

Unit 1 Notes

Introduction: Introduction and Web Development Strategies, History of Web and Internet, Protocols Governing Web, Writing Web Projects, Connecting to Internet, Introduction to Internet services and tools, Introduction to client-server computing.

Core Java: Introduction, Operator, Data type, Variable, Arrays, Methods & Classes, Inheritance, Package and Interface, Exception Handling, String handling

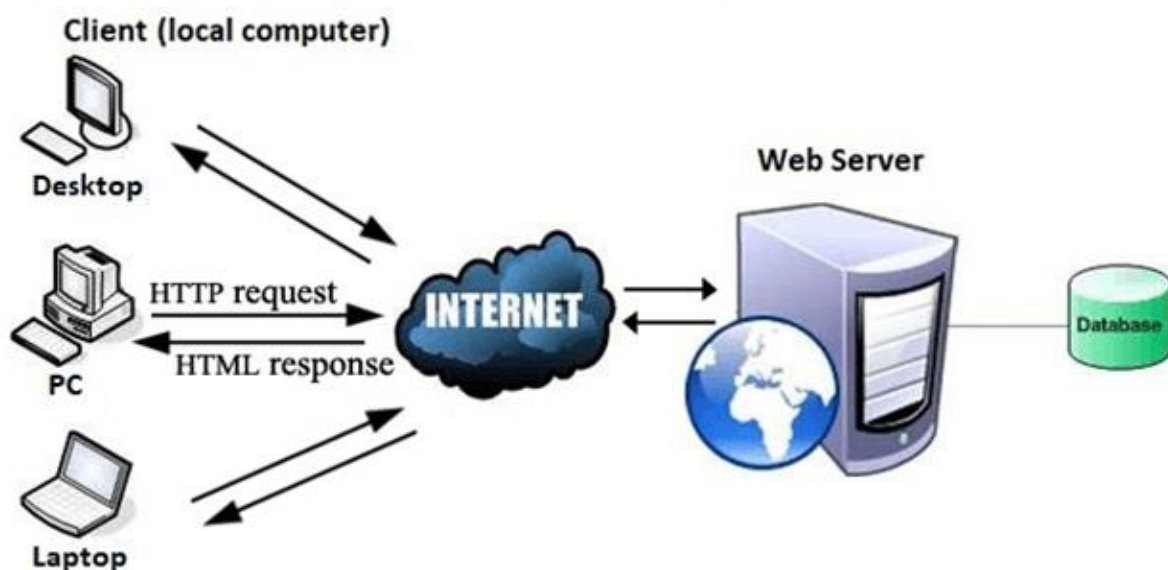
Introduction to Web

- The Web is the common name for the World Wide Web, a subset of the Internet consisting of the pages that can be accessed by a Web browser.
- Many people assume that the Web is the same as the Internet, and use these terms interchangeably.
- However, the term Internet actually refers to the global network of servers that makes the information sharing that happens over the Web possible.
- So, although the Web does make up a large portion of the Internet, but they are not one and same.

What is World Wide Web (WWW)?

- It is a collection of Millions of files stored on thousands of computers (Web Servers) all over the world.
- These files may be HTML Documents, Text Documents, pictures, videos, sounds, programs etc.
- Created by Tim Berners Lee in 1991.

How Web Works?



The client sends a request to the server using Internet. The server sends the response back to the client. The server can also interact with some database management system to manage the data.

History of the Internet

- Internet was developed by ARPANet in 1969.
- ARPANet stands for American Research Project Agency Network






Governing body of the Internet

- There is no central governing body of the Internet. Instead, each constituent network setting follows its own lead and enforces its own policies.
- But even though no sole entity runs the Internet, there are still a few smaller institutions that have a bit of control.

Organizations Governing the Internet

- Internet Governance Forum (IGF)
- Internet Architecture Board (IAB)
- Internet Corporation for Assigned Names and Numbers (ICANN)
- Internet Engineering Task Force (IETF)
- Internet Research Task Force (IRTF)
- World Wide Web Consortium (W3C)

The Intersection of Media Development Principles and Internet Governance

INTERNET GOVERNANCE BODY	PRINCIPLE AT STAKE	TECHNICAL DEBATE
	Freedom of Expression	Domain Names (gTLDs) Management of new, generic Top-Level Domains (gTLDs)
	Media Pluralism	Social Media as News Platforms Algorithms and Media Plurality
	Access to Information	Wireless Internet 5G Cellular Networks and Unlicensed Spectrum Standards
	Privacy	Web Browsing Privacy Encryption
	Secure Access and Trust	Wi-Fi Security Local Area Networks (LAN) Protocols in Diverse Settings

Protocols Governing Web

- HTTP (Hyper Text Transfer Protocol)
- HTTPS (Hyper Text Transfer Protocol Secure)
- SMTP (Simple Mail Transfer Protocol)
- POP3 (Post Office Protocol version 3)
- MIME (Multipurpose Internet Mail Extensions)
- IMAP (Internet Messaging Access Protocol)
- FTP (File Transfer Protocol)
- TELNET (Terminal Networking)

HTTP (Hyper Text Transfer Protocol)

- HTTP is a pull protocol, the user pulls information from a remote site.
- Protocol consists of GET and POST commands to transfer data.
- HTTP uses cached files to speed up transfers
- HTTP Uses LAN accessible cache that is Proxy Server.
- Proxy allows for reduced load on the internet connection

HTTPS (Hyper Text Transfer Protocol Secure)

Hypertext Transfer Protocol Secure (HTTPS) is an extension of the Hypertext Transfer Protocol (HTTP). It is used for secure communication over a computer network, and is widely used on the Internet. In HTTPS, the communication protocol is encrypted using Transport Layer Security (TLS) or, formerly, Secure Sockets Layer (SSL). The protocol is therefore also referred to as HTTP over TLS or HTTP over SSL.

SMTP (Simple Mail Transfer Protocol)

- The protocol is very simple.
- SMTP is a push protocol; information is pushed to a remote site.
- It uses port 25.
- All files are ASCII text.

POP3 (Post Office Protocol version 3)

- It is mail access client.
- It uses port 110.
- Messages are downloaded to client but can be stored on server.
- Does not easily allow multiple clients.

IMAP (Internet Message Access Protocol)

- It is improved version of POP3
- Automatically assigns folders
- Leaves mail on server
- Only transfers as much as needed per message (headers, subject only on list)
- It uses port number 143

MIME (Multipurpose Internet Mail Extensions)

- Generally it is used for encoding.
- Handles Non-ASCII data in an ASCII transfer medium.
- Defines extensions to support binary attachments of arbitrary format
 - Images, Audio, Video and multi-media messages
 - Text having unlimited line length or overall length
 - Multiple objects in a single message
 - Character sets other than ASCII
 - Multi-font messages

FTP (File Transfer Protocol)

- Used to Transfer files between two computers.
- It uses port 21
- Goals of FTP Service
 - Promote sharing of files (programs and/or data)

- Encourage indirect/implicit use of remote computers
- Shield users from variations in file storage among hosts
- Transfer data reliably and efficiently

TELNET (Terminal Networking)

- It is also known as Network Virtual Terminal.
- TELNET is a protocol that provides bi-directional communications using Command Line interface

Web Development Life Cycle

- Information Gathering
- Planning
- Design
- Development
- Testing
- Delivery or Hosting
- Maintenance Phase

What is web browser?

- Web browser is software application.
- It has ability to retrieving, presenting and traversing information resources on the World Wide Web.
- First web browser is Nexus.
- National Center for Supercomputing Applications (NCSA) develop Netscape Navigator in 1994.
- In 1995 IE (Internet Explorer) is developed by Microsoft.

What is Web Applications?

- Web Application is any software which runs on web browser.
- It is created in a browser-supported programming language.
- It is based on Client-Server model

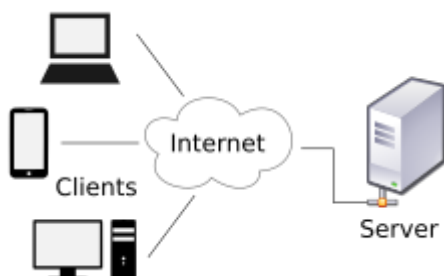
What is Web Team?

- Group of peoples which works together to develop web applications.
- Roles & Responsibilities of the team Members defined clearly.
- There should be no communication barrier between team members.
- Size of team may be varying organization to organization.

<u>Core Team member</u>	<u>Extended Team Member</u>	<u>Special Team Member</u>
1. Project Manager: <ul style="list-style-type: none"> Specify the work. Developing the project plan. Scheduling. Allocation resources. Budgeting and managing the team. 2. Technical lead: <ul style="list-style-type: none"> Managing programmers. Chooses specialized team such as security expert, database programmers. 3. Web Production specialist: <ul style="list-style-type: none"> Integrate the site using html or java script. 4. Creative Lead: determines creative concepts for the site and responsible for site design. 5. Quality Assurance Lead: for testing purpose.	1. Account Manager: It interacts with the client, project manager and creative lead. 2. Programmer: develops applications for the web projects. 3. Network Engineer: configuring a web server. 4. Information architects: It understands how to display information visually to users and how to interact with the website. 5. Content Writer: write contents for the website. 6. Tester: It tests the web project based on the team plan that QA lead writes.	1. Security Experts: security handling and encryption techniques. 2. Audio, Video Engineer 3. 3-D Modeler 4. Web Cast Specialist 5. Media Buyer 6. Strategic Planner

Introduction to client-server computing

Client-server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs, which share their resources with clients. A client usually does not share any of its resources, but it requests content or service from a server. Clients, therefore, initiate communication sessions with servers, which await incoming requests. Examples of computer applications that use the client-server model are email, network printing, and the World Wide Web.



The "client-server" characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services. Servers are classified by the services they provide. For example, a web server serves

web pages and a file server serves computer files. A shared resource may be any of the server computer's software and electronic components, from programs and data to processors and storage devices. The sharing of resources of a server constitutes a service.

Introduction to Java

- Java is a high level class based object oriented programming language
- It is a general purpose programming language intended to let application developers Write Once Run Anywhere (WORA)
- Java was originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform Sun Microsystems was acquired by Oracle in 2010

History of Java

- James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991
- The language was initially called Oak after an oak tree that stood outside Gosling's office
- Later the project went by the name Green and was finally renamed Java from Java coffee, a type of coffee from Indonesia
- Sun Microsystems released the first public implementation as Java 1.0 in 1996
- The latest version is Java 17 released in September 2021

Different technologies in Java

- Java SE (Standard Edition)
- JAVA EE (Enterprise Edition)
- Java ME (Micro Edition)

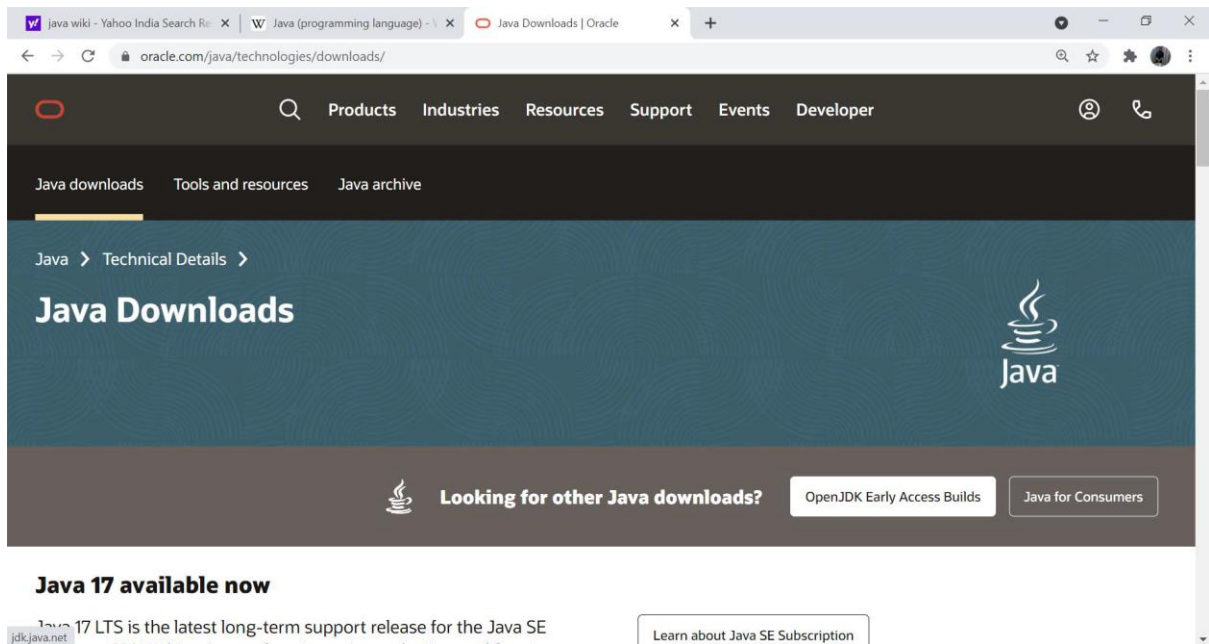
Tools Required for Java Development

- Java Development Kit (JDK)
- Java Runtime Environment (JRE)
- Java IDEs (Eclipse, NetBeans, IntelliJ etc.)

Resource URL:

<https://java.sun.com/>

Java SE 17 Download

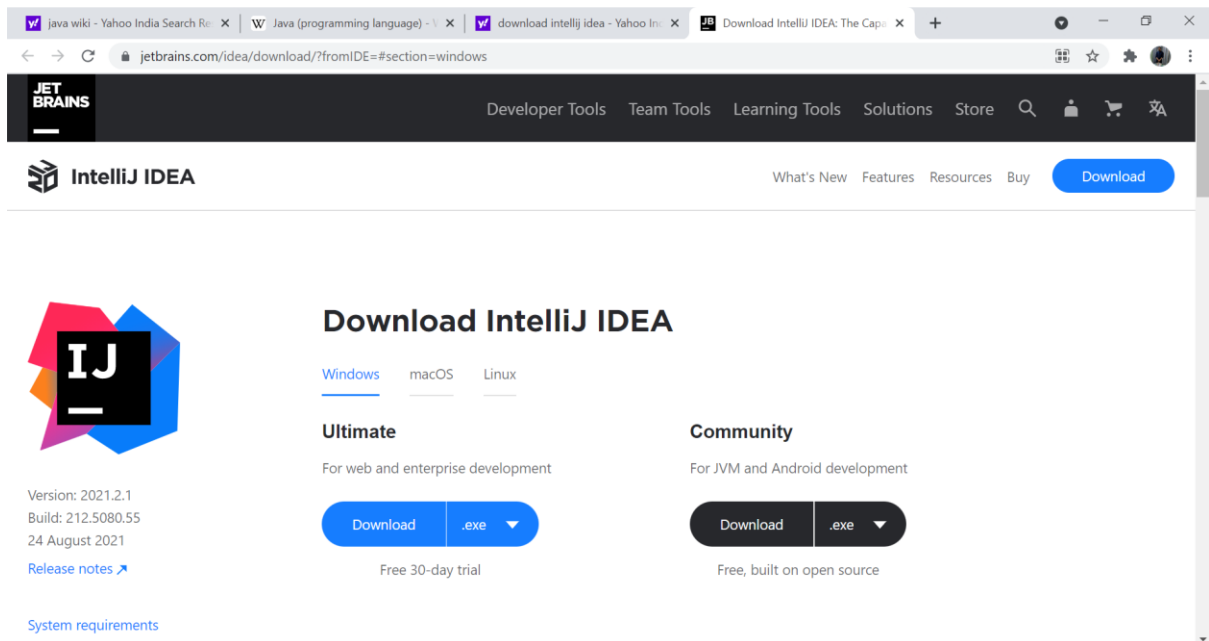


Java 17 available now

Java 17 LTS is the latest long-term support release for the Java SE

Learn about Java SE Subscription

IntelliJ IDEA Community Edition Download



Download IntelliJ IDEA

Windows macOS Linux

Ultimate
For web and enterprise development

Download .exe

Free 30-day trial

Community
For JVM and Android development

Download .exe

Free, built on open source

Writing First Java Program

```
public class Welcome{  
    public static void main(String[] args){  
        System.out.println("Welcome to Java");  
    }  
}
```

Save the program with .java extension e.g. Welcome.java

What is class?

- A user defined data type used to classify different entities like Student, Employee, Book, Department
- A class contains a set of data members and the methods applicable on an entity
- Classes can be of two types
 - Library Classes
 - User Defined Classes

What are library classes?

- The classes which are provided with Java Development Kit for Rapid Application Development (RAD)
 - String class
 - Character class
 - Math class
 - System class
- String class provides the methods applicable on a string like length(), toUpperCase (), toLowerCase ()
- Math class provides the methods required for mathematical operations like pow(), sqrt (), log()
- System class provides reference variables and the methods to interact with the computer system

Understanding System class

- The System class provides three built in reference variables to refer the input and output devices in a computer system
 - in refers to standard input (keyboard)
 - out refers to standard output (monitor)
 - err refers to standard error (monitor)
- These reference variables can be used to call the methods provided in their classes InputStream and OutputStream

Methods applied on out and err reference variables

- print()
 - To print some output to the monitor
- println()
 - To print some output to the monitor with line break
- printf()
 - To print formatted output to the monitor using format specifiers

What is entry point?

- Special method inside a class from where the runtime environment starts execution is called as entry point
- Entry point can be any of three choices
 - `public static void main(String[] args)`
 - `public static void main(String args[])`
 - `public static void main(String... args)`

Syntactical Rules of Java

- All keyword in lower case
- Class names start with capital letter. Any new word get introduced, make its first letter as capital. Such syntax is also known as Pascal Case
 - String class
 - BufferedReader class
 - InputStreamReader class
- Method names start with small letters. Any new word get added make its first letter as capital.
 - `length()`
 - `toUpperCase()`

Compiling a Java Program

- Once the program is ready we can compile the program code using a Java IDE or on the Console
- If compiling on the console we need the Java Compiler `JAVAC.EXE` provided with JDK

Syntax

`JAVAC < programname`

Example

`JAVAC Welcome.java`

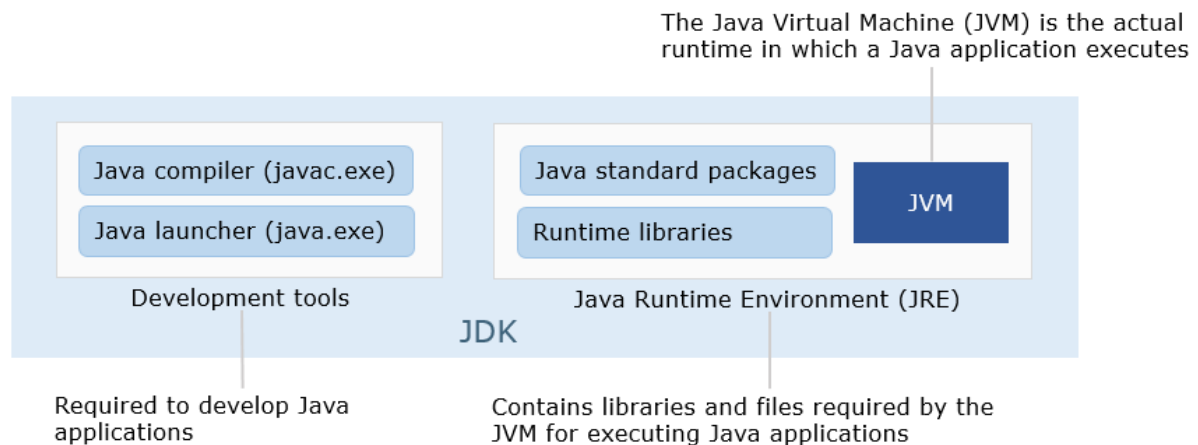
What happens during compilation and execution process?

- During the compilation process, Java code gets converted to another programming language called as Byte Code language and stored as `.class` file

`JAVAC Welcome.java → Welcome.class`

- Here `Welcome.class` file is not in binary format but in Byte Code Format. We can decompile the `Welcome.class` file to convert it back to `.java` file

The development, compilation and execution of Java programs is taken care by JDK which contains 2 main components: Development tools and JRE.



Let us first discuss about development tools. The development tools consist of Java compiler and Java launcher.

- Java compiler (javac.exe) - It is the primary Java compiler. The compiler accepts Java source code and produces Java bytecode conforming to the Java Virtual Machine Specification (JVMS).
- Java launcher (java.exe) - It helps in launching a Java application during execution.

Next, let us discuss about Java Runtime Environment (JRE).

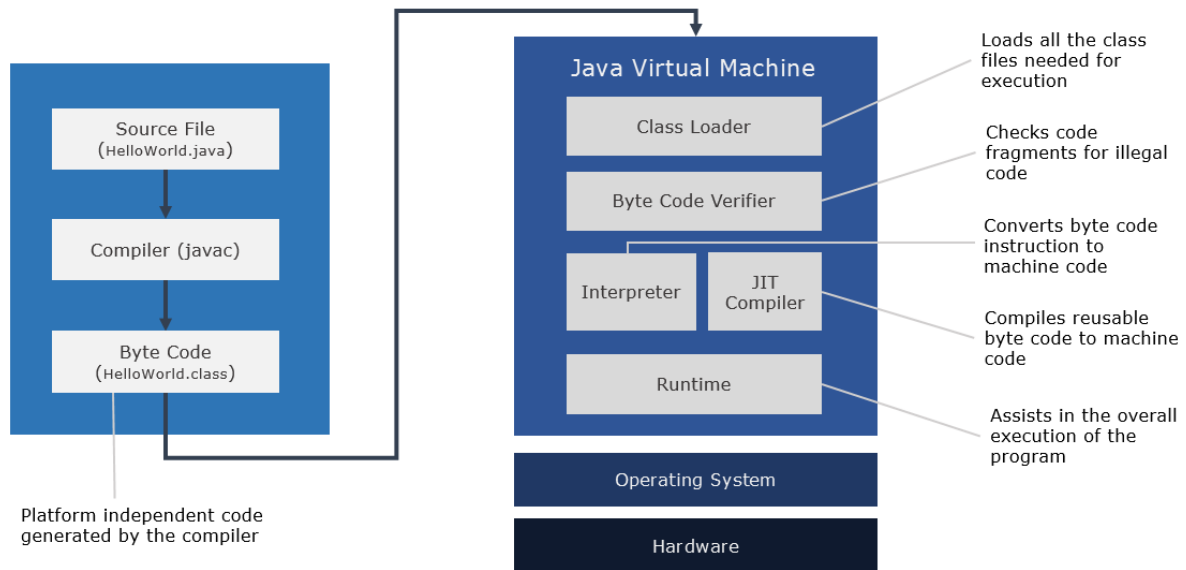
The Java Runtime Environment (JRE) contains Java Virtual Machine(JVM) and the Java standard library (Java Class Library).

- Java Virtual Machine (JVM) - It is the virtual machine that enables the computer to run Java programs.
- Java standard library (Java Class Library) - It is a set of dynamically loadable libraries that Java applications can call at run time. Java Platform is not dependent on a specific operating system and hence applications cannot rely on any of the platform-native libraries. So, the Java Platform provides a set of standard class libraries containing functions common to modern operating systems.

By now, you would have understood the components of JDK. Next, you will see how does a Java program get executed by the computer.

The below diagram shows various stages that a Java program goes through during execution.

The Java source code is saved in a file with .java extension. When we compile a Java program (.java file), .class files (byte code) with the same class names present in .java file are generated by the Java compiler (javac). These .class files go through various steps when we run the program as shown in the below diagram.

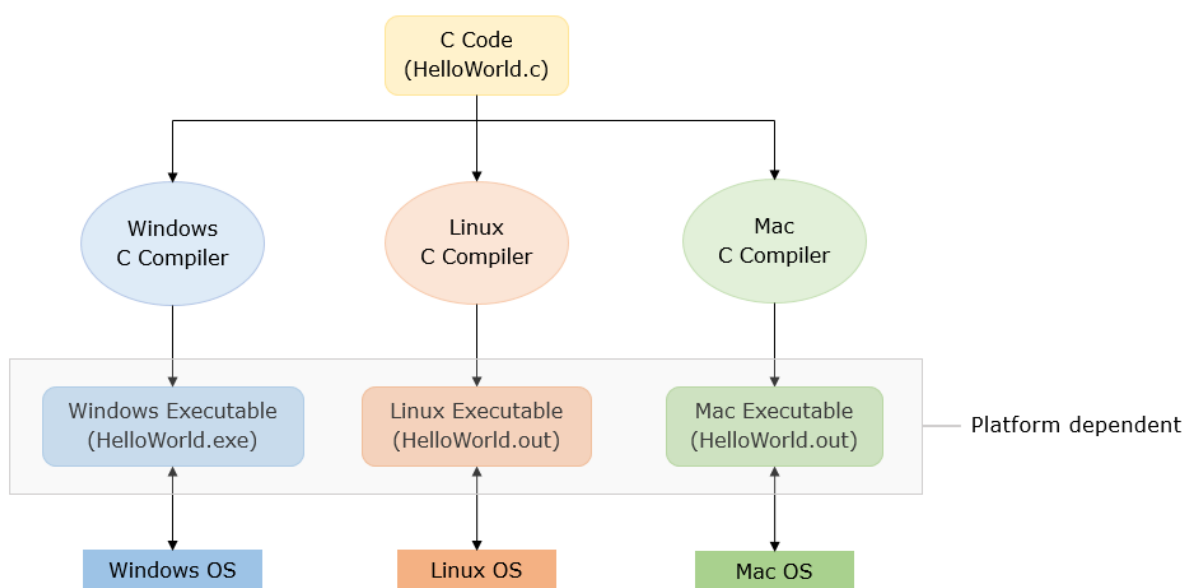


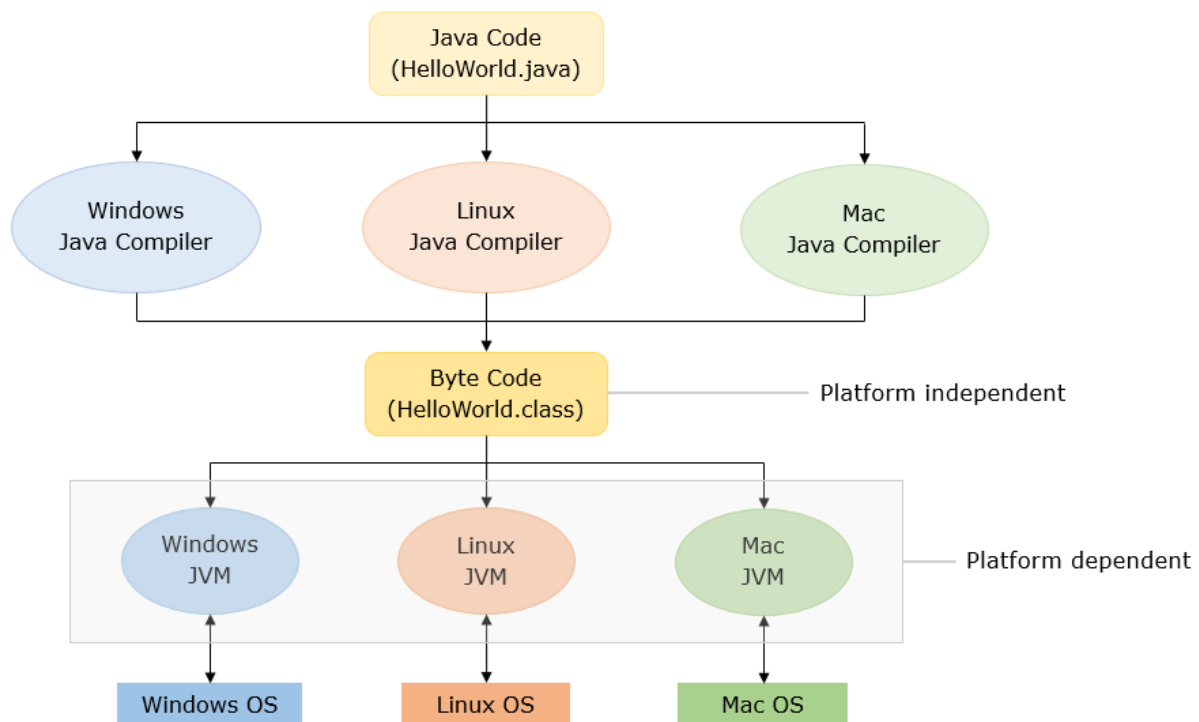
What makes Java Platform independent?

If a program written on a particular platform can run on other platforms without any recompilation, it is known as a **platform independent** program.

Usually larger applications are created by a team of developers. While most of them could be working on the **same operating system** such as Windows, others might be using different operating systems like Mac or Linux. In this scenario, we might have a situation where a program written on Windows needs to be executed on Mac OS also.

Since Java is **platform independent**, it is not a problem. A program written using Java on Windows will execute **without any recompilation** on any other platform.





Java Virtual Machine is platform dependent software, that makes Java Platform independent.

How to run a class?

- If using a Java IDE, then simply use a context menu to compile and run the code
- If using Console then use Java Runtime Environment (JRE) JAVA.EXE provided with JDK

JAVA Welcome

Checking for version of JDK installed in your machine

- While compiling and running the Java programs we need two softwares
 - JAVAC.EXE Java Compiler
 - JAVA.EXE Java Runtime
- Both of these software must be of same version
- Check the versions using following statements
 JAVA --version
 JAVAC --version

File Naming Rules in Java

- A Java program file can have one or more classes
- The class can have any name and file can also have any name of choice
- But, if a class is declared as public using public keyword with it, then the program name can class name must be same
- A Java Program can have many classes but only one public class

Primitive Data Types in Java

- Integer
 - byte 1 byte
 - short 2 bytes
 - int 4 bytes
 - long 8 bytes
- Floating
 - float 4 bytes
 - double 8 bytes
- Character
 - char 2 bytes
- Boolean
 - boolean undefined

Literals or Constant values

- The values that assign to some variable or used in some expression are called as literals or constant values
- Can be of five types
 - Integer Literals
 - Float Literals
 - Character Literals
 - Boolean Literals
 - String Literals

Integer Literal

- All numbers without decimal point are called integer literals
- Such values are of int type by default
- Use l or L suffix for long type values

Example

```
long num =12345L;
```

Types of Integer Literals

- Integer Literals can be of four types
 - Decimal
 - Octal
 - Hexa Decimal
 - Binary
- Decimal numbers can have 10 digits (0 to 9). It is default
- Octal numbers can have 8 digits (0 to 7). Starts with 0
- Hexa Decimal numbers can have 16 digits(0 to 9, A F). Starts with 0x or 0X
- Binary numbers can have two digits (0 or 1). Starts with 0b or 0B

Examples of Integer Literals

- `int a=1234; //decimal`
- `int b=01234; //octal`
- `int c=0X1234; hexa decimal`
- `int d=0B01101; 0B01101; // binary`

Note: If we print these values using `print()`, `println ()` or `printf ()` methods then its output will be printed as decimal number system

```
public class IntegerLiterals {  
    public static void main(String[] args) {  
        int a=1234;        //decimal  
        int b=01234;        //octal  
        int c=0X1234;        //hexa decimal  
        int d=0B01101;    // binary  
        System.out.println(a+", "+b+", "+c+", "+d);  
    }  
}
```

Floating Literals

- Such numbers are of double by default
- Use `f` or `F` as suffix with float type values

Examples

`double x=123.55;`

`float y=123.55; //error`

`float z=123.55F; //correct`

Character Literals

- Characters are enclosed in single quotes
`char ch ='A';`
- Each character has corresponding ASCII value
`char ch =65;`
- Characters from different languages have their Unicode values
`char z='\u0910';`

```

public class CharacterLiterals {
    public static void main(String[] args) {
        char x='A';
        char y=65;
        char z='\u0910';

        System.out.println(x+","+y+","+z);
    }
}

```

String Literals

- Enclosed in double quotes
- Managed by String class

Example

```
String str ="Hello";
```

Boolean Literals

- Can have true or false only

Examples

```
boolean married=false;
boolean rented=true;
```

Introduction to Packages

- Special folders which contains related set of classes are called as packages

Examples

- java.lang package
 - Contains all commonly used classes String, Math, System, Integer, Character etc.
 - It is default Package
- java.util package
 - Contains the collections and utility classes
 - LinkedList , Stack, Queue, Scanner, Date
- java.io package
 - Contains file and memory management related classes
 - BufferedReader, InputStreamReader, File, FileReader , FileWriter etc.

Importing classes from a package

- Before using a class from a package we need to import that class
- We can import single class or all the classes provided in the package using import command

Examples

```
import java.util.*;  
import java.util.LinkedList;  
import java.util.Scanner;
```

Terminologies in Java

- Class
- Instance
- Reference
- Constructor

What is class?

- A set of specifications or blueprint about an entity describing all possible data members and the methods to be applicable on that entity
- Classes can be of two types
 - User Defined classes
 - Library classes
- Use **class** keyword to create a class user defined class

What is instance?

- A real entity created based on class specifications is called as instance
- To create an instance, we require two things
 - **new** keyword
 - Constructor
- An instance can be used to use the data members and the methods of that class

What is constructor?

- Special method inside a class having special features
 - Same name as class name
 - No return type
 - Used to initialize data in data members

Example

- String class
 - String() constructor
- Scanner class
 - Scanner() constructor

Example of instance of String class

```
public class StringInstance {
    public static void main(String[] args) {
        System.out.println(new String(original: "Hello").toLowerCase());
    }
}
```

What is reference?

- Special variable which hold reference of an instance present somewhere in memory
- It can be used to use the properties and methods of the class it is referring

Example

```
String s=new String("Hello");
```

- Here s is called as reference or reference variable.

```
public class StringInstance {
    public static void main(String[] args) {
        String s=new String(original: "Hello");
        System.out.println(s.length());
        System.out.println(s.toLowerCase());
        System.out.println(s.toUpperCase());
    }
}
```

Getting Data Input from user

Method 1: Using Scanner class

- Allows to read any type of data using pre-defined methods
 - String next() -- string without space
 - String nextLine() -- string with space
 - int nextInt()
 - float nextFloat()
 - double nextDouble()
- To use these methods we need instance of Scanner class

```
Scanner sc=new Scanner(System.in);
```

Example

Write a program (ScannerTest.java) to input data of a student rollno (int), name (string), gender (char) and fees (float). Show that data.

Hint: To input a character you can first input the data as string using next() method and then fetch its first character using charAt() method of String class

```
import java.util.Scanner;
public class ScannerTest {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.print("Roll No : ");
        int rollno=sc.nextInt();
        System.out.print("Name : ");
        String name=sc.next();
        System.out.print("Gender : ");
        char gender=sc.next().charAt(0);
        System.out.print("Fees : ");
        float fees=sc.nextFloat();
        System.out.printf("Roll No is %d, Name is %s, Gender is %c, Fees is %f",rollno,name,gender,fees);
    }
}
```

Note: When you try to input a string with space using nextLine() method, it may not work in between. To handle such situations, you can use another method.

Method 2: Using BufferedReader class

When we press a key from keyboard (System.in), a stream of bits get produced (A → 01000001)

Java provides InputStreamReader class to read this stream of bits and convert into readable format

These characters are passed to an instance of another class BufferedReader

To read the data from buffer using readLine() method of BufferedReader class

String readLine() throws IOException

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;
import java.util.*;
public class DataInputTest {
    public static void main(String[] args) throws IOException {
        BufferedReader br=new BufferedReader(new
        InputStreamReader(System.in));
        Scanner sc=new Scanner(System.in);
        System.out.print("Roll No : ");
        int rollno=sc.nextInt();
        System.out.print("Name : ");
        String name=br.readLine();
        System.out.print("Gender : ");
        char gender=sc.next().charAt(0);
        System.out.print("Fees : ");
        float fees=sc.nextFloat();
        System.out.printf("%s %d %c %f",name,rollno,gender,fees);
    }
}
```

```
}  
}
```

What is an array?

Special variable which allows to store multiple values of same data type in continues locations. Each value or item get placed at some index number. Index number starts with 0.

Arrays in Java are treated as objects and created using **new** keyword.

Arrays provide **length** property to get size of array.

Arrays can be created using any of three syntaxes

Syntax 1: Arrays with sample values

```
datatype []arrayname={value1, value2, value3, ...};
```

Example

```
int []num={6,1,2,9,12};
```

Test Case

Write a program having an array with sample values. Show sum of the values in the array.

```
public class SumArray {  
    public static void main(String[] args) {  
        int []num={5,1,8,9,12};  
        int sum=0;  
        for(int i=0;i<num.length;i++)  
            sum=sum+num[i];  
        System.out.println("Sum is "+sum);  
    }  
}
```

Using foreach loop

Java provides a variant of for loop called as foreach loop where you can work on an array without knowing length of it and without using array indexing.

Syntax

```
for(datatype variable: arrayname){  
    statements;  
}
```

Test Case

Write a program (SumArray.java) having an array with sample values. Show sum of the values in the array using foreach loop

```
public class SumArray {  
    public static void main(String[] args) {  
        int []num={5,1,8,9,12};  
        int sum=0;
```

```

        for(int n:num)
            sum=sum+n;
        System.out.println("Sum is "+sum);
    }
}

```

Syntax 2: Creating an array of defined size

`datatype []arrayname=new datatype[size];`

Example

`int []num=new int[10]; //array of 10 integers`

Test Case

Write a program (BigSmall.java) to create an array to store 10 integer values. Input 10 integer values and show the smallest and biggest of given values.

```

import java.util.Scanner;
public class BigSmall {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        int []ar=new int[10];
        for(int i=0;i<ar.length;i++)
            ar[i]=sc.nextInt();
        int min,max;
        min=max=ar[0];
        for(int i=1;i<ar.length;i++){
            if(ar[i]>max) max=ar[i];
            if(ar[i]<min) min=ar[i];
        }
        System.out.printf("Smallest is %d and Biggest is %d",min,max);
    }
}

```

Syntax 3: Arrays with user defined size

`datatype []arrayname=new datatype[variable];`

Test Case

Write a program (BigSmall.java) to ask the user how many numbers to be input. Input that much of integer type of values. Show the smallest and biggest of given values.

```

import java.util.Scanner;
public class BigSmall {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.print("How many numbers to input : ");
        int num=sc.nextInt();
        int []ar=new int[num];
        System.out.printf("Enter %d integer values : ",num);
        for(int i=0;i<ar.length;i++)
            ar[i]=sc.nextInt();
        int min,max;
    }
}

```

```

        min=max=ar[0];
        for(int i=1;i<ar.length;i++){
            if(ar[i]>max) max=ar[i];
            if(ar[i]<min) min=ar[i];
        }
        System.out.printf("Smallest is %d and Biggest is %d",min,max);
    }
}

```

Types of Array

Java provides two types of arrays

- One Dimensional Array
- Array of Array or Jagged Array

When an array can have only one row, it is called one dimensional array. All above examples are of one dimensional array.

When an array contains an array which refers to another array, it is called as array of array or Jagged array. Such arrays allow to manage different number of columns in each row.

Syntax 1: Array of Array having different columns in each row

```
datatype [][]arrayname=new datatype[rows][];
```

```
arrayname[index]=new datatype[size];
```

Example

Declare an array having 4 columns in first row, 5 columns in second row and 6 columns in third row.

```

int [][]ar=new int[3][];
ar[0]=new int[4];
ar[1]=new int[5];
ar[2]=new int[6];

```

Syntax 2: Array of Array having equal number of columns in each row

```
datatype [][]arrayname=new datatype[rows][columns in each row];
```

Example

Declare an array of 3x4.

```
int [][]ar=new int[3][4];
```

Test Case

Write a program (ArrayOfArray.java) to create an array having 4 columns in first row, 5 columns in second row and 6 columns in third row. Input that much of integer values and show of values in each row.

```

import java.util.Scanner;
public class ArrayOfArray {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
    }
}

```

```

int [][]ar=new int[3][];
ar[0]=new int[4];
ar[1]=new int[5];
ar[2]=new int[6];

System.out.println("Enter 15 integers : ");
for(int i=0;i<ar.length;i++){
    for(int j=0;j<ar[i].length;j++){
        ar[i][j]=sc.nextInt();
    }
}
System.out.println("Array with row wise sum is ");
for(int i=0;i<ar.length;i++){
    int sum=0;
    for(int j=0;j<ar[i].length;j++){
        System.out.printf("%d\t",ar[i][j]);
        sum = sum + ar[i][j];
    }
    System.out.printf("= %d\n",sum);
}
}
}

```

Test Case

Write a program (Matrix.java) having two arrays of 2x3 and 3x4. Input the integer type and show the arrays in matrix format along with matrix multiplication.

```

import java.util.Scanner;
public class Matrix {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        int [][]ar1=new int[2][3];
        int [][]ar2=new int[3][4];
        int [][]ar3=new int[2][4];
        int i,j,k;
        System.out.println("Enter 6 items in first array : ");
        for(i=0;i<ar1.length;i++){
            for(j=0;j<ar1[i].length;j++){
                ar1[i][j]=sc.nextInt();
            }
        }
        System.out.println("Enter 12 items in second array : ");
        for(j=0;j<ar2.length;j++){
            for(k=0;k<ar2[j].length;k++){
                ar2[j][k]=sc.nextInt();
            }
        }

        for(i=0;i<ar1.length;i++){
            for(k=0;k<ar3.length;k++){
                for(j=0;j<ar2.length;j++){
                    ar3[i][k]+=ar1[i][j]*ar2[j][k];
                }
            }
        }
        System.out.println("First Matrix is");
        for(i=0;i<ar1.length;i++){
            for(j=0;j<ar1[i].length;j++){
                System.out.printf("%d\t",ar1[i][j]);
            }
        }
    }
}

```

```

    }
    System.out.println();
}
System.out.println("Second Matrix is");
for(j=0;j<ar2.length;j++){
    for(k=0;k<ar2[i].length;k++){
        System.out.printf("%d\t",ar2[j][k]);
    }
    System.out.println();
}
System.out.println("Matrix multiplication : ");
for(i=0;i<ar1.length;i++){
    for(k=0;k<ar1[i].length;k++){
        System.out.printf("%d\t",ar2[i][k]);
    }
    System.out.println();
}
}
}

```

Exception Handling in Java

What is Exception Handling?

- A system to send an error message in runtime from the place a runtime error has occurred to the place a method get called using a set of special classes called as exceptions.
- Java provides thousands of such classes suffixed with Exception
 - IOException
 - SQLException
 - ClassNotFoundException
 - RemoteException
 - InputMismatchException

Sample Code Generating an exception

- Write a program to input a number and show square of it

```

import java.util.Scanner;
public class ExceptionDemo {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter a number : ");
        int num=sc.nextInt();
        System.out.println("Square is "+num);
    }
}

```

If we run this program and input some wrong number then see what happens


```
1 import java.util.Scanner;
2 public class ExceptionDemo {
3     public static void main(String[] args) {
4         Scanner sc=new Scanner(System.in);
5         System.out.print("Enter a number : ");
6         int num=sc.nextInt();
7         System.out.println("Square is "+num);
8     }
9 }
10
```

Run: ExceptionDemo

"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.1\lib\idea_rt..."

Enter a number : 100

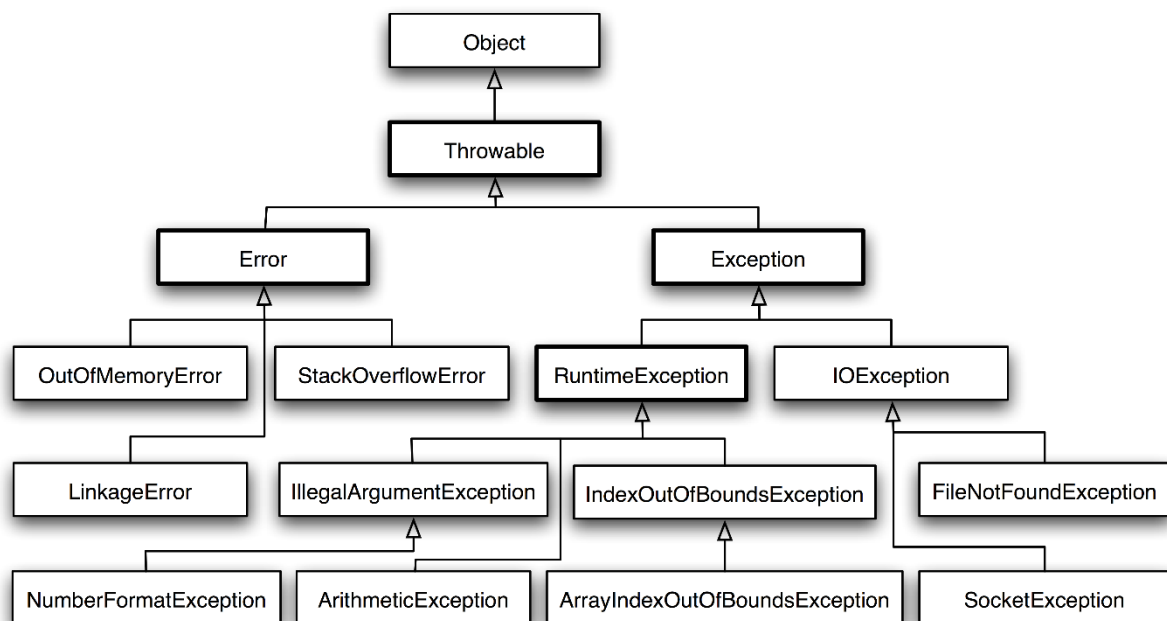
Exception in thread "main" java.util.InputMismatchException Create breakpoint

at java.base/java.util.Scanner.throwFor(Scanner.java:939)
at java.base/java.util.Scanner.next(Scanner.java:1594)
at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
at ExceptionDemo.main(ExceptionDemo.java:6)

Process finished with exit code 1

Hierarchy of Exception classes

- The java.lang.Throwable class act as the root for Java Exception Hierarchy
- A Throwable class defines two child classes
 - Exception
 - Error
- Exceptions are unexpected event that disturbs normal flow of the program. We can handle such events.
- An error is non-recoverable. We cannot handle them.



Types of Java Exceptions

- There are mainly two types of exceptions:
 - Checked
 - Unchecked
- Error is generally considered third type of exception

Difference between Checked and Unchecked Exceptions

- **Checked exceptions** are checked at compile-time. The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException, etc.
- **Unchecked exceptions** are not checked at compile-time, but they are checked at runtime. The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.
- **Error** is irrecoverable but they are also unchecked kind of. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

How to handle such runtime errors?

- Java provides five keywords to handle the runtime errors
 - try
 - catch
 - throw
 - throws
 - finally
- Use try-catch block to try some statements and trap the runtime error
- If we don't know the kind of error may occur in the program code, use the parent classes Throwable or Exception to handle all such exceptions
- Here we can give some generalized message when a runtime error occurs

The screenshot shows an IDE window titled "ExceptionDemo.java" with the following code:

```

1 import java.util.Scanner;
2 public class ExceptionDemo {
3     public static void main(String[] args) {
4         try {
5             Scanner sc = new Scanner(System.in);
6             System.out.print("Enter a number : ");
7             int num = sc.nextInt();
8             System.out.println("Square is " + num);
9         } catch (Throwable ex) {
10             System.err.println("00Ps! A runtime error has been occurred");
11         }
12     }
13 }

```

The Run window at the bottom shows the command used to run the program and the output:

```

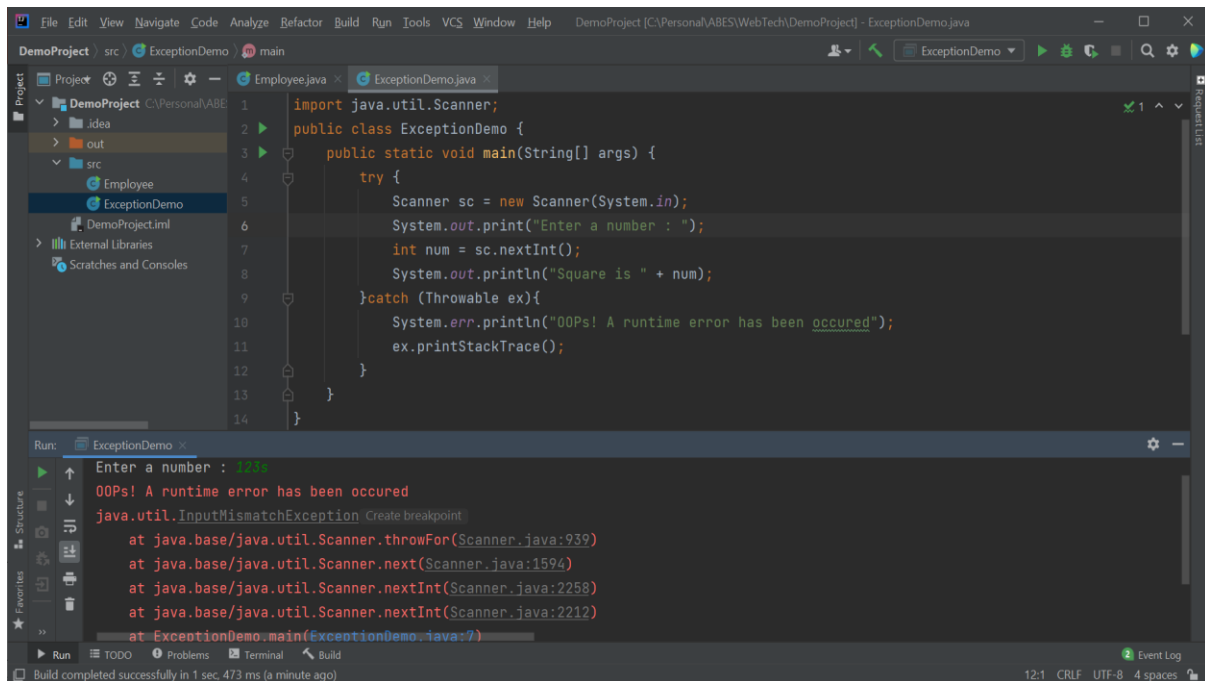
Run: "C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.1\lib\idea_rt
Enter a number : 1234
00Ps! A runtime error has been occurred
Process finished with exit code 0

```

The status bar at the bottom indicates "Build completed successfully in 2 sec, 207 ms (moments ago)".

Tracking for Exception Name for Specialized Messaging

- We can track the exception class name that notified us about a runtime error
- **Throwable** class provides some common methods for all the exceptions
 - void printStackTrace()
 - Prints the complete error information
 - String getMessage()
 - Returns the error message only



The screenshot shows an IDE window with a Java file named `ExceptionDemo.java`. The code is as follows:

```
1 import java.util.Scanner;
2 public class ExceptionDemo {
3     public static void main(String[] args) {
4         try {
5             Scanner sc = new Scanner(System.in);
6             System.out.print("Enter a number : ");
7             int num = sc.nextInt();
8             System.out.println("Square is " + num);
9         } catch (Throwable ex) {
10             System.err.println("Oops! A runtime error has been occurred");
11             ex.printStackTrace();
12         }
13     }
14 }
```

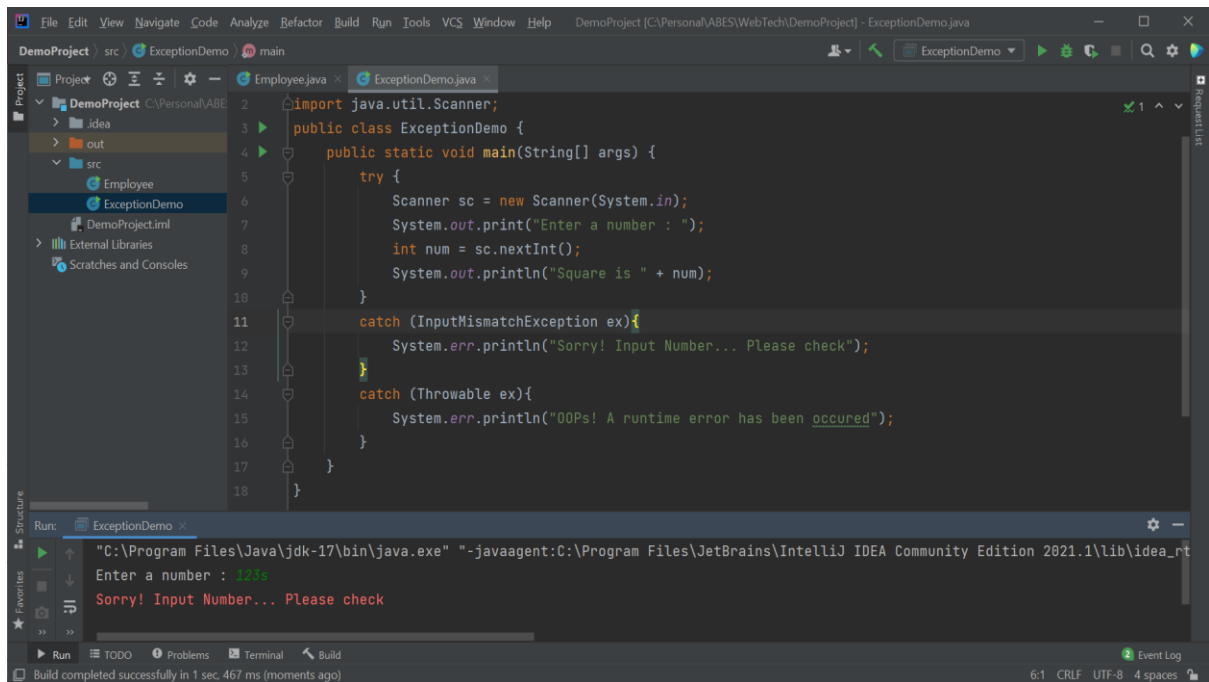
The Run console at the bottom shows the execution output:

```
Enter a number : 1234
Oops! A runtime error has been occurred
java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
    at ExceptionDemo.main(ExceptionDemo.java:7)
```

The status bar at the bottom indicates the build was completed successfully in 1 sec. 473 ms (a minute ago).

Using multiple catch blocks

- The program code may throw multiple kind of exceptions at different moments
- We can use multiple catch blocks to handle different types of exceptions
- Use Throwable class at the bottom to handle those exceptions that are not discovered yet



What is finally block?

- Special block used with try to always execute some code irrespective of an exception
- One try can have many **catch** blocks but only one **finally** block at the bottom

Syntax

```

try{
    statements;
}catch(Exception ex){
    statements;
}
finally{
    statements
}

```

What is throw keyword?

- A keyword used to throw an instance of some exception kind of class when some runtime error occurs in some method or constructor.

Syntax

```

throw <exception kind of instance>;

```

What is throws keyword?

- A keyword used to send an instance of some exception kind of class from one method to another method
- Such feature is also known as **exception propagation**

Syntax

```

<return type> methodname() throws exceptionclass{
}

```

Test Case

- Create a class **MyMath** having a static method **factorial()** which takes a number as argument and returns factorial of given number.
- If the value passed to factorial() method is in negative then throw an instance of **Exception** class with a message “**Sorry! Negative values not allowed**”
- Create a class **Demo** with entry point. Input the data from user and show factorial of that number.
- Handle the exceptions

```

1  public class MyClass {
2      public static int factorial(int n) throws Exception{
3          if(n<0) throw new Exception("Sorry! Negative number is not allowed");
4
5          if(n==0)
6              return 1;
7          else
8              return n*factorial(n-1);
9      }
10 }

```

```

1  import java.util.Scanner;
2  public class Demo {
3      public static void main(String[] args) {
4          Scanner sc=new Scanner(System.in);
5          try {
6              System.out.print("Enter a number : ");
7              int n = sc.nextInt();
8              int f = MyClass.factorial(n);
9              System.out.println("Factorial is : "+f);
10         }catch (Exception ex){
11             System.err.println("Error : "+ex.getMessage());
12         }
13     }
14 }

```

Difference between throw and throws?

- The **throw** keyword is used to throw an instance of some exception kind of class while **throws** carries that instance to next method till try-catch block is reached

String Handling in Java

What is String?

- An alphanumeric data is known a string. It can have alphabets, digits and special characters
- It can have zero or more characters

- Managed by **java.lang.String** class

Different ways of defining strings

- Strings can be defined using two ways
 - As Literal
 - As instance

Examples of Literals

String x=""; //blank string

String y="A"; // single character string

String z="Hello 123 \$"; // Multiple character string

Example of instance

String name=new String("Rohit Kumar");

Methods of String class

- String class provides a variety of methods to work on strings
 - int **length()**
 - boolean **equals**(String s)
 - boolean **equalsIgnoreCase**(String s)
 - String **toUpperCase()**
 - String **toLowerCase()**
 - char **charAt**(int index)
 - int **indexOf**(String s) – Searches from start. Returns -1 if not found
 - int **lastIndexOf**(String s) – Searches from end. Returns -1 if not found
 - int **compareTo**(String s) – returns 0 if equal, >0 for greater or <0
 - String **substring**(int start) – returns string from start to end
 - String **substring**(int start, int end) – returns string having characters end-start from start position

```
import java.util.Locale;
public class StringMethods {
    public static void main(String[] args) {
        String s="Hello India 123 $#@!";
        System.out.println("Length : "+s.length());
        System.out.println("Uppercase : "+s.toUpperCase());
        System.out.println("Lowercase : "+s.toLowerCase());
        System.out.println("1st character is : "+s.charAt(0));
        System.out.println("Substring from 3rd character till end : "+s.substring(2));
        System.out.println("Substring from 3rd character till 9th character : "+s.substring(3,9));
    }
}
```

```
Length : 20
Uppercase : HELLO INDIA 123 $#@!
Lowercase : hello india 123 $#@!
1st character is : H
Substring from 3rd character till end : llo India 123 $#@!
Substring from 3rd character till 9th character : lo Ind
```

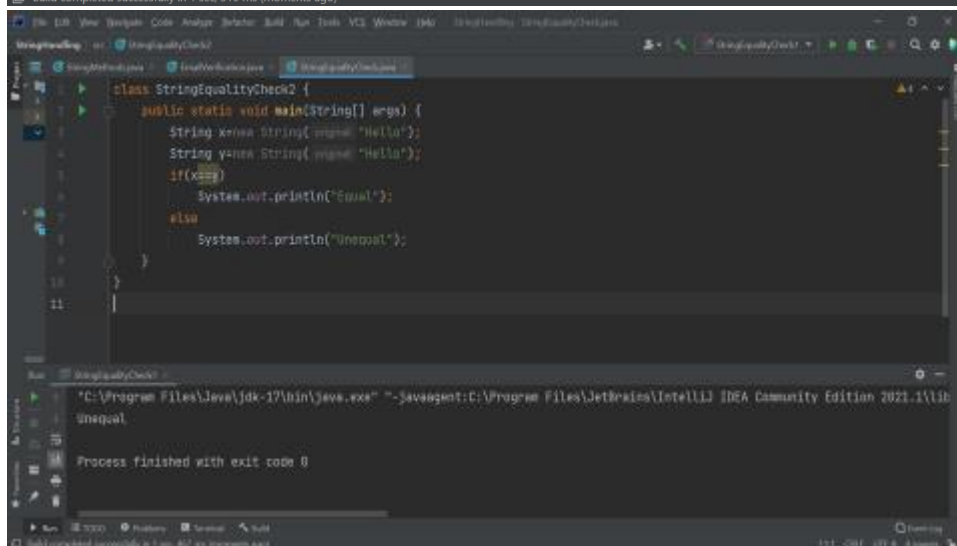
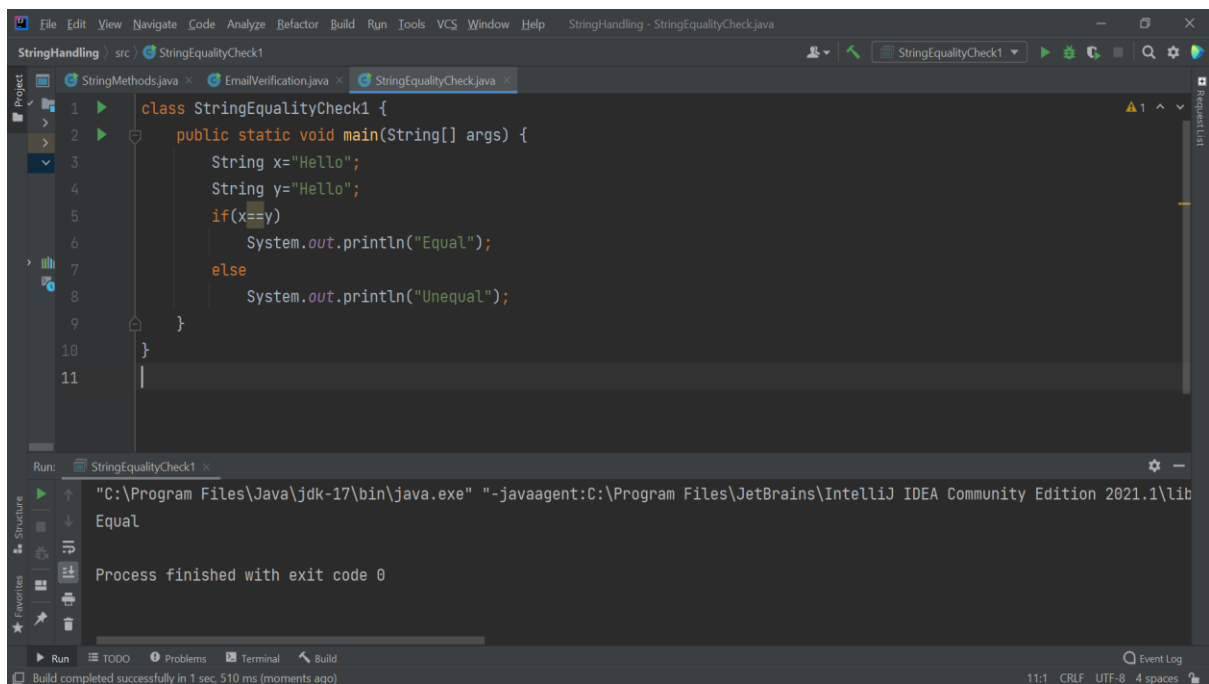
Test Case

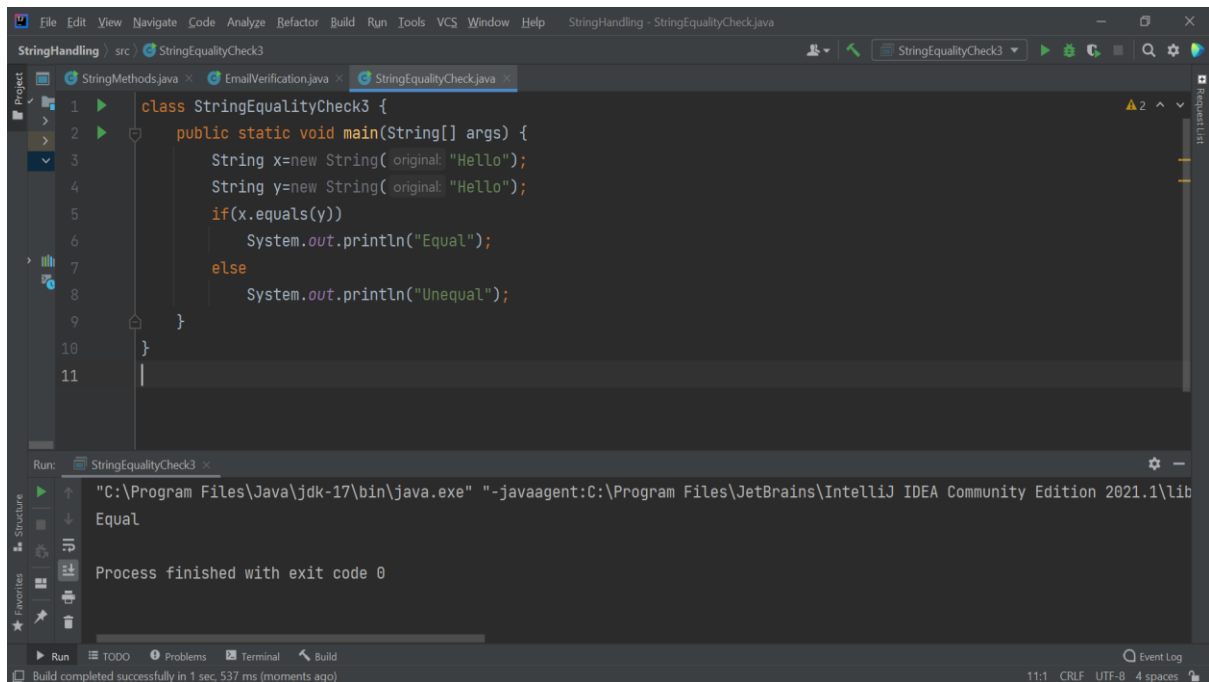
- Write a program to input an email ID and verify it with following rules
 - Minimum length 3
 - Maximum length 20
 - Must have at least one dot (.)
 - Must have one and only one @

```
import java.util.Scanner;
public class EmailVerification {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter the email : ");
        String email=sc.next();
        int length=email.length();
        int dot=email.indexOf(".");
        int atfront=email.indexOf("@");
        int atback=email.lastIndexOf(str: "@");
        if (length>=3 && length<=20 && dot!=-1 && atfront==atback)
            System.out.println(email+" is a valid email id");
        else
            System.err.println(email+" is not a valid email id");
    }
}
```

How to compare strings?

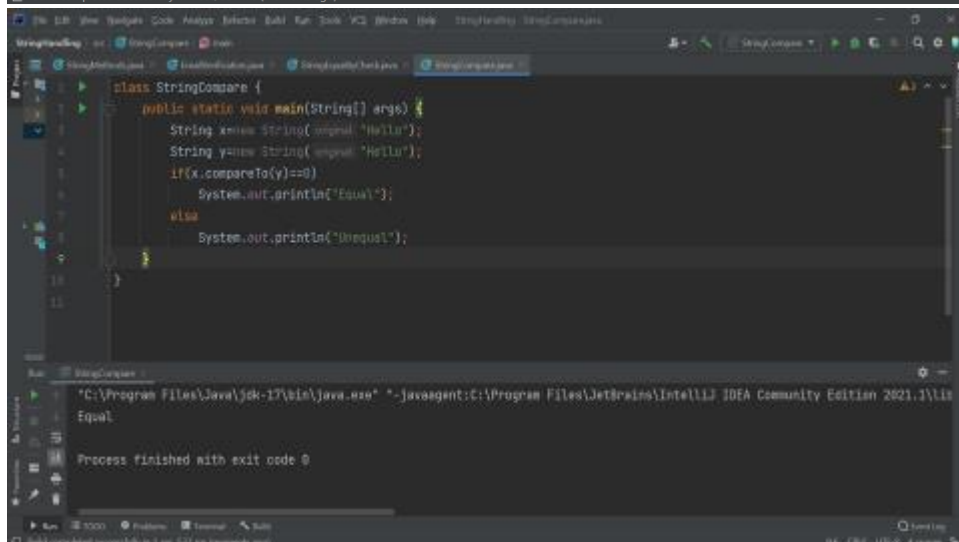
- If a string is managed as literal we can use equality operator (==) or **equals()** or **compareTo()** methods
- If a string is managed as instance use **equals()** or **compareTo()** methods





```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help StringHandling - StringEqualityCheck3.java
StringHandling > src > StringEqualityCheck3
StringEqualityCheck3.java
1 class StringEqualityCheck3 {
2     public static void main(String[] args) {
3         String x=new String( original: "Hello");
4         String y=new String( original: "Hello");
5         if(x.equals(y))
6             System.out.println("Equal");
7         else
8             System.out.println("Unequal");
9     }
10 }
11

Run: StringEqualityCheck3 x
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.1\lib
Equal
Process finished with exit code 0
Build completed successfully in 1 sec, 537 ms (moments ago)
```



```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help StringHandling - StringCompare.java
StringHandling > src > StringCompare.java
StringCompare.java
1 class StringCompare {
2     public static void main(String[] args) {
3         String x=new String( original: "Hello");
4         String y=new String( original: "Hello");
5         if(x.compareTo(y)==0)
6             System.out.println("Equal");
7         else
8             System.out.println("Unequal");
9     }
10 }
11

Run: StringCompare
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.1\lib
Equal
Process finished with exit code 0
Build completed successfully in 1 sec, 527 ms (moments ago)
```

How to do the password masking in Java?

Using readPassword() method Console class. Provided in Java 6 and above.

```
public class PasswordInput {
    public static void main(String[] args) throws Exception {
        java.io.Console console = System.console();
        String username = console.readLine( fmt: "Username: ");
        String password = new String(console.readPassword( fmt: "Password: "));
        System.out.println(username+"/"+password);
    }
}
```

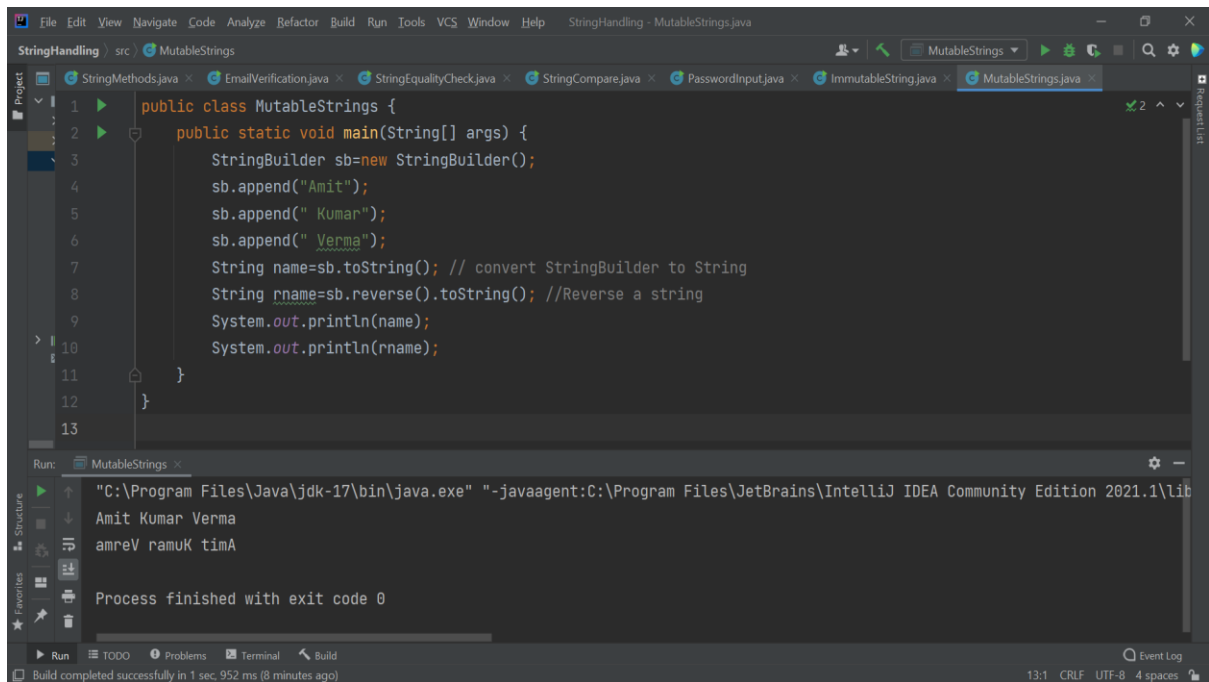
What is immutable strings?

- A string that do not allow to change its contents is called as immutable strings
- String class is used to manage immutable strings
- When adding two strings with addition operator (+) it waste one memory space every time due to immutable nature of String class

```
public class ImmutableString {
    public static void main(String[] args) {
        String name="Rohit";
        name=name+" Kumar";
        name=name+" Verma";
        System.out.println(name);
    }
}
```

What are mutable classes?

- Special classes which allows to change the data in same location are called as mutable classes
- Java provides two classes to manage mutable data
 - StringBuffer class (Java 1.0)
 - StringBuilder class (Java 5.0)
- Use **append()** method to append the data
- Use **reverse()** method to reverse a string



```
1 public class MutableStrings {
2     public static void main(String[] args) {
3         StringBuilder sb=new StringBuilder();
4         sb.append("Amit");
5         sb.append(" Kumar");
6         sb.append(" Verma");
7         String name=sb.toString(); // convert StringBuilder to String
8         String rname=sb.reverse().toString(); //Reverse a string
9         System.out.println(name);
10        System.out.println(rname);
11    }
12 }
13 }
```

Run: MutableStrings

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.1\lib\idea_rt.jar=5000:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.1\bin" -Dfile.encoding=UTF-8
Amit Kumar Verma
amreV ramuK timA
Process finished with exit code 0
```

Build completed successfully in 1 sec, 952 ms (8 minutes ago)

What is OOPs?

OOPs stands for Object Oriented Programming System.

It is a system independent of any programming language to a goal “Building better and scalable project management” using a set of components called as pillars of OOPs

1. Encapsulation
2. Abstraction
3. Polymorphism
4. Inheritance

These core pillars require three other components

1. Class
2. Instance
3. Reference

What is class?

A set of specifications or blueprint about an entity describing all possible data members and the methods to be applicable on that entity.

A class used to create multiple instances of same type.

Use **class** keyword to define a class

Syntax

```
class <entityname>{

    // members
```

}

Types of members inside a class

1. Data Members
2. Methods

Types of methods inside a class

1. Special methods
 - a. Constructor
 - b. Finalizer
 - c. Setter
 - d. Getter
2. General methods

What is constructor?

- Special method having special features
 - Same name as class name
 - No return type
 - Used to initialize data in data members of an instance

What is setter?

- Special methods prefixed with **set**
- Used to update the value in some data member of an instance

What is getter?

- Special methods prefixed with **get**
- Used to return the value in some data member of an instance

Java provides a built-in reference called **this** to refer the current instance

Example

Entity → Customer

Data Members → cid, name, mobile, balance

Special Methods → Constructor, setter for mobile, getter for data members

General methods → deposit(), withdraw()

```

public class Customer {
    int cid;
    String name,mobile;
    double balance;

    public Customer(int cid, String name, String mobile, double
balance) {
        this.cid = cid;
        this.name = name;
        this.mobile = mobile;
        this.balance = balance;
    }

    public void setMobile(String mobile) {
        this.mobile = mobile;
    }

    public int getCid() {
        return cid;
    }

    public String getName() {
        return name;
    }

    public String getMobile() {
        return mobile;
    }

    public double getBalance() {
        return balance;
    }
    void deposit(double amount){
        balance+=amount;
    }
    void withdraw(double amount){
        balance-=amount;
    }
}

```

Create another class PNB with entry point

Create instance of Customer class with sample data

```

public class PNB {
    public static void main(String[] args) {
        Customer c1=new Customer(123,"Harish
Verma","1122334455",5000);
        System.out.println(c1.getName()+" "+c1.getBalance());
        c1.deposit(6000);
        System.out.println(c1.getName()+" "+c1.getBalance());
        c1.withdraw(1200);
        System.out.println(c1.getName()+" "+c1.getBalance());
    }
}

```


Core Components of OOPs

The core components of OOPs are

- Encapsulation
- Abstraction
- Polymorphism
- Inheritance

These core components define the rules for the classes, their members and the instances.

What is encapsulation?

It gives the data binding rule which says that all the data members and the methods to be operated on that data members must be placed under same body called as class.

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates.

```
class <classname>{  
    // members  
    // methods  
}
```

What is abstraction?

It defines the access control on the class and its members using special keywords called as access specifiers.

Java provides four access levels

- Public
- Protected
- Package or Default
- Private

Java provides three keywords to define the access levels on the class and its members

- public
- protected
- private

The public keyword can be applied with class and its members to define the global access of the class and its members. A public class and public member can be access in any other class of any package.

Example

```
public class A{  
    public void hello(){ System.out.println("Hello");}  
}
```

The protected keyword can be applied to the members only. Such members can be used within the same class or in child class of same package or other package. Such members can also be used with instance variables within the same package.

Example

```
class A {  
    protected void hello(){System.out.println("Hello");}  
}
```

The private keyword can be applied to the members only. Such members cannot be accessed outside the class.

Example

```
class A{  
    private int num;  
}
```

The default access is within the current package. If we do not specify any access specifier on the class and its members, such class and its members can be accessed with the current package only. The default access is also called as package access.

What is polymorphism?

A concept that allows multiple formats and usage of an item. It can be of two types

- Compile Time Polymorphism
- Runtime Polymorphism

When the compiler knows about different usage or forms of an item at the time of compilation, it is called as compile time polymorphism. It can be achieved using another concept called as method overloading.

When the runtime environment knows about different usage of an item at the time of running a class, it is called as runtime polymorphism. It can be achieved using another concept called as method overriding.

What is method Overloading?

When we have two or more methods with the same name but different number of argument or different type of arguments, it is called as method overloading.

Return type do not participate in method overloading

Example

Suppose we want to show the area of circle, area of square and area of rectangle. All three functionalities are same. We can create a method area() for all the activities with different number of arguments or type of arguments.

```
class OverloadingDemo{
```



```

    public static void area(int side){
        System.out.println("Area of square is "+ side*side);
    }
    public static void area(int length, int width){
        System.out.println("Area of rectangle is "+ length*width);
    }
    public static void area(double radius){
        System.out.println("Area of circle is "+ Math.PI*radius*radius);
    }

    public static void main(String[] args){
        area(5);        // area of square
        area(5.0);      // area of circle
        area(5,6);      // area of rectangle
    }
}

```

What is Inheritance?

It is most important component of OOPs that provides re-usability of code reducing the code size and improving the performance of the application or project.

It is based on parent-child relationship where members of the parent class can be reused in child class.

Use **extends** keyword to inherit a class into child class.

```

class A{
}
class B extends A{
}

```

Here A is parent class and B is child class

Important Points to remember

- Java allows single inheritance only
- Java provides a library class `java.lang.Object` that is top level class in Java hierarchy
- All classes in Java are child of `Object` class by default
- Java provides `super` keyword to interact with super class

Hierarchy can be written as

Object → A → B

Example

Create the classes to manage data and functionality of different entities in some organization

Vendor → vid, name, email, mobile, itemlist

Employee → empid, name, email, mobile, designation

Customer → cid, name, email, mobile, amountdue

If you see here all three classes have some common members

- name, email and mobile

Instead of managing data and related methods at three places, we can manage only once using inheritance

```
class Common{
    private String name, email, mobile;
    public Common(String name, String email, String mobile){
        this.name=name;
        this.mobile=mobile;
        this.email=email;
    }
    public String getName(){ return name;}
    public String getEmail(){ return email;}
    public String getMobile(){ return mobile;}
}
```

```
class Employee extends Common{
    private int empid;
    private String designation;
    public Employee(int empid, String name, String email, String mobile, String designation){
        super(name, email, mobile);
        this.empid=empid;
        this.designation=designation;
    }
    public int getEmpID(){ return empid;}
    public String getDesignation(){ return designation;}
}
```

```
class InheritanceTest{
    public static void main(String []args){
        Employee e=new Employee(121,"Amit","amit@gmail.com",
"1234567890","Manager");
        System.out.println(e.getName() + "," + e.getDesignation());
    }
}
```

What is method overriding?

When a method of parent class is reused in child class with same signature and different body contents, then it is called as method overriding.

While overriding we can increase the scope of overridden method but cannot decrease it.

Use `@Override` annotation to mark the overriding. It is optional.

```
class A{
    public void show(){
        System.out.println("Welcome to A");
    }
}

class B extends A{
    @Override
    public void show(){
        System.out.println("Welcome to B");
    }
}
```

What is need of Method Overriding?

Method overriding can be used to achieve the runtime polymorphism where one reference of parent class can be used to access multiple childs.

Java provides an inheritance based rule, generally called as Golden Rule of Inheritance. It states that a parent can hold reference to its childs but allows to access only those methods of child whose signature is provided from parent to the child.

```
class A{
    public void show(){
        System.out.println("Welcome to A");
    }
}

class B extends A{
    @Override
    public void show(){
        System.out.println("Welcome to B");
    }
}

class C extends B{
    @Override
```

```

        public void show(){
            System.out.println("Welcome to C");
        }
    }

    class MethodOverridingDemo{
        public static void main(String[] args){
            A x; //reference of parent class
            x=new A();
            x.show();

            x=new B();
            x.show();

            x=new C();
            x.show();
        }
    }

```

Difference between overloading and overriding

- Overloading can be in same class or in child class but overriding can be only be in child class
- Overloading is used to achieve compile time polymorphism while overriding is used to achieve runtime polymorphism
- While overloading, the methods in parent class and in child class can have different scopes but while overriding we can increase the scope of overriding method but cannot decrease it
- While overloading the signature of the methods must be different while overriding the signature must be same
- While overloading no specific marking is required while overriding we can use @Overriding annotation to indicate that overriding is happening

What are different types of classes?

Java provides different type of classes

- Concrete class
- Final class
- Abstract class
- Inner class
- Anonymous Inner class

Concrete class

A class that can be instantiated and can also be inherited is called as concrete class. It is by default.

Example

```
class A{  
}
```

Final class

A class that can be instantiated but cannot be inherited is called as final class. Use final keyword to declare a class as final.

Example

```
final class A{  
}
```

Abstract class

A class that is used for inheritance purpose only but cannot be instantiated is called as abstract class. Use abstract keyword to declare a class as abstract.

Example

```
abstract class A{  
}
```

Inner class

A class within a class is called as inner class. Such class can access all the members of containing class.

Example

```
class A{  
    class B{ // Inner class  
    }  
}
```

Anonymous Inner class

A class within another class without any name is called as anonymous inner class. Such classes have some number rather than name.

Types of methods

Methods can be of three types

- Concrete methods
- Final method
- Abstract method

Concrete method

A method having the signature and body contents that can be overriding in child class is called as concrete methods. It is by default.

Example

```
class A{
    public void hello(){
        System.out.println("Welcome to class A");
    }
}
```

Final method

A method that can be used as it is in child class but cannot be overridden in child class is called as final method. Use final keyword to declare a method as final.

Example

```
class A{
    public final void hello(){
        System.out.println("Welcome to class A");
    }
}
```

Abstract method

A method having only signature but no body contents is called as abstract method. Use abstract keyword to declare the abstract method. Such method are always overridden in child class.

Such methods can be declared at two places

- Inside an abstract class
- Inside an interface

If declared inside an abstract class then abstract keyword is required. If a class contains any abstract method then the class must be declared as abstract.

If declared inside an interface no keyword is required. All methods inside an interface are public and abstract by default.

```
abstract class A{
    public void hello();
}
```

```
interface A{
    void hello();
}
```

What is an interface?

A user defined data type very similar to class that provides a way to implement multiple inheritance in Java. A class can inherit any number of interfaces using **implements** keyword.

```
interface A{  
}  
interface B{  
}  
class X implements A,B{  
}
```

Once interface can also inherit other interfaces as well using **extends** keyword.

```
interface C extends A,B{  
}
```

If inheriting class and interface both, the class must be inherited first.

```
class P{  
}  
class Q extends P implements A,B{  
}
```

What is Multi-Threading?

A system that allows to perform more than operations within the same application is called as multithreading.

Java provides **java.lang.Thread** class to create the threads. Every thread has to perform some task that must be defined inside special method called **run()** that is provided inside **Runnable** interface.

Thread has five states

- Born State
- Ready State
- Running State
- Block State
- Dead State

Methods of Thread class

Constructors

- Thread(Runnable r)
- Thread(Runnable r, String threadName)

Methods

- String getName()
- void setName(String threadName)
- void start()
- void run()
- void sleep(int millisecond)
- void sleep(int millisecond, int nanosecond)
- void wait()
- void notify()
- void notifyAll()
- void suspend()
- void resume()
- void stop()
- static Thread currentThread()

All Java applications are single threaded by default. One thread is always present called as **main** thread.

```
public class ThreadTest {  
    public static void main(String[] args) {  
        String name=Thread.currentThread().getName();  
        System.out.printf("Thread Name is %s",name);  
    }  
}
```


Creating user defined threads

To create your own threads, you need to create a class inherited from Runnable Interface or Thread class to override the run() method. Use Thread.currentThread() method to detect the thread that is currently in process and define the functionality accordingly.

Use sleep() method to define the pause between the threads.

For example,

Lets create three sample threads named as chunnu, munnu and punnu assuming three children who have given three different tasks to be performed at the interval of 100 millisecond. The thread named as chunnu has to print reverse counting 50 to 1, munnu has to print A to Z and punnu has to print 1 to 10.

```
public class MyThread implements Runnable {
    public MyThread(String threadName){
        Thread t=new Thread(this,threadName);
        t.start();
    }
    public void run(){
        String tname=Thread.currentThread().getName();
        if(tname.equals("chunnu")){
            for(int i=50;i>=1;i--){
                System.out.println(tname+" "+i);
                try{Thread.sleep(100);}catch (Exception ex){}
            }
        }
        else if(tname.equals("munnu")){
            for(char ch='A';ch<='Z';ch++){
                System.out.println(tname+" "+ch);
                try{Thread.sleep(100);}catch (Exception ex){}
            }
        }
        else if(tname.equals("punnu")){
            for(int i=1;i<=10;i++){
                System.out.println(tname+" "+i);
                try{Thread.sleep(100);}catch (Exception ex){}
            }
        }
    }
}

public static void main(String[] args) {
    new MyThread("chunnu");
    new MyThread("munnu");
    new MyThread("punnu");
}
```

What is AWT?

AWT stands for Abstract Window Toolkit. It is a set of classes provided in **java.awt** package and used for GUI Development.

The classes are categorized in three major categories

- Containers
- Components
- Helper classes

Container classes are used to contain the components. Top level container for GUI Applications is called as Frame.

- Frame class
- Dialog class
- Panel class
- Window class

Components are used to interact with user for input and output the data

- Label class
- TextField class
- Button class
- TextArea class
- Choice class
- Checkbox class

Helper classes are used to help the containers and the components

- Color
- Font
- Size
- Dimension

All the component classes are inherited from Component class that provides the common methods for all the components and the containers

- setSize(int width, int height)
- setLocation(int x, int y)
- setVisible(boolean value)

- setBackground(Color c)

All the containers are inherited from Container class. It provides the common methods for all the containers

- add(Component c)
- setLayout(LayoutManager lm)

Methods of Frame class

A Java class to be called as GUI Application must inherit Frame class.

Constructor

- Frame()
- Frame(String title)

Methods

- void setTitle(String title)
- String getTitle()

Methods of TextField class

Used to input single line text box and password field

Constructor

- TextField()
- TextField(int columns)

Methods

- void setText(String text)
- String getText()
- void setEchoChar(char ch)

Methods of Button class

Used to create a push button

Constructor

- Button()
- Button(String label)

Methods

- void setLabel(String label)
- String getLabel()

What is Hungarian Notation?

While using controls in a windows application, every control should have proper name that will indicate the type of control along with purpose of the control.

Charles Simony from Hungary gave a notation called as Hungarian notation to name the controls.

Use three character prefix to define the type of control along with purpose of the control e.g. txtName indicates a text field to input the name, btnLogin indicates a button having label as login on it.

Example 1

Create a window of 300x300 with red background

```
import java.awt.*;
public class FirstWindow extends Frame{
    public FirstWindow(){
        setSize(300,300);
        setBackground(Color.red);
        setVisible(true);
    }
    public static void main(String []args){
        new FirstWindow();
    }
}
```

Example 2

Create a login page to input user id and password.

```
import java.awt.*;
public class LoginForm extends Frame{
    Label lblUserID, lblPassword;
    TextField txtUserID, txtPassword;
```

```

Button btnLogin;
public LoginForm(){
    lblUserID=new Label("Login");
    lblPassword=new Label("Password");
    txtUserID=new TextField(10);
    txtPassword=new TextField(10);
    btnLogin=new Button("Login");
    setLayout(new FlowLayout());
    add(lblUserID); add(txtUserID);
    add(lblPassword); add(txtPassword);
    add(btnLogin);
    setSize(300,300);
    setVisible(true);
}
public static void main(String []args){
    new LoginForm();
}
}

```

What is event handing in AWT?

When we perform some activity on a control it is called as an event e.g. click on a button, moving mouse pointer on the frame, checking a checkbox, selecting a value in the drop down, pressing a key in the text field etc.

AWT provides various abstract methods to perform some action when any of such event occurs. These abstract methods are placed inside special interfaces called listeners under java.awt.event package

- ActionListener interface
- MouseListener interface
- MouseMotionListener interface
- KeyListener interface

These interfaces provide pre-defined abstract methods that we need to override to define the task. E.g. ActionListener interface provides an abstract method

```
public void actionPerformed(ActionEvent e)
```

To perform the overridden action, we need to associate the listener with the control using predefined methods like `addActionListener()` where we need to define the instance of the class where the listener is implemented.

Example

In last login form, add the functionality on the login button. When we click on the login button, if the given user id is demouser and password is demopassword then print Authentication Passed else print Authentication Failed.

```
import java.awt.*;
import java.awt.event.*;
public class LoginForm extends Frame implements ActionListener{
    Label lblUserID, lblPassword;
    TextField txtUserID, txtPassword;
    Button btnLogin;
    public LoginForm(){
        lblUserID=new Label("Login");
        lblPassword=new Label("Password");
        txtUserID=new TextField(10);
        txtPassword=new TextField(10);
        btnLogin=new Button("Login");
        btnLogin.addActionListener(this);
        setLayout(new FlowLayout());
        add(lblUserID); add(txtUserID);
        add(lblPassword); add(txtPassword);
        add(btnLogin);
        setSize(300,300);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e){
        String userid=txtUserID.getText();
        String password=txtPassword.getText();
        if(userid.equals("demouser")
password.equals("demopassword")
        System.out.println("Authentication Passed");
        else
```

```

        System.out.println("Authentication Failed");
    }
    public static void main(String []args){
        new LoginForm();
    }
}

```

What are Layout Managers?

Special classes used to define how the components will be placed inside some container. Use `setLayout()` method to define the layout manager.

AWT provides five layout managers

- `FlowLayout`
- `BorderLayout`
- `GridLayout`
- `GridBagLayout`
- `CardLayout`

`FlowLayout` places the controls from left to right then top to bottom.

Example

```
setLayout(new FlowLayout());
```

`BorderLayout` places the controls in five sections

- `East`
- `West`
- `North`
- `South`
- `Center`

`BorderLayout` is default layout for `Frame` and `Center` is default placement. Use `setLayout()` method with the section name to place a control

```
setLayout(new BorderLayout());
add(lblUserID,"East");
```

GridLayout divides the containers in rows and columns.

Example

```
setLayout(new GridLayout(3,4));
```

GridBagLayout divides the containers in into specific size row and columns

Example

```
setLayout(new GridBagLayout());
```

CardLayout is used to place more control in less space.

Example

```
setLayout(new CardLayout());
```

What is Applet?

Special Java classes that get merged into HTML to provide the dynamic programming into HTML.

A class to be called as Applet must be inherited from **java.applet.Applet** class

To execute the Java applets we need the <applet></applet> tag using attributes

- code="classname"
- width="size"
- height="size"

Points to remember

- The class must be public
- No constructor required
- No setVisible() and setSize() methods required

Test Case

Create an applet having a text box to input a number and button to show the factorial of that number in another text box.

```
import java.applet.Applet;
```

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
public class MyApplet extends Applet implements ActionListener{
```



```

TextField txtNum,txtResult;
Button btnFactorial;
public MyApplet(){
    txtNum=new TextField(10);
    txtResult=new TextField(10);
    btnFactorial=new Button("Factorial");
    btnFactorial.addActionListener(this);

    add(txtNum);add(txtResult);add(btnFactorial);

}
public void actionPerformed(ActionEvent e){
    int num=Integer.parseInt(txtNum.getText());
    int f=1;
    for(int i=1;i<=num;i++) f=f*i;
    txtResult.setText(Integer.toString(f));
}
}

```

```

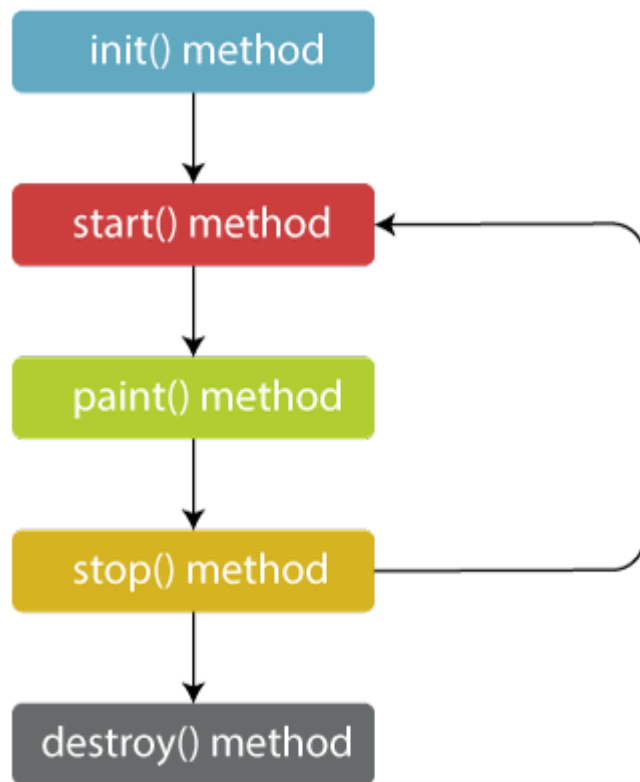
<applet code="MyApplet" width="300" height="300">
</applet>

```

We can test the applets using a built-in tool under JDK called as appletviewer
appletviewer myapplet.html

Life Cycle of Applet

There are various methods that get called during the life cycle of thread under different states



You can override these methods if required

```
class DemoApplet extends Applet {  
    public void init() {  
        // initialized objects  
    }  
    public void start() {  
        // code to start the applet  
    }  
    public void paint(Graphics graphics) {  
        // draw the shapes  
    }  
    public void stop() {  
        // code to stop the applet  
    }  
    public void destroy() {  
        // code to destroy the applet  
    }  
}
```