
     {
       Scanner sc=new Scanner(System.in);
       String st1=sc.nextLine();
       System.out.println("Upper Case :"+st1.toUpperCase());
       System.out.println("Lower Case :"+st1.toLowerCase());
       System.out.println("Original :"+st1);
     }
 }
```

concat

public String concat(String str)

Concatenates the specified string to the end of this string and returns the resultant String.

concat method is similar to ‘+’ usage

s1+s2 is similar to s1.concat(s2)

concat

```
public class String1 {
   public static void main(String args[]) {
      String str1="Direct Assignment";
      String str2="Direct Assignment";
      String str3=new String("Direct Assignment");
      String str4=new String("Direct Assignment");
      str1=str1+str2;
      str3=str3.concat(str4);
      System.out.println(str1);
      System.out.println(str3);
   }
}
```

String Length

The ***length()*** method returns the length of the string.

Eg: System.out.println("Varun".length()); // prints 5

```
public class StringDemo {
   public static void main(String args[]) {
      String str = "Dot saw I was Tod";
      int len = str.length();
      System.out.println( "String Length is : " + len );
   }
}
```

String Length

```
import java.util.Scanner;
class Test
 {
   public static void main(String args[])
     {
       Scanner sc=new Scanner(System.in);
       String st1=sc.nextLine();
       System.out.println(st1.length());
     }
 }
```

charAt

This method returns the character located at the String's specified index. The string indexes start from zero.

Syntax : ***public char charAt(int index)***

```
public class Test {
   public static void main(String args[]) {
      String s = “Hello Welcome to Java";
      char result = s.charAt(8);
      System.out.println(result);
   }
}
```

charAt

```
import java.util.Scanner;
class Test
 {
    public static void main(String args[])
      {
        Scanner sc=new Scanner(System.in);
        String st1=sc.nextLine();
        int len=st1.length();
        System.out.println(st1.charAt(0));
        System.out.println(st1.charAt(len/2));
        System.out.println(st1.charAt(len-1));

      }
 }
```

charAt

```
import java.util.Scanner;
class Test
 {
   public static void main(String args[])
     {
       Scanner sc=new Scanner(System.in);
       String st1=sc.nextLine();
       int len=st1.length();
       for(int i=0;i<len;i++)
         {
           System.out.print(st1.charAt(i)+".");
         }
     }
 }
```

Do It Yourself

Write a Program that will check whether a given String is Palindrome or not

charAt

Write a Program that will check the number of occurrences of each digit in the given input string.

Do It Yourself

Write a Program to check if a string has all the alphabets.

"Farmer jack realized that big yellow quilts were expensive."

charAt

```
import java.util.Scanner;
class Test
 {
    public static void main(String args[])
      {
         Scanner sc=new Scanner(System.in);
         String st1=sc.nextLine();
         int len=st1.length();
         int count=0;
         for(int i=0;i<len;i++)
          {
            if(st1.charAt(i)==' ')                    ‘\s’
                  count++;
          }
         System.out.println(count);
      }
 }
```

CharAt

```
import java.util.Scanner;
class Test
 {
   public static void main(String args[])
     {
        Scanner sc=new Scanner(System.in);
        String st1=sc.nextLine();
        int len=st1.length();
        int count=0;
        for(int i=0;i<len;i++)
         {
           if(st1.charAt(i)=='\s' && st1.charAt(i-1)!='\s')
                 count++;
         }
        System.out.println(count+1);
     }
 }
```

Trim

trim()
Returns a copy of the string, with leading and trailing whitespace omitted

public String trim()

```
class Test
  {
     public static void main(String args[])
        {
        String st1=" Sai Kiran P  ";
        System.out.println(st1.trim());
        }
  }
```

Trim and CharAt

```
import java.util.Scanner;
class Test
 {
   public static void main(String args[])
     {
        Scanner sc=new Scanner(System.in);
        String st1=sc.nextLine();
        st1=st1.trim();
        int len=st1.length();
        int count=0;
        for(int i=0;i<len;i++)
         {
           if(st1.charAt(i)=='\s' && st1.charAt(i-1)!='\s')
                 count++;
         }
        System.out.println(count+1);
     }
 }
```

char Input In Java With Scanner

```
Scanner sc=new Scanner(System.in);
```

```
char ch=sc.nextChar();    X
```

```
char ch=sc.next().charAt(0);
```


```
import java.util.Scanner;
class CharInput
 {
  public static void main(String args[])
   {
        Scanner sc=new Scanner(System.in);
        char ch;
        ch=sc.next().charAt(0);
        System.out.println(ch);
   }
 }
```

indexOf

indexOf – Searches for the first occurrence of a character or substring. Returns -1 if the character does not occur

public int indexOf(int ch)- It searches for the character represented by ch within this string and returns the index of first occurrence of this character

public int indexOf(String str) - It searches for the substring specified by str within this string and returns the index of first occurrence of this substring

String str = “How was your day today?”;
str.indexOf(‘t’);
Str.indexOf(“was”);

indexOf

```
import java.util.Scanner;
class Test
 {
   public static void main(String args[])
     {
       Scanner sc=new Scanner(System.in);
       String st1=sc.nextLine();
       char ch=sc.next().charAt(0);
       System.out.println(st1.indexOf(ch));
     }
 }
```

indexOf

```
import java.util.Scanner;
class Test
 {
    public static void main(String args[])
      {
          Scanner sc=new Scanner(System.in);
          String st1=sc.nextLine();
          String st2=sc.nextLine();
          if(st1.indexOf(st2)==-1)
            System.out.println("Sub String Not Found");
          else
            System.out.println("Found At Index Position :" +st1.indexOf(st2));
      }
 }
```

indexOf

public int indexOf(int ch, int fromIndex)-
It searches for the character represented by ch within this string and returns the index of first occurrence of this character starting from the position specified by fromIndex

public int indexOf(String str, int fromIndex) **–**
Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

String str = “How was your day today?”;
str.indexOf(‘a’, 6);
Str.indexOf(“was”, 2);

indexOf

```
import java.util.Scanner;
class Test
 {
   public static void main(String args[])
     {
        Scanner sc=new Scanner(System.in);
        String st1=sc.nextLine();
        char ch=sc.next().charAt(0);
        if(st1.indexOf(ch)==-1)
          System.out.println("Character is Not Found");
        else if(st1.indexOf(ch,st1.indexOf(ch)+1)==-1)
          System.out.println("Character Occurred Only Once at"+st1.indexOf(ch));
        else
          System.out.println("Character Occurred More than Once at "+st1.indexOf(ch) +"
                  and aslo at "+(st1.indexOf(ch,st1.indexOf(ch)+1)));
     }
 }
```

indexOf

lastIndexOf()
It searches for the last occurrence of a particular character or substring

String str = “How was your day today?”;
str.indexOf(‘a’);
str.lastIndexOf(‘a’);

lastIndexOf

```
import java.util.Scanner;
class Test
 {
   public static void main(String args[])
     {
        Scanner sc=new Scanner(System.in);
        String st1=sc.nextLine();
        char ch=sc.next().charAt(0);
        if(st1.indexOf(ch)==-1)
          System.out.println("Character is Not Found");
        else if(st1.indexOf(ch)==st1.lastIndexOf(ch))
          System.out.println("Character Occurred Only Once at"+st1.indexOf(ch));
        else
          System.out.println("Character Occurred More than Once at "+st1.indexOf(ch) +"
                    and last at "+st1.lastIndexOf(ch));
     }
 }
```

Do It Yourself

Write a Program to display the Index Positions of all the occurrences of a SubString.

substring

substring()
This method returns a new string which is actually a substring of this string. It extracts characters starting from the specified index all the way till the end of the string

public String substring(int beginIndex)
Eg: "unhappy".substring(2) returns "happy“

public String substring(int beginIndex, int endIndex)
Eg: "smiles".substring(1, 5) returns "mile“

Try It Now

```
import java.util.Scanner;
class Test
 {
   public static void main(String args[])
     {
       Scanner sc=new Scanner(System.in);
       String st1=sc.nextLine();
       System.out.println(st1.substring(1,st1.length()-1));
     }
 }
```

Do It Yourself

Given a string of even length, return the first half. So the string "CatDog" yields "Cat".

If the string length is odd number then return the Second Half by skipping the middle character.

Try It Now

```
import java.util.Scanner;
class Test
  {
     public static void main(String args[])
       {
           Scanner sc=new Scanner(System.in);
           String st1=sc.nextLine();
           int len=st1.length();
           int ipos=0;
           int npos=st1.indexOf('\s');
```


```
if(npos==-1)
  System.out.println(st1);
else
  {
    do
      {

            System.out.println(st1.substring(ipos,npos));
            ipos=npos+1;
            npos=st1.indexOf('\s',npos+1);
      }while(npos!=-1);
    System.out.println(st1.substring(ipos,len));
  }
        }
    }
```

Do It Yourself

Write a Program that Displays all the Unique Words of a String

Input : Java is great Python is also great.
Output : Java
Python
also

Do It Yourself

Write a Program that adds required string to a string at a particular position in a string in java

String str="Hello, Welcome to Java";

Input: Position : 6
String to Insert: Sai Kiran

Output: Hello, Sai Kiran Welcome to Java

Compare

equals() Method- This method is used to compare the invoking String to the object specified. It will return true, if the argument is not null and it is String object which contains the same sequence of characters as the invoking String.

public boolean equals(Object anObject)

equalsIgnoreCase() Method- Compares this String to another String, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case.

public boolean equalsIgnoreCase(String anotherString)

equals

```
Class String
{
  public static void main(String args[])
    {
        String s1="Hello";
        String s2="Hello";
        String s3= new String("Hello");
        String s4= new String("Hello");
        System.out.println((s1==s2));
        System.out.println((s1.equals(s2)));
        System.out.println((s3==s4));
        System.out.println((s3.equals(s4)));
    }
}
```

equals

Accept Input of two strings and check if they are same or not.

Compare

compareTo() - Compares two strings and to know which string is bigger or smaller

- We will get a negative integer, if this String object is less than the argument string
- We will get a positive integer if this String object is greater than the argument string.
- We will get a return value 0(zero), if these strings are equal.

public int **compareTo**(String anotherString)

public int **compareToIgnoreCase**(String str)

This method is similar to compareTo() method but this does not take the case of strings into consideration.

Compare

```
public class Test {

   public static void main(String args[]) {
      String str1 = "Strings are immutable";
      String str2 = new String("Strings are immutable");
      String str3 = new String("Integers are not immutable");

      int result = str1.compareTo( str2 );
      System.out.println(result);

      result = str2.compareTo( str3 );
      System.out.println(result);
   }
}
```

Do It Yourself

Given 2 strings, a and b, return a string of the form short+long+short, with the shorter string on the outside and the longer string on the inside.

The strings will not be the same length, but they may be empty (length 0).

If input is "hi" and "hello", then output will be "hihellohi".

startsWith

startsWith() –
Tests if this string starts with the specified prefix.

public boolean startsWith(String prefix)

"January".startsWith("Jan"); // true

endsWith() -
Tests if this string ends with the specified suffix.

public boolean endsWith(String suffix)
"January".endsWith("ry"); // true

Do It Yourself

Given two strings, append them together (known as "concatenation") and return the result.

However, if the concatenation creates a double-char, then omit one of the chars.

If the inputs are “Mark” and “Kate”
*then the ouput should be “**markate**”.*
(The output should be in lowercase)

Replace

replace()- Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar

public String replace(char oldChar, char newChar)

"wipra technalagies".replace('a', 'o') returns "wipro technologies“

String s1="my name is khan my name is java";
String replaceString=s1.replace("is","was");

valueof

valueOf() – This method is used to convert a Datatype into String.

public static String **valueOf**(char[] data)

public static String **valueOf**(char c)

public static String **valueOf**(boolean b)

public static String **valueOf**(int i)

public static String **valueOf**(long l)

public static String **valueOf**(float f)

public static String **valueOf**(double d)

valueof

```
public class Test
  {
     public static void main(String args[]){
     int value=30;
     String s1=String.valueOf(value);
     System.out.println(s1+10);//concatenating string with 10
      }
}
```

toCharArray()

The **java string toCharArray()** method converts this string into character array.

It returns a newly created character array, its length is similar to this string and its contents are initialized with the characters of this string.

```
String s1="hello";
char[] ch=s1.toCharArray();
for(int i=0;i<ch.length;i++)
 {
    System.out.print(ch[i]);
 }
```

split()

The **java string split()** method splits this string against given regular expression and returns a char array.

public *String split(String regex)*
String s1="Welcome to Java";
String[] words=s1.split("\\s");//splits the string based on whitespace

public *String split(String regex,* ***int*** *limit)*
String s1="Welcome to Java";
String[] words=s1.split(\\s,2);//splits the string based on whitespace

Do It Yourself

Given a string, return a new string made of n copies of the first 2 chars of the original string where n is the length of the string.

The string may be any length. If there are fewer than 2 chars, use whatever is there.

If input is "Wipro" then output should be "WiWiWiWiWi".

StringBuffer

String is immutable where as StringBuffer is Mutable

```
StringBuffer str= new StringBuffer();
StringBuffer str2= new StringBuffer(100);
StringBuffer str3=new StringBuffer(“Hello Welcome to Java”);
```

StringBuffer Vs String

```
public class Test {
   public static void main(String args[]) {
         StringBuffer str1=new StringBuffer(5);
         str1.append("Welcome to Java");

         String str2 = new String();
         str2 = str2.concat("Welcome to Java");
         System.out.println(str1);
         System.out.println(str2);
   }
}
```

append

StringBuffer append(String str)
StringBuffer append(int num)

As the name suggests, the append method adds the specified characters at the end of the StringBuffer object

```
StringBuffer str1=new StringBuffer();
StringBuffer str2=new StringBuffer(5);
StringBuffer str3=new StringBuffer("welcome to Java");
str1.append("Welcome to Java");
str2.append("Welcome to Java");
```

Input

```
Scanner sc= new Scanner(System.in);
StringBuffer str1=new StringBuffer(sc.nextLine());
StringBuffer str2=new StringBuffer(5);
str2=sc.nextLine();
StringBuffer str3=new StringBuffer("welcome to Java");
str3.append(sc.nextLine());
```

append

```
public class Test {
   public static void main(String args[]) {
         StringBuffer str1=new StringBuffer("Hello ");
         StringBuffer str2=new StringBuffer("Sai Kiran,");
         StringBuffer str3=new StringBuffer("Welcome to Java ");
         str2.append(str3);
         str1.append(str2);
         System.out.println(str1);
         System.out.println(str2);
         System.out.println(str3);
   }
}
```

insert

The insert methods are used to insert characters at the specified index location

Here are few insert methods:

StringBuffer insert(int index, String str)

StringBuffer insert(int index, char ch)

Index specifies at which point the string will be inserted into the invoking StringBuffer object

insert

```
public class Test {
   public static void main(String args[]) {
         StringBuffer str1=new StringBuffer("Hello, Welcome to Java");
         str1.insert(6,"Sai Kiran P,");
         System.out.println(str1);
   }
}
```

Do It Yourself

Accept Base String as an input.
Accept a String to be Inserted and index position where it is to be inserted.
Check if the index position is valid or not and insert only if it is valid position

Example:
Base String: **Hello Welcome**
String to be Inserted: **Sai Kiran**
Index position: **25**
Output: Not a Valid Index position

delete

delete() - This method is used to delete specified substring within the StringBuffer object

public StringBuffer delete(int start, int end)

delete

```
import java.util.Scanner;
public class Test {
   public static void main(String args[]) {
         StringBuffer str1=new StringBuffer("Hello, Welcome to Java");
         System.out.println(str1.length());
         str1.delete(7,15);
         System.out.println(str1);
         System.out.println(str1.length());
   }
}
```

delete

```
import java.util.Scanner;
public class Test {
   public static void main(String args[]) {
         StringBuffer str1=new StringBuffer("Hello, Welcome to Java");
         System.out.println(str1.length());
         str1.delete(7,15);
         System.out.println(str1);
         System.out.println(str1.length());
   }
}
```

delete

```
import java.util.Scanner;
public class Test {
   public static void main(String args[]) {
         StringBuffer str1=new StringBuffer();
         Scanner sc=new Scanner(System.in);
         str1.append(sc.nextLine());
         String ch=sc.nextLine();
         int pos;
         while((pos=str1.indexOf(ch))!=-1)
            str1.delete(pos,pos+ch.length());
         System.out.println(str1);
   }
}
```

delete

Return a version of the given string, where for every star (*) in the string the star and the chars immediately to its left and right are gone.

So "ab*cd" yields "ad" and "ab**cd" also yields "ad".

replace

replace() - This method is used to replace part of this StringBuffer(substring) with another substring

public StringBuffer replace(int start, int end, String str)

replace

```
import java.util.Scanner;
public class Test {
   public static void main(String args[]) {
         StringBuffer str1=new StringBuffer();
         Scanner sc=new Scanner(System.in);
         str1.append(sc.nextLine());
         String ch=sc.nextLine();
         String rch=sc.nextLine();
         int pos;
         while((pos=str1.indexOf(ch))!=-1)
           str1.(pos,pos+ch.length(),rch);
         System.out.println(str1);
   }
}
```

reverse

Reverse the string

public StringBuffer **reverse**()

```
public class Test {
   public static void main(String args[]) {
         StringBuffer str1=new StringBuffer(“Welcome to Java);
         System.out.println(str1);
         str1.reverse();
         System.out.println(str1);
   }
}
```

DIY

Given two strings, a and b, create a bigger string made of the first char of a, the first char of b, the second char of a, the second char of b, and so on. Any leftover chars go at the end of the result.

If the inputs are "Hello" and "World", then the output is "HWeolrllod".