# JAVA

**Java**

* A programming Language
* OOPS concept follows.
* It’s a platform too, so it is called robust.
* Platform (JRE- > Java Runtime Environment)

**Why Java?**

* Simple
* Object Oriented
* Portable
* Platform independent
* Robust
* Multi threaded
* High Performances

Simple

* Syntax is based upon C++
* Automatic Garbage Collection in Java

### Object-oriented

* Java is an [object-oriented](https://www.javatpoint.com/java-oops-concepts) programming language.
* Objects that incorporates both data and behavior.
* Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. [Object](https://www.javatpoint.com/object-and-class-in-java)
2. Class
3. [Inheritance](https://www.javatpoint.com/inheritance-in-java)
4. [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
5. [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
6. [Encapsulation](https://www.javatpoint.com/encapsulation)

# Features of Java

The primary objective of [Java programming](https://www.javatpoint.com/java-tutorial) language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as java buzzwords.

A list of most important features of Java language is given below.

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

### Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

* Java syntax is based on C++ (so easier for programmers to learn it after C++).
* Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
* There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

### Object-oriented

Java is an [object-oriented](https://www.javatpoint.com/java-oops-concepts) programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. [Object](https://www.javatpoint.com/object-and-class-in-java)
2. Class
3. [Inheritance](https://www.javatpoint.com/inheritance-in-java)
4. [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
5. [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
6. [Encapsulation](https://www.javatpoint.com/encapsulation)

**Platform Independent**

It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

****

### Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

* **No explicit pointer**
* **Java Programs run inside a virtual machine sandbox**

### Robust

* Strong memory management.
* No pointers
* Automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
* Exception handling

### Portable

Carry the Java byte code to any platform.

### Multi-threaded

* A thread is like a separate program, executing concurrently.
* We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread.
* It shares a common memory area.
* Threads are important for multi-media, Web applications, etc.

Compile Time

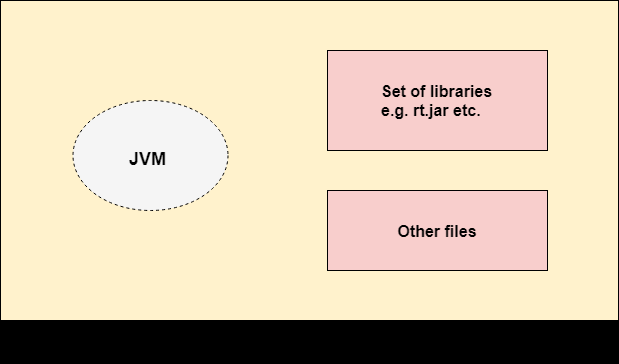
* At compile time java file is complied by Java complier and converts the java code into Byte code.

### JVM

* It physically NOT exists.

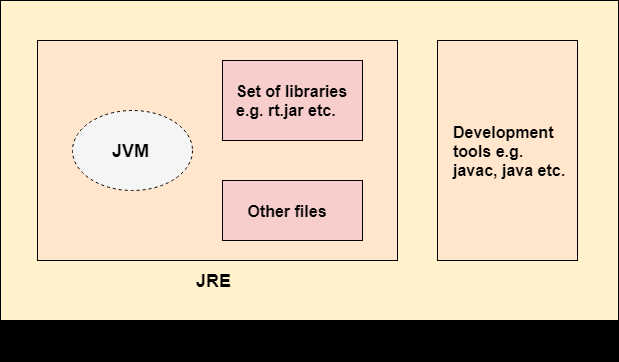
### JRE

* Java Runtime Environment.
* A set of software tools which are used for developing Java applications.
* It is used to provide the runtime environment.
* It is the implementation of JVM.
* It physically exists.
* It contains a set of libraries + other files that JVM uses at runtime.



### JDK

* It physically exists.
* It contains JRE + development tools.



## **Variable**

* It is name of reserved *area allocated in memory*.
* In other words, it is a *name of memory location*.

### Types of Variables

There are three types of variables in java:

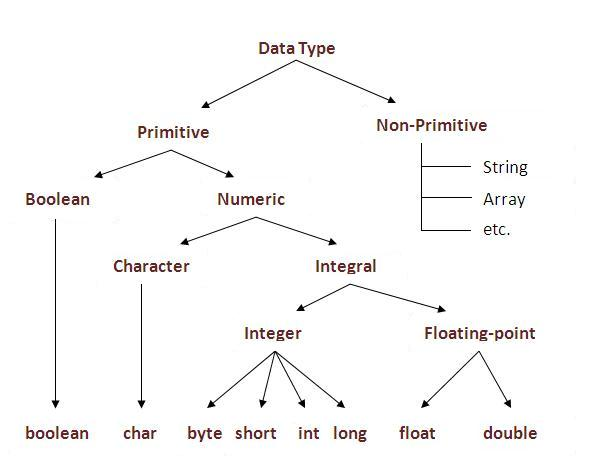
* local variable
  + declared inside the body of the method
  + use this variable only within that method
  + Local variable cannot be defined with "static" keyword.
  + Memory allocated during runtime only.
* instance variable
  + declared inside the class but outside the body of the method
  + It is not declared as static.
  + It is called instance variable because its value is instance specific and is not shared among instances.
  + Memory allocated during runtime only.
* static variable
  + A variable which is declared as static is called static variable.
  + It cannot be local.
  + You can create a single copy of static variable and share among all the instances of the class.
  + Memory allocation for static variable happens only once when the class is loaded in the memory.

Data Types in Java

-> Data types specify the different sizes and values that can be stored in the variable.

There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include Boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.



|  |  |  |
| --- | --- | --- |
| **Data Type** | **Default Value** | **Default size** |
| Boolean | False | 1 bit |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

Java Continue Statement

* The continue statement is used to jump to the next iteration of the loop immediately.
* It can be used with ‘For loop’ or ‘While loop’.

jump-statement;

**continue**;

# Objects and Classes in Java

**Object**

* An entity that has state and behavior is known as an object

**Object Definitions:**

* An object is *a real-world entity*.
* An object is *a runtime entity*.
* The object is *an entity which has state and behavior*.
* The object is *an instance of a class*.

**Class**

* group of objects

A class in Java can contain:

* **Fields**
* **Methods**
* **Constructors**
* **Blocks**
* **Nested class and interface**

**Inheritance**

* Inherit the parent’s behavior & property
* To provide code reusability.

**Polymorphism**

* When one task is performed by different ways.

**Abstraction**

* Hiding internal details and showing functionality

**Encapsulation**

* Binding code & data together into a single unit.

Java Naming conventions

|  |  |
| --- | --- |
| **Name** | **Convention** |
| class name | should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc. |
| interface name | should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc. |
| method name | should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc. |
| variable name | should start with lowercase letter e.g. firstName, orderNumber etc. |
| package name | should be in lowercase letter e.g. java, lang, sql, util etc. |
| constants name | Should be in uppercase letter. e.g. RED, YELLOW, MAX\_PRIORITY etc. |
| Keywords  (this, static) | should be in lowercase letter |

## **Camel Case in java naming conventions**

Java follows camel case syntax for naming the class, interface, method and variable.

If name is combined with two words, second word will start with uppercase letter always e.g. actionPerformed(), firstName, ActionEvent, ActionListener etc.

### Method in Java

* Code Reusability
* Code Optimization

### new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

Main method

* Whenever we start execution that the control start from main method.
* Control & execution will begin from main method only.

Class

* That the complete java code written on class file only.
* If we have multiple class in single java file then file name should be save as where the main method is present.

**Constructor**

* block of codes similar to the method
* is used to initialize the state of an object.

### Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. no return type
3. A Java constructor cannot be abstract, static, final, and synchronized

## **Types of Java constructors**

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

## **Java Default Constructor**

A constructor is called "Default Constructor" when it doesn't have any parameter.

#### Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

### What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

### Java Parameterized Constructor

### A constructor which has a specific number of parameters is called a parameterized constructor.

### Why use the parameterized constructor?

The parameterized constructor is used **to provide different values** to the distinct objects. However, you can provide the same values also.

## **Constructor Overloading in Java**

* Having more than one constructor with different parameter lists.

## **Difference between constructor and method in Java**

There are many differences between constructors and methods. They are given above.

|  |  |
| --- | --- |
| **Constructor** | **Method** |
| A constructor is used to initialize the state of an object. | A method is used to expose the behavior of an object. |
| A constructor must not have a return type. | A method must have a return type. |
| The constructor is invoked implicitly. | The method is invoked explicitly. |
| The Java compiler provides a default constructor if you don't have any constructor in a class. | The method is not provided by the compiler in any case. |
| The constructor name must be same as the class name. | The method name may or may not be same as class name. |

# Java static keyword

* Is used for memory management mainly.
* We can apply java static keyword with variables, methods, blocks and nested class.
* The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

## **1) Java static variable**

If you declare any variable as static, it is known as a static variable.

* The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
* The static variable gets memory only once in the class area at the time of class loading.

### Advantages of static variable

It makes your program **memory efficient** (i.e., it saves memory).

## **2) Java static method**

If you apply static keyword with any method, it is known as static method.

* A static method belongs to the class rather than the object of a class.
* A static method can be invoked without the need for creating an instance of a class.
* A static method can access static data member and can change the value of it.

### Restrictions for the static method

There are two main restrictions for the static method. They are:

1. The static method cannot use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

### Why is the Java main method static?

It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main () method that will lead the problem of extra memory allocation.

## **3) Java static block**

* Is used to initialize the static data member.
* It is executed before the main method at the time of class loading.

### Can we execute a program without main () method?

Ans) No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a java class without the main method.

this keyword in java

There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.

## **Usage of java this keyword**

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

### 1) this: to refer current class instance variable

The ‘this’ keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

### 2) this: to invoke current class method

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method.

### 3) this() : to invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

### Real usage of this() constructor call

The this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining.

#### Rule: Call to this() must be the first statement in constructor.

### 4) this: to pass as an argument in the method

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling.

### 5) this: to pass as argument in the constructor call

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes.

### 6) this keyword can be used to return current class instance

We can return this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive).

# Super Keyword in Java

* The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

## **Usage of Java super Keyword**

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

## **1) super is used to refer immediate parent class instance variable.**

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

## **2) super can be used to invoke parent class method**

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

## **3) super is used to invoke parent class constructor.**

The super keyword can also be used to invoke the parent class constructor.

## **super() & this()**

* super() - to call parent class constructor.
* this() - to call same class constructor.

**NOTE:**

* We can use super() and this() only in constructor not anywhere else, any attempt to do so will lead to compile-time error.
* We have to keep either super() or this() as the first line of the constructor but NOT both simultaneously.

## **super & this keyword**

* super - to call parent class members(variables and methods).
* this - to call same class members(variables and methods).

**NOTE:** We can use both of them anywhere in a class except static areas(static block or method), any attempt to do so will lead to compile-time error.

# Inheritance in Java

* one object acquires all the properties and behaviors of a parent object.
* can reuse methods and fields of the parent class.
* Moreover, you can add new methods and fields in your current class also.
* Inheritance represents the **IS-A relationship** which is also known as a parent-child relationship.

### Why use inheritance in java

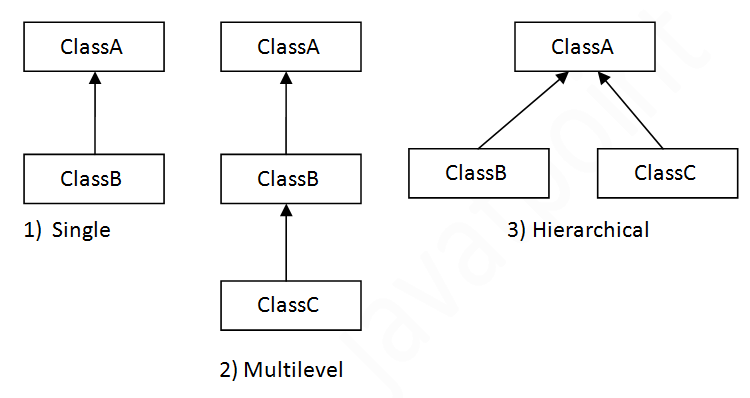
* For Method Overriding (so runtime polymorphism can be achieved).
* For Code Reusability.

Note-

* The **extends keyword** indicates that you are making a new class that derives from an existing class.
* a class which is inherited is called a parent or super class, and the new class is called child or subclass.

## **Types of inheritance in java**

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.



#### Note: Multiple inheritance is not supported in Java through class.

# Method Overloading in Java

* multiple methods having same name but different in parameters

## **Advantage of method overloading**

Method overloading increases the readability of the program.

### Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

#### In java, Method Overloading is not possible by changing the return type of the method only.

### Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only.

Method Overriding in Java

* If subclass (child class) has the same method as declared in the parent class.
* If a subclass provides the specific implementation of the method that has been declared by one of its parent class.

### Usage of Java Method Overriding

* used to provide the specific implementation of a method which is already provided by its superclass.
* used for runtime polymorphism

#### Rules for Java Method Overriding

1. The method must have the same name and parameter as in the parent class
2. There must be an IS-A relationship (inheritance).

### Can we override static method?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | | **Method Overloading** | | **Method Overriding** |
| 1) | Used to increase the readability of the program. | | Used to provide the specific implementation of the method that is already provided by its super class. | |
| 2) | Performed within class. | | Occurs in two classes that have IS-A (inheritance) relationship. | |
| 3) | Parameter must be different. | | Parameter must be same. | |
| 4) | Compile time polymorphism. | | Run time polymorphism. | |
| 5) | Return type can be same or different . | | Return type must be same or covariant | |

No, a static method cannot be overridden. Because the static method is bound with class whereas instance method is bound with an object. Even main method also cant override due to static.

## **Java Method Overloading example**

**class** OverloadingExample{

**static** **int** add(**int** a,**int** b){**return** a+b;}

**static** **int** add(**int** a,**int** b,**int** c){**return** a+b+c;}

}

## **Java Method Overriding example**

**class** Animal{

**void** eat(){System.out.println("eating...");}

}

**class** Dog **extends** Animal{

**void** eat(){System.out.println("eating bread...");}

}

# Final Keyword

* used to restrict the user.

Final can be:

1. variable
2. method
3. class

**Note-**

* a final variable that have no value it is called blank final variable or uninitialized final variable.
* It can be initialized in the constructor only.
* The blank final variable can be static also which will be initialized in the static block only.

## **final variable**

If you make any variable as final, you cannot change the value of final variable(It will be constant).

## **final method**

If you make any method as final, you cannot override it.

## **final class**

If you make any class as final, you cannot extend it.

### final method inherited?

Yes, final method is inherited but you cannot override it.

### What is blank or uninitialized final variable?

* A final variable that is not initialized at the time of declaration is known as blank final variable.
* If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee.
* It can be initialized only in constructor.

**class** Bike10{

**final** **int** speedlimit;//blank final variable

  Bike10(){

  speedlimit=70;

  System.out.println(speedlimit);

  }

**public** **static** **void** main(String args[]){

**new** Bike10();

 }

}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Bike10)

Output: 70

### Can we declare a constructor final?

No, because **constructor is never inherited.**

# Polymorphism in Java

**-** a single action in different ways.

- polymorphism means many forms.

Types

* compile-time polymorphism
* runtime polymorphism.
* We can perform polymorphism in java by method overloading and method overriding.

If you overload a static method in Java, it is the example of **compile time polymorphism**.

## **Runtime Polymorphism in Java**

**Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass.

### Upcasting

* If the reference variable of Parent class refers to the object of Child class, it is known as upcasting.
* For upcasting, we can use the reference variable of class type or an interface type

**class** Bank{

**float** getRateOfInterest(){**return** 0;}

}

**class** SBI **extends** Bank{

**float** getRateOfInterest(){**return** 8.4f;}

}

**class** ICICI **extends** Bank{

**float** getRateOfInterest(){**return** 7.3f;}

}

**class** AXIS **extends** Bank{

**float** getRateOfInterest(){**return** 9.7f;}

}

**class** TestPolymorphism{

**public** **static** **void** main(String args[]){

Bank b;

b=**new** SBI();

System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());

b=**new** ICICI();

System.out.println("ICICI Rate of Interest: "+b.getRateOfInterest());

b=**new** AXIS();

System.out.println("AXIS Rate of Interest: "+b.getRateOfInterest());

}

}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestPolymorphism)

Output:

SBI Rate of Interest: 8.4

ICICI Rate of Interest: 7.3

AXIS Rate of Interest: 9.7

## **Java Runtime Polymorphism with Data Member**

A method is overridden, not the data members, so runtime polymorphism can't be achieved by data members.

# Abstract class in Java

* A class which is declared with the abstract keyword.
* It can have abstract and non-abstract methods (method with the body).

### Abstraction in Java

* **It** is a process of hiding the implementation details and showing only functionality to the user.
* Another way, it shows only essential things to the user and hides the internal details,

### Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

### Abstract class in Java

It needs to be extended and its method implemented. It cannot be instantiated.

#### Points to Remember

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have constructors and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.

### Another example of Abstract class in java

*File: TestBank.java*

**abstract** **class** Bank{

**abstract** **int** getRateOfInterest();

}

**class** SBI **extends** Bank{

**int** getRateOfInterest(){**return** 7;}

}

**class** PNB **extends** Bank{

**int** getRateOfInterest(){**return** 8;}

}

**class** TestBank{

**public** **static** **void** main(String args[]){

Bank b;

b=**new** SBI();

System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");

b=**new** PNB();

System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestBank)

Rate of Interest is: 7 %

Rate of Interest is: 8 %

### Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

### Abstract class having constructor, data member and methods

An abstract class can have a data member, abstract method, method body (non-abstract method), constructor, and even main() method.

#### Rule 1: If there is an abstract method in a class, that class must be abstract.

#### Rule 2: If you are extending an abstract class that has an abstract method, you must either provide the implementation of the method or make this class abstract.

# Interface in Java

* It is a blueprint of a class.
* It has static constants and abstract methods.
* The interface in Java is a mechanism to achieve abstraction.
* There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

It cannot be instantiated just like the abstract class.

Since Java 8, we can have **default and static methods** in an interface.

Since Java 9, we can have **private methods** in an interface.

## **Why use Java interface?**

There are mainly three reasons to use interface. They are given below.

* It is used to achieve abstraction.
* By interface, we can support the functionality of multiple inheritance.
* It can be used to achieve loose coupling.

## **How to declare an interface?**

*File: TestInterface2.java*

**interface** Bank{

**float** rateOfInterest();

}

**class** SBI **implements** Bank{

**public** **float** rateOfInterest(){**return** 9.15f;}

}

**class** PNB **implements** Bank{

**public** **float** rateOfInterest(){**return** 9.7f;}

}

**class** TestInterface2{

**public** **static** **void** main(String[] args){

Bank b=**new** SBI();

System.out.println("ROI: "+b.rateOfInterest());

}}

* An interface is declared by using the interface keyword.
* It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
* A class that implements an interface must implement all the methods declared in the interface.

#### The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.

#### The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.

**Multiple inheritance in Java by interface**

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



## **Multiple inheritance is not supported through class in java, but it is possible by an interface, why?**

As we have explained in the inheritance chapter, multiple inheritance is not supported in the case of class because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class.

## **Java 8 Default Method in Interface**

Since Java 8, we can have method body in interface. But we need to make it default method.

## **Java 8 Static Method in Interface**

Since Java 8, we can have static method in interface.

## **What is marker or tagged interface?**

An interface which has no member is known as a marker or tagged interface, for example, Serializable, Cloneable, Remote, etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

#### Nested Interface in Java

An interface can have another interface which is known as a nested interface.

# Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%)

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface class** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

# Java Package

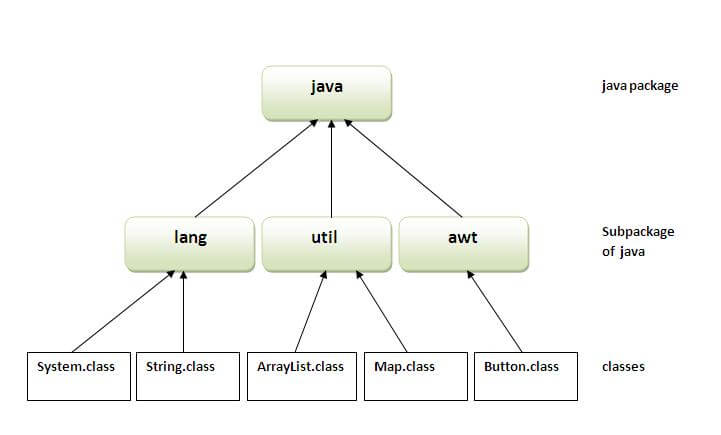
* It is a group of similar types of classes, interfaces and sub-packages.
* Categorized in two form, built-in package and user-defined package.
* There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

## **Advantage of Java Package**

1) Used to categorize the classes and interfaces so that they can be easily maintained.

2) Provides access protection.

3) Removes naming collision.



## **How to access package from another package?**

There are three ways to access the package from outside the package.

1. import package.\*;
2. import package.classname;
3. fully qualified name.

#### 1) Using packagename.\*

- all the classes and interfaces of this package will be accessible but not subpackages.

- The import keyword is used to make the classes and interface of another package accessible to the current package.

#### 2) Using packagename.classname

- only declared class of this package will be accessible.

#### 3) Using fully qualified name

- only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

- It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

#### Note: If you import a package, subpackages will not be imported.

#### Note: Sequence of the program must be package then import then class.

### Ways to load the class files or jar files

|  |
| --- |
| There are two ways to load the class files temporary and permanent. |

* Temporary
  + By setting the classpath in the command prompt
  + By -classpath switch
* Permanent
  + By setting the classpath in the environment variables
  + By creating the jar file, that contains all the class files, and copying the jar file in the jre/lib/ext folder.

#### Rule: There can be only one public class in a java source file and it must be saved by the public class name.

### How to put two public classes in a package?

|  |
| --- |
| If you want to put two public classes in a package, have two java source files containing one public class,  but keep the package name same. |

# Package class

- to get information about the specification and implementation of a package.

- It provides methods such as getName(), getImplementationTitle(), getImplementationVendor(), getImplementationVersion() etc.

# Access Modifiers in java

two types of modifiers in java:

- **access modifiers**

**- non-access modifiers**

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

### 1) private access modifier

|  |
| --- |
| - is accessible only within class. |

### Role of Private Constructor

|  |
| --- |
| If you make any class constructor private, you cannot create the instance of that class from outside the class. |

### 2) default access modifier

|  |
| --- |
| If you don't use any modifier, it is treated as **default** bydefault.  The default modifier is accessible only within package. |

### 3) protected access modifier

The **protected access modifier** is accessible within package and outside the package but through **inheritance** only.

The protected access modifier can be applied on the data member, method and constructor. **It can't be applied on the class.**

### 4) public access modifier

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers. | | | | | |
| **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** | |
| **Private** | Y | N | N | N | |
| **Default** | Y | Y | N | N | |
| **Protected** | Y | Y | Y | N | |
| **Public** | Y | Y | Y | Y | |

### Java access modifiers with method overriding

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

Encapsulation in Java

* is a process of wrapping code and data together into a single unit.
* We can create a fully encapsulated class in Java by making all the data members of the class private.
* Now we can use setter and getter methods to set and get the data in it.
* **Java Bean** class is the example of a fully encapsulated class.

### Advantage of Encapsulation in Java

* By providing only a setter or getter method, you can make the class **read-only or write-only**. In other words, you can skip the getter or setter methods.
* It provides you the **control over the data**.
* It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.
* The encapsulate class is **easy to test**. So, it is better for unit testing.
* The standard IDE's are providing the facility to generate the getters and setters. So, it is **easy and fast to create an encapsulated class** in Java.

Java String

- string is basically an object that represents sequence of char values.

- An array of characters works same as Java string.

**char**[] ch={'j','a','v','a','t','p','o','i','n','t'};

String s=**new** String(ch);

is same as:

String s="javatpoint";

**Java String** class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.

## **CharSequence Interface**

* Can create a string using these three classes(**String**, **StringBuffer** and **StringBuilder** classes)
* The Java **String** is **immutable** which means it cannot be changed. Whenever we change any string, a new instance is created. For **mutable** strings, you can use **StringBuffer** and **StringBuilder** classes.

### What is String in java

* String is a sequence of characters.
* But in Java, string is an object that represents a sequence of characters.
* The java.lang.String class is used to create a string object.

### How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

### 1) String Literal

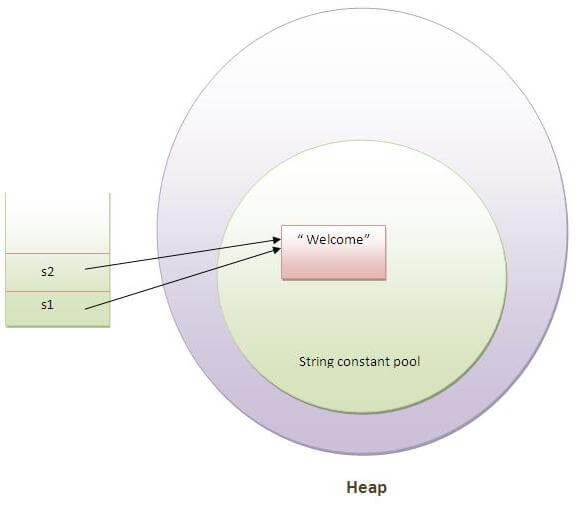
Java String literal is created by using double quotes.

String s="welcome";

* Each time you create a string literal, the JVM checks the "string constant pool" first.
* If the string already exists in the pool, a reference to the pooled instance is returned.
* If the string doesn't exist in the pool, a new string instance is created and placed in the pool.

String s1="Welcome";

String s2="Welcome";//It doesn't create a new instance



In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool, that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

#### Note: String objects are stored in a special memory area known as the "string constant pool".

### Why Java uses the concept of String literal?

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

### 2) By new keyword

String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

### Java String Example

**public** **class** StringExample{

**public** **static** **void** main(String args[]){

String s1="java";//creating string by java string literal

**char** ch[]={'s','t','r','i','n','g','s'};

String s2=**new** String(ch);//converting char array to string

String s3=**new** String("example");//creating java string by new keyword

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=StringExample)

java

strings

example

String Methods

# Java String charAt()

The **java string charAt()** method returns a char value at the given index number.

## **Java String charAt() method example**

**public** **class** CharAtExample{

**public** **static** **void** main(String args[]){

String name="javatpoint";

**char** ch=name.charAt(4);//returns the char value at the 4th index

System.out.println(ch);

}}

# Java String compareTo()

The **java string compareTo()** method compares the given string with current string lexicographically. It returns positive number, negative number or 0.

## **Java String compareTo() method example**

**public** **class** CompareToExample{

**public** **static** **void** main(String args[]){

String s1="hello";

String s2="hello";

String s3="meklo";

String s4="hemlo";

String s5="flag";

System.out.println(s1.compareTo(s2));//0 because both are equal

System.out.println(s1.compareTo(s3));//-5 because "h" is 5 times lower than "m"

System.out.println(s1.compareTo(s4));//-1 because "l" is 1 times lower than "m"

System.out.println(s1.compareTo(s5));//2 because "h" is 2 times greater than "f"

}}

# Java String concat

The **java string concat()** method combines specified string at the end of this string. It returns combined string. It is like appending another string.

## **Java String concat() method example**

**public** **class** ConcatExample{

**public** **static** **void** main(String args[]){

String s1="java string";

s1.concat("is immutable");

System.out.println(s1);

s1=s1.concat(" is immutable so assign it explicitly");

System.out.println(s1);

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=ConcatExample)

java string

java string is immutable so assign it explicitly

# Java String contains()

The **java string contains()** method searches the sequence of characters in this string. It returns *true* if sequence of char values are found in this string otherwise returns *false*.

## **Java String contains() method example**

**class** ContainsExample{

**public** **static** **void** main(String args[]){

String name="what do you know about me";

System.out.println(name.contains("do you know"));

System.out.println(name.contains("about"));

System.out.println(name.contains("hello"));

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=ContainsExample)

true

true

false

# Java String endsWith()

The **java string endsWith()** method checks if this string ends with given suffix. It returns true if this string ends with given suffix else returns false.

## **Java String endsWith() method example**

**public** **class** EndsWithExample{

**public** **static** **void** main(String args[]){

String s1="java by javatpoint";

System.out.println(s1.endsWith("t"));

System.out.println(s1.endsWith("point"));

}}

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=EndsWithExample)

Output:

true

true

# Java String equals()

The **java string equals()** method compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.

## **Java String equals() method example**

**public** **class** EqualsExample{

**public** **static** **void** main(String args[]){

String s1="javatpoint";

String s2="javatpoint";

String s3="JAVATPOINT";

String s4="python";

System.out.println(s1.equals(s2));//true because content and case is same

System.out.println(s1.equals(s3));//false because case is not same

System.out.println(s1.equals(s4));//false because content is not same

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=EqualsExample)

true

false

false

# Java String equalsIgnoreCase()

The **String equalsIgnoreCase()** method compares the two given strings on the basis of content of the string irrespective of case of the string. It is like equals() method but doesn't check case. If any character is not matched, it returns false otherwise it returns true.

## **Java String equalsIgnoreCase() method example**

**public** **class** EqualsIgnoreCaseExample{

**public** **static** **void** main(String args[]){

String s1="javatpoint";

String s2="javatpoint";

String s3="JAVATPOINT";

String s4="python";

System.out.println(s1.equalsIgnoreCase(s2));//true because content and case both are same

System.out.println(s1.equalsIgnoreCase(s3));//true because case is ignored

System.out.println(s1.equalsIgnoreCase(s4));//false because content is not same

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=EqualsIgnoreCaseExample)

true

true

false

# Java String format()

The **java string format()** method returns the formatted string by given locale, format and arguments.

If you don't specify the locale in String.format() method, it uses default locale by calling *Locale.getDefault()*method.

The format() method of java language is like *sprintf()* function in c language and *printf()* method of java language.

## **Java String format() method example**

**public** **class** FormatExample{

**public** **static** **void** main(String args[]){

String name="sonoo";

String sf1=String.format("name is %s",name);

String sf2=String.format("value is %f",32.33434);

String sf3=String.format("value is %32.12f",32.33434);//returns 12 char fractional part filling with 0

System.out.println(sf1);

System.out.println(sf2);

System.out.println(sf3);

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=FormatExample)

name is sonoo

value is 32.334340

value is 32.334340000000

# Java String getBytes()

The **java string getBytes()** method returns the byte array of the string. In other words, it returns sequence of bytes.

## **Java String getBytes() method example**

**public** **class** StringGetBytesExample{

**public** **static** **void** main(String args[]){

String s1="ABCDEFG";

**byte**[] barr=s1.getBytes();

**for**(**int** i=0;i<barr.length;i++){

System.out.println(barr[i]);

}

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=StringGetBytesExample)

Output:

65

66

67

68

69

70

71

# Java String getChars()

The **java string getChars()** method copies the content of this string into specified char array. There are 4 arguments passed in getChars() method. The signature of getChars() method is given below:

## **Java String getChars() method example**

**public** **class** StringGetCharsExample{

**public** **static** **void** main(String args[]){

 String str = **new** String("hello javatpoint how r u");

**char**[] ch = **new** **char**[10];

**try**{

         str.getChars(6, 16, ch, 0);

         System.out.println(ch);

      }**catch**(Exception ex){System.out.println(ex);}

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=StringGetCharsExample)

Output:

javatpoint

# Java String indexOf()

The **java string indexOf()** method returns index of given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

## **Java String indexOf() method example**

**public** **class** IndexOfExample{

**public** **static** **void** main(String args[]){

String s1="this is index of example";

//passing substring

**int** index1=s1.indexOf("is");//returns the index of is substring

**int** index2=s1.indexOf("index");//returns the index of index substring

System.out.println(index1+"  "+index2);//2 8

//passing substring with from index

**int** index3=s1.indexOf("is",4);//returns the index of is substring after 4th index

System.out.println(index3);//5 i.e. the index of another is

//passing char value

**int** index4=s1.indexOf('s');//returns the index of s char value

System.out.println(index4);//3

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=IndexOfExample)

2 8

5

3

# Java String intern()

The **java string intern()** method returns the interned string. It returns the canonical representation of string.

It can be used to return string from memory, if it is created by new keyword. It creates exact copy of heap string object in string constant pool.

## **Java String intern() method example**

**public** **class** InternExample{

**public** **static** **void** main(String args[]){

String s1=**new** String("hello");

String s2="hello";

String s3=s1.intern();//returns string from pool, now it will be same as s2

System.out.println(s1==s2);//false because reference variables are pointing to different instance

System.out.println(s2==s3);//true because reference variables are pointing to same instance

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=InternExample)

false

true

# Java String isEmpty()

The **java string isEmpty()** method checks if this string is empty or not. It returns *true*, if length of string is 0 otherwise false. In other words, true is returned if string is empty otherwise it returns false.

The isEmpty() method of String class is included in java string since JDK 1.6.

The **java string join()** method returns a string joined with given delimiter. In string join method, delimiter is copied for each elements.

In case of null element, "null" is added. The join() method is included in java string since JDK 1.8.

## **Java String isEmpty() method example**

**public** **class** IsEmptyExample{

**public** **static** **void** main(String args[]){

String s1="";

String s2="javatpoint";

System.out.println(s1.isEmpty());

System.out.println(s2.isEmpty());

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=IsEmptyExample)

true

false

# Java String join()

The **java string join()** method returns a string joined with given delimiter. In string join method, delimiter is copied for each elements.

In case of null element, "null" is added. The join() method is included in java string since JDK 1.8.

There are two types of join() methods in java string.

## **Java String join() method example**

**public** **class** StringJoinExample{

**public** **static** **void** main(String args[]){

String joinString1=String.join("-","welcome","to","javatpoint");

System.out.println(joinString1);

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=StringJoinExample)

welcome-to-javatpoint

# Java String lastIndexOf()

The **java string lastIndexOf()** method returns last index of the given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

## **Java String lastIndexOf() method example**

**public** **class** LastIndexOfExample{

**public** **static** **void** main(String args[]){

String s1="this is index of example";//there are 2 's' characters in this sentence

**int** index1=s1.lastIndexOf('s');//returns last index of 's' char value

System.out.println(index1);//6

}}

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=LastIndexOfExample)

Output:

6

# Java String length()

The **java string length()** method length of the string. It returns count of total number of characters. The length of java string is same as the unicode code units of the string.

## **Java String length() method example**

**public** **class** LengthExample{

**public** **static** **void** main(String args[]){

String s1="javatpoint";

String s2="python";

System.out.println("string length is: "+s1.length());//10 is the length of javatpoint string

System.out.println("string length is: "+s2.length());//6 is the length of python string

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=LengthExample)

string length is: 10

string length is: 6

# Java String replace()

The **java string replace()** method returns a string replacing all the old char or CharSequence to new char or CharSequence.

Since JDK 1.5, a new replace() method is introduced, allowing you to replace a sequence of char values.

## **Java String replace(char old, char new) method example**

**public** **class** ReplaceExample1{

**public** **static** **void** main(String args[]){

String s1="javatpoint is a very good website";

String replaceString=s1.replace('a','e');//replaces all occurrences of 'a' to 'e'

System.out.println(replaceString);

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=ReplaceExample1)

jevetpoint is e very good website

# Java String replaceAll()

The **java string replaceAll()** method returns a string replacing all the sequence of characters matching regex and replacement string.

## **Java String replaceAll() example: replace character**

Let's see an example to replace all the occurrences of **a single character**.

**public** **class** ReplaceAllExample1{

**public** **static** **void** main(String args[]){

String s1="javatpoint is a very good website";

String replaceString=s1.replaceAll("a","e");//replaces all occurrences of "a" to "e"

System.out.println(replaceString);

}}

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=ReplaceAllExample1)

jevetpoint is e very good website

# Java String split()

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

## **Java String split() method example**

The given example returns total number of words in a string excluding space only. It also includes special characters.

**public** **class** SplitExample{

**public** **static** **void** main(String args[]){

String s1="java string split method by javatpoint";

String[] words=s1.split("\\s");//splits the string based on whitespace

//using java foreach loop to print elements of string array

**for**(String w:words){

System.out.println(w);

}

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=SplitExample)

java

string

split

method

by

javatpoint

# Java String startsWith()

The **java string startsWith()** method checks if this string starts with given prefix. It returns true if this string starts with given prefix else returns false.

## **Java String startsWith() method example**

**public** **class** StartsWithExample{

**public** **static** **void** main(String args[]){

String s1="java string split method by javatpoint";

System.out.println(s1.startsWith("ja"));

System.out.println(s1.startsWith("java string"));

}}

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=StartsWithExample)

Output:

true

true

# Java String substring()

The **java string substring()** method returns a part of the string.

We pass begin index and end index number position in the java substring method where start index is inclusive and end index is exclusive. In other words, start index starts from 0 whereas end index starts from 1.

There are two types of substring methods in java string.

## **Java String substring() method example**

**public** **class** SubstringExample{

**public** **static** **void** main(String args[]){

String s1="javatpoint";

System.out.println(s1.substring(2,4));//returns va

System.out.println(s1.substring(2));//returns vatpoint

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=SubstringExample)

va

vatpoint

# Java String 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.

## **Java String toCharArray() method example**

**public** **class** StringToCharArrayExample{

**public** **static** **void** main(String args[]){

String s1="hello";

**char**[] ch=s1.toCharArray();

**for**(**int** i=0;i<ch.length;i++){

System.out.print(ch[i]);

}

}}

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=StringToCharArrayExample)

Output:

hello

# Java String toLowerCase()

The **java string toLowerCase()** method returns the string in lowercase letter. In other words, it converts all characters of the string into lower case letter.

The toLowerCase() method works same as toLowerCase(Locale.getDefault()) method. It internally uses the default locale.

## **Java String toLowerCase() method example**

**public** **class** StringLowerExample{

**public** **static** **void** main(String args[]){

String s1="JAVATPOINT HELLO stRIng";

String s1lower=s1.toLowerCase();

System.out.println(s1lower);

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=StringLowerExample)

Output:

javatpoint hello string

# Java String toUpperCase()

The **java string toUpperCase()** method returns the string in uppercase letter. In other words, it converts all characters of the string into upper case letter.

The toUpperCase() method works same as toUpperCase(Locale.getDefault()) method. It internally uses the default locale.

## **Java String toUpperCase() method example**

**public** **class** StringUpperExample{

**public** **static** **void** main(String args[]){

String s1="hello string";

String s1upper=s1.toUpperCase();

System.out.println(s1upper);

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=StringUpperExample)

Output:

HELLO STRING

# Java String trim()

The **java string trim()** method eliminates leading and trailing spaces. The unicode value of space character is '\u0020'. The trim() method in java string checks this unicode value before and after the string, if it exists then removes the spaces and returns the omitted string.

## **Java String trim() method example**

**public** **class** StringTrimExample{

**public** **static** **void** main(String args[]){

String s1="  hello string   ";

System.out.println(s1+"javatpoint");//without trim()

System.out.println(s1.trim()+"javatpoint");//with trim()

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=StringTrimExample)

hello string javatpoint

hello stringjavatpoint

# Java String valueOf()

The **java string valueOf()** method converts different types of values into string. By the help of string valueOf() method, you can convert int to string, long to string, boolean to string, character to string, float to string, double to string, object to string and char array to string.

## **Java String valueOf() method example**

**public** **class** StringValueOfExample{

**public** **static** **void** main(String args[]){

**int** value=30;

String s1=String.valueOf(value);

System.out.println(s1+10);//concatenating string with 10

}}

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=StringValueOfExample)

Output:

3010

# Java Array

* is a collection of similar type of elements that have a contiguous memory location.
* is an object which contains elements of a similar data type.
* It is a data structure where we store similar elements.
* We can store only a fixed set of elements in a Java array.
* Array in java is index-based, the first element of the array is stored at the 0 index.



### Advantages

**Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.

**Random access:** We can get any data located at an index position.

### Disadvantages

**Size Limit:**

* We can store only the fixed size of elements in the array.
* It doesn't grow its size at runtime.
* To solve this problem, collection framework is used in Java which grows automatically.

### Types of Array in java

There are two types of array.

Single Dimensional Array

Multidimensional Array

## **Single Dimensional Array in Java**

### Example of Java Array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

//Java Program to illustrate how to declare, instantiate, initialize

//and traverse the Java array.

**class** Testarray{

**public** **static** **void** main(String args[]){

**int** a[]=**new** **int**[5];//declaration and instantiation

a[0]=10;//initialization

a[1]=20;

a[2]=70;

a[3]=40;

a[4]=50;

//traversing array

**for**(**int** i=0;i<a.length;i++)//length is the property of array

System.out.println(a[i]);

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Testarray)

Output:

10

20

70

40

50

## **Passing Array to Method in Java**

We can pass the java array to method so that we can reuse the same logic on any array.

Let's see the simple example to get the minimum number of an array using a method.

//Java Program to demonstrate the way of passing an array

//to method.

**class** Testarray2{

//creating a method which receives an array as a parameter

**static** **void** min(**int** arr[]){

**int** min=arr[0];

**for**(**int** i=1;i<arr.length;i++)

**if**(min>arr[i])

  min=arr[i];

System.out.println(min);

}

**public** **static** **void** main(String args[]){

**int** a[]={33,3,4,5};//declaring and initializing an array

min(a);//passing array to method

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Testarray2)

Output:

3

## **Anonymous Array in Java**

Java supports the feature of an anonymous array, so you don't need to declare the array while passing an array to the method.

//Java Program to demonstrate the way of passing an anonymous array

//to method.

**public** **class** TestAnonymousArray{

//creating a method which receives an array as a parameter

**static** **void** printArray(**int** arr[]){

**for**(**int** i=0;i<arr.length;i++)

System.out.println(arr[i]);

}

**public** **static** **void** main(String args[]){

printArray(**new** **int**[]{10,22,44,66});//passing anonymous array to method

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestAnonymousArray)

Output:

10

22

44

66

## **Returning Array from the Method**

We can also return an array from the method in Java.

//Java Program to return an array from the method

**class** TestReturnArray{

//creating method which returns an array

**static** **int**[] get(){

**return** **new** **int**[]{10,30,50,90,60};

}

**public** **static** **void** main(String args[]){

//calling method which returns an array

**int** arr[]=get();

//printing the values of an array

**for**(**int** i=0;i<arr.length;i++)

System.out.println(arr[i]);

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestReturnArray)

Output:

10

30

50

90

60

## **Multidimensional Array in Java**

In such case, data is stored in row and column based index (also known as matrix form).

### Example of Multidimensional Java Array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

//Java Program to illustrate the use of multidimensional array

**class** Testarray3{

**public** **static** **void** main(String args[]){

//declaring and initializing 2D array

**int** arr[][]={{1,2,3},{2,4,5},{4,4,5}};

//printing 2D array

**for**(**int** i=0;i<3;i++){

**for**(**int** j=0;j<3;j++){

   System.out.print(arr[i][j]+" ");

 }

 System.out.println();

}

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Testarray3)

Output:

1 2 3

2 4 5

4 4 5

## **Jagged Array in Java**

If we are creating odd number of columns in a 2D array, it is known as a jagged array. In other words, it is an array of arrays with different number of columns.

//Java Program to illustrate the jagged array

**class** TestJaggedArray{

**public** **static** **void** main(String[] args){

        //declaring a 2D array with odd columns

**int** arr[][] = **new** **int**[3][];

        arr[0] = **new** **int**[3];

        arr[1] = **new** **int**[4];

        arr[2] = **new** **int**[2];

        //initializing a jagged array

**int** count = 0;

**for** (**int** i=0; i<arr.length; i++)

**for**(**int** j=0; j<arr[i].length; j++)

                arr[i][j] = count++;

        //printing the data of a 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();//new line

        }

    }

}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestJaggedArray)

Output:

0 1 2

3 4 5 6

7 8

## **What is the class name of Java array?**

In Java, an array is an object. For array object, a proxy class is created whose name can be obtained by getClass().getName() method on the object.

//Java Program to get the class name of array in Java

**class** Testarray4{

**public** **static** **void** main(String args[]){

//declaration and initialization of array

**int** arr[]={4,4,5};

//getting the class name of Java array

Class c=arr.getClass();

String name=c.getName();

//printing the class name of Java array

System.out.println(name);

}}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Testarray4)

Output:

I

## **Copying a Java Array**

We can copy an array to another by the arraycopy() method of System class.

### Example of Copying an Array in Java

//Java Program to copy a source array into a destination array in Java

**class** TestArrayCopyDemo {

**public** **static** **void** main(String[] args) {

        //declaring a source array

**char**[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',

                'i', 'n', 'a', 't', 'e', 'd' };

        //declaring a destination array

**char**[] copyTo = **new** **char**[7];

        //copying array using System.arraycopy() method

        System.arraycopy(copyFrom, 2, copyTo, 0, 7);

        //printing the destination array

        System.out.println(String.valueOf(copyTo));

    }

}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestArrayCopyDemo)

Output:

caffein