subject=java notes

NAME=mansi Mokashi

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

JBM=A Java Business Module is a special class that typically handles the database logic behind JeniuX Service.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

JRE=Java Runtime Environment (JRE) is an open-access software distribution that has a Java class library, specific tools, and a separate JVM. JRE is one of the interrelated components in the Java Development Kit (JDK). It is the most common environment available on devices for running Java programs.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

JDK=The Java Development Kit (JDK) is one of three core technology packages used in Java programming, along with the JVM (Java Virtual Machine) and the JRE (Java Runtime Environment).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CLASS LOADER=Java ClassLoader is an abstract class. It belongs to a java.lang package. It loads classes from different resources. Java ClassLoader is used to load the classes at run time. In other words, JVM performs the linking process at runtime. Classes are loaded into the JVM according to need. If a loaded class depends on another class, that class is loaded as well. When we request to load a class, it delegates the class to its parent. In this way, uniqueness is maintained in the runtime environment. It is essential to execute a Java program.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

JIT=JIT compiler environment variable is properly set, the JVM reads the.class file (bytecode) for interpretation after that it passes to the JIT compiler for further process. After getting the bytecode, the JIT compiler transforms it into the native code (machine-readable code).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

FEATURES OF JAVA=

Simple

Object-Oriented

Portable

Platform independent

Secured

Robust

Architecture neutral

Interpreted

High Performance

Multithreaded

Distributed

Dynamic

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

BYTE CODE=Bytecode is essentially the machine level language which runs on the Java Virtual Machine. Whenever a class is loaded, it gets a stream of bytecode per method of the class.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

JAVA FILE CLASS=The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.

The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PRIMITIVE DATA TYPES(PRIMITIVE)=

Primitive data types in Java are predefined by the Java language and named as the reserved keywords. A primitive data type does not share a state with other primitive values. Java programming language supports the following eight primitive data types.

Boolean data type

byte data type

int data type

long data type

float data type

double data type

char data type

short data type

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

REFRENCE DATA TYPES(NON-PRIMITIVE)=

non-primitive data types are known as reference types. In other words, a variable of class type is called reference data type. It contains the address (or reference) of dynamically created objects.

class

interface

String

Arrays

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

COMMAND LINE ARGUEMENT=

The java command-line argument is an argument i.e. passed at the time of running the java program.

The arguments passed from the console can be received in the java program and it can be used as an input.

\*JAVA CLASS/OBJECT=

Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

A Class is like an object constructor, or a "blueprint" for creating objects.

\*EXCESS SPECIFIERS=**Access specifiers** are the keywords like **"public"**, **"protected"**, **"default"** and **"private"** which has its special meaning in java.

It defines the access scope of the variable, methods, and classes and here the access scope means the area or space where a variable or classes or methods are accessible.

TYPES OF EXCESS SPECIFIERS=

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

­­­­­­­­­­­­­­­­­­­­­\*ENCAPSULATION=

**Encapsulation in Java** is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.

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.

The **Java Bean** class is the example of a fully encapsulated class.

\*GETTER AND SETTER=

Getter and setter methods are frequently used in Java programming. **Getter and setter methods in Java** are widely used to access and manipulate the values of class fields. Usually, class fields are decorated with a private access specifier. Thus, to access them, public access specifiers are used with the getter and setter methods.

this KEYWORD IN JAVA=

In Java, this is a **reference variable** that refers to the current object. Here is given the 6 usage of java this keyword.

USE OF this KEYWORD=

1. [this can be used to refer current class instance variable.](https://www.javatpoint.com/this1)
2. [this can be used to invoke current class method (implicitly)](https://www.javatpoint.com/this2)
3. [this() can be used to invoke current class constructor.](https://www.javatpoint.com/this3)
4. [this can be passed as an argument in the method call.](https://www.javatpoint.com/this4)
5. [this can be passed as argument in the constructor call.](https://www.javatpoint.com/this5)
6. [this can be used to return the current class instance from the method.](https://www.javatpoint.com/this6)



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

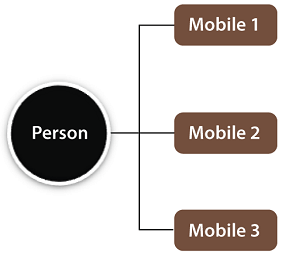
\*ABSTRACTION=Abstract Classes and Methods Data abstraction is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either abstract classes or interfaces (which you will learn more about in the next chapter). The abstract keyword is a non-access modifier, used for classes and methods.

* **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

* **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

**Note:** Abstraction can also be achieved with [Interfaces](https://www.w3schools.com/java/java_interface.asp)

**ASSOCIATION=**Association in Java defines the connection between two classes that are set up through their objects. Association manages **one-to-one, one-to-many**, and **many-to-many** relationships. In [Java](https://www.javatpoint.com/java-tutorial), the multiplicity between objects is defined by the **Association**. It shows how objects communicate with each other and how they use the functionality and services provided by that communicated object. Association manages **one-to-one, one-to-many, many-to-one** and **many-to-many** relationships



AGGREGATION=If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Why use Aggregation?

* For Code Reusability.
* Composition allows us to reuse the code.
* In Java, we can use multiple Inheritance by using the composition concept.
* The Composition provides better test-ability of a class.
* Composition allows us to easily replace the composed class implementation with a better and improved version.
* Composition allows us to dynamically change our program's behavior by changing the member objects at run time.

GENERALIZATION=

* Converting a subclass type into a superclass type is called ‘ Generalization ‘ because we are making the subclass to become more general and its scope is widening. This is also called widening or up casting. Widening is safe because the classes will become more general.

­­­­­­­­­­­­­­­­­­­\*FUNCTION OVERLOADING=Function Overloading in Java occurs when there are functions having the same name but have different numbers of parameters

Function overloading is used to reduce complexity and increase the efficiency of the program by involving more functions

­­­­­­­­­­­­­­­­­­­­\*VARIABLE ARGUMENT(VARARGS)= The varrags allows the method to accept zero or muliple arguments. Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem. If we don't know how many argument we will have to pass in the method, varargs is the better approach.

## **Advantage of Varargs:**

We don't have to provide overloaded methods so less code.

RULES OF VARARGS=

* There can be only one variable argument in the method.
* Variable argument (varargs) must be the last argument.

\*STATIC VARIABLE=Static variable in Java is variable which belongs to the class and initialized only once at the start of the execution. It is a variable which belongs to the class and not to object (instance). Static variables are initialized only once, at the start of the execution.

\*STSTIC METHOD=**Static method in Java** is a method which belongs to the class and not to the object. A static method can access only static data. It is a method which belongs to the class and not to the object(instance). A static method can access only static data. It cannot access non-static data (instance variables).

* A static method can call only other static methods and can not call a non-static method from it.
* A static method can be accessed directly by the class name and doesn’t need any object
* A static method cannot refer to “this” or “super” keywords in anyway

\*STATIC BLOCK=In simpler language whenever we use a [static keyword](https://www.geeksforgeeks.org/static-keyword-java/) and associate it to a block then that block is referred to as a static block

IMP NOTE=**Can we print something on the console in static without creating main() method?**

It is very important question from the interview’s perceptive point. The answer is yes we can print if we are using JDK version 1.6 or previous and if after that  it will throw an. error.

\*static import=The static import feature of Java 5 facilitate the java programmer to access any static member of a class directly. There is no need to qualify it by the class name.

\*WRAPPER CLASS=

* In Java, Wrapper Class is used for converting primitive data type into object and object into a primitive data type. For each primitive data type, a pre-defined class is present which is known as Wrapper class.

# \*Autoboxing and Unboxing in Java

In Java, primitive data types are treated differently so do there comes the introduction of [wrapper classes](https://www.geeksforgeeks.org/wrapper-classes-java/) where two components play a role namely Autoboxing and Unboxing. [Autoboxing](https://www.geeksforgeeks.org/autoboxing-unboxing-java/) refers to the conversion of a primitive value into an object of the corresponding wrapper class is called autoboxing. For example, converting int to Integer class. The Java compiler applies autoboxing when a primitive value is:

* Passed as a parameter to a method that **expects an object** of the corresponding wrapper class.
* Assigned to a variable of the corresponding **wrapper class**.

**Unboxing**on the other hand refers to converting an object of a wrapper type to its corresponding primitive value. For example conversion of Integer to int. The Java compiler applies to unbox when an object of a wrapper class is:

* Passed as a parameter to a method that **expects a value** of the corresponding primitive type.
* Assigned to a variable of the corresponding **primitive type**.

| Primitive Type | Wrapper Class |
| --- | --- |
| boolean | Boolean |
| byte | Byte |
| char | Character |
| float | Float |
| int | Integer |
| long | Long |
| short | Short |
| double | Double |

**INHERITANCE=**

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of [OOPs](https://www.javatpoint.com/java-oops-concepts)

(Object Oriented programming system).

The idea behind inheritance in Java is that you can create new [classes](https://www.javatpoint.com/object-and-class-in-java)

that are built upon existing classes. When you inherit from an existing class, you 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](https://www.javatpoint.com/method-overriding-in-java)

(so [runtime polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)

can be achieved).

* For Code Reusability.

Terms used in Inheritance

* **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
* **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
* **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
* **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

The syntax of Java Inheritance

1. **class** Subclass-name **extends** Superclass-name
2. {
3. //methods and fields
4. }

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

## **Types of inheritance in java**

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

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



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

When one class inherits multiple classes, it is known as multiple inheritance. For Example:



## **Single Inheritance Example**

When a class inherits another class, it is known as a single inheritance.

## **Multilevel Inheritance Example**

When there is a chain of inheritance, it is known as multilevel inheritance.

## **Hierarchical Inheritance Example**

When two or more classes inherits a single class, it is known as hierarchical inheritance.

## **Q) Why multiple inheritance is not supported in java?**

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

# **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.

# **Final Keyword In Java**

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, 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. We will have detailed learning of these. Let's first learn the basics of final keyword.

## **1) Java final variable**

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

## **2) Java final method**

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

## **3) Java final class**

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

### Q) Is final method inherited?

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

### Q) 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.

### Que) Can we initialize blank final variable?

Yes, but only in constructor.

### static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

### Q) What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

### Q) Can we declare a constructor final?

No, because constructor is never inherited.

## **Java Constructors**

A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes:

­­­­­­­­­­­­­­

\*\*DIFFERENCE BETWEEN ABSTRACT CLASS AND INTERFACE=

|  |  |
| --- | --- |
| Abstract Class | Interface |
| It contains both declaration and definition part. | It contains only a declaration part. |
| Multiple inheritance is not achieved by abstract class. | Multiple inheritance is achieved by interface. |
| It contain [constructor](https://www.geeksforgeeks.org/c-sharp-constructors/). | It does not contain [constructor](https://www.geeksforgeeks.org/c-sharp-constructors/). |
| It can contain static members. | It does not contain static members. |
| It can contain different types of access modifiers like public, private, protected etc. | It only contains public access modifier because everything in the interface is public. |
| The performance of an abstract class is fast. | The performance of interface is slow because it requires time to search actual method in the corresponding class. |
| It is used to implement the core identity of class. | It is used to implement peripheral abilities of class. |
| A class can only use one abstract class. | A class can use multiple interface. |
| If many implementations are of the same kind and use common behavior, then it is superior to use abstract class. | If many implementations only share methods, then it is superior to use Interface. |
| Abstract class can contain methods, fields, constants, etc. | Interface can only contains methods, properties, indexers, events. |
| It can be fully, partially or not implemented. | It should be fully implemented. |

## \*\*Checked exceptions

A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation, the programmer should take care of (handle) these exceptions.

For example, if you use **FileReader** class in your program to read data from a file, if the file specified in its constructor doesn't exist, then a FileNotFoundException occurs, and the compiler prompts the programmer to handle the exception.

## \*\*Unchecked exceptions

An unchecked exception is an exception that occurs at the time of execution. These are also called as **Runtime Exceptions.** These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.

For example, if you have declared an array of size 5 in your program, and trying to call the 6th element of the array then an ArrayIndexOutOfBoundsExceptionexception occurs.

# **Java try-catch block**

## **Java try block**

Java **try** block is used to enclose the code that might throw an exception. It must be used within the method.

If an exception occurs at the particular statement in the try block, the rest of the block code will not execute. So, it is recommended not to keep the code in try block that will not throw an exception.

Java try block must be followed by either catch or finally block.

### Syntax of Java try-catch

1. **try**{
2. //code that may throw an exception
3. }**catch**(Exception\_class\_Name ref){}

### Syntax of try-finally block

1. **try**{
2. //code that may throw an exception
3. }**finally**{}

## **Java catch block**

Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception ( i.e., Exception) or the generated exception type. However, the good approach is to declare the generated type of exception.

## **Internal Working of Java try-catch block**



# \*\*throw and throws in Java=

**throw**

The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either [checked or unchecked exception](https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/). The throw keyword is mainly used to throw custom exceptions.

**Syntax:**

**throw *Instance***

Example:**throw new ArithmeticException("/ by zero");**

But this exception i.e, *Instance* must be of type **Throwable** or a subclass of **Throwable**. For example Exception is a sub-class of Throwable and [user defined exceptions typically extend Exception class](https://www.geeksforgeeks.org/g-fact-32-user-defined-custom-exception-in-java/). Unlike C++, data types such as int, char, floats or non-throwable classes cannot be used as exceptions.

The flow of execution of the program stops immediately after the throw statement is executed and the nearest enclosing **try** block is checked to see if it has a **catch** statement that matches the type of exception. If it finds a match, controlled is transferred to that statement otherwise next enclosing **try** block is checked and so on. If no matching**catch**is found then the default exception handler will halt the program.

**throws**

throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.

**Syntax:**

**type method\_name(parameters) throws exception\_list**

exception\_list is a comma separated list of all the

exceptions which a method might throw.

In a program, if there is a chance of raising an exception then compiler always warn us about it and compulsorily we should handle that checked exception, Otherwise we will get compile time error saying **unreported exception XXX must be caught or declared to be thrown**. To prevent this compile time error we can handle the exception in two ways:

1. By using [try catch](https://www.geeksforgeeks.org/flow-control-in-try-catch-finally-in-java/)
2. By using **throws** keyword

We can use throws keyword to delegate the responsibility of exception handling to the caller (It may be a method or JVM) then caller method is responsible to handle that exception.

**Important points to remember about throws keyword:**

* throws keyword is required only for checked exception and usage of throws keyword for unchecked exception is meaningless.
* throws keyword is required only to convince compiler and usage of throws keyword does not prevent abnormal termination of program.
* By the help of throws keyword we can provide information to the caller of the method about the exception.

\*\*PACKAGES=

A package is simply a container that groups related types (Java classes, interfaces, enumerations and annotations). For example, in core Java, the ResultSet interface belongs to java.sql package. The package contains all the related types that are needed for SQL query and database connection.

\*\*jarfiles=

 public class JarFile extends ZipFile The JarFile class is used to read the contents of a jar file from any file that can be opened with java.io.RandomAccessFile. It extends the class java.util.zip.ZipFile with support for reading an optional Manifest entry.

Java 8 provides following features for Java Programming:

* Lambda expressions,
* Method references,
* Functional interfaces,
* Stream API,
* Default methods,
* Base64 Encode Decode,
* Static methods in interface,
* Optional class,
* Collectors class,
* ForEach() method,
* Nashorn JavaScript Engine,
* Parallel Array Sorting,
* Type and Repating Annotations,
* IO Enhancements,
* Concurrency Enhancements,
* JDBC Enhancements etc.

\*\*FUNCTION INTERFACE=

The Function Interface is a part of the java.util.function package which has been introduced since Java 8, to implement functional programming in Java. It represents a function which takes in one argument and produces a result.

\*\*java typecasting=Type casting is when you assign a value of one primitive data type to another type

# \*\*String class=String is a sequence of characters. In java, objects of String are immutable which means a constant and cannot be changed once created.

# **Java StringBuffer Class**

Java StringBuffer class is used to create mutable (modifiable) String objects. The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| StringBuffer() | It creates an empty String buffer with the initial capacity of 16. |
| StringBuffer(String str) | It creates a String buffer with the specified string.. |
| StringBuffer(int capacity) | It creates an empty String buffer with the specified capacity as length. |

# **Life cycle of a Thread (Thread States)**

In Java, a thread always exists in any one of the following states. These states are:

1. New
2. Active
3. Blocked / Waiting
4. Timed Waiting
5. Terminated

## **Explanation of Different Thread States**

**New:** Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.

**Active:** When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **runnable**, and the other is **running**.

* **Runnable:** A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time. It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread the running state.  
  A program implementing multithreading acquires a fixed slice of time to each individual thread. Each and every thread runs for a short span of time and when that allocated time slice is over, the thread voluntarily gives up the CPU to the other thread, so that the other threads can also run for their slice of time. Whenever such a scenario occurs, all those threads that are willing to run, waiting for their turn to run, lie in the runnable state. In the runnable state, there is a queue where the threads lie.
* **Running:** When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

**Blocked or Waiting:** Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.

**Timed Waiting:** Sometimes, waiting for leads to starvation. For example, a thread (its name is A) has entered the critical section of a code and is not willing to leave that critical section. In such a scenario, another thread (its name is B) has to wait forever, which leads to starvation. To avoid such scenario, a timed waiting state is given to thread B. Thus, thread lies in the waiting state for a specific span of time, and not forever. A real example of timed waiting is when we invoke the sleep() method on a specific thread. The sleep() method puts the thread in the timed wait state. After the time runs out, the thread wakes up and start its execution from when it has left earlier.

**Terminated:** A thread reaches the termination state because of the following reasons:

* When a thread has finished its job, then it exists or terminates normally.
* **Abnormal termination:** It occurs when some unusual events such as an unhandled exception or segmentation fault.

A terminated thread means the thread is no more in the system. In other words, the thread is dead, and there is no way one can respawn (active after kill) the dead thread.

# **Java Threads | How to create a thread in Java**

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

### Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

### Commonly used Constructors of Thread class:

* Thread()
* Thread(String name)
* Thread(Runnable r)
* Thread(Runnable r,String name)

### Commonly used methods of Thread class:

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(depricated).
16. **public void resume():** is used to resume the suspended thread(depricated).
17. **public void stop():** is used to stop the thread(depricated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

### Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1. **public void run():** is used to perform action for a thread.

### Starting a thread:

The **start() method** of Thread class is used to start a newly created thread. It performs the following tasks:

* A new thread starts(with new callstack).
* The thread moves from New state to the Runnable state.
* When the thread gets a chance to execute, its target run() method will run.

# **Synchronization in Java**

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

### Why use Synchronization?

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

### Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

### Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
   1. Synchronized method.
   2. Synchronized block.
   3. Static synchronization.
2. Cooperation (Inter-thread communication in java)

### Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. It can be achieved by using the following three ways:

1. By Using Synchronized Method
2. By Using Synchronized Block
3. By Using Static Synchronization

\*\*DEAD LOCK=Deadlock in Java is **a condition where two or more threads are blocked forever, waiting for each other**. This usually happens when multiple threads need the same locks but obtain them in different orders. Multithreaded Programming in Java suffers from the deadlock situation because of the synchronized keyword.

# **Java Garbage Collection**

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

### Advantage of Garbage Collection

* It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
* It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

## **How can an object be unreferenced?**

There are many ways

* By nulling the reference
* By assigning a reference to another
* By anonymous object etc.

# **Daemon Thread in Java**

**Daemon thread in Java** is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

There are many java daemon threads running automatically e.g. gc, finalizer etc.

You can see all the detail by typing the jconsole in the command prompt. The jconsole tool provides information about the loaded classes, memory usage, running threads etc.

## **Points to remember for Daemon Thread in Java**

* It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
* Its life depends on user threads.
* It is a low priority thread.

### Why JVM terminates the daemon thread if there is no user thread?

The sole purpose of the daemon thread is that it provides services to user thread for background supporting task. If there is no user thread, why should JVM keep running this thread. That is why JVM terminates the daemon thread if there is no user thread.

### Methods for Java Daemon thread by Thread class

The java.lang.Thread class provides two methods for java daemon thread.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1) | public void setDaemon(boolean status) | is used to mark the current thread as daemon thread or user thread. |
| 2) | public boolean isDaemon() | is used to check that current is daemon. |

# **Inter-thread Communication in Java**

**Inter-thread communication** or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

* wait()
* notify()
* notifyAll()

### 1) wait() method

The wait() method causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

36.9M

663

Prime Ministers of India | List of Prime Minister of India (1947-2020)

|  |  |
| --- | --- |
| **Method** | **Description** |
| public final void wait()throws InterruptedException | It waits until object is notified. |
| public final void wait(long timeout)throws InterruptedException | It waits for the specified amount of time. |

### 2) notify() method

The notify() method wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.

**Syntax:**

1. **public** **final** **void** notify()

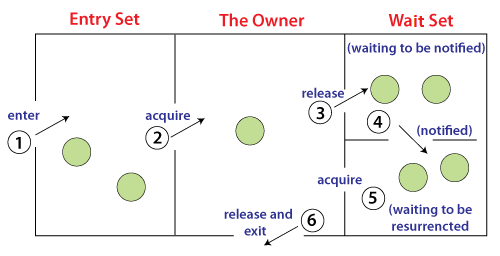
### 3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor.

**Syntax:**

1. **public** **final** **void** notifyAll()

## **\*\*Understanding the process of inter-thread communication**



The point to point explanation of the above diagram is as follows:

1. Threads enter to acquire lock.
2. Lock is acquired by on thread.
3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4. If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.

### Why wait(), notify() and notifyAll() methods are defined in Object class not Thread class?

It is because they are related to lock and object has a lock.

### Difference between wait and sleep?

Let's see the important differences between wait and sleep methods.

|  |  |
| --- | --- |
| **wait()** | **sleep()** |
| The wait() method releases the lock. | The sleep() method doesn't release the lock. |
| It is a method of Object class | It is a method of Thread class |
| It is the non-static method | It is the static method |
| It should be notified by notify() or notifyAll() methods | After the specified amount of time, sleep is completed. |

# **Collections in Java**

1. [Hierarchy of Collection Framework](https://www.javatpoint.com/collections-in-java" \l "collectionhierarchy)
2. [Collection interface](https://www.javatpoint.com/collections-in-java" \l "collectionmethods)
3. [Iterator interface](https://www.javatpoint.com/collections-in-java" \l "collectioniterator)

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes ([ArrayList](https://www.javatpoint.com/java-arraylist)

, Vector, [LinkedList](https://www.javatpoint.com/java-linkedlist)

, [PriorityQueue](https://www.javatpoint.com/java-priorityqueue)

, HashSet, LinkedHashSet, TreeSet).

#### **What is Collection in Java**

A Collection represents a single unit of objects, i.e., a,group

#### **What is a framework in Java**

* It provides readymade architecture.
* It represents a set of classes and interfaces.
* It is optional.

#### **What is Collection framework**

The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes
2. Algorithm

### Hierarchy of Collection Framework

Let us see the hierarchy of Collection framework. The **java.util** package contains all the [classes](https://www.javatpoint.com/object-and-class-in-java)

and [interfaces](https://www.javatpoint.com/interface-in-java)

for the Collection framework



### Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | public boolean add(E e) | It is used to insert an element in this collection. |
| 2 | public boolean addAll(Collection<? extends E> c) | It is used to insert the specified collection elements in the invoking collection. |
| 3 | public boolean remove(Object element) | It is used to delete an element from the collection. |
| 4 | public boolean removeAll(Collection<?> c) | It is used to delete all the elements of the specified collection from the invoking collection. |
| 5 | default boolean removeIf(Predicate<? super E> filter) | It is used to delete all the elements of the collection that satisfy the specified predicate. |
| 6 | public boolean retainAll(Collection<?> c) | It is used to delete all the elements of invoking collection except the specified collection. |
| 7 | public int size() | It returns the total number of elements in the collection. |
| 8 | public void clear() | It removes the total number of elements from the collection. |
| 9 | public boolean contains(Object element) | It is used to search an element. |
| 10 | public boolean containsAll(Collection<?> c) | It is used to search the specified collection in the collection. |
| 11 | public Iterator iterator() | It returns an iterator. |
| 12 | public Object[] toArray() | It converts collection into array. |
| 13 | public <T> T[] toArray(T[] a) | It converts collection into array. Here, the runtime type of the returned array is that of the specified array. |
| 14 | public boolean isEmpty() | It checks if collection is empty. |
| 15 | default Stream<E> parallelStream() | It returns a possibly parallel Stream with the collection as its source. |
| 16 | default Stream<E> stream() | It returns a sequential Stream with the collection as its source. |
| 17 | default Spliterator<E> spliterator() | It generates a Spliterator over the specified elements in the collection. |
| 18 | public boolean equals(Object element) | It matches two collections. |
| 19 | public int hashCode() | It returns the hash code number of the collection. |

### Iterator interface

|  |
| --- |
| Iterator interface provides the facility of iterating the elements in a forward direction only. |

#### **Methods of Iterator interface**

There are only three methods in the Iterator interface. They are:

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | public boolean hasNext() | It returns true if the iterator has more elements otherwise it returns false. |
| 2 | public Object next() | It returns the element and moves the cursor pointer to the next element. |
| 3 | public void remove() | It removes the last elements returned by the iterator. It is less used. |

| [List](https://www.geeksforgeeks.org/list-interface-java-examples/) | [Set](https://www.geeksforgeeks.org/set-in-java/) | [Map](https://www.geeksforgeeks.org/map-interface-java-examples/) |
| --- | --- | --- |
| The list interface allows duplicate elements | Set does not allow duplicate elements. | The map does not allow duplicate elements |
| The list maintains insertion order. | Set do not maintain any insertion order. | The map also does not maintain any insertion order. |
| We can add any number of null values. | But in set almost only one null value. | The map allows a single null key at most and any number of null values. |
| List implementation classes are [Array List](https://www.geeksforgeeks.org/arraylist-in-java/), [LinkedList](https://www.geeksforgeeks.org/linked-list-in-java/). | Set implementation classes are [HashSet](https://www.geeksforgeeks.org/hashset-in-java/), [LinkedHashSet](https://www.geeksforgeeks.org/linkedhashset-in-java-with-examples/), and [TreeSet](https://www.geeksforgeeks.org/treeset-in-java-with-examples/). | Map implementation classes are [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/), [HashTable](https://www.geeksforgeeks.org/hashtable-in-java/), [TreeMap](https://www.geeksforgeeks.org/treemap-in-java/), [ConcurrentHashMap](https://www.geeksforgeeks.org/concurrenthashmap-in-java/), and [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples/). |
| The list provides get() method to get the element at a specified index. | Set does not provide get method to get the elements at a specified index | The map does not  provide get method to get the elements at a specified index |
| If you need to access the elements frequently by using the index then we can use the list | If you want to create a collection of unique elements then we can use set | If you want to store the data in the form of key/value pair then we can use the map. |
| To traverse the list elements by using Listlterator. | Iterator can be used traverse the set elements | Through keyset, value, and entry set. |

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to  store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory. | Manipulation with LinkedList is **faster** than  ArrayList because it uses a doubly linked list,  so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and**  **queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |
| 5) The memory location for the elements of an ArrayList is contiguous. | The location for the elements of a linked list  is not contagious. |
| 6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList. | There is no case of default capacity in a LinkedList.  In LinkedList, an empty list is created when a  LinkedList is initialized. |
| 7) To be precise, an ArrayList is a resizable array. | LinkedList implements the doubly linked list  of the list interface. |