

# Object Oriented Programming with Java

---

**Prof. ,** Assistant Professor  
Computer Science & Engineering





# CHAPTER-1

## Fundamentals - I

# Unit-1 FUNDAMENTALS - I

- **Topics:**

Review of OOP

Objects and classes in Java

Defining classes

Methods

Access Specifiers

Static members

Constructors

Garbage collection

Arrays

Strings

Packages

# What is java?

- The Java is Object oriented programming language.
- It is derived from c and c++.
- Java programs can be either application or applets.
- Java is a platform independent language that means it can run on different OS.
- The Java platform consists of JVM and Java API (Application programming Interface).
- The API provides pre-written libraries and standard functionality with classes and interfaces.

# Java Development Tools

- In Java , Java Developer's Kit (JDK) is a software environment provided by Sun Microsystems, which contains JRE(Java Runtime Environment) + Java language development tools (compiler, interpreter, debugger, applet viewer etc..)
- JDK=JRE + Development tools
- JRE=JVM + set of libraries
- JDK comes in various versions and can be downloaded free from the Sun Microsystems

## Types of programs

### •Applets

- Execute on HTML Browsers.
- Have several security restrictions.



### •Servlet

- Execute on Sophisticated Servers.



### •GUI Applications

- Are Interpreted and Executed.
- Use the current's platform's GUI widgets.
- Mostly use Java's AWT packages.

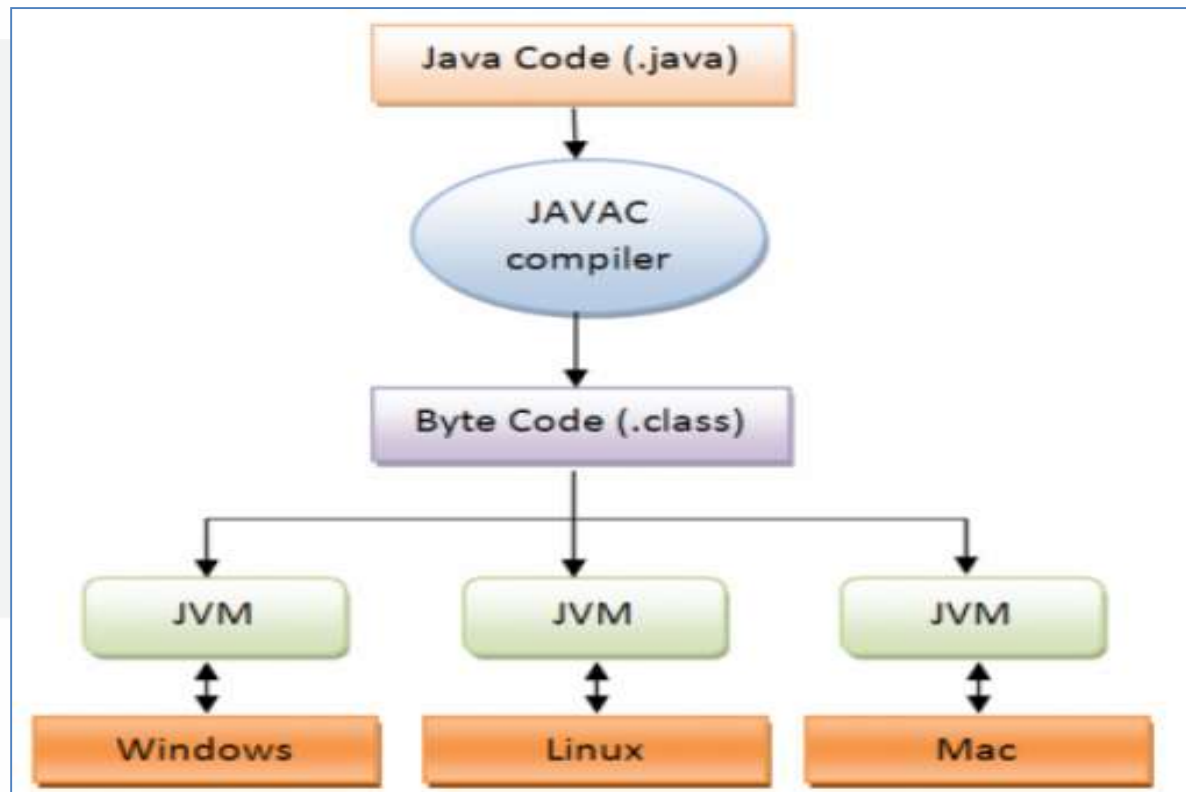


### •CLI Applications

- Runs on command line

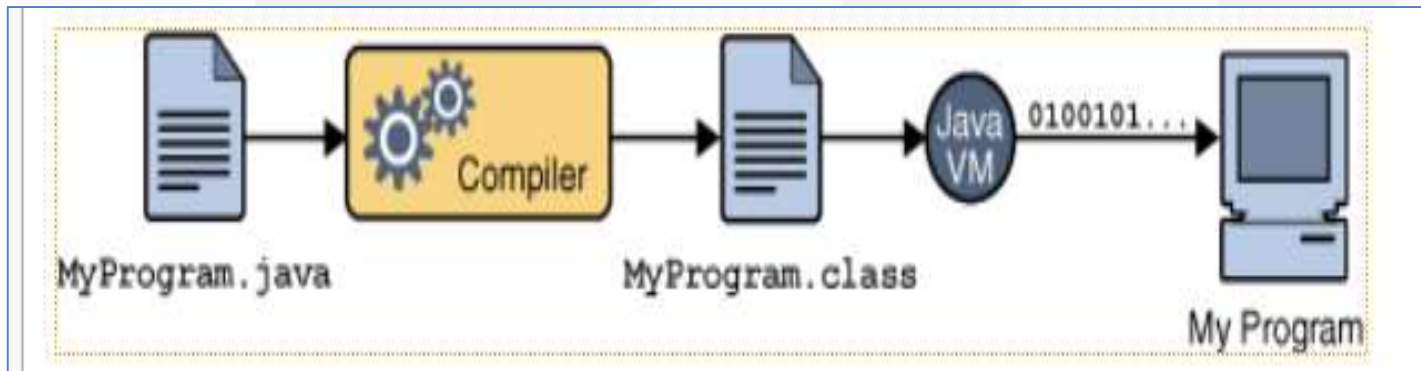


# Java is Platform Independent



# WORA(Write Once Run Anywhere)

D I I







## Features of Java

- Java is **simple**
- Java is **object-oriented**
- Java is **distributed**
- Java is **interpreted**
- Java is **robust**
- Java is **portable**
- Java's **performance**
- Java is **multithreaded**
- Java is **dynamic**
- Java is **Secure**

## Cont ...

- **Java is simple:**
  - Derived from C(syntax) & C++(features)
  - Easy to write & Compile
- **Java is object-oriented**
  - Class
  - Object
  - Inheritance
  - Polymorphism
  - Encapsulation
  - Everything is an object
  - Java supports all features of OOP:

## Cont ...

- **Java is distributed:**

- Java is known as distributed language for creating application on networks
- Java supports TCP/IP & UDP Protocol.
- RMI
- Support for Internet communication primitives

## Cont ...

- **Java is Interpreted**

- Compiling the java source code into an intermediate representation called BYTECODE
- It can run on every platform having Java Virtual Machine installed
- Java Virtual Machine
  - Runtime Environment to execute java programs
  - Interprets the bytecode and converts into platform specific code
  - Secure execution of the programs without any side effects

- **Java is Portable:**

- Write Once Run Anywhere



## Cont ...

- **Java is Robust:**

- The program should run effectively on multiple platforms
- Strictly typed language
- Code checking at compile time as well as run time
- Memory management using the Garbage Collector
  - ❖ Automatic deallocation of memory with unused Objects
- Exception Handling
  - ❖ Prevent program from crashing due to runtime error

## Cont ...

- Java is **Multithreaded**
  - Multithreaded means handling multiple tasks simultaneous.
  - Multithreading means a single program having different threads executing independently at the same time.

## Cont ...

- Java is **dynamic**
  - By connecting to the Internet, a user immediately has access to thousands of programs and other computers.
  - During the execution of a program, Java can dynamically load classes that it requires either from the local hard drive, from another computer on the local area network or from a computer somewhere on the Internet.
  - Java is capable dynamically linking in new class libraries, methods, and objects.

## Cont ...

- Java is **Secure**

- The Java language has built-in capabilities to ensure that violations of security do not occur.
- Threat or viruses and abuse of resources are everywhere. Java system not only verifies all memory access but also ensure that no viruses are communicated with an Applet.
- Absence of pointer in java ensures that program cannot gain access to memory locations without proper authorization.



# Java Environment

- Java includes many development tools, classes and methods
  - Development tools are part of Java Development Kit (JDK) and
  - The classes and methods are part of **Java Standard Library (JSL)**, also known as **Application Programming Interface (API)**.
- JDK constitutes of tools like **java compiler**, java interpreter and many.
- **API** includes hundreds of **classes** and **methods** grouped into several **packages** according to their functionality.

## Other components

- JIT – improves performance of JVM by directly compiling some sensitive code (code which executes above a certain threshold) into native instruction. It caches the compiled code hence improving the performance.
- JRE -Runtime Environment
- JAR – Java Archive



# First Java Program

## Example.java

```
class Example
{
    public static void main(String args[])
    {
        System.out.println("This is a simple Java program");
    }
}
```

## Source file name must match the class name – Why?

- a source file is officially called a *compilation unit*
- Java compiler requires that a source file use the **.java** filename extension
- Compilation:
  - javac Example.java
- Will create a file named **Example.class**
  - The name is **same as that of class name in source file (except the extension!)**
  - Contains the compiled **bytecode** of the **Example.java** source

## Source file name must match the class name – Why?

- To actually run the program the command is
  - `java Example`
- specifying the name of the class that you want to execute
- **Searches for the .class file with name Example (Example.class)**
- Hence, both needs to be same



- Objects and classes are basic concepts of Object Oriented Programming which revolve around the real life entities.
- Class is a container for variables and methods.
- Class defines a new data type that is used to create the object of that type.
- Thus, a class is a template for an object, and an object is an instance of a class.
- A class is declared by use of the class keyword.

# Class Syntax

```
class classname
{
    type instance-variable1;
    type instance-variable2;
    // ...
    type instance-variableN;
    type methodname1(parameter-list)
    {
        // body of method
    }
    type methodname2(parameter-list)
    {
        // body of method
    }
    // .....
    type methodnameN(parameter-list)
    {
        // body of method
    }
}
```



# Class

- The data, or variables, defined within a class are called instance variables because each instance of the class contains its own copy of these variables.
- The data for one object is separate and unique from the data for another.
- The code is contained within methods. The methods and variables defined within a class are called members of the class.

Note: Java classes do not need to have a `main( )` method. You only specify `main()` method if that class is the starting point for your program.

## Class Example

Example:

```
class Box
{
    int width,height, depth;
    void set()
    {
    }
    void set1(int a)
    {
    }
    int set2()
    {
    }
    int set3(int a)
    {
    }
}
```

# Objects

- A java object is one instance of class.
- A object is a block of memory that contains space to store all the instance variables.
- Objects are created by using new operator. The new operator creates an object of the specified class in the heap area of memory and returns a reference to the object created. It is dynamically allocated memory to the object.
- Syntax:

*classname objectname = new classname();*

## Object Example

```
Box b1=new Box();
```

❓ Now when we create another object reference and give to reference of previous object

```
Box b1; b1=new Box();
```

```
Box b2=b1
```

❓ The above stat. can not create new object in the memory but b1 and b2 will both refer to the same object.

## Example: Class

A class with three variable members:

```
class Box
{
    double width;
    double height;
    double depth;
}
```

A new `Box` object is created and a new value assigned to its width variable:

```
Box myBox = new Box();
myBox.width = 100;
```

## Example: Class Usage

```
class BoxDemo {  
    public static void main(String args[]) {  
        Box mybox = new Box();  
        double vol;  
  
        mybox.width = 10;  
        mybox.height = 20;  
        mybox.depth = 15;  
  
        vol = mybox.width * mybox.height * mybox.depth;  
        System.out.println("Volume is " + vol);  
    }  
}
```

# Compilation and Execution

Place the `Box` class definitions in file `Box.java`:

```
class Box { ... }
```

Place the `BoxDemo` class definitions in file `BoxDemo.java`:

```
class BoxDemo  
{ public static void main(...)  
  { ... }  
}
```

Compilation and execution:

```
> javac BoxDemo.java  
> java BoxDemo
```

# Variable Independence

Each object has its own copy of the instance variables: changing the variables of one object has no effect on the variables of another object.

Consider this example:

```
class BoxDemo2
{ public static void main(String args[])
    { Box mybox1 = new Box();
      Box mybox2 = new Box();
      double vol;

      mybox1.width = 10;
      mybox1.height = 20;
      mybox1.depth = 15;
```



```
mybox2.width    = 3;  
mybox2.height   = 6;  
mybox2.depth    = 9;
```

```
vol = mybox1.width * mybox1.height * mybox1.depth;  
System.out.println("Volume is " + vol);
```

```
vol = mybox2.width * mybox2.height * mybox2.depth;  
System.out.println("Volume is " + vol);
```

```
    }  
}
```

What are the printed volumes of both boxes?



## OBJECTS Example

```
class Book
{
}

class JavaBook
{
    public static void main(String args[])
    {
        Book b=new Book();
    }
}
```

```
class College
{
}

class Kits
{
    Public static void main(String
    args[])
    {
        College c=new College();
    }
}
```



# Declaring Objects

Obtaining objects of a class is a two-stage process:

1) Declare a variable of the class type:

```
Box myBox;
```

The value of myBox is a reference to an object, if one exists, or `null`.  
At this moment, the value of myBox is `null`.

2) Acquire an actual, physical copy of an object and assign its address to the variable. How to do this?

## Operator new

Allocates memory for a `Box` object and returns its address:

```
Box myBox = new Box();
```

The address is then stored in the `myBox` reference variable.

`Box()` is a class constructor - a class may declare its own constructor or rely on the default constructor provided by the Java environment.

# Memory Allocation

Memory is allocated for objects dynamically.

This has both advantages and disadvantages:

- 1) as many objects are created as needed
- 2) allocation is uncertain – memory may be insufficient

Variables of simple types do not require **new**:

```
int n = 1;
```

In the interest of efficiency, Java does not implement simple types as objects. Variables of simple types hold values, not references.

## Assigning Reference Variables

Assignment copies address, not the actual value:

```
Box b1 = new Box();
```

```
Box b2 = b1;
```

Both variables point to the same object.

Variables are not in any way connected. After

```
b1 = null;
```

b2 still refers to the original object.

# Methods

General form of a method definition:

```
type  name(parameter-list)
{    ... return  value;    ...
}
```

Components:

- 1) **type** - type of values returned by the method. If a method does not return any value, its return type must be **void**.
- 2) **name** is the name of the method
- 3) **parameter-list** is a sequence of type-identifier lists separated by commas
- 4) **return value** indicates what value is returned by the method.

## Example: Method

Classes declare methods to hide their internal data structures, as well as for their own internal use:

Within a class, we can refer directly to its member variables:

```
class Box
{
    double width, height, depth;
    void volume()
    {
        System.out.print("Volume is : ");
        System.out.println(width * height * depth);
    }
}
```





## Example: Method

When an instance variable is accessed by code that is not part of the class in which that variable is defined, access must be done through an object:

```
class BoxDemo3
{
    public static void main(String args[])
    {
        Box mybox1 = new Box();
        Box mybox2 = new Box();
        mybox1.width = 10;
        mybox1.height = 20;
        mybox1.depth = 15;
        mybox1.volume();
        mybox2.volume();

        mybox2.width = 3;
        mybox2.height = 6;
        mybox2.depth = 9;
    }
}
```

## Value-Returning Method

The type of an expression returning value from a method must agree with the return type of the method:

```
class Box
{
    double width;
    double height;
    double depth;

    double volume()
    {
        return width * height * depth;
    }
}
```



```
class BoxDemo4
{ public static void main(String args[])
  { Box mybox1 = new Box();
    Box mybox2 = new Box();
    double vol;
    mybox1.width = 10;
    mybox2.width = 3;
    mybox1.height = 20;
    mybox2.height = 6;
    mybox1.depth = 15;
    mybox2.depth = 9;
    vol = mybox1.volume();
    System.out.println("Volume is " + vol);
    vol = mybox2.volume();
    System.out.println("Volume is " + vol);
  } }
```

# Parameterized Method

Parameters increase generality and applicability of a method:

1) method without parameters

```
int square() { return 10*10; }
```

2) method with parameters

```
int square(int i) { return i*i; }
```

**Parameter:** a variable receiving value at the time the method is invoked.

**Argument:** a value passed to the method when it is invoked.

## Example: Parameterized Method

```
class Box
{
    double width;
    double height;
    double depth;

    double volume()
    {
        return width * height * depth;
    }

    void setDim(double w, double h, double d)
    {
        width = w; height = h; depth = d;
    }
}
```



```
class BoxDemo5
{ public static void main(String args[])
    { Box mybox1 = new Box();
      Box mybox2 = new Box();
      double vol;

      mybox1.setDim(10, 20, 15);
      mybox2.setDim(3, 6, 9);

      vol = mybox1.volume();
      System.out.println("Volume is " + vol);
      vol = mybox2.volume();
      System.out.println("Volume is " + vol);
    }
}
```

## Method overloading

```
class overloaddemo
{
void test()
{ System.out.println("no parameters."); }
void test(float a)
{ System.out.println("value of float a="+a); }
void test(int a ,int b)
{
System.out.println("value of a="+a);
System.out.println("value of b="+b);
}
int test(int a)
{ return a*a; }
}
```

```
class m1
{
public static void main(String args[])
{
overloaddemo ob= new overloaddemo();
ob.test();
ob.test(10.2f);
ob.test(10,20);

int r=ob.test(10);
System.out.println("value of r="+r);
}
}
```

**output:**

no parameters  
value of float a=10.2  
value of float a=10  
value of float b=20  
value of float r=100



# Constructor

A constructor initializes the instance variables of an object.

It is called immediately after the object is created but before the **new** operator completes.

- 1) it is syntactically similar to a method:
- 2) it has the same name as the name of its class
- 3) it is written without return type; the default return type of a class constructor is the same class

When the class has no constructor, the default constructor automatically initializes all its instance variables with zero.

## Constructor : Example

```
class One
{
    One()
    { //Initialization }
}
```

```
class Book
{
    Book()
    {
        // Some Initialization
    }
}
```

## Example: Constructor

```
class Box
{
    double width;
    double height;
    double depth;

    Box()
    {
        System.out.println("Constructing Box");
        width = 10; height = 10; depth = 10;
    }

    double volume()
    {
        return width * height * depth;
    }
}
```



```
class BoxDemo6
{
    public static void main(String args[])
    {
        Box mybox1 = new Box();
        Box mybox2 = new Box();
        double vol;

        vol = mybox1.volume();
        System.out.println("Volume is " + vol);

        vol = mybox2.volume();
        System.out.println("Volume is " + vol);
    }
}
```

## Parameterized Constructor

So far, all boxes have the same dimensions.

We need a constructor able to create boxes with different dimensions:

```
class Box
{
    double width;
    double height;
    double depth;

    Box(double w, double h, double d)
    {
        width = w; height = h; depth = d;
    }

    double volume() { return width * height * depth; }
}
```

```
class BoxDemo7
{ public static void main(String args[])
    { Box mybox1 = new Box(10, 20, 15);
      Box mybox2 = new Box(3, 6, 9);
      double vol;

      vol = mybox1.volume();
      System.out.println("Volume is " + vol);

      vol = mybox2.volume();
      System.out.println("Volume is " + vol);
    }
}
```

## Copy constructor

```
class student
{ int id;
  String name;
  student(int i,String n1)
  { id = i;
    name = n1;
  }
  student (student s)
  { id=s.id;
    name=s.name;
  }
  void display()
  {   System.out.println(id+" "+name);
  }
}
```

```
class m1
{
  public static void main(String args[])
  {
    student s1 = new student(1,"Java");
    student s2 = new student(s1);
    s1.display();
    s2.display();
  }
}
```

## Output:

1 Java

1 Java

PU



# Constructor overloading

```
class Box
{ int width,height, depth;
Box()
{ width = 10;
  height = 20;
  depth = 30;
}
Box(int w ,int h, int d)
{ width = w;
  height = h;
  depth = d;
}
Box(int l)
{ width = height = depth =l }
int display()
{ return width * height * depth; }
}
```

```
class m1
{
public static void main(String args[])
{
Box b1 = new Box();
Box b2= new Box(1,2,3);
Box b3= new Box(4);
int v=b1.display();
System.out.println("volume of v=" + v);
v=b2.display();
System.out.println("volume of v=" + v);
v=b3.display();
System.out.println("volume of v=" + v);
}
}
```

output:

volume of  $v_1 = 6000$

volume of  $v=6$

volume of  $v=64$

## Passing object as parameter

```
class test
{
    int a,b;
    test( int i ,int j)
    {
        a = i;
        b = j;
    }
    void equal(test T)
    {
        if(a==T. a && b==T.b)
            System.out.println("value of 2 objs r equal");
        else
            System.out.println("value of 2 objects r not equal");
    }
}
```

```
class m1
{
    public static void main(String args[])
    {
        test t1= new test(10,20);
        test t2= new test(10,20);
        test t3=new test(30,40);
        t1.equal(t2);
        t1.equal(t3);
    }
}
```

**output:**

value of 2 objects r equal

value of 2 objects r not equal

## Array of Objects

- Unlike traditional array which store values like string, integer, Boolean, etc. array of objects stores objects.
- The array elements store the location of reference variables of the object.



## Example:

```
import java.util.Scanner;
class Employee
{
    int Id;
    String Name;
    int Age;
    long Salary;

    void GetData()
    {
        Scanner sc = new Scanner(System.in);

        System.out.print("\n\tEnter Employee Id : ");
        Id = Integer.parseInt(sc.nextLine());

        System.out.print("\n\tEnter Employee Name : ");
        Name = sc.nextLine();

        System.out.print("\n\tEnter Employee Age : ");
        Age = Integer.parseInt(sc.nextLine());

        System.out.print("\n\tEnter Employee Salary : ");
        Salary = Integer.parseInt(sc.nextLine());
    }
}
```



```
void PutData()
{
    System.out.print("\n\t" + Id + "\t" +Name + "\t" +Age + "\t" +Salary);
}
```

```
}
class abc
{
    public static void main(String args[])
    {
        Employee[] Emp = new Employee[3];
        int i;
        for(i=0;i<3;i++)
            Emp[i] = new Employee();    // Allocating memory to each object

        for(i=0;i<3;i++)
        {
            System.out.print("\nEnter details of " + (i+1) + " Employee\n");
            Emp[i].GetData();
        }

        System.out.print("\nDetails of Employees\n");
        for(i=0;i<3;i++)
            Emp[i].PutData();
    }
}
```

## Output:

```
C:\Users\PANKAJ PATEL\Desktop>java abc
Enter details of 1 Employee
    Enter Employee Id : 1
    Enter Employee Name : dsf
    Enter Employee Age : 22
    Enter Employee Salary : 2321
Enter details of 2 Employee
    Enter Employee Id : 2
    Enter Employee Name : dfs
    Enter Employee Age : 23
    Enter Employee Salary : 2324
Enter details of 3 Employee
    Enter Employee Id : 2
    Enter Employee Name : dfgdf
    Enter Employee Age : 34
    Enter Employee Salary : 435345
Details of Employees
    1      dsf      22      2321
    2      dfs      23      2324
    2      dfgdf    34      435345
```



## Example:

```
class Car
{
    int power;
    int Serial_no;
    Car(int p, int s)
    {
        power = p;
        Serial_no = s;
    }
    void disp()
    {
        System.out.println(power + " " + Serial_no);
    }
}

class abc
{
    public static void main(String[] args)
    {
        Car[] c = new Car[2];

        c[0] = new Car(800,111);
        c[1] = new Car(480,157);

        c[0].disp();
        c[1].disp();
    }
}
```

## Output:

```
C:\Users\PANKAJ PATEL\Desktop>javac abc.java
C:\Users\PANKAJ PATEL\Desktop>java abc
800    111
480    157
```

## Return Object from Method

```
class Test
{
    int a;

    Test(int i)
    {
        a = i;
    }

    Test incrByTen()
    {
        Test temp = new Test(a+10);
        return temp;
    }
}
```



```
public class JavaProgram
{
    public static void main(String args[])
    {

        Test obj1 = new Test(2);
        Test obj2;

        obj2 = obj1.incrByTen();

        System.out.println("obj1.a : " + obj1.a);
        System.out.println("obj2.a : " + obj2.a);

        obj2 = obj2.incrByTen();

        System.out.println("obj2.a after second increase : " + obj2.a);

    }
}
```

## Output

Obj 1.a : 10

Obj 2.a : 20

Obj 2.a after second increase : 30

PU

# Garbage Collection

- Java Virtual Machine allocates objects on the heap dynamically using new operator.
- Other language put the burden on the programmer to free these objects using delete() or free() function when they are no longer needed.
- Java language does not provide this type of function because java runtime environment automatically reclaims the memory for objects that are no longer associated with a reference variable.
- This memory reclamation process is known as garbage collection.
- Garbage collection involves:
  - 1.keeping a count of all references to each object
  - 2.periodically reclaiming memory for all objects with a reference count to zero.

- Example :

```
void a method() { Sum d1=new Sum(): }
```

- new operation creates an instances of Sum,its memory address is store in d1and its reference count is incremented to 1.when this method return or finishes reference variable d1 goes out of scope and reference count is decrement to 0.At this point Sum instance is reclaimed by garbage collection.
- Garbage collection is automatically runs periodically.you can manually invoke the garbage collection at any time using `java.lang.System.gc()` method .

## Finalize() Method

A constructor helps to initialize an object just after it has been created.

In contrast, the finalize method is invoked just before the object is destroyed:

1) implemented inside a class as:

```
Protected void finalize() { }
```

Finalize() is the method of Object class.

This method is called just before an object is garbage collected.

Finalize() method overrides to dispose system resources, perform clean-up activities and minimize memory leaks.



```
class abc
{ // Here overriding finalize method
    public void finalize()
    {      System.out.println("finalize method overridden");    }

    public static void main(String[] args) throws Throwable
    { abc aa=new abc();
      aa.finalize();
      // Requesting JVM to call Garbage Collector method
      System.gc();
      System.out.println("Main Completes");
    } }
```

## This keyword

- “this” keyword is used inside any instance method or constructor of class.
- “this” is a reference variable that refers to the current object whose Method or constructor is being invoked(call). It is a keyword in java language represents current class object .
- To differentiate between instance variable and formal parameter , the data member of the class must be preceded by **"this"**.
- If any variable is preceded by **"this"** **JVM treats that variable as class variable.**
- Static method can not use “this” keyword.

## problem without “this”

```
class Student
{
int id;
String name;
Student(int id,String name)
{
id = id;
name = name;
}
void display()
{
System.out.println(id+" "+name);
}
}
```

```
class a1
{
public static void main(String args[])
{
Student s1 = new Student(1,"xyz");
Student s2 = new Student(2,"pqr");
s1.display();
s2.display();
}
}
```



**output:**

0 null

0 null

PU

## Solution with “this”

```
class Student
{
int id;
String name;
Student(int id,String name)
{
this.id = id;
this.name = name;
}
void display()
{
System.out.println(id+" "+name);
}
}
```

```
class a1
{
public static void main(String args[]){
Student s1 = new Student(1,"xyz");
Student s2 = new Student(2,"pqr");
s1.display();
s2.display();
}
```



output:

1 xyz

2 pqr

PU

## Static Keyword

- You cannot use the static keyword with a class unless it is an inner class. A static inner class is a nested class which is a static member of the outer class.
- It can be accessed without instantiating the outer class, using other static members. Just like static members, a static nested class does not have access to the instance variables and methods of the outer class.



```
public class Outer {  
    Java Arrays with Answers  
    static class Nested_Demo {  
        public void my_method() {  
            System.out.println("This is my nested class");  
        }  
    }  
    public static void main(String args[]) {  
        Outer.Nested_Demo nested = new Outer.Nested_Demo();  
        nested.my_method();  
    }  
}
```

## Output

```
This is my nested class
```



## Access specifiers

| Visibility                                      | Public | Protected                           | Default                | Private |
|---|--------|-------------------------------------|------------------------|---------|
| From the same class                             | Yes    | Yes                                 | Yes                    | Yes     |
| From any class in the same package              | Yes    | Yes                                 | Yes                    | No      |
| From a subclass in the same package             | Yes    | Yes ( <i>Package, Inheritance</i> ) | Yes ( <i>Package</i> ) | No      |
| From a subclass outside the same package        | Yes    | Yes ( <i>Inheritance</i> )          | No                     | No      |
| From any non-subclass class outside the package | Yes    | No                                  | No                     | No      |

## Exercise

A program to create employee class. Also use it by creating objects of it

- Eno
- Ename
- baseSalary
- noofdaysworked
- Tax
- Employee() - constructor
- computeSalary()
- computeTax()
- displayEmployee()

# ARRAYS

The entire array  
has a single name

Each value has a numeric *index*

scores

|    |    |    |    |    |     |     |
|----|----|----|----|----|-----|-----|
| 0  | 1  | 2  | 3  | 4  | 5   | 6   |
| 50 | 12 | 45 | 78 | 66 | 100 | 125 |

An array of size N is indexed from zero to N-1

INTEGER

- An array can be of any type.
- Specific element in an array is accessed by its index.
- Can have more than one dimension

# DECLARAING ARRAYS

- The general form of **1-d** array declaration is:-

*type*    *var\_name* [ ];

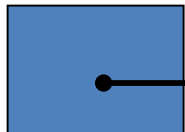
*int, float, char*

*array name*

E.g.:--- *int* *scores* [ ];

*scores*

*int*[] *scores* = new *int*[10];



NULL

1. Even though an array variable "scores" is declared, but there is no array actually existing.
2. "score" is set to NULL, i.e. an array with NO VALUE.

## Declaring Arrays Examples

```
double[] prices = new double[500];
```

```
boolean[] flags;
```

```
flags = new boolean[20];
```

```
char[] codes = new char[1750];
```

## Array using Enhanced for Loop

```
class newforloop
{
    public static void main (String [] args)
    {
        String name[]= {"John","Tom","Sam"};
        for(String n:name)
        System.out.println(n+"\n");
        int i;
        for(i=0;i<3;i++)
        System.out.println(name[i]+"\\n");
    }
}
```

## Bounds Checking

- ❑ Once an array is created, it has a fixed size
- ❑ An index used in an array reference must specify a valid element
- ❑ That is, the index value must be in bounds (**0** to **N-1**)
- ❑ The Java interpreter throws an **ArrayIndexOutOfBoundsException** if an array index is out of bounds
- ❑ This is called *automatic bounds checking*

## ARRAY LENGTH

- ❑ In JAVA, all arrays store the allocated size in a variable named **length**.
- ❑ It is referenced using the array name:  
  
**scores.length**
- ❑ Note that **length** holds the number of elements, not the largest index.
- ❑ This information will be useful in the manipulation of arrays when their sizes are not known



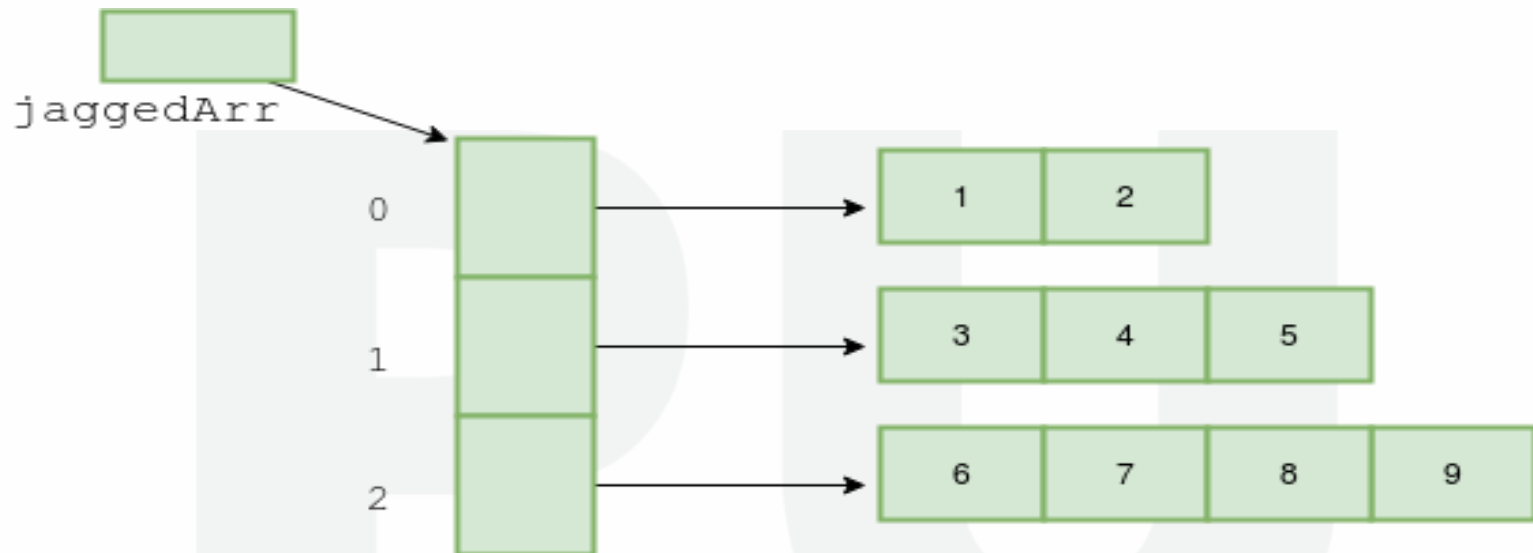
## Jagged Arrays

- Multidimensional Arrays having different lengths of each sub array

- Ex.

```
int p[][] = { {3,5,8,3,9,0},  
              {4,2,6,7,1},  
              {3,1,5,6},  
              {2,3} }
```

# Memory Representation of jagged array



**Fig: Memory Representation of Jagged Array**

## Example: to create a Jagged Array

```
int arr[][] = new int[2][];      // Declaring a 2D array with 2 rows
arr[0] = new int[3]; // First row with 3 elements
arr[1] = new int[2]; // Second row with 2 elements
int count = 0;      // Initializing array
for (int i=0; i<arr.length; i++)
    for(int j=0; j<arr[i].length; j++)
        arr[i][j] = count++;
// Displaying the values of 2D Jagged array
for (int i=0; i<arr.length; i++){
    for (int j=0; j<arr[i].length; j++)
        System.out.print(arr[i][j] + " ");
    System.out.println();
}
```

## Exercise

```
class ArrayTest
{
    String[] firstNames = { "Dennis", "Grace", "Bjarne", "James" };
    String[] lastNames = new String[firstNames.length];
    void printNames()
    {
        for (int i = 0; i < firstNames.length; i++)
            System.out.println(firstNames[i] + " " + lastNames[i]);
    }
    public static void main (String args[]) {
        ArrayTest a = new ArrayTest();
        a.printNames();
        a.lastNames[0] = "Ritchie";
        a.lastNames[1] = "Hopper";
        a.lastNames[2] = "Stroustrup";
        a.lastNames[3] = "Gosling";
        a.printNames();  }}
```



```
C:\Users\PANKAJ PATEL\Desktop>javac abc.java
```

```
C:\Users\PANKAJ PATEL\Desktop>java abc
```

```
Dennis null
```

```
Grace null
```

```
Bjarne null
```

```
James null
```

```
-----
```

```
Dennis Ritchie
```

```
Grace Hopper
```

```
Bjarne Stroustrup
```

```
James Gosling
```

## Exercise

- ☐ A program to find average of all numbers stored in an array
- ☐ A program to reverse the values of an array
- ☐ A program to compute the series  $n^2$  and  $n^3$  using an array
- ☐ A program to compute the matrix multiplication using multidimensional arrays
- ☐ A program to find the transpose of a given matrix

# Strings

- In java, four predefined classes are provided that either represent strings or provide functionality to manipulate them. Those classes are:
  - String
  - StringBuffer
  - StringBuilder
- String, StringBuffer, and StringBuilder classes are defined in java.lang package and all are final.
- All three implement the CharSequence interface.

## Why String is Immutable or Final?

- String has been widely used as parameter for many java classes e.g. for opening network connection we can pass hostname and port number as string ,
- we can pass database URL as string for opening database connection,
- we can open any file in Java by passing name of file as argument to File I/O classes.
- In case if String is not immutable , this would lead serious security threat , means some one can access to any file for which he has authorization and then can change the file name either deliberately or accidentally.



## Why String Handling?

.The internet of things(IOT) is the network of physical objects not limited to devices, vehicles, buildinString handling is required to perform following operations on some string:

- compare two strings
- search for a substring
- concatenate two strings
- change the case of letters within a string
- And many more...

## Creating String objects

```
class StringDemo
{
    public static void main(String args[])
    {
        String strOb1="Hello";
        String strOb2="String";
        String strOb3=strOb1 + " and " + strOb2; System.out.println(strOb1);
        System.out.println(strOb2); System.out.println(strOb3);
    }
}
```

# String Class

## String Constructor:

```
public String ()  
public String (String s)  
public String (char [] c)  
public String (byte [] b)  
public String (char [] c, int offset, int no_of_chars)  
public String (byte [] b, int offset, int no_of_bytes)
```

## Examples

```
char [] a = {'c', 'o', 'n', 'g', 'r', 'a', 't', 's'};
```

```
byte [] b = {65, 66, 67, 68, 69, 70, 71, 72};
```

```
String s1 = new String (a); System.out.println(s1);
```

```
String s2 = new String (a, 1,5); System.out.println(s2);
```

```
String s3 = new String (s1); System.out.println(s3);
```

```
String s4 = new String (b); System.out.println(s4);
```

```
String s5 = new String (b, 4, 4); System.out.println(s5);
```

# String Concatenation with Other Data Types

We can concatenate strings with other types of data.

Example:

```
int age = 9;
```

```
String s = "He is " + age + " years old."; System.out.println(s);
```

## Methods of String class

### **String Length:**

length() returns the length of the string i.e. number of characters.

*int length()*

Example:

```
char chars[] = { 'a', 'b', 'c' }; String s = new String(chars);  
System.out.println(s.length());
```

## Character Extraction

**charAt():** used to obtain the character from the specified index from a string.

*public char charAt (int index);*

Example:

```
char ch;
```

```
ch = "abc".charAt(1);
```

## Methods Cont...

```
public void getChars(int srhStartIndex, int srhEndIndex, char[] destArray, int  
destStartIndex)
```

### Parameters:

srhStartIndex : Index of the first character in the string to copy.

srhEndIndex : Index after the last character in the string to copy.

destArray : Destination array where chars will get copied.

destStartIndex : Index in the array starting from where the chars will be pushed into the array.

**Example:** String s = "MNOPQ";  
char b[] = new char [10];  
s.getChars(0, 4, b, 4);  
System.out.println(b);



## What is Internet of things(IOT) ?

**toCharArray():** returns a character array initialized by the contents of the string.

```
public char [] toChar Array();
```

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

# String Comparison

**equals():** used to compare two strings for equality. Comparison is case-sensitive.

*public boolean equals (Object str)*

**equalsIgnoreCase( ):** To perform a comparison that ignores case differences.

Note:

This method is defined in Object class and overridden in String class.

**equals(),** in Object class, compares the value of reference not the content.

In String class, equals method is overridden for content-wise comparison of two strings.



## Example

```
class equalsDemo {  
    public static void main(String args[])  
    {  
        String s1 = "Hello"; String s2 = "Hello";  
        String s3 = "Good-bye"; String s4 = "HELLO";  
        System.out.println(s1 + " equals " + s2 + " -> " + s1.equals(s2));  
        System.out.println(s1 + " equals " + s3 + " -> " +  
s1.equals(s3));  
        System.out.println(s1 + " equals " + s4 + " -> " + s1.equals(s4));  
        System.out.println(s1 + " equalsIgnoreCase " + s4 + " -> "  
+s1.equalsIgnoreCase(s4));  
    }  
}
```

# String Comparison

## **startsWith( ) and endsWith( ):**

The `startsWith( )` method determines whether a given String begins with a specified string.

Conversely, `endsWith( )` determines whether the String in question ends with a specified string.

*`boolean startsWith(String str)`*

*`boolean endsWith(String str)`*

# String Comparison

## **compareTo( ):**

A string is less than another if it comes before the other in dictionary order.

A string is greater than another if it comes after the other in dictionary order.

*int compareTo(String str)*

| Value             | Meaning  |
|-------------------|--|
| Less than zero    | The invoking string is less than <i>str</i> .    |
| Greater than zero | The invoking string is greater than <i>str</i> . |
| Zero              | The two strings are equal.                       |

## Exercise— Arrange strings in ascending order

```
String arr[] = {"Now", "is", "the", "time", "for", "all", "good", "men", "to",  
               "come", "to", "the", "aid", "of", "their", "country"};
```

PU

## Solution

```
class SortString {  
    static String arr[] = {"Now", "is", "the", "time", "for", "all", "good", "men", "to", "come",  
        "to", "the", "aid", "of", "their", "country"};  
    public static void main(String args[])  
    {  
        for(int j = 0; j < arr.length; j++)  
        {  
            for(int i = j + 1; i < arr.length; i++)  
            {  
                if(arr[i].compareTo(arr[j]) < 0)  
                {  
                    String t = arr[j];  
                    arr[j] = arr[i];  
                    arr[i] = t;  
                }  
            }  
            System.out.println(arr[j]);  
        }  
    }  
}
```

## Searching Strings

The String class provides two methods that allow us to search a string for a specified character or substring:

**indexOf( )**: Searches for the first occurrence of a character or substring.  
*int indexOf(int ch)*

**lastIndexOf( )**: Searches for the last occurrence of a character or substring.  
*int lastIndexOf(int ch)*

To search for the first or last occurrence of a substring, use  
*int indexOf(String str) int lastIndexOf(String str)*



## Example

```
String s = "MMBBNGACCR"
```

```
int index = s.indexOf('G');
```

```
int index = s.indexOf(' ');
```

```
int index = s.indexOf('m');
```

```
int index = s.indexOf('C');
```

```
int index = s.lastIndexOf('C');
```

```
int index = s.lastIndexOf('B');
```

We can specify a starting point for the search using these forms:

*int indexOf(int ch, int startIndex)*

*int lastIndexOf(int ch, int startIndex) int indexOf(String str, int startIndex)*

*int lastIndexOf(String str, int startIndex)*

Here, `startIndex` specifies the index at which point the search begins.

For `indexOf( )`, the search runs from `startIndex` to the end of the string.

For `lastIndexOf( )`, the search runs from `startIndex` to zero.

## Example

```
class indexOfDemo {  
    public static void main(String args[])  
    {  
        String s = "Hello Students" + "Welcome to Java .";  
        System.out.println(s);  
        System.out.println("indexOf(t) = " + s.indexOf('t'));  
        System.out.println("lastIndexOf(t) = " + s.lastIndexOf('t'));  
        System.out.println("indexOf(the) = " + s.indexOf("the"));  
        System.out.println("lastIndexOf(the) = " + s.lastIndexOf("the"));  
        System.out.println("indexOf(t, 10) = " + s.indexOf('t', 10));  
        System.out.println("lastIndexOf(t, 60) = " + s.lastIndexOf('t', 60));  
        System.out.println("indexOf(the, 10) = " + s.indexOf("the", 10));  
        System.out.println("lastIndexOf(the, 60) = " + s.lastIndexOf("the", 60));  
    }  
}
```

## Modifying a String

Because String objects are immutable, whenever we want to modify a String, it will construct a new copy of the string with modifications.

`substring()`: used to extract a part of a string.

*`public String substring (int start_index)`*

*`public String substring (int start_index, int end_index)`*

Example: `String s = "ABCDEFGH";`

`String t = s.substring(2);    System.out.println (t); String u = s.substring (1, 4);`

`System.out.println (u);`

*Note: Substring from start\_index to end\_index-1 will be returned.*

**concat( ):** used to concatenate two strings.

*String concat(String str)*

- This method creates a new object that contains the invoking string with the contents of str appended to the end.
- concat( ) performs the same function as +.      Example:
- String s1 = "one"; String s2 = s1.concat("two");
- It generates the same result as the following sequence: String s1 = "one"; String s2 = s1 + "two";

**replace( ):** The replace( ) method has two forms.

The first replaces all occurrences of one character in the invoking string with another character. It has the following general form:

*String replace(char original, char replacement)*

Here, original specifies the character to be replaced by the character specified by replacement.

**Example:** String s = "Hello".replace('l', 'w');

The second form of replace( ) replaces one character sequence with another. It has this general form:

*String replace(CharSequence original, CharSequence replacement)*

## **trim( )**

The trim( ) method returns a copy of the invoking string from which any leading and trailing whitespace has been removed.

*String trim( )*

Example:

```
String s = "    Hello World    ".trim();
```

This puts the string “Hello World” into s.

# StringBuilder class

A **mutable** sequence of characters (Unlike String which is immutable)

## Constructors

### **StringBuilder()**

Constructs a string builder with no characters in it and an initial capacity of 16 characters

### **StringBuilder(int capacity)**

Constructs a string builder with no characters in it and an initial capacity specified by the capacity argument.

### **StringBuilder(String str)**

Constructs a string builder initialized to the contents of the specified string



## StringBuilder class – important methods

### ***StringBuilder append(String str)***

Appends the specified string to this character sequence

### ***int capacity()***

Returns the current capacity

### ***StringBuilder delete(int start, int end)***

Removes the characters in a substring of this sequence

### ***StringBuilder insert(int offset, String str)***

Inserts the string into this character sequence

*Note: Can also work with other datatypes like Boolean, integer, float, double and Objects...*

# StringBuffer class - Constructors

A **thread-safe, mutable** sequence of characters

## Constructors

### **StringBuffer()**

Constructs a string builder with no characters in it and an initial capacity of 16 characters

### **StringBuffer(int capacity)**

Constructs a string builder with no characters in it and an initial capacity specified by the capacity argument.

### **StringBuffer(String str)**

Constructs a string builder initialized to the contents of the specified string

## StringBuffer class – Important methods

All methods are synchronized

***StringBuffer append(String str)***

Appends the specified string to this character sequence

***int capacity()***

Returns the current capacity

***StringBuffer delete(int start, int end)***

Removes the characters in a substring of this sequence

***StringBuffer insert(int offset, String str)***

Inserts the string into this character sequence

*Note: Can also work with other datatypes like Boolean, integer, float, double and Objects...*

# Packages

- Package is a container for classes.
- A **java package is a group of similar types of classes, interfaces and sub-packages.**
- Package in java can be categorized in two form:
  - 1) built-in package :- Existing Java package (ex: java, lang, awt, javax, swing, net, io, util, sql etc)
  - 2) user-defined package:- Java package created by user(ex: p1, mypack1, p2 etc.)
- Packages are stored in a hierarchical manner and are explicitly imported into new class definitions.
- You can define classes inside a package that are not accessible by code outside that package.
- You can also define class members that are only known to other members of the same package.

## Advantage of Java Package

- 1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- 2) Java package provides access protection.
- 3) Java package removes naming collision.
- 4) To make searching/locating and usage of classes easily.



- You can create a package by just including a package command as the first statement in a Java source file. Any classes declared within that file will be the member of that specified package.
- The package statement defines a name space in which classes are stored. If you omit the package statement, the class names are put into the default package, which has no name.

**Syntax: package packagename;**

**EX : package MyPackage;**

**OR package java.awt.image;**

**OR package pkg1.pkg2.pkg3;**

- Java uses file system directories to store packages.
- For example, if you have package MyPackage then the .class files for any classes you declare to be part of MyPackage must be stored in a directory called MyPackage.

**Filename: A.java (Same package class)**

```
package p1;  
class A  
{  
    void showA()  
    {  
        System.out.println("I am A class of same package");  
    }  
}
```

**Filename: B.java (Same package class)**

```
package p1;  
class B  
{  
    void showB()  
    {  
        System.out.println("I am B of same package");  
    }  
}
```





**Filename: ABC.java(Same package class)**

```
package p1;  
class ABC  
{  
    public static void main(String s[])  
    {  
        A a1=new A();  
        a1.showA();  
        B b1=new B();  
        b1.showB();  
    }  
}
```



## Output:

```
C:\Users\PANKAJ PATEL\Desktop\package eg>javac -d . a.java
C:\Users\PANKAJ PATEL\Desktop\package eg>javac -d . b.java
C:\Users\PANKAJ PATEL\Desktop\package eg>javac -d . abc.java
C:\Users\PANKAJ PATEL\Desktop\package eg>java p1/abc
I am class A of same package
I am class B of same package
```

```
C:\Users\PANKAJ PATEL\Desktop\package eg>java p1.abc
I am class A of same package
I am class B of same package
```

## Import a package

### Syntax:

```
import pkg1[.pkg2].(classname | *);
```

### Ex:

builtin package: import java.util.Date;

or import java.io.\*;

User define package : import p1.D;

or import p1.\*;

# × ○ DIGITAL LEARNING CONTENT



## Parul<sup>®</sup> University



[www.paruluniversity.ac.in](http://www.paruluniversity.ac.in)

