**What is OOPS?**

### **OBJECT ORIENTED PROGRAMMING (OOP)** is a programming concept that works on the principles of abstraction, encapsulation, inheritance, and polymorphism. It allows users to create the objects that they want and then, create methods to handle those objects. The basic concept of OOPs is to create objects, re-use them throughout the program, and manipulate these objects to get results.

### 1) Class

The class is a group of similar entities. It is only an logical component and not the physical entity.

### 2) Object

An object can be defined as an instance of a class, and there can be multiple instances of a class in a program.

### 4) Polymorphism

Polymorphism refers to the ability of a variable, object or function to take on multiple forms.

### 5) Abstraction(hiding internal working)

An abstraction is an act of representing essential features without including background details.

For example, while driving a car, you do not have to be concerned with its internal working. Here you just need to concern about parts like steering wheel, Gears, accelerator, etc.

### 6) Encapsulation

Encapsulation is an OOP technique of wrapping the data and code. In this OOPS concept, the variables of a class are always hidden from other classes. It can only be accessed using the methods of their current class. For example - in school, a student cannot exist without a class.

**Learn Encapsulation with an Example**

To understand what is encapsulation in detail consider the following bank account class with deposit and show balance methods

class Account {

private int account\_number;

private int account\_balance;

public void show Data() {

//code to show data

}

public void deposit(int a) {

if (a < 0) {

//show error

} else

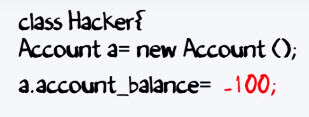
account\_balance = account\_balance + a;

}

}

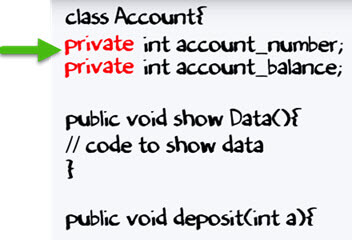
Suppose a hacker managed to gain access to the code of your bank account. Now, he tries to deposit amount -100 into your account by two ways. Let see his first method or approach.

**Approach 1:**He tries to deposit an invalid amount (say -100) into your bank account by manipulating the code.

[](https://www.guru99.com/images/java/052016_0638_LearnJavaEn1.jpg)

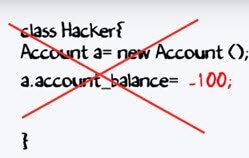
Now, the question is – *Is that possible?* Let investigate.

Usually, a variable in a class are set as "private" as shown below. It can only be accessed with the methods defined in the class. No other class or object can access them.

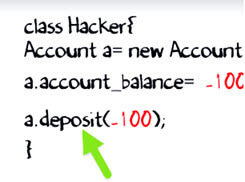
[](https://www.guru99.com/images/java/052016_0638_LearnJavaEn2.jpg)

If a data member is private, it means it can only be accessed within the same class. No outside class can access private data member or variable of other class.

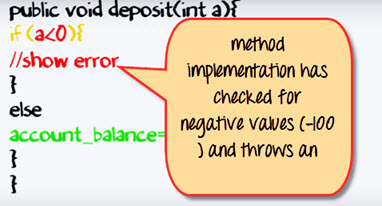
So in our case hacker cannot deposit amount -100 to your account.

[](https://www.guru99.com/images/java/052016_0638_LearnJavaEn3.jpg)

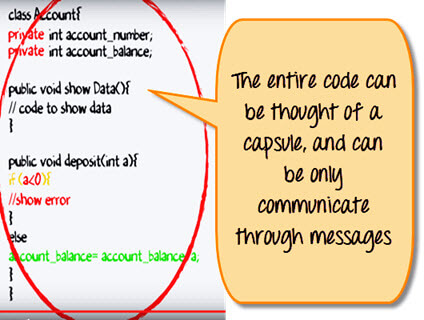
**Approach 2**: Hacker's first approach failed to deposit the amount. Next, he tries to do deposit a amount -100 by using "deposit" method.

[](https://www.guru99.com/images/java/052016_0638_LearnJavaEn4.jpg)

But method implementation has a check for negative values. So the second approach also fails.

[](https://www.guru99.com/images/java/052016_0638_LearnJavaEn5.jpg)

Thus, you never expose your data to an external party. Which makes your application secure.

[](https://www.guru99.com/images/java/052016_0638_LearnJavaEn6.jpg)

The entire code can be thought of a capsule, and you can only communicate through the messages. Hence the name encapsulation.

## Data Hiding in Java

Frequently, Java encapsulation is referred as **data hiding**. But more than data hiding, encapsulation concept is meant for better management or grouping of related data.

To achieve a lesser degree of encapsulation in Java, you can use modifiers like "protected" or "public". With encapsulation, developers can change one part of the code easily without affecting other.

## Getter and Setter Methods in Java

If a data member is declared "private", then it can only be accessed within the same class. No outside class can access data member of that class. If you need to access these variables, you have to use public "getter" and "setter" methods.

Getter and Setter's methods are used to create, modify, delete and view the variables values.

The following code is an example of getter and setter methods:

class Account{

private int account\_number;

private int account\_balance;

// getter method

public int getBalance() {

return this.account\_balance;

}

// setter method

public void setNumber(int num) {

this.account\_number = num;

}

}

In above example, getBalance() method is getter method that reads value of variable account\_balance and setNumber() method is setter method that sets or update value for variable account\_number.

## Abstraction vs. Encapsulation

Often encapsulation is misunderstood with Abstraction. Lets study-

* Encapsulation is more about "How" to achieve a functionality
* Abstraction is more about "What" a class can do.

A simple example to understand this difference is a mobile phone. Where the complex logic in the circuit board is encapsulated in a touch screen, and the interface is provided to abstract it out.

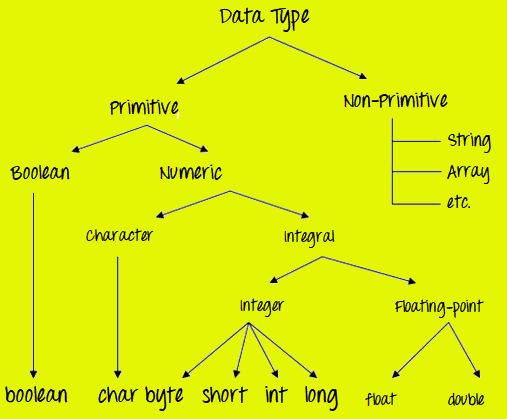
## Advantages of Encapsulation in Java

* Encapsulation is binding the data with its related functionalities. Here functionalities mean "methods" and data means "variables"
* So we keep variable and methods in one place. That place is "class." Class is the base for encapsulation.
* With Java Encapsulation, you can hide (restrict access) to critical data members in your code, which improves security
* As we discussed earlier, if a data member is declared "private", then it can only be accessed within the same class. No outside class can access data member (variable) of other class.
* However, if you need to access these variables, you have to use **public "getter" and "setter"** methods.

## Data Types in Java

Data types classify the different values to be stored in the variable. In java, there are two types of data types:

1. Primitive Data Types
2. Non-primitive Data Types

[](https://www.guru99.com/images/java/DataTypes.png)

### Primitive Data Types

Primitive Data Types are predefined and available within the Java language. Primitive values do not share state with other primitive values.

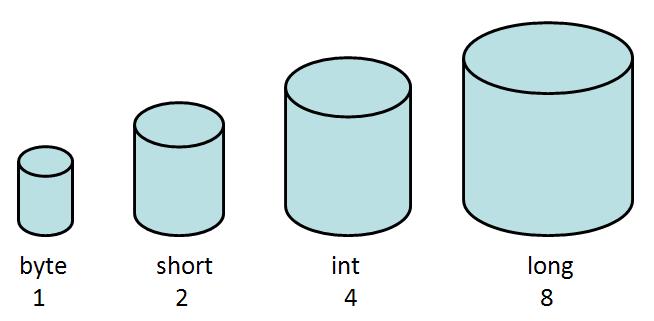
There are 8 primitive types: byte, short, int, long, char, float, double, and boolean **Integer data types**

byte (1 byte)

short (2 bytes)

int (4 bytes)

long (8 bytes)

[](https://www.guru99.com/images/uploads/2012/07/java-varaibles.jpg)

**Floating Data Type**

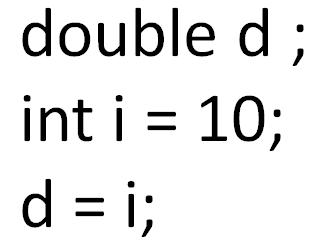
float (4 bytes)

double (8 bytes)

## Java Variable Type Conversion & Type Casting

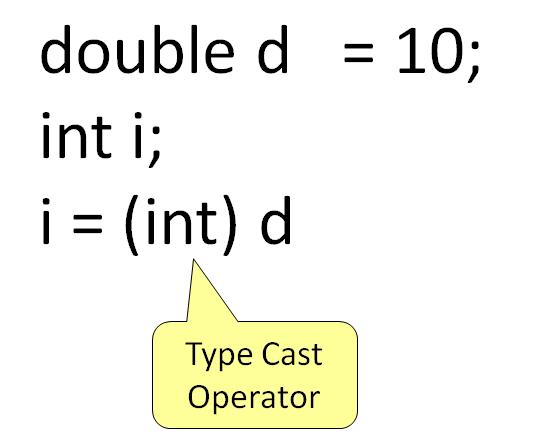
A variable of one type can receive the value of another type. Here there are 2 cases -

**Case 1)**Variable of smaller capacity is be assigned to another variable of bigger capacity.

[](https://www.guru99.com/images/uploads/2012/07/java-type-conversion.jpg)

This process is Automatic, and non-explicit is known as **Conversion**

**Case 2)**Variable of larger capacity is be assigned to another variable of smaller capacity

[](https://www.guru99.com/images/uploads/2012/07/java-type-cast-operator.jpg)

In such cases, you have to explicitly specify the **type cast operator. This process is known as Type Casting.**

Object:

Object Is instance of class.when we call object we are saying object to invoke or call methods we created in class.

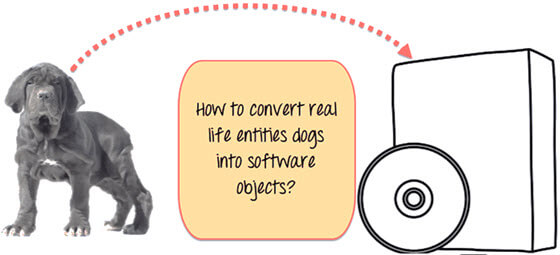
**Syntax**

ClassName ReferenceVariable = new ClassName();

**Understand the concept of Java Classes and Objects with an example.**

Let's take an example of developing a pet management system, specially meant for dogs. You will need various information about the dogs like different breeds of the dogs, the age, size, etc.

You need to model real-life beings, i.e., dogs into software entities.

[](https://www.guru99.com/images/java/052016_0704_ObjectsandC1.jpg)

Moreover, the million dollar question is, how you design such software? **Here is the solution-**

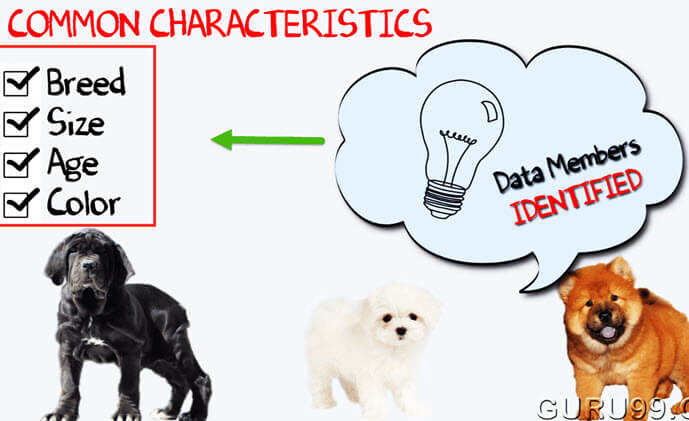
First, let's do an exercise.

You can see the picture of three different breeds of dogs below.

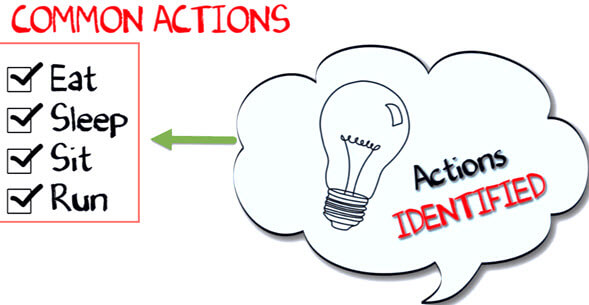
[](https://www.guru99.com/images/java/052016_0704_ObjectsandC2.jpg)

Stop here right now! List down the differences between them.

Some of the differences you might have listed out maybe breed, age, size, color, etc. If you think for a minute, these differences are also some common characteristics shared by these dogs. These characteristics (breed, age, size, color) can form a data members for your object.

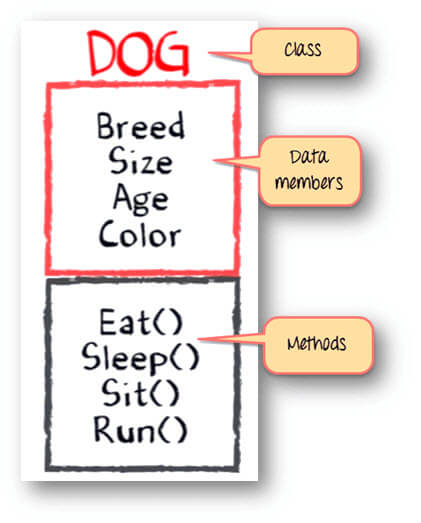
[](https://www.guru99.com/images/java/052016_0704_ObjectsandC3.jpg)

Next, list out the common behaviors of these dogs like sleep, sit, eat, etc. So these will be the actions of our software objects.

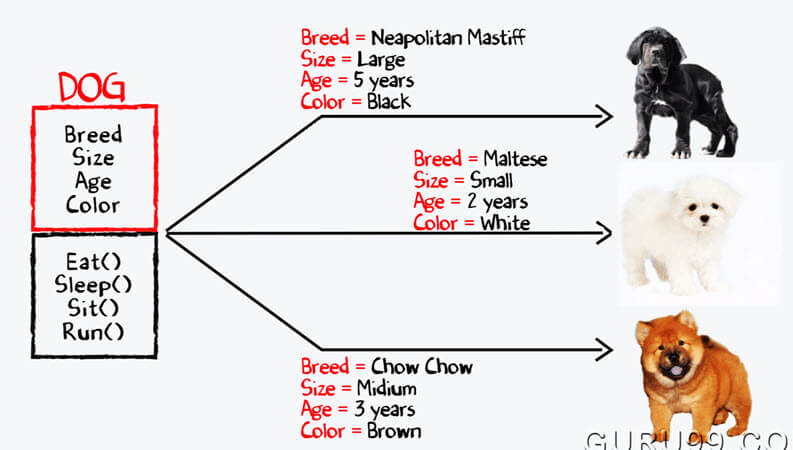
[](https://www.guru99.com/images/java/052016_0704_ObjectsandC4.jpg)

So far we have defined following things,

* **Class** - Dogs
* **Data members** or **objects**- size, age, color, breed, etc.
* **Methods**- eat, sleep, sit and run.

[](https://www.guru99.com/images/java/052016_0704_ObjectsandC5.jpg)

Now, for different values of data members (breed size, age, and color) in Java class, you will get different dog objects.

[](https://www.guru99.com/images/java/052016_0704_ObjectsandC6.jpg)

You can design any program using this OOPs approach.

While creating a class, one must follow the following principles.

* **Single Responsibility Principle (SRP)-** A class should have only one reason to change
* **Open Closed Responsibility (OCP)-** It should be able to extend any classes without modifying it
* **Liskov Substitution Responsibility (LSR)-** Derived classes must be substitutable for their base classes
* **Dependency Inversion Principle (DIP)-** Depend on abstraction and not on concretions
* **Interface Segregation Principle (ISP)-** Prepare fine grained interfaces that are client specific.

**Example Code: Class and Object**

// Class Declaration

public class Dog {

// Instance Variables

String breed;

String size;

int age;

String color;

// method 1

public String getInfo() {

return ("Breed is: "+breed+" Size is:"+size+" Age is:"+age+" color is: "+color);

}

public static void main(String[] args) {

Dog maltese = new Dog();

maltese.breed="Maltese";

maltese.size="Small";

maltese.age=2;

maltese.color="white";

System.out.println(maltese.getInfo());

}

}

**Output:**

Breed is: Maltese Size is:Small Age is:2 color is: white

**Object and Class Example: main outside class**

In previous program, we are creating main() method inside the class. Now, we create classes and define main() method in another class. This is a better way than previous one.

// Class Declaration

class Dog {

// Instance Variables

String breed;

String size;

int age;

String color;

// method 1

public String getInfo() {

return ("Breed is: "+breed+" Size is:"+size+" Age is:"+age+" color is: "+color);

}

}

public class Execute{

public static void main(String[] args) {

Dog maltese = new Dog();

maltese.breed="Maltese";

maltese.size="Small";

maltese.age=2;

maltese.color="white";

System.out.println(maltese.getInfo());

}

}

## Java ArrayList Example

import java.util.ArrayList;

class Test\_ArrayList {

public static void main(String[] args) {

//Creating a generic ArrayList

ArrayList<String> arlTest = new ArrayList<String>();

//Size of arrayList

System.out.println("Size of ArrayList at creation: " + arlTest.size());

//Lets add some elements to it

arlTest.add("D");

arlTest.add("U");

arlTest.add("K");

arlTest.add("E");

//Recheck the size after adding elements

System.out.println("Size of ArrayList after adding elements: " + arlTest.size());

//Display all contents of ArrayList

System.out.println("List of all elements: " + arlTest);

//Remove some elements from the list

arlTest.remove("D");

System.out.println("See contents after removing one element: " + arlTest);

//Remove element by index

arlTest.remove(2);

System.out.println("See contents after removing element by index: " + arlTest);

//Check size after removing elements

System.out.println("Size of arrayList after removing elements: " + arlTest.size());

System.out.println("List of all elements after removing elements: " + arlTest);

//Check if the list contains "K"

System.out.println(arlTest.contains("K"));

}

}

**Output:**

Size of ArrayList at creation: 0

Size of ArrayList after adding elements: 4

List of all elements: [D, U, K, E]

See contents after removing one element: [U, K, E]

See contents after removing element by index: [U, K]

Size of arrayList after removing elements: 2

List of all elements after removing elements: [U, K]

## What Is An Array Of Objects?

**JAVA ARRAY OF OBJECT**, as defined by its name, stores an **array of objects**. Unlike a traditional array that store values like string, integer, Boolean, etc an array of objects stores OBJECTS. The array elements store the location of the reference variables of the object.

**Syntax:**

Class obj[]= new Class[array\_length]

**Example: To create Array Of Objects**  
**Step 1)** Copy the following code into an editor

class ObjectArray{

public static void main(String args[]){

Account obj[] = new Account[2] ;

//obj[0] = new Account();

//obj[1] = new Account();

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

obj[1].setData(3,4);

System.out.println("For Array Element 0");

obj[0].showData();

System.out.println("For Array Element 1");

obj[1].showData();

}

}

class Account{

int a;

int b;

public void setData(int c,int d){

a=c;

b=d;

}

public void showData(){

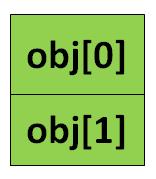
System.out.println("Value of a ="+a);

System.out.println("Value of b ="+b);

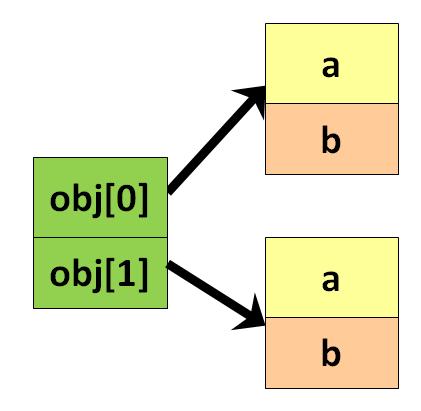
}

}

**Step 2)** Save , Compile & Run the Code.  
  
**Step 3)** Error=? Try and debug before proceeding to step 4.  
  
**Step 4)** The line of code, Account obj[] = new Account[2]; exactly creates an array of two reference variables as shown below

[](https://www.guru99.com/images/uploads/2012/07/java-array-of-objects.jpg)

**Step 5)** Uncomment Line # 4 & 5. This step creates objects and assigns them to the reference variable array as shown below. Your code must run now.

[](https://www.guru99.com/images/uploads/2012/07/java-array-of-objects-1.jpg)

**Output:**

For Array Element 0

Value of a =1

Value of b =2

For Array Element 1

Value of a =3

Value of b =4

## Why use Strings?

One of the primary functions of modern computer science, is processing human language.

Similarly to how numbers are important to math, language symbols are important to meaning and decision making. Although it may not be visible to computer users, computers process language in the background as precisely and accurately as a calculator. Help dialogs provide instructions. Menus provide choices. And data displays show statuses, errors, and real-time changes to the language.

As a Java programmer, one of your main tools for storing and processing language is going to be the String class.

## String Syntax Examples

Now, let’s get to some syntax,after all, we need to write this in Java code isn’t it.

String is an array of characters, represented as:

//String is an array of characters

char[] arrSample = {'R', 'O', 'S', 'E'};

String strSample\_1 = new String (arrSample);

In technical terms, the String is defined as follows in the above example-

= new (argument);

Now we always cannot write our strings as arrays; hence we can define the String in Java as follows:

//Representation of String

String strSample\_2 = "ROSE";

In technical terms, the above is represented as:

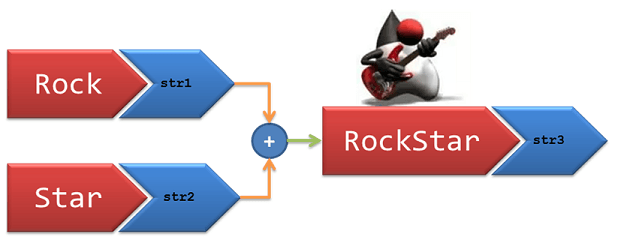
= ;

The **String Class** Java **extends** the **Object class.**

## String Concatenation:

Concatenation is joining of two or more strings.

Have a look at the below picture-

[](https://www.guru99.com/images/uploads/2012/06/Java-String-Concatenation.png)

We have two strings str1 = “Rock” and str2 = “Star”

If we add up these two strings, we should have a result as str3= “RockStar”.

Check the below code snippet,and it explains the two methods to perform string concatenation.

First is using “**concat**” method of String class and second is using arithmetic “+” operator. Both results in the same output

public class Sample\_String{

public static void main(String[] args){

//String Concatenation

String str1 = "Rock";

String str2 = "Star";

//Method 1 : Using concat

String str3 = str1.concat(str2);

System.out.println(str3);

//Method 2 : Using "+" operator

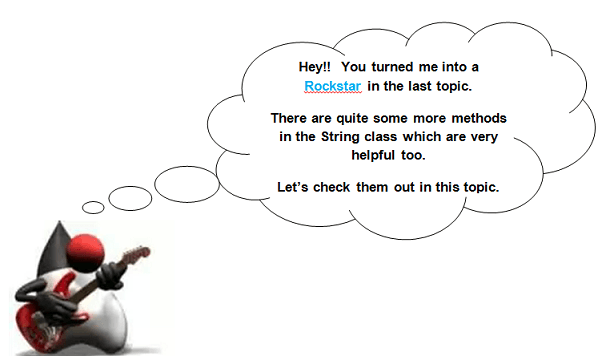
String str4 = str1 + str2;

System.out.println(str4);

}

}

## Important Java string methods :

[](https://www.guru99.com/images/uploads/2012/06/Java-String-Concatenation1.png)

Let’s ask the Java String class a few questions and see if it can answer them ;)

### String "Length" Method

How will you determine the length of given String? I have provided a method called as “length”. Use it against the String you need to find the length.

public class Sample\_String{

public static void main(String[] args){ //Our sample string for this tutorial

String str\_Sample = "RockStar";

//Length of a String

System.out.println("Length of String: " + str\_Sample.length());}}

**output:**

Length of String: 8

### String "indexOf" Method

If I know the length, how would I find which character is in which position? In short, how will I find the index of a character?

You answered yourself, buddy, there is an “indexOf” method that will help you determine the location of a specific character that you specify.

public class Sample\_String{

public static void main(String[] args){//Character at position

String str\_Sample = "RockStar";

System.out.println("Character at position 5: " + str\_Sample.charAt(5));

//Index of a given character

System.out.println("Index of character 'S': " + str\_Sample.indexOf('S'));}}

**Output:**

Character at position 5: t

Index of character 'S': 4

### String "charAt" Method

Similar to the above question, given the index, how do I know the character at that location?

Simple one again!! Use the “charAt” method and provide the index whose character you need to find.

public class Sample\_String{

public static void main(String[] args){//Character at position

String str\_Sample = "RockStar";

System.out.println("Character at position 5: " + str\_Sample.charAt(5));}}

**Output:**

Character at position 5: t

### String "CompareTo" Method

Do I want to check if the String that was generated by some method is equal to something that I want to verify with? How do I compare two Strings?

Use the method “compareTo” and specify the String that you would like to compare.

Use “compareToIgnoreCase” in case you don’t want the result to be case sensitive.

The result will have the value 0 if the argument string is equal to this string; a value less than 0 if this string is lexicographically less than the string argument; and a value greater than 0 if this string is lexicographically greater than the string argument.

public class Sample\_String{

public static void main(String[] args){//Compare to a String

String str\_Sample = "RockStar";

System.out.println("Compare To 'ROCKSTAR': " + str\_Sample.compareTo("rockstar"));

//Compare to - Ignore case

System.out.println("Compare To 'ROCKSTAR' - Case Ignored: " + str\_Sample.compareToIgnoreCase("ROCKSTAR"));}}

**Output:**

Compare To 'ROCKSTAR': -32

Compare To 'ROCKSTAR' - Case Ignored: 0

### String "Contain" Method

I partially know what the string should have contained, how do I confirm if the String contains a sequence of characters I specify?

Use the method “contains” and specify the characters you need to check.

Returns **true** if and only if this string contains the specified sequence of char values.

public class Sample\_String{

public static void main(String[] args){ //Check if String contains a sequence

String str\_Sample = "RockStar";

System.out.println("Contains sequence 'tar': " + str\_Sample.contains("tar"));}}

**Output:**

Contains sequence 'tar': true

### String "endsWith" Method

How do I confirm if a String ends with a particular suffix? Again you answered it. Use the “endsWith” method and specify the suffix in the arguments.

Returns **true** if the character sequence represented by the argument is a suffix of the character sequence represented by this object.

public class Sample\_String{

public static void main(String[] args){ //Check if ends with a particular sequence

String str\_Sample = "RockStar";

System.out.println("EndsWith character 'r': " + str\_Sample.endsWith("r"));}}

**Output:**

EndsWith character 'r': true

### String "replaceAll" & "replaceFirst" Method

I want to modify my String at several places and replace several parts of the String?

Java String Replace, replaceAll and replaceFirst methods. You can specify the part of the String you want to replace and the replacement String in the arguments.

public class Sample\_String{

public static void main(String[] args){//Replace Rock with the word Duke

String str\_Sample = "RockStar";

System.out.println("Replace 'Rock' with 'Duke': " + str\_Sample.replace("Rock", "Duke"));}}

**Output:**

Replace 'Rock' with 'Duke': DukeStar

### String Java "tolowercase" & Java "touppercase" Method

I want my entire String to be shown in lower case or Uppercase?

Just use the “toLowercase()” or “ToUpperCase()” methods against the Strings that need to be converted.

public class Sample\_String{

public static void main(String[] args){//Convert to LowerCase

String str\_Sample = "RockStar";

System.out.println("Convert to LowerCase: " + str\_Sample.toLowerCase());

//Convert to UpperCase

System.out.println("Convert to UpperCase: " + str\_Sample.toUpperCase());}}

**Output:**

Convert to LowerCase: rockstar

Convert to UpperCase: ROCKSTAR

## Important Points to Note:

* **String is a Final class**; i.e once created the value cannot be altered. Thus String objects are called immutable.
* The Java Virtual Machine(JVM) creates a memory location especially for Strings called **String Constant Pool**. That’s why String can be initialized without ‘new’ keyword.
* String class falls under**java.lang.String hierarchy**. But there is no need to import this class. Java platform provides them automatically.
* String**reference can be overridden but that does not delete the content**; i.e., if

String h1 = "hello";

h1 = "hello"+"world";

then "hello" String does not get deleted. It just loses its handle.

* **Multiple references** can be used for same String but it will **occur in the same place**; i.e., if

String h1 = "hello";

String h2 = "hello";

String h3 = "hello";

then only one pool for String “hello” is created in the memory with 3 references-h1,h2,h3

* If a **number is quoted in “ ”** then it**becomes a string**, not a number anymore. That means if

String S1 ="The number is: "+ "123"+"456";

System.out.println(S1);

then it will print: The number is: 123456

If the initialization is like this:

String S1 = "The number is: "+(123+456);

System.out.println(S1);

then it will print: The number is:579 That's all to Strings!

**Constructors:**

**Each class has atleast one constructors. The constructors iniatialized the object created. Each time**

**new object is constructed then constructors is iniatialized.**

\*no argument constructors.

\*parameterized constructors.

{object has states and behaviour}

{class is block/blueprint which describes the type of state and behaviour}

Important:

* Inclusive range of byte is -128 (-2^7) to (2^7 -1)
* Inclusive range of short is (-2^15) to (2^15 -1)
* Inclusive range of int is (-2^31) to (2^31 -1)
* Inclusive range of byte is (-2^63)to (2^63 -1)

Access Control Modifiers

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are −

* Visible to the package, the default. No modifiers are needed.
* Visible to the class only (private).
* Visible to the world (public).
* Visible to the package and all subclasses (protected).

Non-Access Modifiers

Java provides a number of non-access modifiers to achieve many other functionality.

* The *static* modifier for creating class methods and variables.
* The *final* modifier for finalizing the implementations of classes, methods, and variables.
* The *abstract* modifier for creating abstract classes and methods.
* The *synchronized* and *volatile* modifiers, which are used for threads.

### **Conditional Operator ( ? : )**

Conditional operator is also known as the **ternary operator**. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable. The operator is written as −

variable x = (expression) ? value if true : value if false

Following is an example −

**Example**

[Live Demo](http://tpcg.io/xJJ2kP)

public class Test {

public static void main(String args[]) {

int a, b;

a = 10;

b = (a == 1) ? 20: 30;

System.out.println( "Value of b is : " + b );

b = (a == 10) ? 20: 30;

System.out.println( "Value of b is : " + b );

}

}

Output:

Value of b is: 30

Value of b is : 20