ARULMIGU PALANIANDAVR ARTS COLLEGE FOR WOMEN
(Autonomous)
(Re-Accredited with 'B++' Grade by NAAC 3rd Cycle)
Run by Arulmigu Dhandayuthapani Swamy Thirukoil, H.R & C.E Dept. Government of Tamil Nadu
A Government Aided College - Affiliated to Mother Teresa Women's University, Kodaikanal
CHINNAKALAYAMPUTHUR(PO), PALANI - 624615


*DEPARTMENT OF COMPUTER SCIENCE*

*&*

*COMPUTER APPLICATION*

**LEARNING RESOURCE**

# JAVA PROGRAMMING

# INTRODUCTION

Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).

The latest release of the Java Standard Edition is Java SE 8. With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms. For example: J2EE for Enterprise Applications, J2ME for Mobile Applications.

The new J2 versions were renamed as Java SE, Java EE, and Java ME respectively. Java is guaranteed to be **Write Once, Run Anywhere**.

Java is:

- **Object Oriented:** In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

- **Platform Independent:** Unlike many other programming languages including Cand C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is beingrun on.

- **Simple:** Java is designed to be easy to learn. If you understand the basic conceptof OOP Java, it would be easy to master.

- **Secure:** With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

- **Architecture-neutral:** Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

- **Portable:** Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.

- **Robust:** Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

- **Multithreaded:** With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

- **Interpreted:** Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The

development process is more rapid and analytical since the linking is an incremental and light-weight process.

- **High Performance:** With the use of Just-In-Time compilers, Java enables high performance.

- **Distributed:** Java is designed for the distributed environment of the internet.

- **Dynamic:** Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

## History of Java

James Gosling initiated Java language project in June 1991 for use in one of his many set- top box projects. The language, initially called ‘Oak’ after an oak tree that stood outside Gosling's office, also went by the name ‘Green’ and ended up later being renamed as Java,from a list of random words.

Sun released the first public implementation as Java 1.0 in 1995. It promised **Write Once, Run Anywhere** (WORA), providing no-cost run-times on popular platforms.

On 13 November, 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

On 8 May, 2007, Sun finished the process, making all of Java's core code free and open- source, aside from a small portion of code

to which Sun did not hold the copyright.

You will also need the following softwares:

- Linux 7.1 or Windows xp/7/8 operating system
- Java JDK 8
- Microsoft Notepad or any other text editor

This tutorial will provide the necessary skills to create GUI, networking, and webapplications using Java.

## Java - Environment Setup

Java Programming environment setup online, so that you can compile and execute all the available examples online at the same time when you are doing your theory work. This gives you confidence in what you are reading and to check the result with different options. Feel free to modify any example and execute it online.

Try the following example using **Try it** option available at the top right corner of the following sample code box:

```
public class MyFirstJavaProgram {

    public static void main(String []args) {
       System.out.println("Hello World");
    }
}
```

For most of the examples given in this tutorial, you will find the

**Try it** option, which you can use to execute your programs and enjoy your learning.

## Local Environment Setup

If you are still willing to set up your environment for Java programming language, then this section guides you on how to download and set up Java on your machine. Following are the steps to set up the environment.

Java SE is freely available from the link Download Java. You can download a version basedon your operating system.

Follow the instructions to download Java and run the **.exe** to install Java on your machine. Once you installed Java on your machine, you will need to set environment variables to point to correct installation directories:

### Setting Up the Path for Windows

Assuming you have installed Java in c:\Program Files\java\jdk directory:

- Right-click on 'My Computer' and select 'Properties'.
- Click the 'Environment variables' button under the 'Advanced' tab.
- Now, alter the 'Path' variable so that it also contains the path to the Java executable. Example, if the path is currently set to 'C:\WINDOWS\SYSTEM32', then change your path to read 'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'.

### Setting Up the Path for Linux, UNIX, Solaris, FreeBSD

Environment variable PATH should be set to point to where the Java binaries have been installed. Refer to your shell

documentation, if you have trouble doing this.

Example, if you use **bash** as your shell, then you would add the following line to the end of your **'.bashrc: export PATH=/path/to/java:$PATH'**

---

### Popular Java Editors

To write your Java programs, you will need a text editor. There are even more sophisticated IDEs available in the market. But for now, you can consider one of the following:

- **Notepad:** On Windows machine, you can use any simple text editor like Notepad (Recommended for this tutorial), TextPad.

- **Netbeans:** A Java IDE that is open-source and free, which can be downloaded from http://www.netbeans.org/index.html.

- **Eclipse:** A Java IDE developed by the eclipse open-source community and can be downloaded from http://www.eclipse.org/.

## Java – Basic Syntax

- **Object -** Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behavior such as wagging their tail, barking, eating. An object is an instance of a class.

- **Class -** A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.

- **Methods -** A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.

- **Instance Variables -** Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

### First Java Program

Let us look at a simple code that will print the words **Hello World**.

```
public class MyFirstJavaProgram {

    /* This is my first java program.
     * This will print 'Hello World' as the output
     */

    public static void main(String []args) { System.out.println("Hello World"); //
        prints Hello World
    }
```

Let's look at how to save the file, compile, and run the program. Please follow thesubsequent steps:

- Open notepad and add the code as above.

- Save the file as: MyFirstJavaProgram.java.

- Open a command prompt window and go to the directory where you saved theclass. Assume it's

- Type 'javac MyFirstJavaProgram.java' and press enter to compile your code. If there are no errors in your code, the command prompt will take you to the next line (Assumption : The path variable is set).

- Now, type ' java MyFirstJavaProgram ' to run your program.

- You will be able to see ' Hello World ' printed on the window.

```
C:\> javac MyFirstJavaProgram.javaC:\> java
MyFirstJavaProgram Hello World
```

**BasicSyntax**

About Java programs, it is very important to keep in mind the following points.

- **Case Sensitivity** - Java is case sensitive, which means identifier **Hello**and **hello** would have different meaning in Java.

- **Class Names** - For all class names the first letter should be in Upper Case.

  If several words are used to form a name of the class, each

inner word's first lettershould be in Upper Case.

**Example**: class MyFirstJavaClass

- **Method Names -** All method names should start with a Lower Case letter.

  If several words are used to form the name of the method, then each inner word'sfirst letter should be in Upper Case.

  **Example**: public void myMethodName()

- **Program File Name -** Name of the program file should exactly match the classname.

  When saving the file, you should save it using the class name (Remember Java iscase sensitive) and append '.java' to the end of the name (if the file name and theclass name do not match, your program will not compile).

  **Example**: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as 'MyFirstJavaProgram.java'

- **public static void main(String args[]) -** Java program processing starts from the main() method which is a mandatory part of every Java program.

**JavaIdentifiers**

All Java components require names. Names used for classes, variables, and methods arecalled **identifiers**.

In Java, there are several points to remember about identifiers. They are as follows:

- All identifiers should begin with a letter (A to Z or a to z), currency character ($)or an underscore (_).
- After the first character, identifiers can have any combination of characters.
- A key word cannot be used as an identifier.
- Most importantly, identifiers are case sensitive.
- Examples of legal identifiers: age, $salary, _value, 1_value.
- Examples of illegal identifiers: 123abc, -salary.

---

**JavaModifiers**

Like other languages, it is possible to modify classes, methods, etc., by using modifiers. There are two categories of modifiers:

- **Access Modifiers:** default, public , protected, private
- **Non-access Modifiers:** final, abstract, strictfp

We will be looking into more details about modifiers in the next section.

**JavaVariables**

Following are the types of variables in Java:

- Local Variables
- Class Variables (Static Variables)
- Instance Variables (Non-static Variables)

**JavaArrays**

Arrays are objects that store multiple variables of the same type. However, an array itselfis an object on the heap.

**JavaEnums**

Enums were introduced in Java 5.0. Enums restrict a variable to have one of only a few predefined values. The values in this enumerated list are called enums.

With the use of enums it is possible to reduce the number of bugs in your code. For example, if we consider an application for a fresh juice shop, it would be possible to restrict the glass size to small, medium, and large. This would make sure that it would notallow anyone to order any size other than small, medium, or large.

```
class FreshJuice {

    enum FreshJuiceSize{ SMALL, MEDIUM, LARGE }
    FreshJuiceSize size;
}

public class FreshJuiceTest {

    public static void main(String args[]){FreshJuice
        juice = new FreshJuice();
        juice.size = FreshJuice.FreshJuiceSize.MEDIUM ;
        System.out.println("Size: " + juice.size);
    }
}
```

**Example**

The above example will produce the following result:

```
Size: MEDIUM
```

**Note:** Enums can be declared as their own or inside a class. Methods, variables,constructors can be defined inside enums as well.

---

**JavaKeywords**

The following list shows the reserved words in Java. These reserved words may not beused as constant or variable or any other identifier names.

| | | | |
|---|---|---|---|
| abstract | assert | boolean | break |
| byte | case | catch | char |

| | | | |
|---|---|---|---|
| class | const | continue | default |
| do | double | else | enum |
| extends | final | finally | float |
| for | goto | if | implements |
| import | instanceof | int | interface |
| long | native | new | package |
| private | protected | public | return |
| short | static | strictfp | super |
| switch | synchronized | this | throw |
| throws | transient | try | void |
| volatile | while | | |

**CommentsinJava**

Java supports single-line and multi-line comments very similar to C and C++. Allcharacters available inside any comment are ignored by Java compiler.

```
public class MyFirstJavaProgram{

    /* This is my first java program.
     *  This will print 'Hello World' as the output
     *  This is an example of multi-line comments.
     */

    public static void main(String []args){
        // This is an example of single line comment
        /* This is also an example of single line comment. */System.out.println("Hello World");
    }
}
```

**Using Blank Lines**

A line containing only white space, possibly with a comment, is known as a blank line, andJava totally ignores it.

**Inheritance**

In Java, classes can be derived from classes. Basically, if you need to create a new class and here is already a class that has some of the code you require, then it is possible to derive your new class from the already existing code.

This concept allows you to reuse the fields and methods of the existing class without having to rewrite the code in a new class. In this scenario, the existing class is called the **superclass** and the derived class is called the **subclass**.

**Interfaces**

In Java language, an interface can be defined as a contract between objects on how to communicate with each other. Interfaces play a vital role when it comes to the concept ofinheritance.

An interface defines the methods, a deriving class (subclass) should use. But theimplementation of the methods is totally up to the subclass.

## Java – Objects & Classes

Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts:

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects
- Instance
- Method
- Message Parsing

In this chapter, we will look into the concepts - Classes and Objects.

- **Object -** Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object isan instance of a class.

- **Class -** A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

**Objects in Java**

All these objects have a state and a behavior.object with a real-world object, they have very similar characteristics.

Software objects also have a state and a behavior. A software object's state is stored in fields and behavior is shown via methods.

So in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods.

**Classes in Java**

A class is a blueprint from which individual objects are created.Following is a sample of a class.

```
public class
    Dog{String
    breed; int
    ageC String
    color;

    void barking(){
    }

    void hungry(){
    }
```

A class can contain any of the following variable types.

- **Local variables:** Variables defined inside methods,

constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.

- **Instance variables:** Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

- **Class variables:** Class variables are variables declared within a class, outside anymethod, with the static keyword.

A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry() and sleeping() are methods.
Following are some of the important topics that need to be discussed when looking into classes of the Java Language.

---

**Constructors**

When discussing about classes, one of the most important sub topic would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main ruleof constructors is that they should have the same name as the class. A class can have more than one

constructor.

Following is an example of a constructor:

```
public class
    Puppy{public
    Puppy(){
    }

    public Puppy(String name){
        // This constructor has one parameter, name.
    }
}
```

Java also supports Singleton Classes where you would be able to create only one instanceof a class.

The Singleton's purpose is to control object creation, limiting the number of objects to only one. Since there is only one Singleton instance, any instance fields of a Singleton will occuronly once per class, just like static fields. Singletons often control access to resources, such as database connections or sockets.

For example, if you have a license for only one connection for your database or your JDBC driver has trouble with multithreading, the Singleton makes sure that only one connectionis made or that only one thread can access the connection at a time.

**Implementing Singletons**

**Example 1**

The easiest implementation consists of a private constructor and a field to hold its result, and a static accessor method with a name

like getInstance().

The private field can be assigned from within a static initializer block or, more simply, using an initializer. The getInstance( ) method (which must be public) then simply returnsthis instance

```
// File Name:
Singleton.javapublic class
Singleton {

    private static Singleton singleton = new Singleton( );



    /* A private Constructor prevents any other
     * class from instantiating.
     */
    private Singleton(){ }

    /* Static 'instance' method */
    public static Singleton getInstance( ) {
        return singleton;
    }
    /* Other methods protected by singleton-ness
    */protected static void demoMethod( ) {
        System.out.println("demoMethod for singleton");
    }
}
```

The ClassicSingleton class maintains a static reference to the lone

singleton instance and returns that reference from the static getInstance() method.

Here, ClassicSingleton class employs a technique known as lazy instantiation to create the singleton; as a result, the singleton instance is not created until the getInstance() methodis called for the first time. This technique ensures that singleton instances are created onlywhen needed.

---

**Creating an Object**

As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class:

- **Declaration:** A variable declaration with a variable name with an object type.
- **Instantiation:** The 'new' keyword is used to create the object.
- **Initialization:** The 'new' keyword is followed by a call to a constructor. This callinitializes the new object.

Following is an example of creating an object:

```
public class Puppy{

    public Puppy(String name){
        // This constructor has one parameter, name.
        System.out.println("Passed Name is :" + name );
    }

    public static void main(String []args){
        // Following statement would create an object myPuppyPuppy myPuppy = new
        Puppy( "tommy" );
    }
}
```

If we compile and run the above program, then it will produce the following result:

```
Passed Name is :tommy
```

## Accessing Instance Variables and Methods

Instance variables and methods are accessed via created objects. To access an instancevariable, following is the fully qualified path:

```
/* First create an object */ ObjectReference =
new Constructor();

/* Now call a variable as follows */
ObjectReference.variableName;

/* Now you can call a class method as follows */
ObjectReference.MethodName();
```

## Source File Declaration Rules

As the last part of this section, let's now look into the source file declaration rules. These rules are essential when declaring classes, import statements and package statements ina source file.

- There can be only one public class per source file.

- A source file can have multiple non-public classes.

- The public class name should be the name of the source file as well which should be appended by **.java** at the end. For example: the class name is public class Employee{} then the source file should be as Employee.java.

- If the class is defined inside a package, then the package statement should be thefirst statement in the source file.

- If import statements are present, then they must be written between the package statement and the class declaration. If there are no package statements, then the import statement should be the first line in the source file.

- Import and package statements will imply to all the classes present in the source file. It is not possible to declare different import and/or package statements to different classes in the source file.

Classes have several access levels and there are different types of classes; abstract classes, final classes, etc. We will be explaining about all these in the access modifiers chapter.

Apart from the above mentioned types of classes, Java also has some special classes calledInner classes and Anonymous classes.

## String

Every string we create is actually an object of type String.

String constants are actually **String objects**.

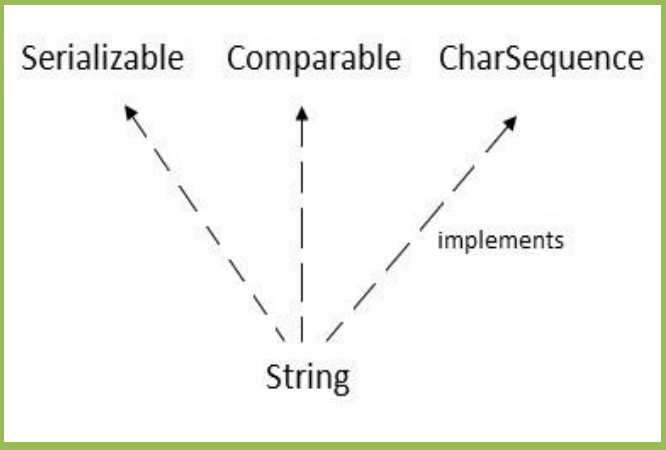
Example:

```
System.out.println("This is a String, too");
```

Objects of type String are immutable i.e. once a String object is created, its contents cannot be altered.

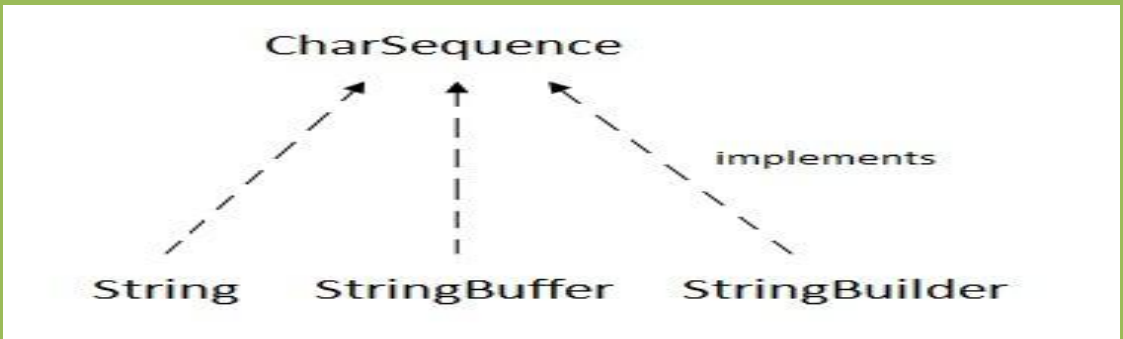
String Class

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



CharSequence Interface

- The CharSequence interface is used to represent sequence of characters. It is implemented by String, StringBuffer and StringBuilder classes. It means, we can create string in java by using these 3 classes.




- In java, 4 predefined classes are provided that either represent strings or provide functionality to manipulate them. Those classes are:
  - String
  - StringBuffer
  - StringBuilder
  - StringTokenizer

String, StringBuffer, and StringBuilder classes are defined in java.lang package and all are final.

All of them implement the CharSequence interface.

**How to create String object?**

There are two ways to create String object:

By string literal

By new keyword

**1) String Literal**

Java String literal is created by using double quotes. For Example:

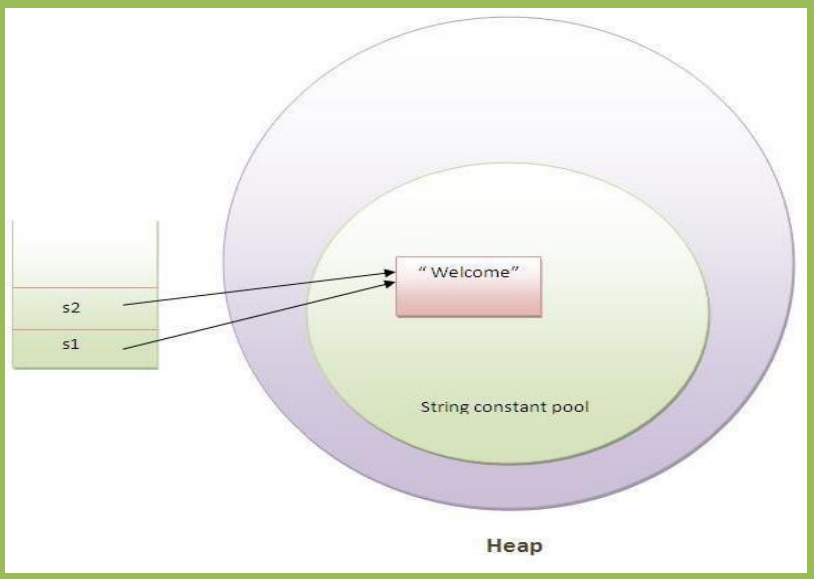
String s="welcome";

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 string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

String s1="Welcome";

String s2="Welcome";

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




2) By new keyword

String s=new String("Welcome");

- 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 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);
 }
}
```

**Why String Handling?**

String handling is required to perform following operations on some string:

Compare two strings

Search for a substring

Concatenate two strings

Change the case of letters within a string

**Creating String objects**

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

**String Constructor**

public
String
()
public
String
(String
)
public
String
(char

[])

public

String

(byte

[])

public String (char [], int offset, int no_of_chars) public String (byte [], int offset, int no_of _bytes)

**Examples**

```
char [] a = {'c', 'o', 'n', 'g', 'r', 'a', 't', 's'};
byte [] b = {82, 65, 86, 73, 75, 65, 78, 84};

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

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

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

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

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

**String Concatenation**

- Concatenating Strings:

```
String age = "9";
String s = "He is " + age + " years old."; System.out.println(s);
```

- Using concatenation to prevent long lines:

```
String longStr = “This could have been” +
                 “a very long line that would have” +
                 “wrapped around.  But string”+
                “concatenation  prevents this.”;
System.out.println(longStr);
```

String Concatenation with Other Data Types

- We can concatenate strings with other types of data.

Example:

```
int age = 9;
String s = "He is " + age + " years old.";
System.out.println(s);
```

**Methods of String class**

1.String Length:

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

```
public int length()
```

Example:

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

2. concat( ): used to concatenate two strings. String

concat(String str)

- This method creates a new object that contains the invoking string with the contents of str appended to the end.

- concat( ) performs the same function as +.

Example:

String s1 = "one"; String s2 = s1.concat("two");

- It generates the same result as the following sequence: String s1 = "one"; String s2 = s1 + "two";

**Character Extraction**

3. charAt(): used to obtain the character from the specified index from a string. public char charAt (int index);
4. toCharArray(): returns a character array initialized by the contents of the string. public char [] toCharArray();

Example: String s = “India”; char c[] = s.toCharArray(); for (int i=0; i<c.length; i++)
{ if (c[i]>= 65 && c[i]<=90) c[i] += 32;
System.out.print(c[i]); }

5. getChars(): used to obtain set of characters from the string.

public void getChars(int start_index, int end_index, char[], int offset)

Example: String s = “KAMAL”; char b[] = new char [10]; b[0] = ‘N’; b[1] = ‘E’; b[2] = ‘E’; b[3] = ‘L’; s.getChars(0, 4, b, 4);

System.out.println(b);

## STRING COMPARISON

There are three ways to compare string in java:

1.By equals() method

2.By = = operator

3.By compareTo() method

### 1. By equals() Method

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

  public boolean equals (Object str)

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

Note:

- This method is defined in Object class and overridden in String class.
- equals(), in Object class, compares the value of reference not the content.
- In String class, equals method is overridden for content-wise comparison of two strings.

**Example**

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

```
                    s1.equals(s4));
                    System.out.println(s1 + " equalsIgnoreCase " + s4 + " ->
    “                                                    +s1.equalsIgnoreCase(s4));
                    }
            }
```

**2. String compare by == operator**

 The = = operator compares references not values.

```
class Teststringcomparison3
{   public   static   void
    main(String args[])
     {
            String s1="Sachin"; String s2="Sachin";
            String s3=new String("Sachin");
            System.out.println(s1==s2);
//true  (because  both  refer  to  same  instance)
            System.out.println(s1==s3);
//false(because  s3  refers  to  instance  created  in
    nonpool) }
}
```

**3. String compare by compareTo() method**

The String compareTo() method compares values lexicographically and returns an integer value that

describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

**s1 == s2** :0

**s1 > s2** :positive value  **s1 < s2** :negative value

String Comparison

- startsWith( ) and endsWith( ):
  - The startsWith( ) method determines whether a given String begins with a specified string.
  - Conversely, endsWith( ) determines whether the String in question ends with a specified string.

boolean startsWith(String str) boolean endsWith(String str)

- Example

```
String s="Sachin";
System.out.println(s.startsWith("Sa"));
//true
System.out.println(s.endsWith("n"));
//true
```

**String Comparison**

compareTo( ):

- A string is less than another if it comes before the other in dictionary order.
- A string is greater than another if it comes after the other in dictionary order. int compareTo(String str)

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

**Example**

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

```
                = arr[j]; arr[j] = arr[i];
        arr[i] = t;
                                                }
                                        }
                    System.out.println(arr[j]);
                }
            }
        }
```

**Searching Strings**

- The String class provides two methods that allow us to search a string for a specified character or substring:
  indexOf( ): Searches for the first occurrence of a character or substring. int indexOf(int ch)
  lastIndexOf( ): Searches for the last occurrence of a character or substring. int lastIndexOf(int ch)
- To search for the first or last occurrence of a substring, use int indexOf(String str) int lastIndexOf(String str)
- We can specify a starting point for the search using these forms:
  int indexOf(int ch, int startIndex) int lastIndexOf(int ch, int startIndex) int indexOf(String str, int startIndex) int lastIndexOf(String str, int startIndex)
- Here, startIndex specifies the index at which point the search begins.
- For indexOf( ), the search runs from startIndex to the end of the string.
- For lastIndexOf( ), the search runs from startIndex to zero.

**Example**

```
class indexOfDemo {
    public static void main(String args[]) {
            String s = "Now is the time for all good men " +
```

```
                    "to come to the aid of their country.";
        System.out.println(s);
        System.out.println("indexOf(t) = " + s.indexOf('t'));
        System.out.println("lastIndexOf(t) = " + s.lastIndexOf('t'));
        System.out.println("indexOf(the) = " + s.indexOf("the"));
        System.out.println("lastIndexOf(the) = " + s.lastIndexOf("the"));
        System.out.println("indexOf(t, 10) = " + s.indexOf('t', 10));
        System.out.println("lastIndexOf(t, 60) = " + s.lastIndexOf('t', 60));
        System.out.println("indexOf(the, 10) = " + s.indexOf("the", 10));
        System.out.println("lastIndexOf(the, 60) = " + s.lastIndexOf("the", 60));
        }
}
```

**Modifying a String**

- Because String objects are immutable, whenever we want to modify a String, it will construct a new copy of the string with modifications.
- substring(): used to extract a part of a string.

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

Example: String s = “ABCDEFG”;

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

Note: Substring from start_index to end_index-1 will be returned.

**Example of java substring**

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

```
        String s="SachinTendulkar";
        System.out.println(s.substring(6));
        //Tendulkar
        System.out.println(s.substring(0,6));
        //Sachin
}
}
```

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

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

  String replace(char original, char replacement)
- Here, original specifies the character to be replaced by the character specified by replacement.

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

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

  String replace(CharSequence original, CharSequence replacement)
- Example:

  String s1="Java is a programming language. Java is a plat form. Java is an Island.";

  String replaceString=s1.replace("Java","Kava");

//replaces all occurrences of "Java" to "Kava"

  System.out.println(replaceString);

**trim( )**

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

S

tring trim( ) Example:

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

This puts the string “Hello World” into s.

- The string trim() method eliminates white spaces before and after string.
- Example

String s=" Sachin ";

System.out.println(s);// Sachin

System.out.println(s.trim());//Sachin

**Changing the Case of Characters Within a String toLowerCase() & toUpperCase()**

 Both methods return a String object that contains the uppercase or lowercase equivalent of the invoking String.

String

toLowerCase( )

String

toUpperCase( )

Example

String s="Sachin";

System.out.println(s.toUpperCase()); //SACHIN

System.out.println(s.toLowerCase());

//sachin

System.out.println(s);

//Sachin(no change in original)

break Statement

continue Statement

return Statement

**break Statement**

break statement has three uses:

- Terminates a statement sequence in a switch statement
- Used to exit a loop
- Used as a “civilized” form of goto

- The break statement has two forms:
  - Labeled
  - Unlabeled

**Unlabeled break**

- An unlabeled break is used to terminate a for, while, or do-while loop and switch statement.

**Labeled break Statement**

- Java defines an expanded form of the break statement. **break** label;
- By using this form of break, we can break out of one or more blocks of code.
- When this form of break executes, control is transferred out of the named block.

- An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
- An exception is an abnormal condition that arises in a code sequence at run time.
- A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code.
- In other words, “An exception is a run-time error.”

**Exception Handling**

- An Exception is a run-time error that can be handled programmatically in the application and does not result in abnormal program termination.
- Exception handling is a mechanism that facilitates programmatic handling of run-time errors.
- In java, each run-time error is represented by an object.

**Exception (Class Hierarchy)**

- At the root of the class hierarchy, there is a class named ‘Throwable’ which represents the basic features of run-time errors.
- There are two non-abstract sub-classes of Throwable.
- Exception : can be handled • Error : can’t be handled
- RuntimeException is the sub-class of Exception.
- Each exception is a run-time error but all run-time errors are not exceptions.

Checked Exception

 Checked Exceptions are those, that have to be either caught or declared to be thrown in the method in which they are raised.

Unchecked Exception

- Unchecked Exceptions are those that are not forced by the compiler either to be caught or to be thrown.
- Unchecked Exceptions either represent common programming errors or those run-time errors that can’t be handled through exception handling.

Commonly used sub-classes of Exception

- ArithmeticException
- ArrayIndexOutOfBoundsException
- NumberFormatException
- NullPointerException
- IOException

- When the Java run-time system detects the attempt to divide by zero, it constructs a new exception object and then throws this exception.

- Once an exception has been thrown, it must be caught by an exception handler and dealt with immediately.

Why Exception Handling?

- When the default exception handler is provided by the Java run-time system , why Exception Handling?
- Exception Handling is needed because:
- It allows to fix the error, customize the message .
- It prevents the program from automatically terminating

**Exception Handling**

**Keywords for Exception Handling**

- try
- catch
- throw
- throws
- finally

**Keywords for Exception Handling try**

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

try {

Statements whose execution may cause an exception }

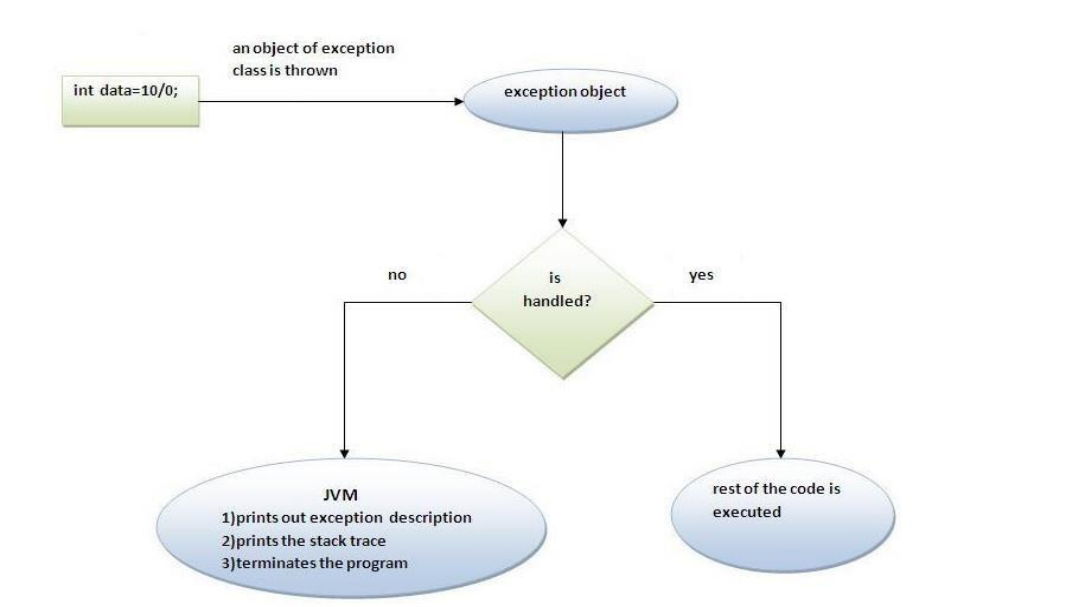
Note: try block is always used either with catch or finally or with both.

**Keywords for Exception Handling catch**

- catch is used to define a handler.
- It contains statements that are to be executed when the exception represented by the catch block is generated.

- If program executes normally, then the statements of catch block will not executed.
- If no catch block is found in program, exception is caught by JVM and program is terminated.
- It must be used after the try block only.
- You can use multiple catch block with a single try.

## Internal working of java try-catch block




## Nested try block

- Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.
- **Syntax:**
- ....
- try
- {
- statement 1;
- statement 2;
- try
- {
- statement 1;
- statement 2;
- }

```
catch(Exception e)
{
}
}
catch(Exception e)
{
}
....
```

Java finally block

- Java finally block is always executed whether exception is handled or not.
- **Java finally block** is a block that is used to execute important code such as closing connection, stream etc.
- Java finally block follows try or catch block.

# Multithreading

Threads are the backbone of multithreading. We are living in a real-world which in itself is caught on the web surrounded by lots of applications. With the advancement in technologies we cannot achieve the speed required to run them simultaneously unless we introduce the concept of multi tasking efficiently. It is achieved by the concept of thread.

**Real-life Example**

Suppose you are using two tasks at a time on the computer, be it using Microsoft Word and listening to music. These two tasks are called **processes**. So you start typing in Word and at the same time start music app, this is called **multitasking**. Now you committed a mistake in a Word and spell check shows exception, this means Word is a process that is broken down into sub-processes. Now if a machine is dual-core then one process or task is been handled by one core and music is been handled by another core.

In the above example, we come across both multiprocessing and multithreading. These are somehow indirectly used to achieve multitasking. In this way the mechanism of dividing the tasks is called multithreading in which every process or task is called by a thread where a thread is responsible for when to execute, when to stop and how long to be in a waiting state. Hence, a **thread** is the smallest unit of processing whereas **multitasking** is a process of executing multiple tasks at a time.

**Multitasking is being achieved in two ways**:

1. **Multiprocessing**: Process-based multitasking is a heavyweight process and occupies different address spaces in memory. Hence, while switching from one process to another, it will require some time be it very small, causing a lag because of switching. This happens as registers will be loaded in memory maps and the list will be updated.
2. **Multithreading**: Thread-based multitasking is a lightweight process and occupies the same address space. Hence, while switching cost of communication will be very less.

**Below is the Lifecycle of a Thread been illustrated**

1. **New**: When a thread is just created.
2. **Runnable**: When a start() method is called over thread processed by the thread scheduler.
   - Case A: Can be a running thread
   - Case B: Can not be a running thread
3. **Running**: When it hits case 1 means the scheduler has selected it to be run the thread from runnable state to run state.
4. **Blocked**: When it hits case 2 meaning the scheduler has selected not to allow a thread to change state from runnable to run.
5. **Terminated**: When the run() method exists or stop() method is called over a thread.

If we do incorporate threads in operating systems one can perceive that the process scheduling algorithms in operating systems are strongly deep-down working on the same concept incorporating thread in **Gantt charts**. A few of the most popular are listed below which wraps up all of them and are used practically in software development.

- First In First Out
- Last In First Out
- Round Robin Scheduling

Now Imagine the concept of **Deadlock in operating systems with threads** – how the switching is getting computed over internally if one only has an overview of them.

So far we have understood multithreading and thread conceptually, so we can conclude **advantages of multithreading** before moving to any other concept or getting to programs in multithreading.

- The user is not blocked as threads are independent even if there is an issue with one thread then only the corresponding process will be stopped rest all the operations will be computed successfully.
- Saves time as too many operations are carried over at the same time causing work to get finished as if threads are not used the only one process will be handled by the processor.
- Threads are independent though sharing the same address space.

So we have touched all main concepts of multithreading but the question striving in the head is left. why do we need it, where to use it and how? Now, we will discuss all three scenarios why multithreading is needed and where it is implemented via the help of programs in which we will be further learning more about threads and their methods. We need multithreading in four scenarios as listed.

- Thread Class
- Mobile applications
    - Asynchronous thread
- Web applications
- Game Development

**Note:** By default we only have one main thread which is responsible for main thread exception as you have encountered even without having any prior knowledge of multithreading

## Two Ways to Implement Multithreading

- **Using Thread Class**
- **Using Runnable Interface**

### **Method 1: Using** Thread Class

Java provides Thread class to achieve programming invoking threads thereby some major methods of thread class are shown below in the tabular format with which we deal frequently along the action performed by them.

| Methods | Action Performed |
|---|---|
| isDaemon() | It checks whether the current thread is daemon or not |
| start() | It starts the execution of the thread |
| run() | It does the executable operations statements in the body of this method over a thread |
| sleep() | It is a static method that puts the thread to sleep for a certain time been passed as an argument to it |
| wait() | It sets the thread back in waiting state. |
| notify() | It gives out a notification to one thread that is in waiting state |
| notifyAll() | It gives out a notification to all the thread in the waiting state |
| setDaemon() | It set the current thread as Daemon thread |
| stop() | It is used to stop the execution of the thread |

| Methods | Action Performed |
|---|---|
| resume() | It is used to resume the suspended thread. |

**Pre-requisites:** Basic syntax and methods to deal with threads

Now let us come up with how to set the name of the thread. By default, threads are named thread-0, thread-1, and so on. But there is also a method that is often used as **setName()** method. Also corresponding to it there is a method **getName()** which returns the name of the thread be it default or settled already by using setName() method. The syntax is as follows:

**Syntax:**

**(a)** Returning the name of the thread

public String getName() ;

**(b)** Changing the name of the thread

public void setName(String name);

Taking a step further, let us dive into the implementation part to understand more concepts about multithreading. So, there are basically two ways of implementing multithreading:

**Illustration:** Consider if one has to multiply all elements by 2 and there are 500 elements in an array.

```
// Case 1

// Java Program to illustrate Creation and execution of

// thread via start() and run() method in Single inheritance
```

```
// Class 1

// Helper thread Class extending main Thread Class

class MyThread1 extends Thread {



    // Method inside MyThread2

    // run() method which is called as

    // soon as thread is started

    public void run()

    {



        // Print statement when the thread is called

        System.out.println("Thread1 is running");

    }

}
```

```
// Class 2
// Main thread Class extending main Thread Class
class MyThread2 extends Thread {

    // Method inside MyThread2
    // run() method which is called
    // as soon as thread is started
    public void run()
    {

        // run() method which is called as soon as thread is
        // started

        // Print statement when the thread is called
        System.out.println("Thread2 is running");
```

```
    }
}


// Class 3
// Main Class
class GFG {


    // Main method
    public static void main(String[] args)
    {


        // Creating a thread object of our thread class
        MyThread1 obj1 = new MyThread1();
        MyThread2 obj2 = new MyThread2();
```

```
        // Getting the threads to the run state



        // This thread will transcend from runnable to run

        // as start() method will look for run() and execute

        // it

        obj1.start();



        // This thread will also transcend from runnable to

        // run as start() method will look for run() and

        // execute it

        obj2.start();

    }

}
```

**Output:**

**Case 1:**

Thread1 is running

Thread2 is running

Here we have created our two thread classes for each thread. In the main method**,** we are simply creating objects of these thread classes where objects are now threads. So in main**,** we call thread using start() method over both the threads. Now start() method starts the thread and lookup for their run() method to run. Here both of our thread classes were having run() methods, so both threads are put to run state from runnable by the scheduler, and output on the console is justified.

**Case 2:**

Thread 1 is running

Here we have created our two thread classes for each thread. In the main method, we are simply creating objects of these thread classes where objects are now threads. So in main, we call thread using start() method over both the threads. Now start() method starts the thread and lookup their run() method to run. Here only class 1 is having the run() method to make the thread transcend from runnable to run state to execute whereas thread 2 is only created but not put to run state by the scheduler as its corresponding run() method was missing. Hence, only thread 1 is called rest thread 2 is created only and is in the runnable state later blocked by scheduler because the corresponding run() method was missing.

**Case 3:**

Thread 2

Thread 1 is running

### Method 2: Using Runnable Interface

Another way to achieve multithreading in java is via the Runnable interface. Here as we have seen in the above example in way 1 where Thread class is extended. Here Runnable interface being a functional interface has its own run() method. Here classes are implemented to the Runnable interface. Later on, in the main() method, Runnable reference is created for the classes that are implemented in order to make bondage with Thread class to run our own corresponding run() methods. Further, while creating an object of Thread class we will pass these references in Thread class as its constructor allows only one runnable object, which is passed as a parameter while creating Thread class object in a main()

method. Now lastly just like what we did in Thread class, start() method is invoked over the runnable object who are now already linked with Thread class objects, so the execution begins for our run() methods in case of Runnable interface.

## Special Methods of Threads

Now let us discuss various methods that are there for threads. Here we will be discussing major methods in order to have a practical understanding of threads and multithreading which are sequential as follows:

1. start() Method
2. suspend() Method
3. stop() Method
4. wait() Method
5. notify() Method
6. notifyAll() Method
7. sleep() Method
   - Output Without sleep() Method
   - Output with sleep() method in Serial Execution Processes (Blocking methods approach)
   - Output with sleep() method in Parallel Execution Processes (Unblocking methods approach)
8. join() Method

## Priorities in Threads

**Priorities in threads** is a concept where each thread is having a priority which is represented by numbers ranging from 1 to 10.

- The default priority is set to 5 as excepted.
- Minimum priority is set to 1.

- Maximum priority is set to 10.

Here 3 constants are defined in it namely as follows:

1. public static int NORM_PRIORITY
2. public static int MIN_PRIORITY
3. public static int MAX_PRIORITY

Let us discuss it with an example to get how internally the work is getting executed. Here we will be using the knowledge gathered above as follows:

- We will use currentThread() method to get the name of the current thread. User can also use setName() method if he/she wants to make names of thread as per choice for understanding purposes.
- getName() method will be used to get the name of the thread.

If we look carefully we do see the outputs for cases 1 and 2 are equivalent. This signifies that when the user is not even aware of the priority threads still NORM_PRIORITY is showcasing the same result as to what default priority is. It is because the default priority of running thread as soon as the corresponding start() method is called is executed as per setting priorities for all the thread to 5 which is equivalent to the priority of NORM case. This is because both the outputs are equivalent to each other. While in case 3 priority is set to a minimum on a scale of 1 to 10 so do the same in case 4 where priority is assigned to 10 on the same scale. Hence, all the outputs in terms of priorities are justified. Now let us move ahead onto an important aspect of priority threading been incorporated in daily life – Daemon thread

**Daemon thread** is basically a service provider thread that provides services to the user thread. The scope for this thread start() or be it terminate() is completely dependent on the user's thread as it supports in the backend for user threads being getting run. As soon as the user thread is terminated daemon thread is also terminated at the same time as being the service provider thread.

Hence, the characteristics of the Daemon thread are as follows:

- It is only the service provider thread not responsible for interpretation in user threads.

- So, it is a low-priority thread.
- It is a dependent thread as it has no existence on its own.
- JVM terminates the thread as soon as user threads are terminated and come back into play as the user's thread starts.
- Yes, you guess the most popular example is garbage collector in java. Some other examples do include 'finalizer'.

**Exceptions:** IllegalArgumentException as return type while setting a Daemon thread is boolean so do apply carefully.

**Note:** To get rid of the exception users thread should only start after setting it to daemon thread. The other way of starting prior setting it to daemon will not work as it will pop-out IllegalArgumentException

As discussed above in the Thread class two most widely used method is as follows:

Let us discuss the implementation of the Daemon thread before jumping onto the garbage collector.

```
// Java Program to show Working of Daemon Thread
// with users threads

import java.io.*;
// Importing Thread class from java.util package
import java.util.*;

// Class 1
// Helper Class extending Thread class
class CheckingMyDaemonThread extends Thread {

    // Method
    // run() method which is invoked as soon as
    // thread start via start()
```

```
    public void run()
    {

        // Checking whether the thread is daemon thread or
        // not
        if (Thread.currentThread().isDaemon()) {

            // Print statement when Daemon thread is called
            System.out.println(
                "I am daemon thread and I am working");
        }

        else {

            // Print statement whenever users thread is
            // called
            System.out.println(
                "I am user thread and I am working");
        }
    }
}

// Class 2
// Main Class
class GFG {

    // Main driver method
    public static void main(String[] args)
```

```
    {

        // Creating threads in the main body
        CheckingMyDaemonThread t1
            = new CheckingMyDaemonThread();
        CheckingMyDaemonThread t2
            = new CheckingMyDaemonThread();
        CheckingMyDaemonThread t3
            = new CheckingMyDaemonThread();

        // Setting thread named 't2' as our Daemon thread
        t2.setDaemon(true);

        // Starting all 3 threads using start() method
        t1.start();
        t2.start();
        t3.start();

        // Now start() will automatically
        // invoke run() method
    }
}
```

Another way to achieve the same is through **Thread Group** in which as the name suggests multiple threads are treated as a single object and later on all the operations are carried on over this object itself aiding in providing a substitute for the Thread Pool.

**Note:**

While implementing ThreadGroup do note that ThreadGroup is a part of '**java.lang.ThreadGroup'** class not a part of Thread class in java so do peek out constructors and methods of ThreadGroup class before moving ahead keeping a check over deprecated methods in his class so as not to face any ambiguity further.

Here main() method in itself is a thread because of which you do see Exception in main() while running the program because of which **system.main thread exception** is thrown sometimes while execution of the program.

## Synchronization

It is the mechanism that bounds the access of multiple threads to share a common resource hence is suggested to be useful where only one thread at a time is granted the access to run over.

It is implemented in the program by using 'synchronized' keyword.

Now let's finally discuss some advantages and disadvantages of synchronization before implementing the same. For more depth in synchronization, one can also learn object level lock and class level lock and do notice the differences between two to get a fair understanding of the same before implementing the same.

**Why synchronization is required?**

Data inconsistency issues are the primary issue where multiple threads are accessing the common memory which sometimes results in faults in order to avoid that a thread is overlooked by another thread if it fails out.

1. Data integrity
2. To work with a common shared resource which is very essential in the real world such as in banking systems.

**Note:** Do not go for synchronized keyword unless it is most needed, remember this as there is no priority setup for threads, so if the main thread runs before or after other thread the output of the program would be different.

The biggest advantage of synchronization is the increase in idiotic resistance as one can not choose arbitrarily an object to lock on as a result string literal can not be locked or be the content. Hence, these bad practices are not possible to perform on synchronized method block.

As we have seen humongous advantages and get to know how important it is but there comes disadvantage with it.

**Disadvantage:** Performance issues will arise as during the execution of one thread all the other threads are put to a blocking state and do note they are not in waiting state. This causes a performance drop if the time taken for one thread is too long.

As perceived from the image in which we are getting that count variable being shared resource is updating randomly. It is because of multithreading for which this concept becomes a necessity.

- **Case 1:** If 'main thread' executes first then count will be incremented followed by a 'thread T' in synchronization
- **Case 2:** If 'thread T' executes first then count will not increment followed by 'main thread' in synchronization

**Implementation:** Let us take a sample program to observe this 0 1 count conflict

**Example:**

```
// Java Program to illustrate Output Conflict between
// Execution of Main thread vs Thread created

// count = 1 if main thread executes first
// count = 1 if created thread executes first
```

```
// Importing basic required libraries
import java.io.*;
import java.util.*;

// Class 1
// Helper Class extending Thread class
class MyThread extends Thread {

    // Declaring and initializing initial count to zero
    int count = 0;

    // Method 1
    // To increment the count above by unity
    void increment() { count++; }

    // Method 2
    // run method for thread invoked after
    // created thread has started
    public void run()
    {

        // Call method in this method
        increment();

        // Print and display the count
        System.out.println("Count : " + count);
    }
```

```
}

// Class 2
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Creating the above our Thread class object
        // in the main() method
        MyThread t1 = new MyThread();

        // start() method to start execution of created
        // thread that will look for run() method
        t1.start();
    }
}
```

**Output Explanation:**

Here the count is incremented to 1 meaning ‘main thread‘ has executed prior to ‘created thread‘. We have run it many times and compiled and run once again wherein all cases here main thread is executing faster than created thread but do remember output may vary. Our created thread can execute prior to ‘main thread‘ leading to ‘Count : 0’ as an output on the console.

Now another topic that arises in dealing with synchronization in threads is Thread safety in java synchronization is the new concept that arises out in synchronization so let us discuss it considering

- A real-life scenario followed by
- Pictorial representation as an illustration followed by
- Technically description and implementation

**Real-life Scenario**

Suppose a person is withdrawing some amount of money from the bank and at the same time the ATM card registered with the same account number is carrying on withdrawal operation by some other user. Now suppose if withdrawing some amount of money from net banking makes funds in account lesser than the amount which needed to be withdrawal or the other way. This makes the bank unsafe as more funds are debited from the account than was actually present in the account making the bank very unsafe and is not seen in daily life. So what banks do is that they only let one transaction at a time. Once it is over then another is permitted.

**Illustration:**

Interpreting the same technology as there are two different processes going on which object in case of parallel execution is over headed by threads. Now possessing such traits over threads such that they should look after for before execution or in simpler words making them synchronized. This mechanism is referred to as Thread Safe with the use of the keyword ‘synchronized‘ before the common shared method/function to be performed parallel.

**Technical Description:**

As we know Java has a feature, Multithreading, which is a process of running multiple threads simultaneously. When multiple threads are working on the same data, and the value of our data is changing, that scenario is not thread-safe, and we will get inconsistent results. When a thread is already working on an object and preventing another thread from working on the same object, this process is called Thread-Safety. Now there are several ways to achieve thread-safety in our program namely as follows:

1. Using Synchronization
2. Using Volatile Keyword

3. Using Atomic Variable
4. Using Final Keyword

**Conclusion:** Hence, if we are accessing one thread at a time then we can say thread-safe program and if multiple threads are getting accessed then the program is said to be thread-unsafe that is one resource at a time can not be shared by multiple threads at a time.

**Implementation:**

- Java Program to illustrate Incomplete Thread Iterations returning counter value to Zero irrespective of iteration bound
- Java Program to Illustrate Complete Thread Iterations illustrating join() Method
- Java Program to Illustrate thread-unsafe or non-synchronizing programs as of incomplete iterations
- Java Program to Illustrate Thread Safe And synchronized Programs as of Complete iterations using '**synchronized**' Keyword.

## ONE MARK QUESTION AND ANSWER

1. The insulation of the data from direct access by the program is called _____.

**Ans: Data hiding**

2. The linking of a Procedure call to the code at runtime is called _____.

**Ans: Dynamic Binding.**

3. Define Polymorphism.

**Ans: The ability to take more than one form.**

4. The old name of Java was _____ .

**Ans: Oak.**

5. For interpretation of java program , _____ command is used.

**Ans: java.**

6. _____ command is used to compile a java program.

**Ans: javac.**

7. What do you mean by javap?

[A] java Compiler [B] java Interpreter

[C] java Disassembler [D] java Debugger

**Ans: [C] java Disassembler**

8. Java intermediate code is known as _____.

**Ans: Byte code.**

9. Hot java is _____.

**Ans: Web browser.**

10. What is the full form of JVM?

**Ans: Java Virtual Machine.**

11. Java contains Struct and Union datatypes (T/F).

**Ans: False.**

12. What is the Expansion of JSL ?

**Ans: Java Standard Library.**

13. JDK stands for _____.

**Ans: Java Development Kit.**

14. _____ is a software that interprets Java Byte codes.

[A] Java Virtual Machine [B] Java compiler

[C] Java Debugger [D] Java API

**Ans: [A] Java Virtual Machine**

15. Smallest Individual units in a programs are known as _____.

**Ans : Tokens.**

16. In java ,the data items are called _____ .

**Ans: Fields.**

17. Objects in java are created using _____ operator.

**Ans: new.**

18. Static variables and Static methods are referred to as _____ and _____ .

**Ans: Class Variables and Class Methods.**

19. _____ method can be called without using object.

**Ans: Static method.**

20. The mechanism of deriving a new class from an old one is

called_____.

**Ans : Inheritance**

21. _____keyword is used only within a subclass constructor method.

**Ans : Super**

22. **State True or False**

Java run-time is an automatic garbage collecting system.

**Ans : True**

23. _____ methods are used to destroy the objects created by the

constructor methods.

**Ans : finalize( )**

24. Methods that have same name , but different parameter list and different definition is

known as _____.

**Ans : Method Overloading**

25. _____method can't be overridden.

**Ans : final**

26. Which of the following feature is not supported by java ?

[A] Multithreading [B] Reflection

[C] Operator Overloading [D] Garbage Collection

**Ans: [C] Operator Overloading**

27._____class cannot be subclassed in java.

**Ans : Final class**

28. **Say Yes or No**

Can we declare abstract static method ?

[A] Yes [B] No

**Ans : [B] No**

29. If method have same name as class name and method don't have any return type, then it is known as _____.

**Ans : Constructor**

30. _____ method can be defined only once in a program.

**Ans : main method**

31. _____are grouping a variety if classes and / or interfaces.

**Ans : Packages**

32. API stands for _____.

**Ans : Application Programming Interface**.

33. _____ package support a write network programming.

[A] java.lang [B] java.net

[C] java.applet [D] java.util

**Ans: [B] java.net**

34. **State True or False**

When an interface extends two or more interfaces , they are separated by colon.

**Ans : False**

35._____ acts as container for data & methods.

**Ans : Class**

36._____ statement is used to access the package.

**Ans : import statement**

37. State True or False

i ) Packages begins with lowercase letters.

ii)Java does not support package hierarchy.

**Ans :i)True ii)False**

38. _____ keyword is used to refer to member of base class from a sub class.

**Ans:super**

39. A tiny program which has a single flow of control is called_____.

**Ans:Thread**

40. The ability of a language to support multithread is referred to as_____.

**Ans:Concurrency**

41. A program that contains multiple flows of control is known as_____.

**Ans:Multiplied Program**

42 _____ is an operating system concept in which multiple task are performed simultaneously.

**Ans:Multitasking**

43. The process of assigning time to thread is known as _____.

**Ans:Time-slicing**

44. What is the name of the method used to start a thread execution?

[A]init(); [B]start(); [C]run(); [D]resume();

**Ans:[B]start();**

45. _____ method is the heart and soul of any thead.

**Ans:run()**

46. An Applet developed locally and stored in a local system is known as

_____.

**Ans:Local applet**

47. To run applet_____command is used.

**Ans:Appletviewer**

48. State True or False

(i)Applet do not use the main() method for initiating the execution of the code.

(ii)Applet can communicate with other serves on the network.

**Ans: i)True ii)False**

49. The Applet class is contained in _____package.

[A] java.applet [B] java.awt

[C] java.io[D] java.util

**Ans: java.applet**

50. To display a text on the applet_____method is used.

**Ans:drawstring()**

51. _____method is called to display the result of an applet code on the screen.

**Ans:paint()**

52. The Graphics class is contained in _____package.

[A] java.applet [B] java.awt

[C]java.io [D]java.util

53. URL stands for _____.

**Ans:Uniform Resource Locator**

54. Applet Progarms can be run by_____ and _____.

**Ans:Appletviewer,java-enabled web browser**

55. StateTrue or False

Applet's getParameter() method can be used to fetch parameter values from HTML file.

**Ans:i)True**

56.Which package contains color class?

[A]java.applet

[B]java.awt

[C]java.graphics

[D]java.lang

**Ans:[B]java.awt**

57.The event listener corresponding to handling keyboard events is the

_____

**Ans:KeyListener**

58._____method can be used to draw a rounded rectangle in a applet.

**Ans:drawRoundRect()**

59. _____method can be used to draw a ellipse in a applet?

**Ans:drawOval()**

60.What is the syntax of paint() method?

**Ans:public void paint(Graphics g)**

61.What is a file?

**Ans:A file is a collection of related records placed in a particular area on the disk**.

62.Storing and managing data in a file is known as _____

**Ans: File Processing**

63.The process of reding and writing objects in a file is called as _____

**Ans: Object Serialization**

64.What are the basic types of java Streams?

**Ans:InputStream,OutputStream**

65.The_____ package contains a large number of stream classes for provide capabilities for processing all types of data.

a) java.io b)java.awt c)java.appletd)java.lang

**Ans:a)java.io**

66.What are the two kinds of character stream classes.

**Ans:Reader Stream Classes and Writer Stream Classes**

67.The Method flush() belongs to _____ classes.

a)Input Stream b)Output Stream

c)DataInputStream d)DataOutputStream

**Ans: b)Output Stream**

68.The Methods skip(n),available() belongs to _____ stream class.

a)input b)Output c)DataInput d)DataOutput

**Ans:a)Input**

69.The class DataInputStream extends FilterInputStream and implements the interface DataInput(T/F)

**Ans:TRUE**

70.The_____ class is an abstract class which acts as a base class for all the other writer stream classes.

a)Writer b)Buffered Writer c)Filter Writer d)Print Writer

**Ans: a) Writer**

71.FileOutputStream class is used for reading bytes from a file. (T/F)

**Ans: FALSE**

72.What are the Two ways of initializing the file stream objects?

**Ans:Direct and Indirect**

73. The function " Singnals that an I/O exception of some sort has occurred " related to which of the following exception.

[A] EOF Exception [B] FileNotFoundException

[C] Interrupted Exception [D] IO Exception

**Ans: [D] IO Exception**

74. What are the two commonly used classes for handling bytes?

**Ans: FileInputStream and FileOutputStream.**

75. In a Random Access File an existing file can be updated using _____ mode.

[A] r [B] rw

[C] w [D] None of the above . **Ans: [B] rw**