

JAVA PROGRAMMING

Chap 1 : Java Fundamentals

| Program: Master of Computer Application (MCA) | | | | Semester: I | |
|---|----------------------------------|------------------------------------|--------|---|--|
| Course: Java Programming | | | | Code: | |
| Teaching Scheme | | | | Evaluation Scheme | |
| Lecture (Hours per week) | Practical (Hours per week) | Tutorial (Hours per week) | Credit | Internal Continuous Assessment (ICA) (Marks - 50) | Term End Examinations (TEE) (Marks - 100) |
| 2 | 4 | 0 | 4 | Marks Scaled to 50 | Marks Scaled to 50 |
| Prerequisite: NA | | | | | |
| Course Objective This course will impart knowledge of object-oriented programming, building graphical user interface and database connectivity using Java. | | | | | |
| Course Outcomes After completion of the course, students will be able to - | | | | | |
| <ol style="list-style-type: none"> 1. Implement programs using object oriented programming paradigm 2. Implement programs using collection and generics concepts 3. Develop GUI application with database connectivity | | | | | |

| Detailed Syllabus: | | |
|---------------------------|--|-----------------|
| Unit | Description | Duration |
| 1 | Java Fundamentals Overview of Java, Using Blocks of code, Lexical Issues, Java Class Libraries, Data Types, Variables and Arrays, Operators, Control Statements, Command Line Arguments. | 02 |
| 2 | Classes and Methods Class fundamentals, Declaring Objects, Constructors, Methods, Overloading of methods, Access control, Static and final variables. | 04 |
| 3 | Inheritance Inheritance Basics, method overriding, using abstract classes, using final with inheritance. | 04 |
| 4 | Packages and Interfaces Packages, Access Protection, importing packages, , Interfaces: Defining an Interface, Implementing Interfaces , Applying Interfaces, Variables in Interfaces. | 03 |
| 5 | Exception Handling Exception handling fundamentals, exception types, uncaught exceptions, using try and catch, throw, throws, finally, Java's built-in exceptions, creating your own exceptions. | 02 |
| 6 | Programs using String Handling String Constructors, Special String operators, Character Extraction, String Comparison, Searching Strings and Modifying Strings, Buffer class and its methods. | 02 |
| 7 | Generics and Collections | 05 |

| | | |
|---|---|----|
| | Generics: Introduction, A Generic class with Type Parameters, General Form of a Generic class, Bounded Types, Using wildcard arguments, Creating a Generic Method, Generic class Hierarchies, Collection: Collection Framework, ArrayList class, List Iterator interface, Linked List class, TreeSet class | |
| 8 | GUI design and Event Handling using Java FX Introduction, JavaFX Architecture, application structure, JavaFX, Text, Effect, Anim, UI controls. Types of Events, Processing Events in JavaFX, Event Delivery Process, Event Handlers. | 05 |
| 9 | Java and Database Programming JDBC Architecture, Types of Drivers, JDBC components, JDBC classes and Interfaces, steps for querying the database with JDBC, Database connection, querying and updating database tables, passing parameters to a statement. | 03 |
| | Total | 30 |

Text Books:

1. Herbert Schildt, *Java The Complete Reference*, 11th Edition, Oracle Press, 2020.
2. Sergey Grinev, *Mastering JavaFX10*, Packt Publishing, 2018.

Reference Books:

1. Cay Horstmann, *Core Java Volume I- Fundamentals*, Pearson Education Inc., 2020.
2. Carl Dea, Gerrit Grunwald, José Pereda, Sean Phillips, Mark Heckler, *JavaFX 9 by Example*, 3rd Edition, Apress, 2017.

- Java is a programming language and a platform.
- Java is a high level, robust, object-oriented and secure programming language.
- Java was developed by Sun Microsystems (which is now the subsidiary of Oracle) in the year 1995.
- James Gosling is known as the father of Java.
- Before Java, its name was Oak. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

- There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application : Also known as desktop applications or window-based applications are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

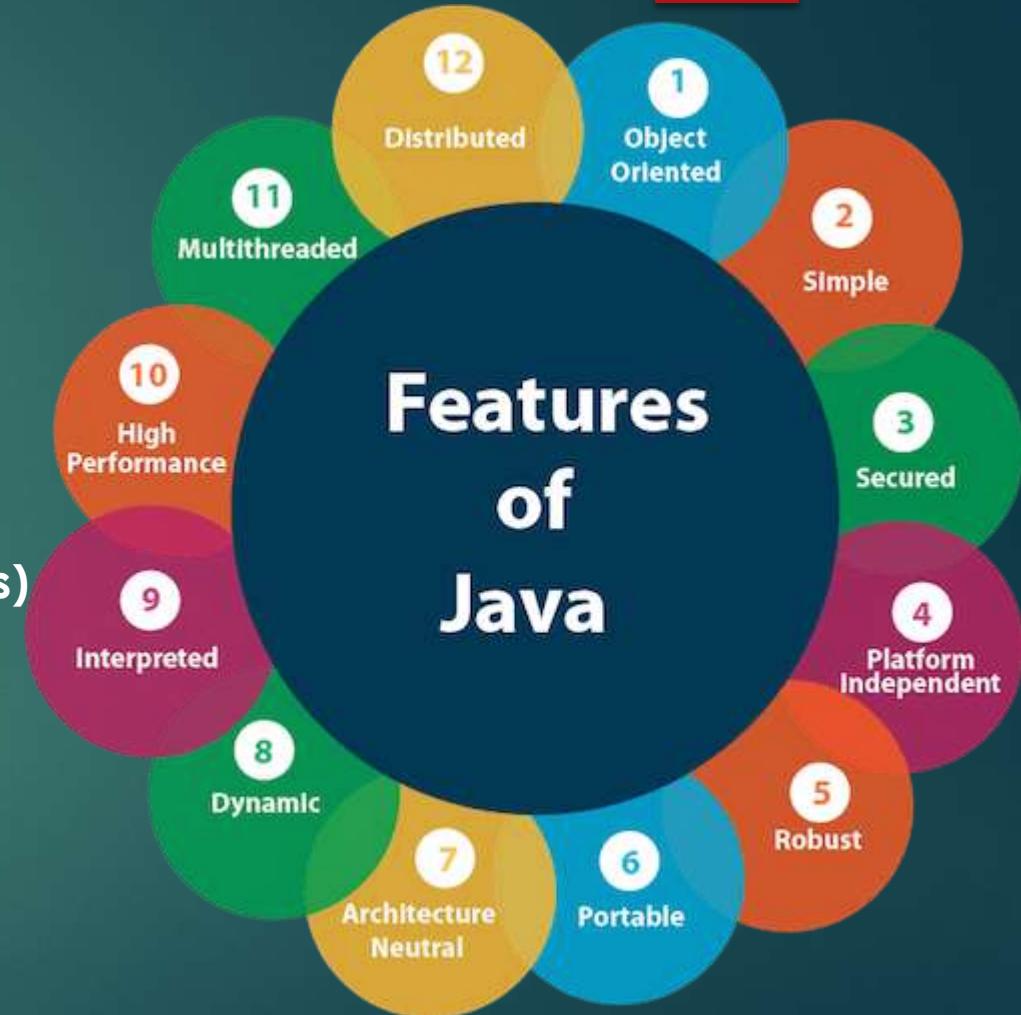
2) Web Application : An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) Enterprise Application : An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application : An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Features of Java

- 1) Simple - (No pointers, no operator overloading, no garbage collection, no multiple inheritance)
- 2) Object-Oriented
- 3) Portable - (bytecode)
- 4) Platform independent - (bytecode)
- 5) Secured - (no explicit pointer)
- 6) Robust - (strong MM Mgmt, Exception Handling, Auto Garbage Collection, no pointers)
- 7) Architecture neutral - (bytecode, fixed size of primitive datatypes)
- 8) Interpreted
- 9) High Performance
- 10) Multithreaded
- 11) Distributed - (helps to create distributed applns using RMI & EJB)
- 12) Dynamic - (uses JIT)
- 13) Garbage Collected



Basic concepts of OOPs are:

- 1) Object
- 2) Class
- 3) Inheritance
- 4) Polymorphism
- 5) Abstraction
- 6) Encapsulation

| C++ | Java |
|--|---|
| It is object oriented but not purely. That is a program can be written in C++ without using classes and objects. | It is purely object oriented that is no program can be written without classes and objects. |
| C++ is platform dependent | Java is platform independent |
| C++ supports the goto statement. | Java doesn't support the goto statement. |
| C++ supports pointers. This makes program system dependent. | Java doesn't support pointers to avoid platform dependency. |
| C++ supports multiple and hybrid inheritance | Java does not support multiple and hybrid inheritance. A slight implementation of the same can be done using a special feature called as Interface. |
| Supports operator overloading | Does not allow operator overloading |
| Supports 3 access specifiers : public, protected and private | Supports 5 access specifiers : public, protected, private, default and private protected |
| Destructors are used in C++ | Java is garbage collected that is the objects are automatically destroyed once their use is over |
| C++ doesn't have exception handling | Java has exception handling |
| Doesn't support multithreading | Supports multithreading |

C++

C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.

Java

Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.

C++ supports structures and unions.

Java doesn't support structures and unions

C++ doesn't support documentation comments.

Java supports documentation comment (/** ... */) to create documentation for java source code.

Has header files

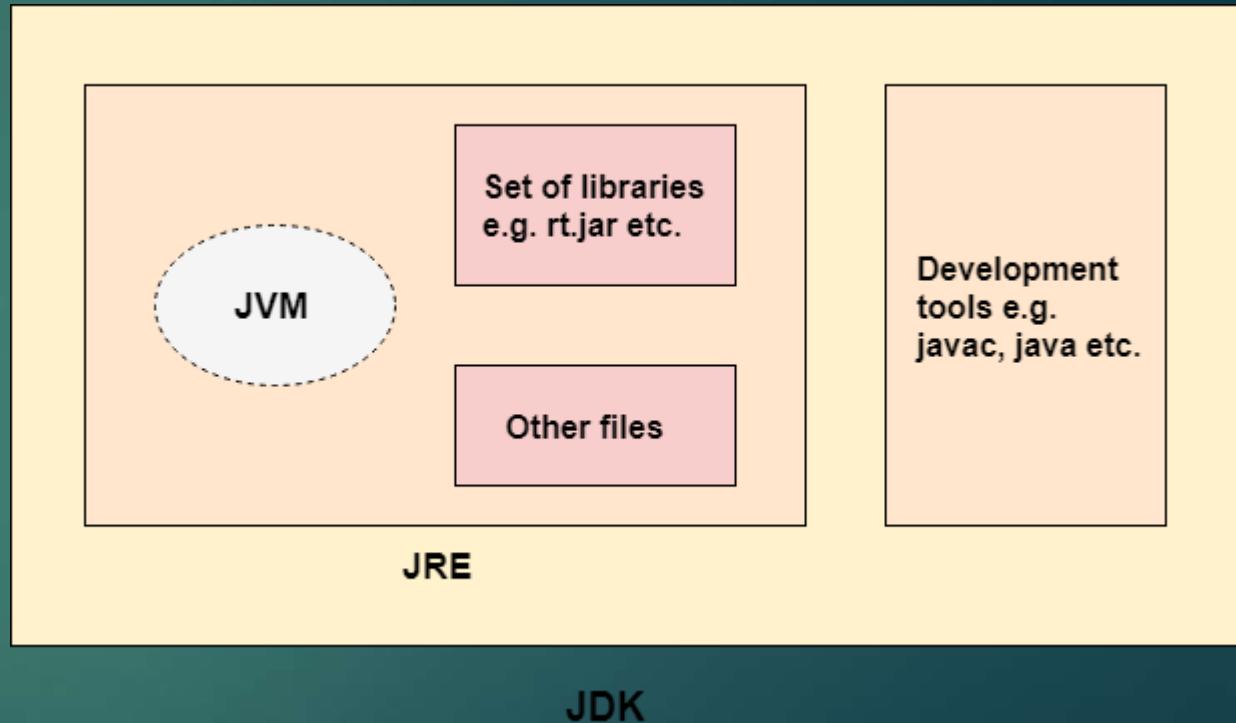
uses the import keyword to include different classes and methods

mainly used for system programming.

mainly used for application programming

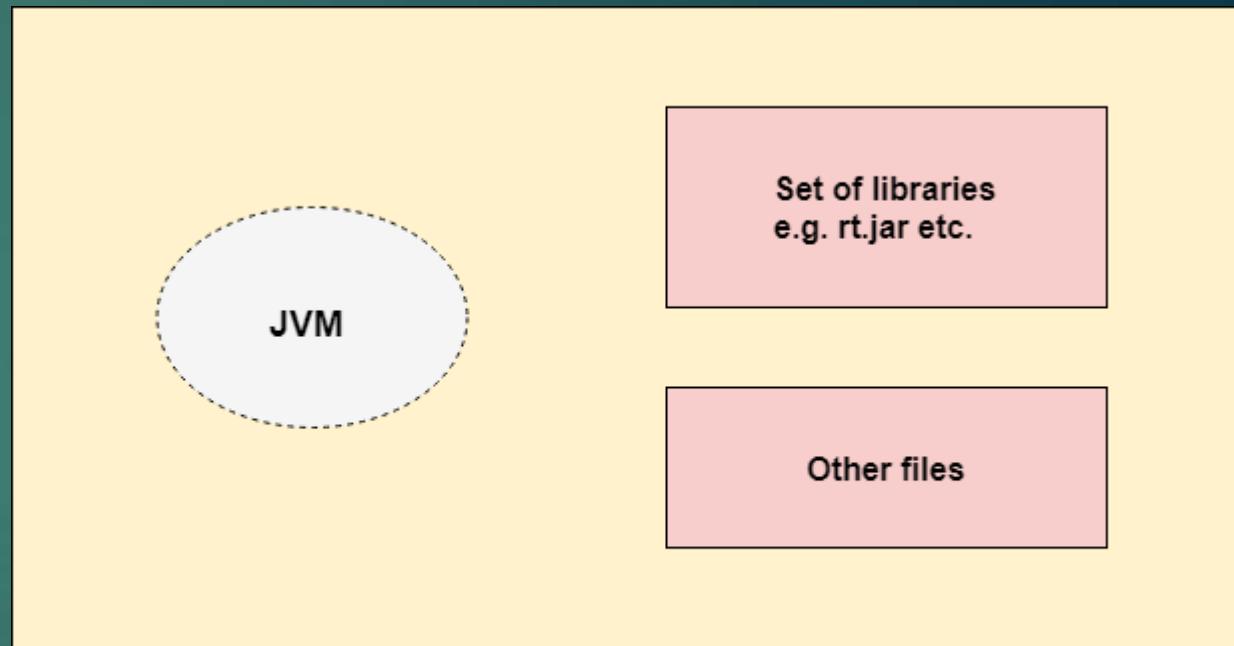
Java Development Kit (JDK)

- ▶ JDK is an acronym for Java Development Kit.
- ▶ The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets.
- ▶ It physically exists. It contains JRE + development tools.
- ▶ The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



Java Runtime Environment (JRE)

- ▶ JRE is an acronym for Java Runtime Environment.
- ▶ It is also written as Java RTE.
- ▶ The Java Runtime Environment is a set of software tools which are used for developing Java applications.
- ▶ It is used to provide the runtime environment.
- ▶ It is the implementation of JVM.
- ▶ It physically exists.
- ▶ It contains a set of libraries + other files that JVM uses at runtime.
- ▶ The implementation of JVM is also actively released by other companies besides Sun Micro Systems.



JRE

Java Virtual Machine (JVM)

- ▶ JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.
- ▶ JVM is an interpreter. It is the main component of Java that makes java platform independent.
- ▶ It is because the bytecode file can be taken onto any system and the JVM residing in that system will interpret and translate the byte code into machine readable code.
- ▶ The JVM performs following operation:
 - ▶ Loads code
 - ▶ Verifies code
 - ▶ Executes code
 - ▶ Provides runtime environment

Tokens of Java

- ▶ Character Set
- ▶ Keywords
- ▶ Identifiers
- ▶ Constants & variables
- ▶ Datatypes
- ▶ operators

Character Set

- ▶ Alphabets: Both lowercase (a, b, c, d, e, etc.) and uppercase (A, B, C, D, E, etc.).
- ▶ Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- ▶ Special symbols: _, (,), {, }, [,], +, -, *, /, %, !, &, |, ~, ^, <, =, >, \$, #, ?, Comma (,), Dot (.), Colon (:), Semi-colon (;), Single quote ('), Double quote ("), Back slash (\).
- ▶ White space: Space, Tab, New line.

Keywords

- ▶ There are 50 keywords currently defined in the Java language.
- ▶ These keywords, combined with the syntax of the operators and separators, form the foundation of the Java language.
- ▶ **These keywords cannot be used as identifiers.**
- ▶ Thus, they cannot be used as names for a variable, class, or method.
- ▶ In addition to the keywords, Java reserves the following: **true**, **false**, and **null**. These are values defined by Java. You may not use these words for the names of variables, classes, and so on.

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

Table 2-1 Java Keywords

Identifiers

- ▶ Identifiers are used to name things, such as classes, variables, and methods.
- ▶ An identifier may be any descriptive sequence of **uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters**. (The dollar-sign character is not intended for general use.)
- ▶ Following rules needs to be followed while creating identifiers
 1. An identifier can consist of Alphabets, digits, and two special symbols _ (underscore) and \$ (dollar)
 2. An identifier cannot start with a digit. It has to start with an alphabet or _ or \$
 3. No other special symbols apart from _ and \$ are allowed. Not even blank spaces.
 4. An identifier cannot be a keyword.
 5. It is case sensitive. That is firstName, FirstName, firstname are three different identifiers.

Identifiers

Exercise : Identify valid and invalid identifiers from the list given below –

1. simple_interest
2. char
3. 3friends
4. _3\$friends
5. Simple interest
6. #3friends
7. void
8. Void
9. \$xyz

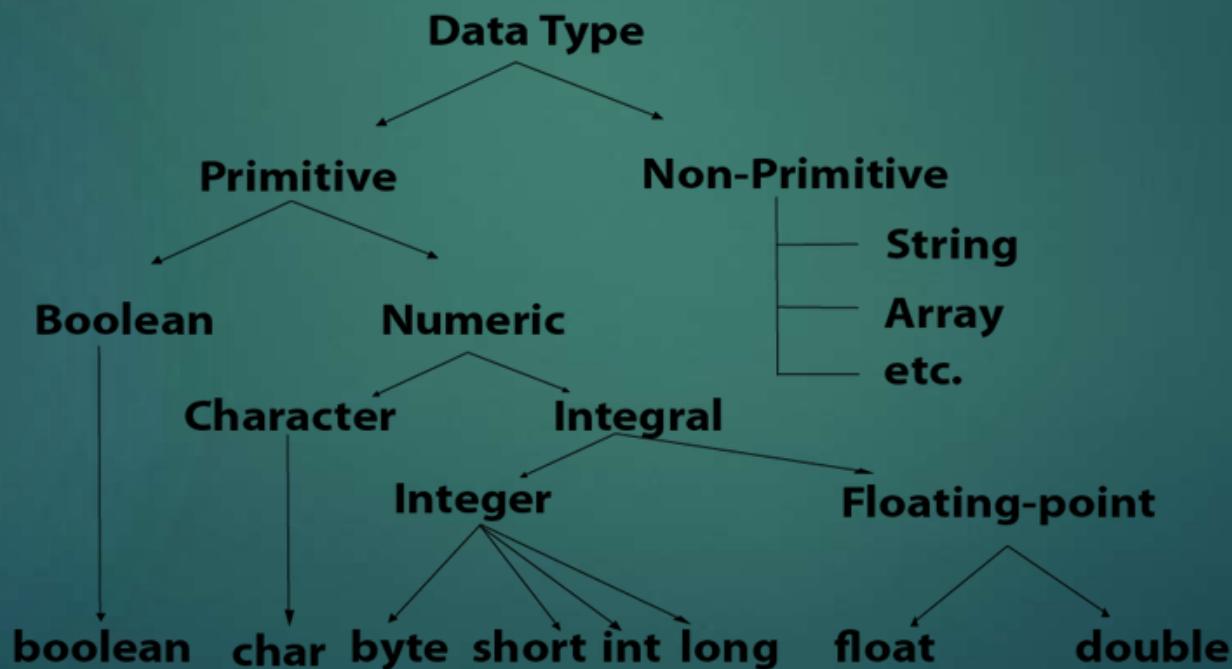
Constants and variables

- ▶ A **constant** is an entity in programming that is immutable.
- ▶ In other words, the value that cannot be changed after assigning it.
- ▶ In java constants can be defined by using the keyword “final” before the datatype.
- ▶ Constants are used to declare values that remain constant like value of pi.
- ▶ Eg : final double pi=3.14;

- ▶ **Variables** are containers for storing data values.
- ▶ A variable is assigned with a data type.
- ▶ A variable is the name of a reserved area allocated in memory.
- ▶ In other words, it is a name of the memory location.
- ▶ It is a combination of "vary + able" which means its value can be changed.
- ▶ eg: **int** data=50; //Here data is a variable

Datatypes

- ▶ Data types specify the different sizes and values that can be stored in the variable.
- ▶ There are two types of data types in Java:
- ▶ **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
- ▶ **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.



Datatypes

| Data Type | Default Value | Default size | Range |
|-----------|---------------|--------------|---|
| boolean | false | 1 bit | true or false |
| char | '\u0000' | 2 byte | '\u0000' (or 0) to '\uffff' (or 65,535 inclusive). |
| byte | 0 | 1 byte | -128 to 127 (inclusive) |
| short | 0 | 2 byte | -32,768 to 32,767 (inclusive) |
| int | 0 | 4 byte | - 2,147,483,648 to 2,147,483,647 (inclusive) |
| long | 0L | 8 byte | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (inclusive) |
| float | 0.0f | 4 byte | -3.4e38 to 3.4e38 |
| double | 0.0d | 8 byte | Its value range is unlimited |

Operators

► There are many types of operators in Java which are given below:

1. Unary Operator
2. Arithmetic Operator
3. Shift Operator
4. Relational Operator
5. Bitwise Operator
6. Logical Operator
7. Ternary Operator
8. Assignment Operator

Operators

► Operator precedence

| Operator Type | Category | Precedence |
|---------------|----------------------|---|
| Unary | postfix | <code>expr++ expr--</code> |
| | prefix | <code>++expr --expr +expr -expr ~ !</code> |
| Arithmetic | multiplicative | <code>* / %</code> |
| | additive | <code>+ -</code> |
| Shift | shift | <code><< >> >>></code> |
| Relational | comparison | <code>< > <= >= instanceof</code> |
| | equality | <code>== !=</code> |
| Bitwise | bitwise AND | <code>&</code> |
| | bitwise exclusive OR | <code>^</code> |
| | bitwise inclusive OR | <code> </code> |
| Logical | logical AND | <code>&&</code> |
| | logical OR | <code> </code> |
| Ternary | ternary | <code>? :</code> |
| Assignment | assignment | <code>= += -= *= /= %= &= ^= = <=>= >>>=</code> |

Ternary Operator

- The operator that requires three operands is called as Ternary Operator.
- Java supports only one ternary operator ? :
- Syntax : (condition) ? (Value if true) : (value if false)

```
import java.util.*;  
  
class Ternary{  
  
    public static void main(String args[]){  
  
        int a, b, large;  
  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter 2 numbers");  
  
        a=sc.nextInt();  
        b=sc.nextInt();  
  
        large=(a>b)?a:b;  
  
        System.out.println("Largest number is :" +large);  
    } }
```

```
F:\Java>javac Ternary.java  
  
F:\Java>java Ternary  
Enter 2 numbers  
15  
34  
Largest number is : 34
```

Comments

- ▶ Single line Comments - // comments
- ▶ Multi line comments - /* comments */
- ▶ Documentation comments - /** comments */

Input/Output in Java

Output in Java

- ▶ `System.out.println("Hello");`
- ▶ `System` – class of `Java.lang` package
- ▶ `out` – variable of class `System` corresponding to the std o/p device i.e. monitor
- ▶ `println()/print()` – static methods for displaying content

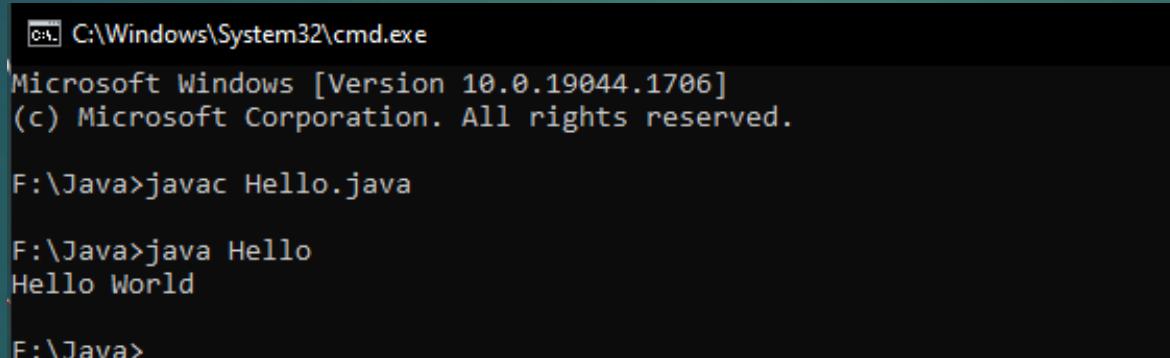
Input in Java

- ▶ `BufferedReader` – `readLine()`
- ▶ `Scanner` – `nextInt()`, `nextLong()`, `nextFloat()`, `nextDouble()`, `next()`, `nextLine()`

Simple Java Program

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

Output :

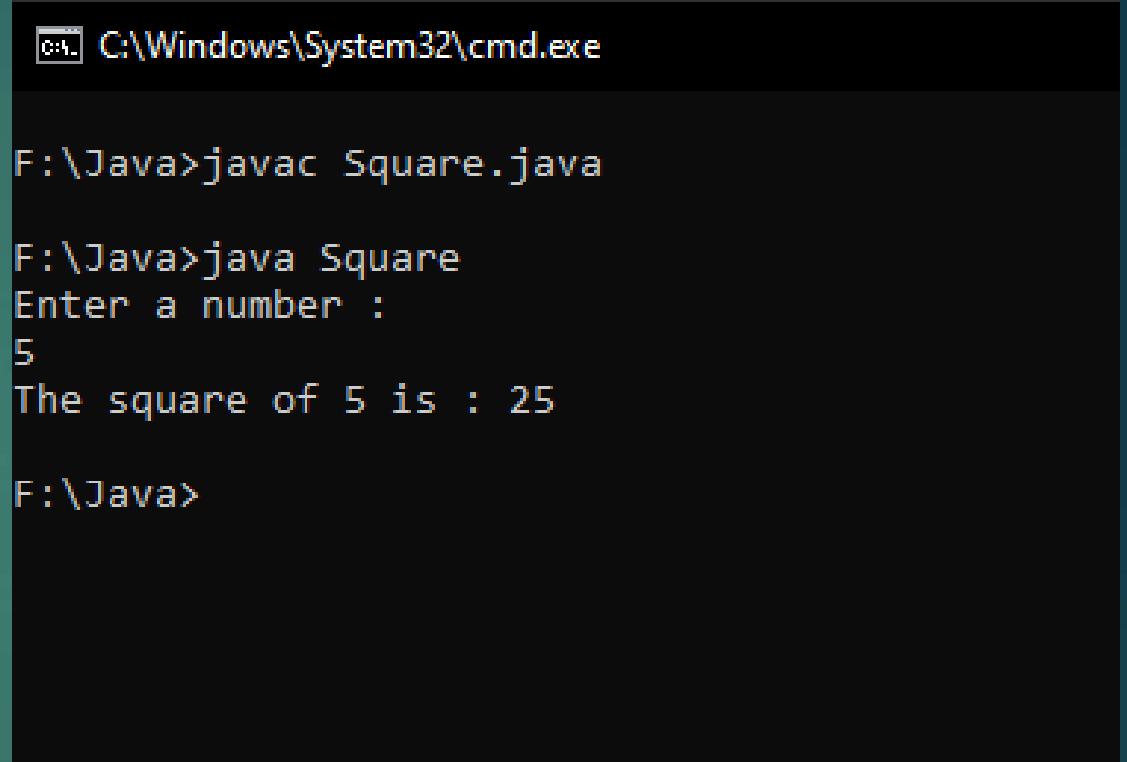


The image shows a screenshot of a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The window displays the following text:

```
Microsoft Windows [Version 10.0.19044.1706]  
(c) Microsoft Corporation. All rights reserved.  
F:\Java>javac Hello.java  
F:\Java>java Hello  
Hello World  
F:\Java>
```

Program for accepting input (Scanner)

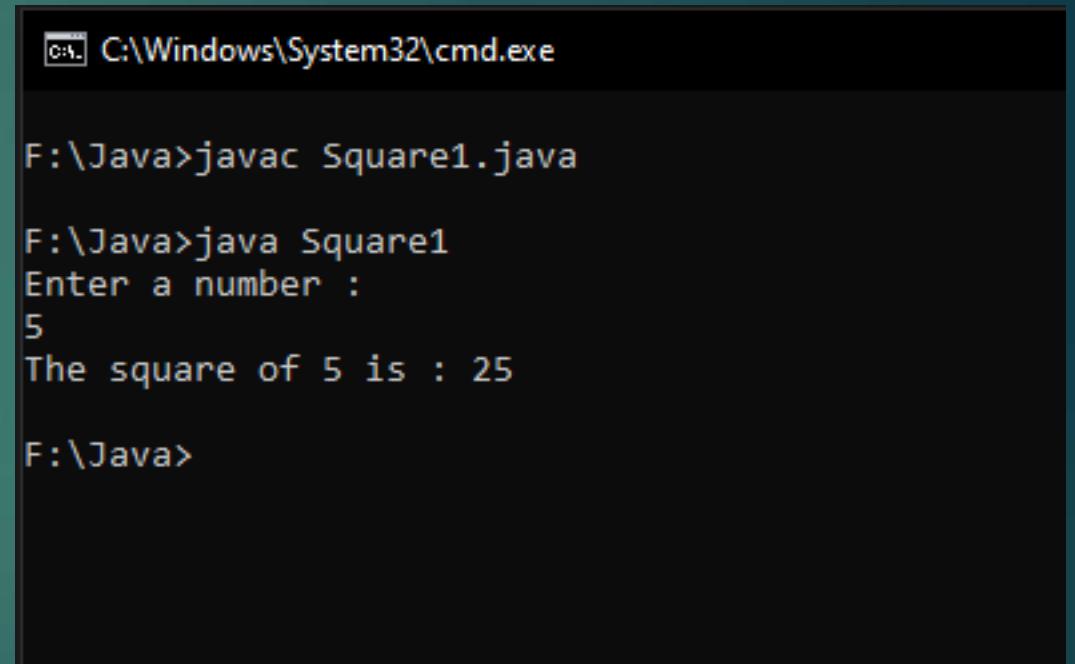
```
import java.util.*;  
  
class Square{  
  
    public static void main(String args[])  
    {  
  
        int i, res;  
  
        Scanner sc=new Scanner (System.in);  
  
        System.out.println("Enter a number : ");  
  
        i=sc.nextInt();  
  
        res=i*i;  
  
        System.out.println("The square of " + i + " is :  
" + res);  
  
    }  
}
```



```
C:\Windows\System32\cmd.exe  
  
F:\Java>javac Square.java  
  
F:\Java>java Square  
Enter a number :  
5  
The square of 5 is : 25  
  
F:\Java>
```

Program for accepting input (BufferedReader)

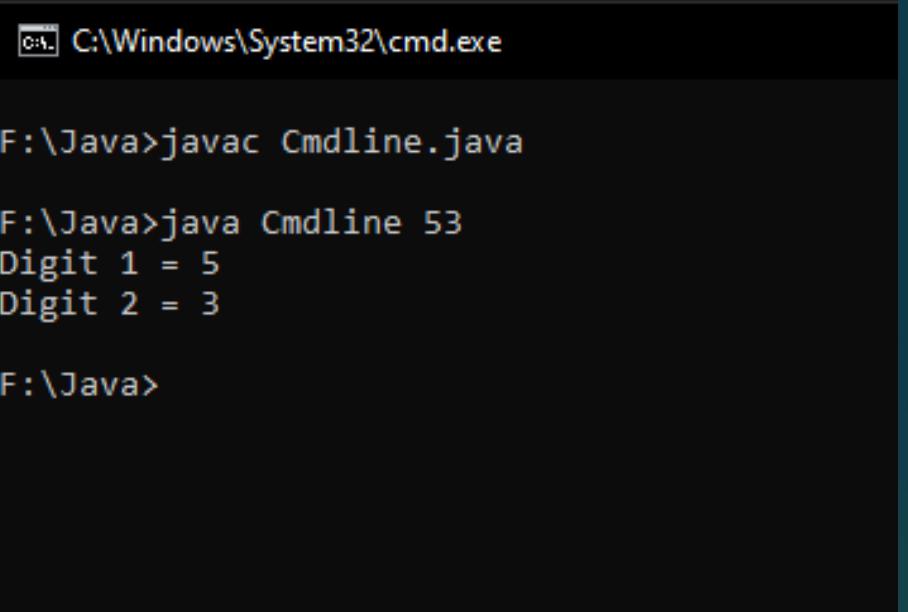
```
import java.io.*;
class Square{
    public static void main(String args[]) throws IOException
    {
        int i, res;
        BufferedReader br=new BufferedReader (new
InputStreamReader(System.in));
        String str;
        System.out.println("Enter a number :");
        str=br.readLine();
        i=Integer.parseInt(str);
        res=i*i;
        System.out.println("The square of " + i + " is : " + res);
    }
}
```



```
C:\Windows\System32\cmd.exe
F:\Java>javac Square1.java
F:\Java>java Square1
Enter a number :
5
The square of 5 is : 25
F:\Java>
```

Program for accepting input (command line)

```
class Cmdline{  
    public static void main(String args[])  
    {  
        int n,d1,d2;  
        n=Integer.parseInt(args[0]);  
        d2=n%10;  
        d1=n/10;System.out.println("Digit 1 = "+d1+"\nDigit 2 = "+d2);  
    }  
}
```



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The command 'javac Cmdline.java' is entered, followed by 'java Cmdline 53'. The output displays 'Digit 1 = 5' and 'Digit 2 = 3', indicating the program correctly extracted the digits from the input number 53.

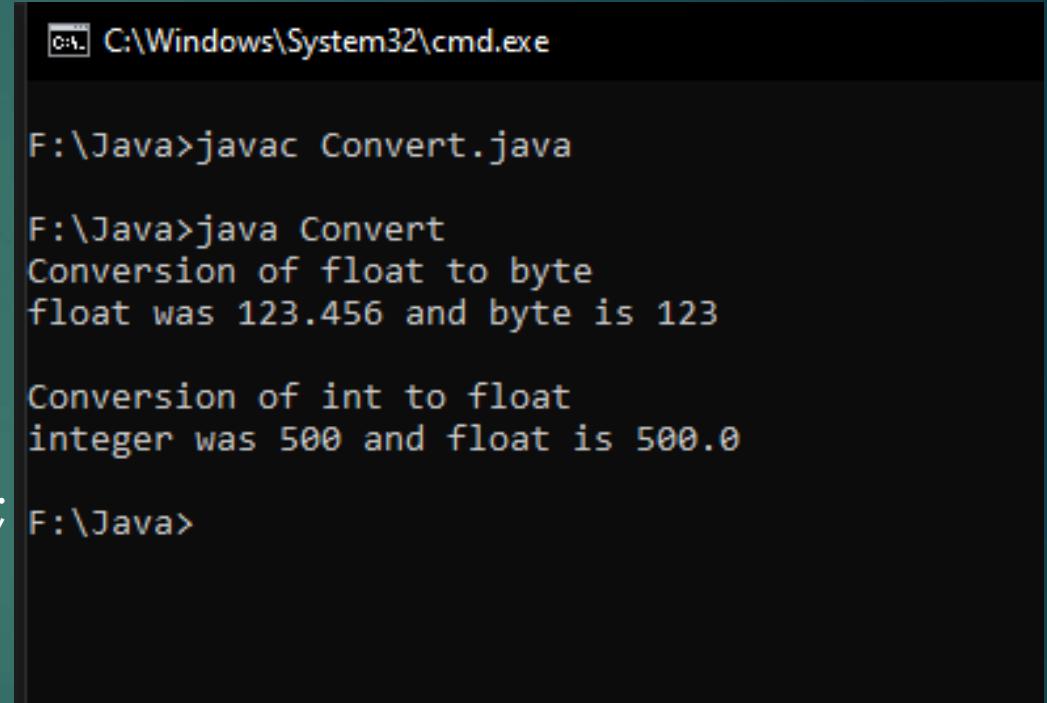
```
C:\Windows\System32\cmd.exe  
F:\Java>javac Cmdline.java  
F:\Java>java Cmdline 53  
Digit 1 = 5  
Digit 2 = 3  
F:\Java>
```

Typecasting in Java

- ▶ **Type casting** is a method or process that converts a data type into another data type in both ways manually and automatically.
- ▶ The automatic conversion is done by the compiler and manual conversion performed by the programmer
- ▶ Converting a lower data type into a higher one is called as **implicit conversion** or **casting down**.
- ▶ It is done automatically.
- ▶ It takes place when:
 - ▶ Both data types must be compatible with each other.
 - ▶ The target type must be larger than the source type.
- ▶ Eg : typecasting int to long
- ▶ Converting a higher data type into a lower one is called as **explicit conversion** or **casting up**.
- ▶ It is done manually by the programmer.
- ▶ If we do not perform casting then the compiler reports a compile-time error.
- ▶ Eg : typecasting float to byte
- ▶ Syntax for explicit typecasting –
`(target datatype)source datatype variable`
- ▶ Eg :
`float x;
(byte) x;`

Typecasting in Java

```
class Convert{  
    public static void main(String args[])  
    {  
        byte b;  
        int i=500;  
        float f=123.456f;  
        System.out.println("Conversion of float to byte");  
        b=(byte) f; // explicit  
        typecasting  
        System.out.println("float was "+f+ " and byte is "+b+"\n");  
        System.out.println("Conversion of int to float");  
        f=i; // implicit typecasting  
        System.out.println("integer was "+i+ " and float is "+f);  
    }  
}
```



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The command 'javac Convert.java' is entered and executed. The output shows two conversions: float to byte and int to float, with their respective values printed.

```
C:\Windows\System32\cmd.exe  
F:\Java>javac Convert.java  
F:\Java>java Convert  
Conversion of float to byte  
float was 123.456 and byte is 123  
Conversion of int to float  
integer was 500 and float is 500.0  
F:\Java>
```

Control statements

- ▶ Control statements are of two types
 - ▶ Conditional statements – if-else, switch case
 - ▶ Iterative statements – for, while, do while

Conditional Statements

If-Else statements

```
import java.util.*;  
  
class PositiveNegativeExample {  
  
public static void main(String[] args) {  
    int n;  
    Scanner sc=new Scanner (System.in);  
    System.out.println("Enter n : ");  
    n=sc.nextInt();  
    if(n>0){  
        System.out.println("POSITIVE");  
    }  
    else if(n<0){  
        System.out.println("NEGATIVE");  
    }  
    else{  
        System.out.println("ZERO");  
    }  
}
```

Switch statements

```
import java.util.*;  
  
public class Main {  
  
public static void main(String[] args) {  
    int day;  
    Scanner sc=new Scanner (System.in);  
    System.out.println("Enter the day number : ");  
    day=sc.nextInt();  
    switch (day) {  
        case 1:  
            System.out.println("Monday");  
            break;  
        case 2:  
            System.out.println("Tuesday");  
            break;  
        case 3:  
            System.out.println("Wednesday");  
            break;
```

```
        case 4:  
            System.out.println("Thursday");  
            break;  
        case 5:  
            System.out.println("Friday");  
            break;  
        case 6:  
            System.out.println("Saturday");  
            break;  
        case 7:  
            System.out.println("Sunday");  
            break;  
    }  
}
```

Conditional Statements

WAP to check if the user entered year is leap or not

```
import java.util.*;
class Leap {
    public static void main(String[] args) {
        int yr;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the yr");
        yr=sc.nextInt();
        if((yr%4==0 && yr%100!=0) | | (yr%100==0 && yr%400==0))
            System.out.println("Leap Year");
        else
            System.out.println("Not a Leap Year");
    }
}
```

```
F:\Java>javac Leap.java
```

```
F:\Java>java Leap
```

```
Enter the yr
2024
```

```
Leap Year
```

```
F:\Java>java Leap
```

```
Enter the yr
2023
```

```
Not a Leap Year
```

Conditional Statements

- WAP to display the class according to the marks scored by student using switch case. The marks scored is taken input from the user and the class is displayed based on the range given below

```
import java.util.*;
class Marks {
    public static void main(String[] args) {
        int marks;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter marks out of 100");
        marks=sc.nextInt();
        if (marks==100)
            System.out.println("Distinction");
        switch(marks/10)
        {
            case 0:
            case 1:
            case 2:
            case 3: System.out.println("Fail");
            break;
            case 4: System.out.println("Pass Class");
            break;
            case 5: System.out.println("Second Class");
            break;
            case 6: System.out.println("First Class");
            break;
            case 7:
            case 8:
            case 9: System.out.println("Distinction");
            break;
            default: System.out.println("InvalidMarks");
        } } }
```

```
case 6: System.out.println("First Class");
break;
case 7:
case 8:
case 9: System.out.println("Distinction");
break;
default: System.out.println("InvalidMarks");
} } }
```

| Marks | Class |
|--------|-------------|
| 70-100 | Distinction |
| 60-69 | First |
| 50-59 | Second |
| 40-49 | Pass |
| 0-39 | Fail |

```
F:\Java>javac Marks.java
F:\Java>java Marks
Enter marks out of 100
85
Distinction
F:\Java>java Marks
Enter marks out of 100
37
Fail
```

Iterative Statements

For loop

```
import java.util.*;  
  
class Natural{  
  
    public static void main(String args[])  
    {  
  
        int i, n;  
  
        Scanner sc=new Scanner  
(System.in);  
  
        System.out.println("Enter n : ");  
  
        n=sc.nextInt();  
  
        for(i=1;i<=n;i++)  
        {  
  
            System.out.println(i);  
  
        }  
    }  
}
```

While loop

```
import java.util.*;  
  
class NaturalWhile{  
  
    public static void main(String args[])  
    {  
  
        int i, n;  
  
        Scanner sc=new Scanner (System.in);  
  
        System.out.println("Enter n : ");  
  
        n=sc.nextInt();  
  
        i=1;  
  
        while(i<=n)  
        {  
  
            System.out.println(i);  
  
            i++;  
        }  
    }  
}
```

Do-while loop

```
import java.util.*;  
  
class NaturalWhile{  
  
    public static void main(String args[])  
    {  
  
        int i, n;  
  
        Scanner sc=new Scanner  
(System.in);  
  
        System.out.println("Enter n : ");  
  
        n=sc.nextInt();  
  
        i=1;  
  
        do  
        {  
  
            System.out.println(i);  
  
            i++;  
        }while(i<=n);  
    }  
}
```

Iterative Statements

WAP to find factorial of a number

```
import java.util.*;
class Fact {
public static void main(String[] args) {
int n,i,fact=1;
Scanner sc=new Scanner(System.in);
System.out.println("Enter the number");
n=sc.nextInt();
for(i = 1; i <= n; i++)
{ fact = fact * i; }
System.out.println("Factorial of " + n + " is " + fact);
}
```

```
F:\Java>java Fact
Enter the number
5
Factorial of 5 is 120
```

WAP to display the pattern

```
import java.util.*;
class StarPattern {
public static void main(String[] args) {
int n,i,j;
Scanner sc=new Scanner(System.in);
System.out.println("Enter the number");
n=sc.nextInt();
for(i=1;i<=n;i++)
{
    for(j=1;j<=i;j++)
    {
        System.out.print("* ");
    }
    System.out.println();
}
}
```

```
F:\Java>javac StarPattern.java
F:\Java>java StarPattern
Enter the number
6
*
* *
* * *
* * * *
* * * * *
* * * * *
```

Arrays (single dimensional)

- ▶ an array is a collection of multiple data of same datatype
- ▶ Indexing in an array starts from 0. The last index is n-1 where n is the size of the array.
- ▶ Syntax for declaring array : datatype array_name[]={datatype[array_size]};
- ▶ Eg : int arr[]={int[10]};

Arrays (single dimensional)

WAP to input and display n elements in an array

```
import java.util.*;
class Array{
public static void main(String args[])
{
int i,n;
Scanner sc=new Scanner(System.in);
System.out.println("Enter no. of elements :");
n=sc.nextInt();
int a[]=new int[n];
for(i=0;i<=n-1;i++)
{
System.out.println("Enter a number :");
a[i]=sc.nextInt();
}
```

```
System.out.println("The elements in the array are :");
for(i=0;i<=n-1;i++)
{
System.out.print(a[i]+"\t");
} }}
```

```
F:\Java>javac Array.java
F:\Java>java Array
Enter no. of elements :
5
Enter a number :
1
Enter a number :
6
Enter a number :
7
Enter a number :
5
Enter a number :
4
The elements in the array are :
1       6       7       5       4
F:\Java>
```

Arrays (single dimensional)

WAP to sort an array in ascending order

```
import java.util.*;
class SortedArray{
public static void main(String args[])
{
int i,n,temp;
Scanner sc=new Scanner(System.in);
System.out.println("Enter no. of elements : ");
n=sc.nextInt();
int a[]=new int[n];
for(i=0;i<=n-1;i++)
{
System.out.println("Enter a number : ");
a[i]=sc.nextInt();
}
for(i=0;i<=n-2;i++)
{
    for(int j=0;j<=n-2;j++)
    {
        if(a[j]>a[j+1])
        {
            temp=a[j];
            a[j]=a[j+1];
            a[j+1]=temp;
        }
    }
}
System.out.println("Sorted Array: ");
for(i=0;i<=n-1;i++)
{
System.out.print(a[i]+"\t");
}
}
```

```
F:\Java>javac SortedArray.java
F:\Java>java SortedArray
Enter no. of elements :
5
Enter a number :
4
Enter a number :
9
Enter a number :
5
Enter a number :
2
Enter a number :
7
Sorted Array:
2        4        5        7
F:\Java>
```

Arrays (multi dimensional)

- ▶ The Java multidimensional arrays are arranged as an array of arrays i.e. each element of a multi-dimensional array is another array.
- ▶ The representation of the elements is in rows and columns.
- ▶ Thus, you can get a total number of elements in a multidimensional array by multiplying row size with column size.
- ▶ So if you have a two-dimensional array of 3×4 , then the total number of elements in this array = $3 \times 4 = 12$
- ▶ The simplest of the multi-dimensional array is a two-dimensional array.
- ▶ A simple definition of 2D arrays is: A 2D array is an array of one-dimensional arrays.
- ▶ In Java, a two-dimensional array is stored in the form of rows and columns and is represented in the form of a matrix.
- ▶ The general declaration of a two-dimensional array is,
- ▶ `data_type[][] array_name = new data_type[row_size][column_size];`
- ▶ Eg : `int a[][]=new int[3][4];`

Arrays (multi dimensional)

```
import java.util.*;
class Array2d
{
    public static void main(String[] args) {
        int i,j,m,n;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter no. of rows and columns :");
        m=sc.nextInt();
        n=sc.nextInt();
        int a[][] = new int[m][n];
        for (i=0;i<=m-1; i++) {
            for (j=0;j<=n-1; j++) {
                System.out.println("Enter a number :");
                a[i][j]=sc.nextInt();
            }
        }
    }
}
```

```
System.out.println("The Array in matrix form is:");
for (i=0;i<=m-1;i++) {
    for (j=0;j<=n-1;j++) {
        System.out.print(a[i][j]+"\t");
    }
}
System.out.println();
} } }
```

```
F:\Java>javac Array2d.java
F:\Java>java Array2d
Enter no. of rows and columns :
3
2
Enter a number :
1
Enter a number :
2
Enter a number :
3
Enter a number :
4
Enter a number :
5
Enter a number :
6
The Array in matrix form is:
1      2
3      4
5      6
```

Arrays (multi dimensional)

WAP to find and display sum of the diagonal elements of a square matrix

```
import java.util.*;
class Diagonal
{
    public static void main(String[] args) {
        int i,j,m,n,sum=0;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter no. of rows and columns :");
        m=sc.nextInt();
        n=sc.nextInt();
        int a[][] = new int[m][n];
        for (i=0;i<=m-1; i++) {
            for (j=0;j<=n-1; j++) {
                System.out.println("Enter a number :");
                a[i][j]=sc.nextInt();
            }
        }
    }
}
```

```
for (i=0;i<=m-1;i++) {
    for (j=0;j<=n-1;j++) {
        if(i==j)
            sum=sum+a[i][j];
    }
}
System.out.println("Sum= "+sum);
} }
```

```
F:\Java>javac SumofDiagonalElements.java
F:\Java>java Diagonal
Enter no. of rows and columns :
2
2
Enter a number :
4
Enter a number :
3
Enter a number :
9
Enter a number :
1
Sum= 5
```

Programming Practice Questions

- ▶ WAP to accept a number and display its equivalent ASCII character using type casting.
 - ▶ WAP to find largest of three numbers accepted from the user through command line using ternary operator.
 - ▶ WAP to display first n odd numbers
 - ▶ WAP to calculate and display sum of square of first n natural numbers.
 - ▶ WAP to display first n elements of Fibonacci series.
 - ▶ WAP to calculate value of following series – $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n}$
 - ▶ WAP to calculate value of following series – $1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \dots + \frac{1}{n!}$
 - ▶ WAP to calculate and display value of s for $t=1,5,10,15,20,\dots,100$. Formula : $s = s_0 + v_0 + \frac{1}{2} a t^2$
 - ▶ WAP to display following patterns

*
**

* * * * *

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

```

      1
    1 2 1
  1 2 3 2 1
1 2 3 4 3 2 1

```

A
B C
D E F
G H I J
K L M N O

*

1
2 2
3 3 3
4 4 4 4
5 5 5 5 5

Programming Practice Questions

- ▶ WAP to count number of digits in a user entered number using while loop.
- ▶ WAP to display a user entered number as a sequence of individual digits using while loop. Eg : 1691 should be displayed as 1 6 9 1
- ▶ WAP to check if a user entered number is divisible by 3 and 5 or not.
- ▶ WAP to check if the entered number is Armstrong number or not.
- ▶ WAP to display the class according to the marks scored by student using if else if. The marks scored is taken input from the user and the class is displayed based on the range given below

| Marks | Class |
|--------|-------------|
| 70-100 | Distinction |
| 60-69 | First |
| 50-59 | Second |
| 40-49 | Pass |
| 0-39 | Fail |

- ▶ WAP to display a user digit number in words using switch case. Eg : i/p : 123 o/p : One Two Three
- ▶ WAP to find GCD and LCM of user entered two numbers.
- ▶ WAP to find largest element from the array.

Programming Practice Questions

- ▶ WAP to display average of n integers stored in an array
- ▶ WAP to search an element in an array and if found display the index.
- ▶ WAP to sort an array.
- ▶ WAP to add two matrices of size m x n
- ▶ WAP to find transpose of a matrix of size m x n
- ▶ The annual examination results of 5 students are tabulated as follows

| Roll No | Subject 1 | Subject 2 | Subject 3 |
|---------|-----------|-----------|-----------|
| 1 | | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |

WAP to read the data and determine the following

- i) Total marks obtained by each student
- ii) The highest total marks and the student who obtained highest total marks.

JAVA PROGRAMMING

Chap 2 : Classes and Methods

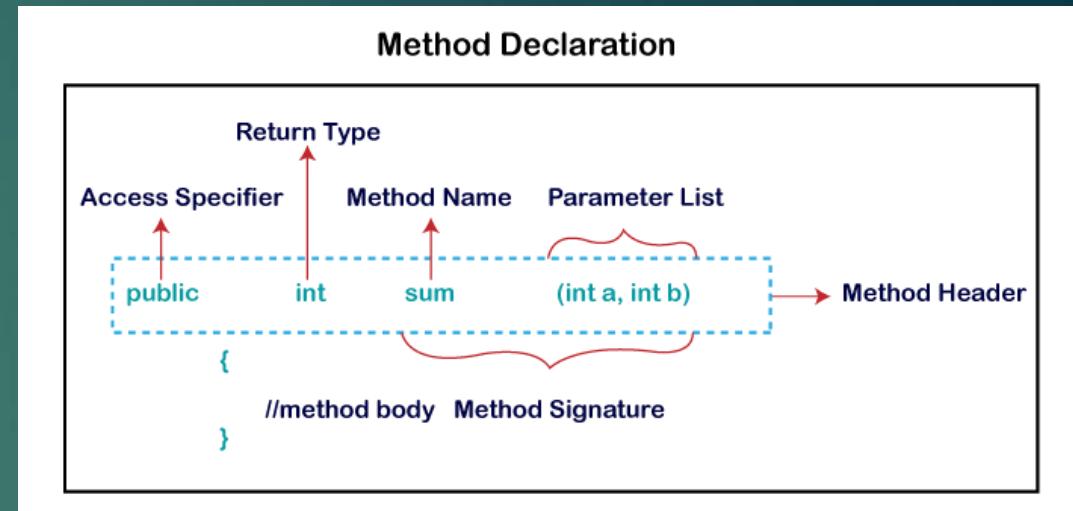
- ▶ In object-oriented programming technique, we design a program using objects and classes.
- ▶ An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only.
- ▶ Method in Java is a collection of instructions that performs a specific task.
- ▶ It provides the reusability of code. We can also easily modify code using methods.

Methods in Java

- ▶ A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.
- ▶ It is used to achieve the **reusability** of code.
- ▶ We write a method once and use it many times.
- ▶ We do not require to write code again and again.
- ▶ It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code.
- ▶ The method is executed only when we call or invoke it.
- ▶ The most important method in Java is the **main()** method.
- ▶ The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments.

Methods in Java

- ▶ It has six components that are known as **method header**, as we have shown in the following figure
- ▶ **Method Signature:** Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.
- ▶ **Access Specifier:** Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **five** types of access specifier
- ▶ **Return Type:** Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.
- ▶ **Method Name:** It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. A method is invoked by its name.
- ▶ **Parameter List:** It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, we leave the parentheses blank. The parameters whose values are passed to the method are called as **actual parameters** and the variables in which the parameter values are received are called as **formal parameters**. **Number of formal parameters should always be equal to number of actual parameters.**
- ▶ **Method Body:** It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.



Methods in Java (Static Method Examples)

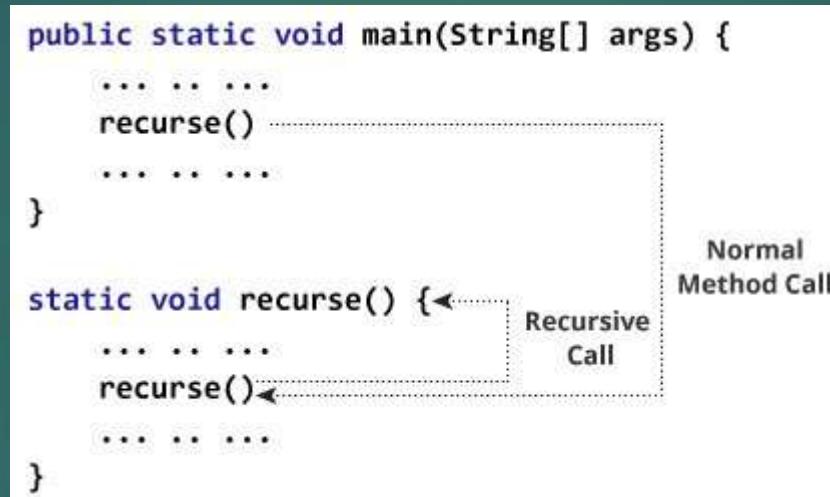
```
import java.util.*;
class Add{
    public static void main(String args[])
    {
        int a,b,sum;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter two nos.");
        a=sc.nextInt();
        b=sc.nextInt();
        sum=add(a,b);
        System.out.println("The Sum is :" +sum);
    }
    public static int add(int x, int y)
    {
        return(x+y);
    }
}
```

```
import java.util.*;
class Add {
    public static void main(String args[])
    {
        int a,b,sum;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter two nos.");
        a=sc.nextInt();
        b=sc.nextInt();
        add(a,b);
    }
    public static void add(int x, int y)
    {
        System.out.println("The Sum is :" +(x+y));
    }
}
```

```
F:\Java>javac Add.java
F:\Java>java Add
Enter two nos.
5
8
Sum is : 13
```

Recursive Methods

- ▶ a method that calls itself is known as a recursive method.
- ▶ And, this process is known as recursion.



- ▶ In order to stop the recursive call, we need to provide some conditions inside the method. Otherwise, the method will be called infinitely.
- ▶ Hence, we use the if...else statement(or similar approach) to terminate the recursive call inside the method.

Recursive Methods

```
import java.util.*;  
  
class Factorial{  
  
public static void main(String[] args){  
  
int n, f;  
Scanner sc=new Scanner(System.in);  
  
System.out.println("Enter the  
number");  
  
n=sc.nextInt();  
  
f=fact(n);  
  
System.out.println("factorial of "+n+" is  
:"+f);  
}  
  
public static int fact(int x)  
{  
  
if(x==1)  
{  
}
```

```
        return 1;  
  
else  
  
return(x*fact(x-1));  
}  
}
```

```
F:\Java>java Factorial  
Enter the number  
5  
factorial of 5 is : 120  
  
F:\Java>
```

Java Access Specifiers

| Access Location \ Access Modifier | Public | Protected (subclass and pkg) | Default (friendly) (in same pkg) | Private | Private Protected (class & its sub class) |
|-----------------------------------|--------|------------------------------|----------------------------------|---------|---|
| Same Class | Yes | Yes | Yes | Yes | Yes |
| Sub class in same package | Yes | Yes | Yes | No | Yes |
| Other classes in same package | Yes | Yes | Yes | No | No |
| Subclass in other packages | Yes | Yes | No | No | Yes |
| Non-subclasses in other packages | Yes | No | No | No | No |

Java Class

- ▶ A class is a group of objects which have common properties.
- ▶ It is a template or blueprint from which objects are created.
- ▶ It is a logical entity. It can't be physical.
- ▶ A class in Java can contain:
 - ▶ **Fields**
 - ▶ **Methods**
 - ▶ **Constructors**
 - ▶ **Blocks**
 - ▶ **Nested class and interface**
- ▶ Syntax to declare a class:

```
class <class_name>{  
    field;  
    method;  
}
```

Java Objects

- ▶ An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc.
- ▶ It can be physical or logical (tangible and intangible)
- ▶ An object has three characteristics:
 - ▶ **State:** represents the data (value) of an object.
 - ▶ **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
 - ▶ **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.
- ▶ **An object is an instance of a class.** A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.
- ▶ **Object Definitions:**
 - ▶ An object is a *real-world entity*.
 - ▶ An object is a *runtime entity*.
 - ▶ The object is an *entity which has state and behavior*.
 - ▶ The object is an *instance of a class*.

Java Objects

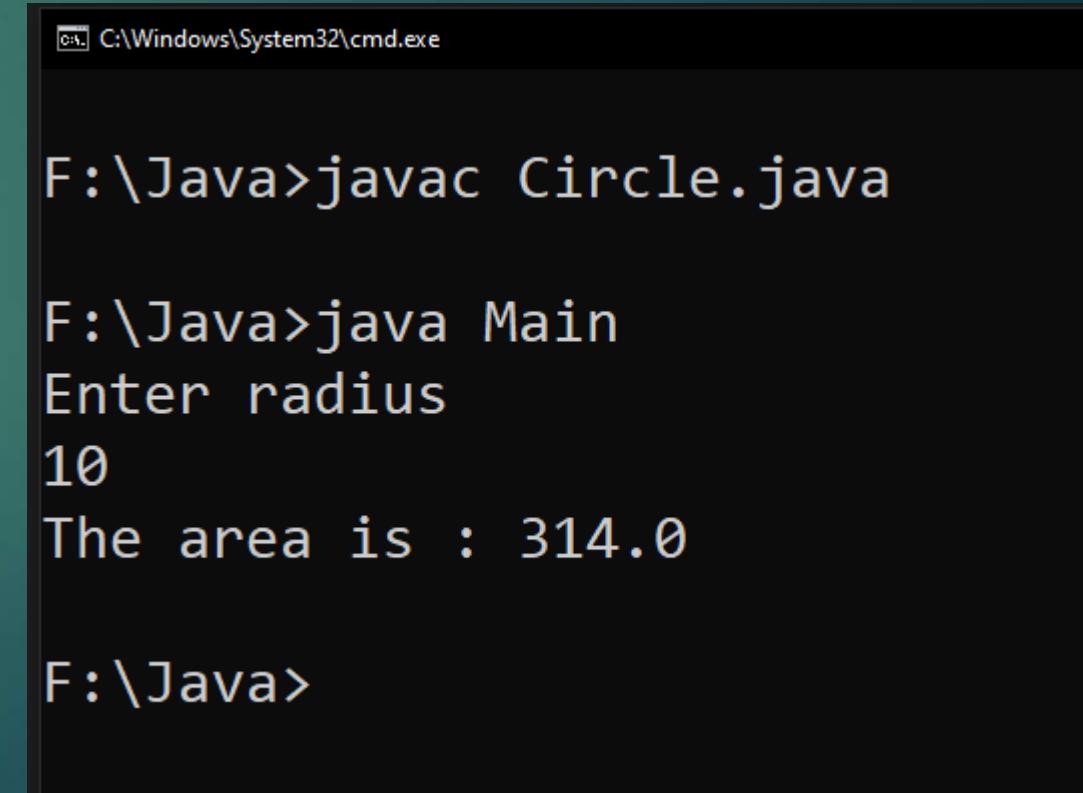
- ▶ We can make objects of the class and memory space will be allocated to the fields of that object.
- ▶ The methods of a class can be accessed using the objects using the dot(.) operator.
- ▶ Syntax of making an object of a class is –

```
class_name object_name = new class_name(parameters to be passed)
```

Eg : WAP to create a class Circle. It should have three methods namely accept radius, calculate area and display the area.

```
import java.util.*;  
  
class Circle{  
    private float r,area;  
    public void accept(float x)  
    {  
        r=x;  
    }  
    public void calculate()  
    {  
        area=3.14f*r*r;  
    }  
    public void display()  
    {  
        System.out.println("The area is :  
"+area);  
    }  
}
```

```
class Main{  
    public static void main(String args[])  
    {  
        float rad;  
        Scanner sc=new Scanner(System.in);  
        System.out.println("Enter radius");  
        rad=sc.nextFloat();  
        Circle c=new Circle();  
        c.accept(rad);  
        c.calculate();  
        c.display();  
    }  
}
```



```
C:\Windows\System32\cmd.exe  
F:\Java>javac Circle.java  
F:\Java>java Main  
Enter radius  
10  
The area is : 314.0  
F:\Java>
```

Constructor

- ▶ A constructor in Java is a **special method** that is used to initialize objects.
- ▶ The constructor is called when an object of a class is created.
- ▶ It can be used to set initial values for object attributes
- ▶ At the time of calling constructor, memory for the object is allocated
- ▶ Every time an object is created using the new() keyword, at least one constructor is called.
- ▶ It calls a default constructor if there is no constructor available in the class.
- ▶ In such case, Java compiler provides a default constructor by default.
- ▶ There are two types of constructors in Java: no-arg(i.e. default) constructor, and parameterized constructor.
- ▶ It is called constructor because it constructs the values at the time of object creation.
- ▶ It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Constructor

- ▶ Rules for creating constructors :
 - ▶ Constructor should always be public/default/protected.
 - ▶ Name of the constructor should always be same as that of the class
 - ▶ Constructor should not have a return type, **not even void**.
 - ▶ Constructor is automatically called when an object of the class is created
 - ▶ There can be more than one constructor having same name but different parameter list. This is known as **Constructor Overloading**
- ▶ There are the following reasons to use constructors:
 - ▶ We use constructors to initialize the object with the default or initial state. The default values for primitives may not be what you are looking for.
 - ▶ Another reason to use constructor is that it informs about dependencies. In other words, using the constructor, we can request the user of that class for required dependencies.
 - ▶ We can find out what it needs in order to use this class, just by looking at the constructor.
- ▶ In short, we use the constructor to **initialize the instance variable of the class**.

Default Constructor Example

```
import java.util.*;  
class CircleDefConst{  
    private float r,area;  
    CircleDefConst()  
{  
        Scanner sc=new  
        Scanner(System.in);  
        System.out.println("Enter Radius");  
        r=sc.nextFloat();  
    }  
    public void calculate()  
{  
        area=3.14f*r*r;  
    }  
    public void display()  
{  
        System.out.println("The area is :  
        "+area);  
    } }
```

```
class Main{  
    public static void main(String args[])  
{  
        // Default Constructor  
        CircleDefConst c=new CircleDefConst();  
        c.calculate();  
        c.display();  
    } }
```

```
F:\Java>java Main  
Enter radius  
10  
The area is : 314.0  
F:\Java>
```

Parameterized Constructor Example

```
import java.util.*;  
  
class Circle{  
    private float r,area;  
    Circle(float x)  
    {  
        r=x;  
    }  
    void calculate()  
    {  
        area=3.14f*r*r;  
    }  
    void display()  
    {  
        System.out.println("The area is : "+area);  
    }  
}  
  
class Main{  
    public static void main(String args[])  
    {  
        float rad;  
        Scanner sc=new Scanner(System.in);  
        System.out.println("Enter radius");  
        rad=sc.nextFloat();  
        // parameterized constructor  
        Circle c=new Circle(rad);  
        c.calculate();  
        c.display();  
    }  
}
```

```
F:\Java>java Main  
Enter radius  
10  
The area is : 314.0  
  
F:\Java>
```

Constructor Overloading

- ▶ Multiple constructors of same class with different parameters is called **constructor overloading**
- ▶ Eg : Program on next slide (slide 18)

```
import java.util.Scanner;  
class Student  
{  
    private String name;  
    private int roll_no;  
    //parameterized constructor  
    Student(String n, int rn)  
    {  
        name=n;  
        roll_no=rn;  
    }  
    //Default constructor  
    Student()  
    {  
        name = "John";  
        roll_no = 1;  
    }  
}
```

```
//method for printing the values  
  
void show()  
{  
    System.out.println("Name of the student:  
    "+name);  
    System.out.println("Roll No of the student:  
    "+roll_no);  
}  
}  
  
class Main{  
    public static void main(String args[])  
    {  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Enter the name of the  
        student: ");  
        String fname = sc.nextLine();  
        System.out.println("Enter the roll no of the  
        student: ");  
        int rno = sc.nextInt();  
    }  
}
```

//Calling the parameterized constructor

```
System.out.println("calling Show() method  
for the parameterized constructor: ");  
Student s=new Student(fname, rno);  
s.show();
```

//Calling the default constructor

```
System.out.println("Show() method for the  
default constructor: ");  
Student s1=new Student();  
s1.show();  
} }
```

```
F:\Java>javac Student.java
```

```
F:\Java>java Main  
Enter the name of the student:  
abc  
Enter the roll no of the student:  
5  
calling Show() method for the parameterized constructor:  
Name of the student: abc  
Roll No of the student: 5  
Show() method for the default constructor:  
Name of the student: John  
Roll No of the student: 1
```

```
F:\Java>
```

Method Overloading

- ▶ Multiple methods with same name in same class or in base class and derived class with different parameters is called **method overloading**
- ▶ Eg1 : WAP to demonstrate Method Overloading by overloading the methods for calculating Area of circle, rectangle and triangle

```
class MethodOverloading{  
    private float a;  
    public void area(float r)  
    {  
        a=3.14f*r*r;  
        System.out.println("Area of circle is :" + a);  
    }  
    public void area(float l, float b)  
    {  
        a=l*b;  
        System.out.println("Area of rectangle is :" + a);  
    }  
    public void area(float s1, float s2, float s3)  
    {  
        float s;  
        s=0.5f*(s1+s2+s3);  
        s=s*(s-s1)*(s-s2)*(s-s3);  
        a=(float)(Math.sqrt(s));  
        System.out.println("Area of triangle is :" + a);  
    }  
}
```

```
class Main {  
    public static void main(String args[]){  
        MethodOverloading obj=new MethodOverloading();  
        obj.area(10);  
        obj.area(10,10);  
        obj.area(10,10,10);  
    }  
}
```

```
F:\Java>javac MethodOverloading.java  
  
F:\Java>java Main  
Area of circle is :314.0  
Area of rectangle is :100.0  
Area of triangle is :43.30127  
  
F:\Java>
```

Method Overloading

- ▶ Eg2 : WAP to demonstrate Method Overloading in a base and derived class by overloading the methods for displaying the value of variable contained in the class.

```
class Parent{  
    public void display(int x)  
    {  
        System.out.println("Value of x is :" + x);  
    }  
}  
  
class Child extends Parent{  
    public void display(int x, int y)  
    {  
        System.out.println("Value of x is :" + x + "\nValue of y is :" + y);  
    }  
}  
  
class Main {  
    public static void main(String args[]){  
        Child c=new Child();  
        c.display(10);  
        c.display(5,5);  
    }  
}
```

```
F:\Java>javac MethodOverloading.java  
  
F:\Java>java Main  
Value of x is :10  
Value of x is :5  
Value of y is :5  
  
F:\Java>
```

Array of Objects

- ▶ Array of objects is created to store information about multiple objects having similar functionality.
- ▶ Syntax : class_name object_name[] = new class_name[size of the array]
- ▶ Eg : Student s[]=new Student[10];
- ▶ Note: memory is not allocated to each object with the above declaration. We need to use “new” keyword with every object of the array separately for allocating memory.
- ▶ Problem : WAP to create an array of objects of a class student. The class should have field members name, id, total marks and marks in three subjects namely Physics, Chemistry and Maths. Accept information of ‘n’ students and display them in tabular form.

```
import java.util.Scanner;
class Student
{
    private String name;
    private int id,p,c,m,t;
    void accept()
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter name, ID, Marks in Physics,
Chemistry and Maths : ");
        name=sc.nextLine();
        id=sc.nextInt();
        p=sc.nextInt();
        c=sc.nextInt();
        m=sc.nextInt();
        t=p+c+m;
    }
    void display()
    {
        System.out.println(name+"\t"+id+"\t"+p+"\t"+c+"\t"+m+"\t"
+t);
    }
}
```

```
class Main{
    public static void main(String args[])
    {
        int i,n;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter total number of students: ");
        n = sc.nextInt();

//creating array of objects of size n

        Student s[] = new Student[n];

        for(i=0;i<=n-1;i++)
        {
            System.out.println("For student "+(i+1));
            s[i]=new Student();
            s[i].accept();
        }
        System.out.println("Name\tID\tPhy\tChem\tMaths\tTotal");
        for(i=0;i<=n-1;i++)
        {
            s[i].display();
        }
    }
}
```

C:\Windows\System32\cmd.exe

F:\Java>javac Array_of_objects.java

F:\Java>java Main

Enter total number of students:

2

For student 1

Enter name, ID, Marks in Physics, Chemistry and Maths :

John

1

50

60

40

For student 2

Enter name, ID, Marks in Physics, Chemistry and Maths :

Mary

2

50

55

70

| Name | ID | Phy | Chem | Maths | Total |
|------|----|-----|------|-------|-------|
| John | 1 | 50 | 60 | 40 | 150 |
| Mary | 2 | 50 | 55 | 70 | 175 |

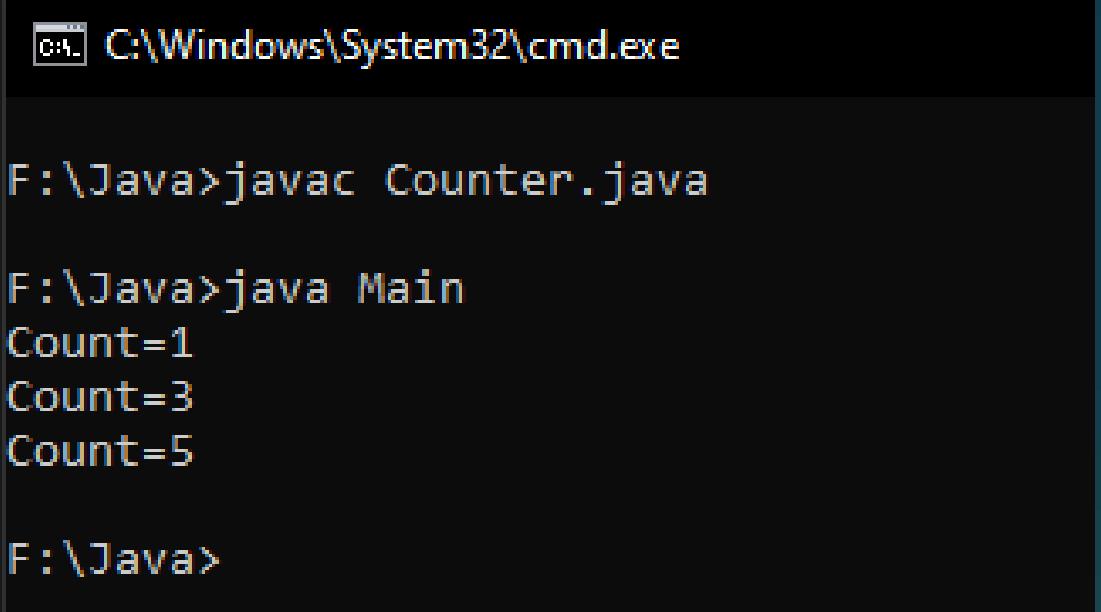
F:\Java>

Static Class Members

- ▶ A method or a field can be static.
- ▶ **static** keyword is used to declare class members as static
- ▶ A static method is a method that works on a class and not on the object of the class.
- ▶ To call a static method we **need not** make object of that class
- ▶ A static method is called by syntax : `class_name.method_name()`
- ▶ A static field member or a static variable will be common for all the objects of that class.
- ▶ A common memory location is allocated for a static variable for all objects made of that class
- ▶ One of the major advantage of using static variable is that we can use it to count number of objects made of the class. Since each object will access the same memory location for static variable, we can increment its value in the constructor of class.
- ▶ Whenever the object is created, this constructor will be called and this static variable will be incremented.

```
import java.util.Scanner;
class Counter
{
private static int count;
Counter()
{
count++;
}
static void display()
{
System.out.println("Count="+count);
}
}

class Main{
public static void main(String args[])
{
Counter c1=new Counter();
Counter.display();
Counter c2=new Counter();
Counter c3=new Counter();
Counter.display();
Counter c4=new Counter();
Counter c5=new Counter();
Counter.display();
}
}
```



A screenshot of a Windows Command Prompt window titled "C:\Windows\System32\cmd.exe". The window shows the following text:
F:\Java>javac Counter.java

F:\Java>java Main
Count=1
Count=3
Count=5

F:\Java>

“this” keyword

- ▶ **this** keyword refers to the current object.
- ▶ **this** can also be used to:
 - ▶ Invoke current class constructor
 - ▶ Invoke current class method
 - ▶ Return the current class object
 - ▶ Pass an argument in the method call
 - ▶ Pass an argument in the constructor call

Usage of Java this Keyword

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

- | | | | |
|----|--|----|--|
| 01 | this can be used to refer current class instance variable. | 04 | this can be passed as an argument in the method call. |
| 02 | this can be used to invoke current class method (implicity) | 05 | this can be passed as argument in the constructor call. |
| 03 | this() can be used to invoke current class Constructor. | 06 | this can be used to return the current class instance from the method |

Eg1 : Without this keyword

```
class Student1{  
int rollno;  
String name;  
float fee;  
Student1(int rollno,String name,float fee){  
rollno=rollno;  
name=name;  
fee=fee;  
}  
void display(){System.out.println(rollno+"<"+name+"<"+fee);}  
}  
class Main{  
public static void main(String args[]){  
Student1 s1=new Student1(111,"ankit",5000f);  
Student1 s2=new Student1(112,"sumit",6000f);  
s1.display();  
s2.display();  
}}
```

```
F:\Java>javac Student1.java
```

```
F:\Java>java Main  
0 null 0.0  
0 null 0.0
```

```
F:\Java>
```

Eg2 : With this keyword

```
class Student1{  
int rollno;  
String name;  
float fee;  
Student1(int rollno,String name,float fee){  
this.rollno=rollno;  
this.name=name;  
this.fee=fee;  
}  
void display(){System.out.println(rollno+"<"+name+"<"+fee);}  
}  
class Main {  
public static void main(String args[]){  
Student1 s1=new Student1(111,"ankit",5000f);  
Student1 s2=new Student1(112,"sumit",6000f);  
s1.display();  
s2.display();  
}}
```

```
F:\Java>javac Student1.java
```

```
F:\Java>java Main  
111 ankit 5000.0  
112 sumit 6000.0
```

```
F:\Java>
```

In the above eg1, parameters (formal arguments) and instance variables are same. So, we are using this keyword in eg2 to distinguish local variable and instance variable.

Programming Practice Questions

- ▶ Create a class Employee with data members `emp_id`, `name`, `designation` and `salary`. Write methods `get_details()` to take employee details from the user and `show_grade()` to display grade of employees based on salary range as given below and `show_details()` to display employee details.

| Salary Range | Grade |
|--------------|-------|
| <10000 | D |
| 10000-24999 | C |
| 25000-49999 | B |
| >50000 | A |

- ▶ Create a class student with appropriate data members and methods to store and display Roll No, Name and marks stored by students in Physics, Chemistry and Maths.
- ▶ WAP to demonstrate Method Overloading by overloading the method for calculating sum. Consider taking different parameter list in terms of no. of parameters and type of parameters.
- ▶ WAP to demonstrate Constructor Overloading by overloading the constructor for taking radius input for calculating area of a circle.
- ▶ WAP to implement a class Stack using array with two methods push and pop to add and remove elements from the stack. Addition of elements should not be allowed if the stack is full and popping should not be allowed if the stack is empty.

JAVA PROGRAMMING

Chap 3 : Inheritance

- ▶ Inheritance is an important pillar of OOP(Object-Oriented Programming).
- ▶ It is the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class.
- ▶ In Java, inheritance means creating new classes based on existing ones.
- ▶ A class that inherits from another class can reuse the methods and fields of that class.
- ▶ In addition, you can add new fields and methods to your current class as well.
- ▶ Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.
- ▶ Important terminologies used in Inheritance:
 - ▶ **Class:** Class is a set of objects which shares common characteristics/ behavior and common properties/ attributes. Class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
 - ▶ **Super Class/Parent Class/Base Class:** The class whose features are inherited is known as a superclass(or a base class or a parent class).
 - ▶ **Sub Class/Child Class/Derived Class:** The class that inherits the other class is known as a subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
 - ▶ **Reusability:** Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

- ▶ Why do we need Inheritance in Java?
 - ▶ **Code Reusability:** The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class code.
 - ▶ **Method Overriding:** Method Overriding is achievable only through Inheritance. It is one of the ways by which java achieves Run Time Polymorphism.
 - ▶ **Abstraction:** The concept of abstract where we do not have to provide all details is achieved through inheritance. Abstraction only shows the functionality to the user.
- ▶ **Syntax for inheritance**
 - ▶ The **extends keyword** is used for inheritance in java. Using the extends keyword indicates you are derived from an existing class. In other words, “extends” refers to increased functionality.
 - ▶ Syntax : `class derived_class extends base_class`

```
{  
    //methods and fields  
}
```
- ▶ Types of Inheritance :
 - ▶ Single Inheritance
 - ▶ Multilevel Inheritance
 - ▶ Hierarchical Inheritance

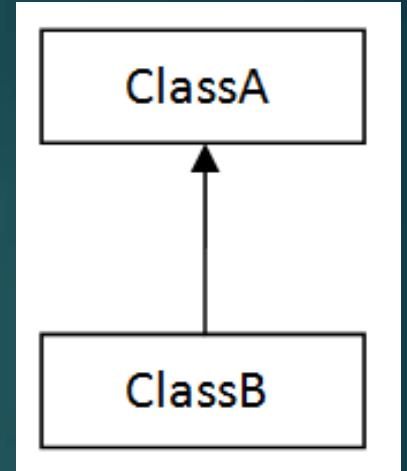
Single Inheritance

Here only one class is derived from another class.

```
import java.util.*;
class Data {          // Base class Data
    float r;
    public void read(float x)
    {
        r=x;
    }
}
class Area extends Data // creating child
class Area
{
    private float a;
    public void calculate()
    {
        a=3.14f*r*r;
    }
    public void display()
    {
        System.out.println("Area is :" +a);
    }
}
```

```
class Main
{
    public static void main(String args[])
    {
        float rad;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter Radius : ");
        rad=sc.nextFloat();
        Area ar=new Area();
        ar.read(rad);
        ar.calculate();
        ar.display();
    }
}
```

```
Enter Radius :
10
Area is : 314.0
```



Multilevel Inheritance

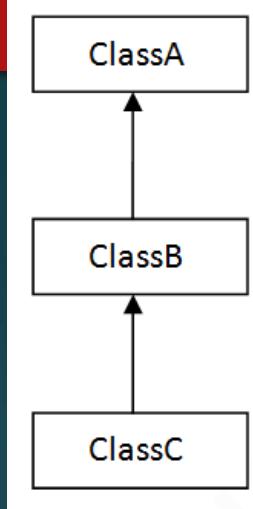
Here one class is derived from a class which is in turn derived from another class.

```
import java.util.*;  
class Data { // Base class Data  
    float r;  
    public void read(float x)  
{  
    r=x;  
}  
  
class Area extends Data // creating  
child class Area of parent class Data  
{  
    protected float a;  
    public void calculate()  
{  
    a=3.14f*r*r;  
}  
    public void display()  
{  
    System.out.println("Area is :" +a);  
}}
```

```
class Volume extends Area // creating child  
class Volume of parent class Area  
{  
    private float vol;  
    public void compute()  
{  
    vol=a*r*4/3;  
}  
    public void show()  
{  
    System.out.println("Volume is :" +vol);  
}  
  
class Main  
{  
    public static void main(String args[])  
{  
        float rad;  
        Scanner sc=new Scanner(System.in);  
        System.out.println("Enter Radius :");  
        rad=sc.nextFloat();
```

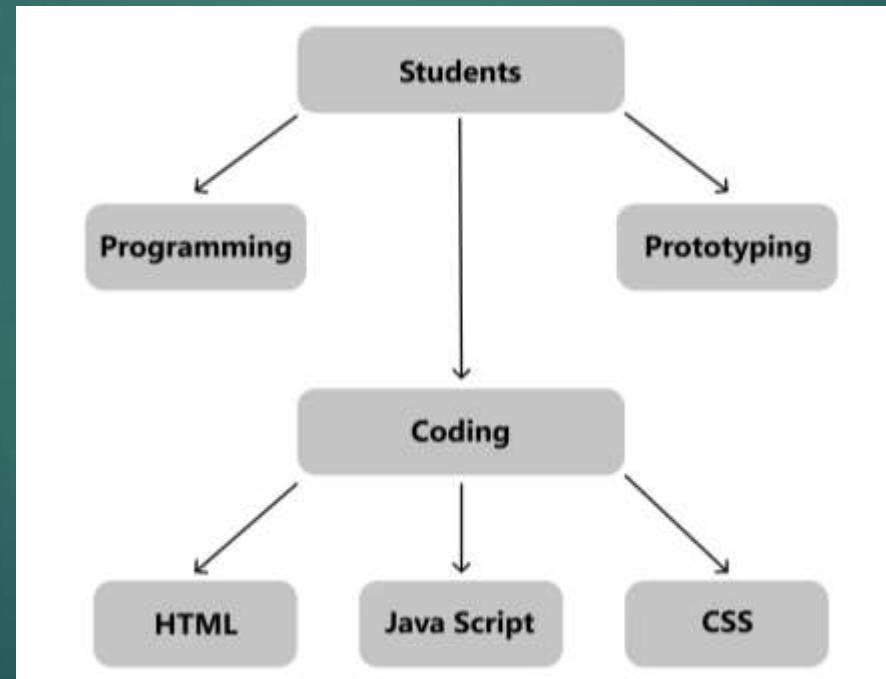
```
Volume v=new Volume();  
v.read(rad);  
v.calculate();  
v.display();  
v.compute();  
v.show();  
}
```

```
Enter Radius :  
10  
Area is : 314.0  
Volume is : 4186.6665
```



Hierarchical Inheritance

- ▶ When multiple classes are derived from a class and further more classes are derived from these derived classes
- ▶ one class serves as a superclass (base class) for more than one subclass
- ▶ Example from codes



Hierarchical Inheritance

// parent class

```
class Employee {  
    double salary = 50000;  
  
    void displaySalary() {  
        System.out.println("Employee Salary:  
Rs." + salary);  
    }  
}
```

// child class 1

```
class FullTimeEmployee extends  
Employee{  
    double hike = 0.50;  
  
    void incrementSalary() {  
        salary = salary + (salary * hike);  
        System.out.println("Full Time Employee  
Salary after increment : " + salary);  
    }  
}
```

// child class 2

```
class InternEmployee extends Employee{  
    double hike = 0.25;  
  
    void incrementSalary() {  
        salary = salary + (salary * hike);  
        System.out.println("Intern Salary after increment : " + salary);  
    }  
}
```

class Main {

```
public static void main(String[] args) {  
    // object created  
    FullTimeEmployee emp1 = new FullTimeEmployee();  
    InternEmployee emp2 = new InternEmployee();  
    System.out.println("Salary of a full-time employee before  
incrementing:");  
    emp1.displaySalary();  
    emp1.incrementSalary(); // salary incremented of Full time Employee  
    System.out.println("Salary of an intern before incrementing:");  
    emp2.displaySalary();  
    emp2.incrementSalary(); // salary incremented of Intern  
}
```

Hierarchical Inheritance

```
Salary of a full-time employee before incrementing:  
Employee Salary: Rs.50000.0  
Full Time Employee Salary after increement : 75000.0  
Salary of an intern before incrementing:  
Employee Salary: Rs.50000.0  
Intern Salary after increement : 62500.0
```

Method Overriding

- ▶ If a base class and derived class have a method with same name but different parameter list, then it is called as method overloading.
- ▶ But, **If a base class and derived class have a method with same name and same parameter list, then it is called as method overriding.**
- ▶ Usage of Java Method Overriding
 - ▶ Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
 - ▶ Method overriding is used for runtime polymorphism
- ▶ Rules for Java Method Overriding
 - ▶ The method must have the same name as in the parent class
 - ▶ The method must have the same parameter as in the parent class.
 - ▶ There must be an IS-A relationship (inheritance).

Method Overriding

```
class Bank{  
int getRateOfInterest()  
{  
return 0;  
}}
```

```
class SBI extends Bank{  
int getRateOfInterest()  
{  
return 8;  
}}
```

```
class ICICI extends Bank{  
int getRateOfInterest()  
{  
return 7;  
}}
```

```
class AXIS extends Bank{  
int getRateOfInterest()  
{  
return 9;  
}}
```

```
class Main{  
public static void main(String args[]){  
SBI s=new SBI();  
ICICI i=new ICICI();  
AXIS a=new AXIS();  
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());  
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());  
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());  
}}
```

```
SBI Rate of Interest: 8  
ICICI Rate of Interest: 7  
AXIS Rate of Interest: 9
```

Final class and “final” keyword

- ▶ The final keyword can be preceded to any member of a class or to the class itself.
- ▶ Making anything final has following implications –
 - ▶ If the field member is declared final then the variable value cannot be changed i.e. it becomes constant.
 - ▶ If a method is declared as final then that method cannot be overridden
 - ▶ If a class is declared as final then that class cannot have any sub class i.e. no class can be derived from a final class

Final class and “final” keyword

- ▶ **Final variable :** If you make any variable as final, you cannot change the value of final variable(It will be constant).
- ▶ Example of final variable : There is a final variable speedlimit, we are going to change the value of this variable, but it can't be changed because final variable once assigned a value can never be changed.

```
class Bike{  
    final int speedlimit=90; //final variable  
    void run(){  
        speedlimit=400;      // changing value of speedlimit  
    }  
}  
  
class Main{  
    public static void main(String args[]){  
        Bike b=new Bike();  
        b.run();  
    }  
}
```

```
Main.java:4: error: cannot assign a value to final variable speedlimit  
    speedlimit=400;      // changing value of speedlimit  
               ^  
1 error
```

Final class and “final” keyword

- **Final method :** If you make any method as final, you cannot override it.

```
class Bike{  
    final void run()  
{  
        System.out.println("running");  
    }  
}
```

```
class Honda extends Bike{  
    void run()          // overriding final method run()  
    {  
        System.out.println("running safely with 100kmph");  
    }  
}
```

```
class Main{  
    public static void main(String args[]){  
        Honda h= new Honda();  
        h.run();  
    }  
}
```

```
Main.java:10: error: run() in Honda cannot override run() in Bike  
        void run()          // overriding final method run()  
                  ^  
    overridden method is final  
1 error
```

Final class and “final” keyword

- **Final class** : If you make any class as final, you cannot extend it.

```
final class Bike{  
}  
  
class Honda extends Bike{      // inheriting from final class bike  
    void run()  
    {  
        System.out.println("running safely with 100kmph");  
    }  
}  
  
class Main{  
    public static void main(String args[]){  
        Honda h= new Honda();  
        h.run();  
    }  
}
```

Main.java:4: error: cannot inherit from final Bike
class Honda extends Bike{
^
1 error

Java Abstract class and method

- ▶ Abstract classes are used to declare common characteristics of subclasses.
- ▶ Abstract classes are declared with keyword **abstract** followed by class definition. They provide template for subclasses.
- ▶ **No object can be made of abstract class. It can be used as a base class for other classes that are derived from the abstract class.**
- ▶ An abstract class can contain fields and methods. It can have abstract and non-abstract methods (method with the body).
- ▶ **Methods of abstract class that has only declaration and no definition are known as Abstract Methods.**
- ▶ **Abstract methods must be overridden.**
- ▶ If a class has any abstract method, the class becomes abstract and must be declared as abstract.

Rules for Java Abstract class



Java Abstract class and method

```
import java.util.*;
abstract class Abst {      //abstract class
    protected float r,vol;
    public void read(float x) // non-abstract method
    {
        r=x;
    }
    public abstract void calculate(); // abstract method
    public void display()
    {
        System.out.println("Volume = "+vol);
    }
}

class Sphere extends Abst {
    public void calculate() // overriding abstract
method
    {
        vol=3.14f*r*r*r*4/3;
    }
}

class Hemisphere extends Abst {
    public void calculate() // overriding abstract method
    {
        vol=3.14f*r*r*r*2/3;
    }
}
```

```
class Main
{
    public static void main(String args[])
    {
        float rad;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter Radius : ");
        rad=sc.nextFloat();
        Sphere s=new Sphere();
        s.read(rad);
        s.calculate();
        System.out.println("For Sphere : ");
        s.display();
        Hemisphere h=new Hemisphere();
        h.read(rad);
        h.calculate();
        System.out.println("For Hemisphere : ");
        h.display();
    }
}
```

```
Enter Radius :
10
For Sphere :
Volume = 4186.6665
For Hemisphere :
Volume = 2093.3333
```

The **super** keyword

- ▶ The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.
- ▶ If you want to access a member of a base class from the derived class, then the **super** keyword is used.
- ▶ This is especially to access the constructors and methods of base class.
- ▶ Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Super Keyword

1

Super can be used to refer immediate parent class instance variable.

2

Super can be used to invoke immediate parent class method.

3

`super()` can be used to invoke immediate parent class constructor.

The super keyword

```
class Parent{  
public Parent(int a)  
{  
    System.out.println("Inside the  
constructor of Parent class : "+a);  
}  
  
public void display(int x)  
{  
    System.out.println("x = "+x);  
}
```

```
class Child extends Parent{  
public Child(int a)  
{  
    super(a);  
    System.out.println("Inside the  
constructor of Child class : "+a);  
}  
  
public void display(int y)  
{  
    super.display(y);  
    System.out.println("y = "+y);  
}}
```

```
class Main{  
public static void main(String args[])  
{  
    Child c = new Child(3);  
    c.display(5);  
}}
```

```
Inside the constructor of Parent class : 3  
Inside the constructor of Child class : 3  
x = 3  
y = 5
```

Programming Practice Questions

Design a class hierarchy for a library system. Create a base class *Book* with attributes *title* and *author*, and a constructor that initializes these attributes. Then, create a derived class *Ebook* that extends *Book* with additional attributes *file_size* and *format* and *Printed book* that extends *Book* with attributes *edition* and *no of copies sold*. Ensure that the constructor of *Ebook* and *Printed book* properly initializes attributes from both the base class and the derived class. (use *super* keyword)

Create an abstract class *Appliance* with abstract methods *switch_on()* and *switch_off()*. Derive two classes *WashingMachine* and *Refrigerator* from *Appliance* and implement the abstract methods with specific behaviors for each appliance. Ensure that the abstract class cannot be instantiated directly and that the derived classes provide concrete implementations for the abstract methods.

Design a class hierarchy for a university system. Create a base class *Person* with attributes *name* and *age*, and methods like *getDetails()* and *display_details()*. Then, create derived classes *Student* and *Professor* that inherit from *Person* and add specific attributes and methods. For example, *Student* might have a *studentID* and *major* field, while *Professor* might have a *facultyID*, *salary* and *department*. Override *getDetails()* and *display details()* methods to input and display all the details of student and professor.

Implement a class hierarchy for a banking system. Create a base class *Account* with attributes *accountNumber* and *balance*, a constructor to initialize these attributes and methods *deposit()*, *withdraw()* and *getBalance()*. Create a derived class *SavingsAccount* that adds an *interestRate* attribute and provides its own constructor that initializes all attributes. Also calculate the interest and add it to the balance using *deposit* method().

Programming Practice Questions

- ▶ You are tasked with designing a system for a university that includes different types of people: *Professor*, *Student*, and *Staff*. All these types of people share some common attributes but also have their unique attributes. You need to use *hierarchical inheritance* to model this system, ensuring that constructors are properly used to initialize the base and derived classes. Create a Base Class *Person* having Fields *name* and *id* and a constructor to initialize them. It will also have *getdetails()* and *display()* methods to input and display the details. Create Derived Classes *Professor*, *Student* and *Staff* that inherits from *Person*. *Professor* has an additional field *department*. Initialize *name*, *id*, and *department* using the base class constructor. Class *Student* has additional field *major* and *gpa*. Initialize *name*, *id*, *major*, and *gpa* using the base class constructor. Class *Staff* has additional field *designation*. Initialize *name*, *id*, and *position* using the base class constructor. Create instances of *Professor*, *Student*, and *Staff* classes and print their details using *display()* method.
- ▶ Design a class Hierarchy for a zoo. Create a base class *Animal* which has data members *Name* and *Species*. It has methods *getHabitat()* to return default habitat description and *display()* to display the details of the animal. Class *Mammal* is derived from *Animal* which has field *hasFur* of type *Boolean*. A class *Lion* inherits class *Mammal* and has data member *endangered* of type *Boolean*. Override *display()* and *getHabitat()* methods to display animal details and Habitat description of *Mammal* as well as *Lion*. Use appropriate constructors to initialize all the data members.
- ▶ Design an abstract class *Character* for a game to serve as the base class for various types of characters. Each character type has specific actions but shares common functionality such as name, attacking and defending. This class accepts the *character name* through constructor and has abstract methods *attack()* and *defend()* that defines how the character attacks or defends. Class *Warrior* and *Mage* inherits class *Character* and overrides *attack()* method to display what weapon is used by the character to attack (eg: sword, fireball, axe, javelin, etc.) and *defend()* method to display the technique used by the character to defend (eg: uses shield, uses magic spell, counter attack, etc.).

JAVA PROGRAMMING

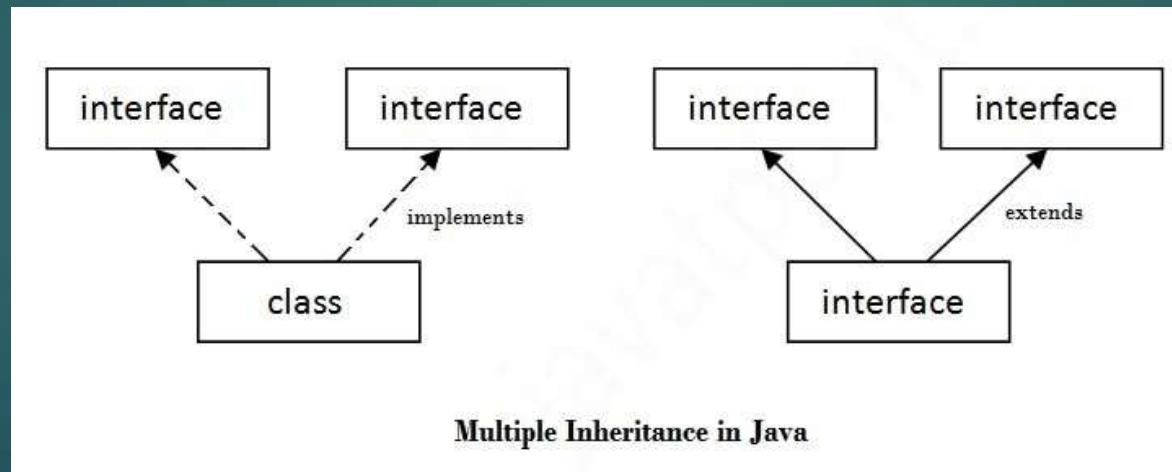
Chap 4 : Packages and
Interfaces

Interfaces

- ▶ Multiple Inheritance is not allowed in Java.
- ▶ Instead Interface is used to implement multiple inheritance.
- ▶ An **interface** in Java is a blueprint of a class. It has static constants and abstract methods.
- ▶ There can be only abstract methods in the Java interface, having no method body.
- ▶ It is used to achieve abstraction and multiple inheritance in Java.
- ▶ Java Interface also represents the IS-A relationship (i.e. Inheritance)
- ▶ **Syntax for inheritance**
- ▶ An interface is declared by using the interface keyword.
- ▶ It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
- ▶ A class that implements an interface must implement all the methods declared in the interface.

```
Syntax : interface <interface_name>
{
    // declare constant fields
    // declare methods that are abstract by default.
}
```

- ▶ a class extends another class, an interface extends another interface, but a class implements an interface.
- ▶ Interface do not have constructor.
- ▶ If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract and method definitions may be provided in the subclass.
- ▶ Any class can extend only 1 class but can implement infinite number of interface.



Abstract Class vs Interface

| Abstract Class | Interface |
|--|--|
| It is a class that contains one or more abstract methods, which are to be defined by sub class | It contains only method declarations and no definitions |
| Abstract class definition begins with keyword "abstract" followed by class definition | Interface definition begins with keyword "interface" followed by interface definition |
| Abstract classes can have general methods defined along with some methods with only declarations | Interfaces have all methods with only declarations that are defined in subclasses i.e. classes that implements the interface |
| Variables in abstract classes need not be public, static and final | All Variables in an interface is by default public, static and final |
| Abstract classes doesn't support multiple inheritance | Interfaces supports multiple inheritance |
| An abstract class can contain private and protected members | An interface can only have public members |
| Abstract classes are fast | Interfaces are slow as it requires extra indirection to find corresponding methods in the actual class |

Interface Example

```
import java.util.*;
interface Base {
    public void read(float x);
    public void calculate();
    public void display();
}
class Sphere implements Base {
    protected float r,vol;
    public void read(float x)
    {
        r=x;
    }
    public void calculate()
    {
        vol=3.14f*r*r*r*4/3;
    }
    public void display()
    {
        System.out.println("Volume of
Sphere = "+vol);
    }
}
```

```
class Hemisphere implements Base {
    protected float r,vol;
    public void read(float x)
    {
        r=x;
    }
    public void calculate()
    {
        vol=3.14f*r*r*r*2/3;
    }
    public void display()
    {
        System.out.println("Volume of
Hemisphere = "+vol);
    }
}
```

```
class Main
{
    public static void main(String args[])
    {
        float rad;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter Radius : ");
        rad=sc.nextFloat();
        Sphere s=new Sphere();
        s.read(rad);
        s.calculate();
        s.display();
        Hemisphere h=new Hemisphere();
        h.read(rad);
        h.calculate();
        h.display();
    }
}
```

```
Enter Radius :
10
Volume of Sphere = 4186.6665
Volume of Hemisphere = 2093.3333
```

Packages

- ▶ Package is a way to organize code i.e. similar function or related classes must be in a package.
- ▶ Thus, one package will have all the classes of similar functionality while another package for another type of classes.
- ▶ A **java package** is a group of similar types of classes, interfaces and sub-packages.
- ▶ Package in java can be categorized in two form, built-in package and user-defined package.
- ▶ There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
- ▶ Advantage of Java Package
 - ▶ Java package is used to categorize the classes and interfaces so that they can be easily maintained.
 - ▶ Java package provides access protection.
 - ▶ Java package removes naming collision.
 - ▶ Making searching/locating and usage of classes, interfaces, enumerations and annotations easier
- ▶ All we need to do is put related classes into packages. After that, we can simply write an import class from existing packages and use it in our program.
- ▶ We can reuse existing classes from the packages as many time as we need it in our program.

Creating a Package

Step-1 : To create a package we use the keyword “package” followed by the name of the package

Step-2 : Create a class and write members of the class

Step-3 : Store the program file in a folder with the same name as that of the package and store the file as the name of the class that has main() method. Eg : if the package name is “world”, then the folder in which the program is stored should also be “world”. If the name of the class having the main() method is “Hello”, then the program should be stored as “Hello.java”

Step-4 : Compile the program as usual using javac. This has to be done in the folder of the package. In case of above eg, in the folder “world” we compile the file as javac Hello.java

Step-5 : Traverse out of the package folder and execute the program with syntax –

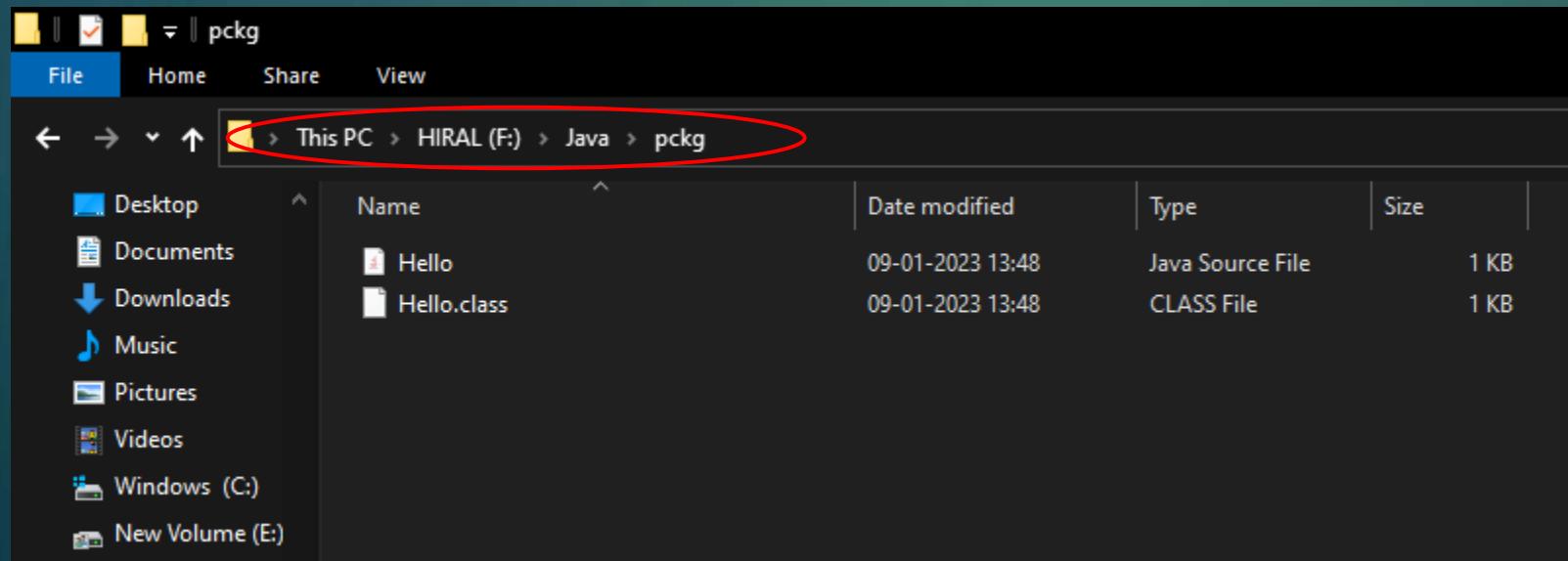
```
java package_name.class_name.
```

In the above eg, java world.Hello

Creating a Package

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

```
F:\Java\pckg>javac Hello.java  
  
F:\Java\pckg>cd..  
  
F:\Java>java pckg.Hello  
Hello World !!!  
  
F:\Java>
```

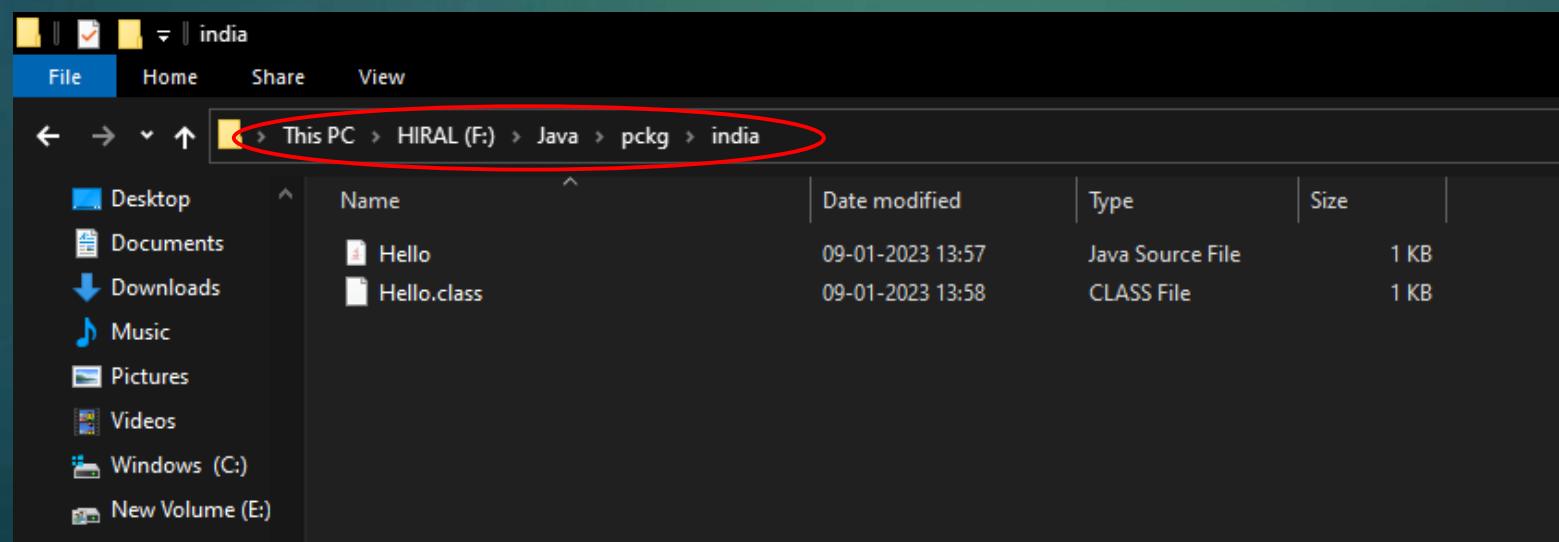


Creating a Sub-Package

- To create a sub-package we use the same process that we used to create the package.
- The folder of the sub package name has to be stored inside the folder with the package name.

```
package pckg.india;  
class Hello  
{  
    public static void main(String args[])  
    {  
        System.out.println("Hello Indians !!!");  
    }  
}
```

```
F:\Java\pckg\india>javac Hello.java  
F:\Java\pckg\india>cd..  
F:\Java\pckg>cd..  
F:\Java>java pckg.india.Hello  
Hello Indians !!!  
F:\Java>
```



Importing a Package

- ▶ We have imported many built in packages like java.io, java.util,etc. as of now.
- ▶ To import self made packages we follow the following steps –
- ▶ Create the package as described in previous slides.
- ▶ This package is to be imported by another program outside the folder with package name
- ▶ That is we write two programs, one the package and another that imports the package. The package is to be stored in the folder with same name.

Importing a Package

The Package (Hello1.java)

```
package pckg.india;
public class Hello1
{
    public void display()
    {
        System.out.println("Hello Indians !!!");
    }
}
```

Importing Package (packimp.java)

```
import pckg.india.*;
class packimp
{
    public static void main(String args[])
    {
        Hello1 h=new Hello1();
        h.display();
    }
}
```

```
F:\Java>cd pckg
F:\Java\pckg>cd india
F:\Java\pckg\india>javac Hello1.java
F:\Java\pckg\india>cd..
F:\Java\pckg>cd..
F:\Java>javac packimp.java
F:\Java>java packimp
Hello Indians !!!
```

Exploring Important Java Packages

- ▶ Java.lang
- ▶ Java.io
- ▶ Java.util

Wrapper Classes

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

Programming Practice Questions

- ▶ Design an interface Shape with methods `calculateArea()` and `displayArea()`. Implement this interface in two classes, Circle and Rectangle. The Circle class should have a constructor that takes the radius, and the Rectangle class should have a constructor that takes width and height. Write a program that creates instances of Circle and Rectangle, and calculates and prints their areas.
- ▶ Create an interface `SortStrategy` with a method `void sort(int[] array)`. Implement this interface in classes, `BubbleSort`, `QuickSort` and `MergeSort`. Each class should provide its own implementation of the sorting algorithm. Write a program that uses different sorting strategies to sort an array of integers.
- ▶ Design an interface named `TemperatureConverter` with a method `convertToCelsius()` to convert temperatures to Celsius. Implement this converter in classes `FahrenheitToCelsiusConverter` and `KelvinToCelsiusConverter` to convert the temperature in different units to Celsius.
- ▶ Define an interface Shape with methods `getArea` and `getPerimeter`. Define another interface Volume with a method `getVolume` to calculate Volume of the shape. Implement classes Cube and Sphere that implements both Shape and Volume interfaces, allowing them to calculate their area, perimeter and Volume.

Programming Practice Questions

- ▶ Create a basic Java application with three packages: employee, department and company. In employee package, create a class Employee with fields name, id, and department, and methods to get and print these fields. In department package, create a class Department with fields departmentName and location, and methods to get and print these fields. Write a main method in a class located in company Package that creates instances of Employee and Department, sets their fields, and prints their details.
- ▶ Create a package hierarchy for a university system with the following structure: university, university.students, and university.courses. In university package, create a base class UniversityMember with common fields like name and id, and methods for displaying member information. In university.students, create classes Student and TeachingAdjunct that extends UniversityMember. Class Student includes additional fields for major and gpa and overrides method to display student details. Class TeachingAdjunct includes fields for no. of hours worked and stipend per hour and method to calculate total stipend alongwith displaying TeachingAdjunct Details by overriding method of base class. In university.courses, create a class Course with fields for courseName and courseCode, and methods to print course details. Write a class in university.main that demonstrates creating instances of Student and Course, and shows how they interact within the university system.

JAVA PROGRAMMING

Chap 5 : String Handling

- ▶ A string is a sequence of characters surrounded by double quotations.
- ▶ In a java programming language, a string is the object of a built-in class String.
- ▶ In the background, the string values are organized as an array of a character data type.
- ▶ **The string created using a character array** can not be extended.
- ▶ **It does not allow to append more characters after its definition, but it can be modified.**
- ▶ The Java String is immutable by default which means it cannot be changed.
- ▶ Whenever we change any string, a new instance is created.
- ▶ For mutable strings, you can use StringBuffer and StringBuilder classes.
- ▶ Syntax :

```
char[] name = {'J', 'a', 'v', 'a', ' ', 'T', 'u', 't', 'o', 'r', 'i', 'a', 'l', 's'};  
// name[14] = '@'; //ArrayIndexOutOfBoundsException when trying to add more characters to the string  
System.out.println(name);  
name[4] = '-'; // modifying the string  
System.out.println(name);
```

Java Tutorials
Java-Tutorials

- ▶ The String class defined in the package `java.lang` package.
- ▶ The String class implements `Serializable`, `Comparable`, and `CharSequence` interfaces.
- ▶ **Java String** class provides a lot of methods to perform operations on strings such as `compare()`, `concat()`, `equals()`, `split()`, `length()`, `replace()`, `compareTo()`, `intern()`, `substring()` etc.
- ▶ The string created using the String class can be extended.
- ▶ It allows us to add more characters after its definition, and also it can be modified.
- ▶ Syntax :

```
String name = "Java Tutorials";
System.out.println(name);
name = "Java Programming";
System.out.println(name);
```

Java Tutorials
Java Programming

Creating String object in Java

- ▶ In java, we can use the following two ways to create a string object.
 - ▶ Using string literal
 - ▶ Using String constructor
- ▶ Syntax :

```
String title = "Java Tutorials"; // Using literals
```

```
String name = new String("Java Programming"); // Using constructor
```

```
System.out.println(title);
```

```
System.out.println(name);
```

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

```
String s=new String(ch); //converting char array to string using constructor
```

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

Java Tutorials
Java Programming
Strings

Why are String objects immutable?

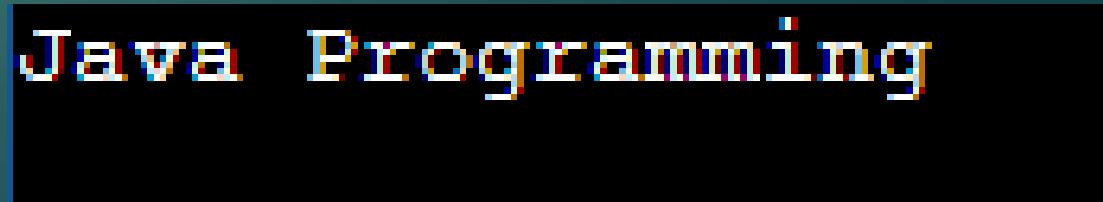
- ▶ A String is an unavoidable type of variable while writing any application program.
- ▶ String references are used to store various attributes like username, password, etc.
- ▶ In Java, String objects are immutable. Immutable simply means unmodifiable or unchangeable.
- ▶ Once String object is created its data or state can't be changed but a new String object is created.
- ▶ Let's try to understand the concept of immutability by the example given below:

```
String s="Java";  
s.concat(" Programming"); //concat() method appends the string at the end  
System.out.println(s); //will print Java because strings are immutable objects
```



- ▶ Now it can be understood that Java is not changed but a new object is created with Java Programming and s reference variable still refers to "Java" and not to "Java Programming". That is why String is known as immutable.
- ▶ But if we explicitly assign it to the reference variable, it will refer to "Java Programming" object. Please notice that still "Java" object is not modified, only that the reference variable refers to the new object.

```
String s="Java";  
s=s.concat(" Programming");  
System.out.println(s);
```



String Handling Methods

| Method | Description | Return Value |
|-----------------------------|--|--------------|
| charAt(int) | Finds the character at given index | char |
| length() | Finds the length of given string | int |
| compareTo(String) | Compares two strings | int |
| compareTolgnoreCase(String) | Compares two strings, ignoring case | int |
| concat(String) | Concatenates the object string with argument string. | String |
| contains(String) | Checks whether a string contains sub-string | boolean |
| contentEquals(String) | Checks whether two strings are same | boolean |
| equals(String) | Checks whether two strings are same | boolean |
| equalsIgnoreCase(String) | Checks whether two strings are same, ignoring case | boolean |
| startsWith(String) | Checks whether a string starts with the specified string | boolean |
| endsWith(String) | Checks whether a string ends with the specified string | boolean |
| getBytes() | Converts string value to bytes | byte[] |

String Handling Methods

| | | |
|----------------------------|--|----------|
| hashCode() | Finds the hash code of a string | int |
| indexOf(String) | Finds the first index of argument string in object string | int |
| lastIndexOf(String) | Finds the last index of argument string in object string | int |
| isEmpty() | Checks whether a string is empty or not | boolean |
| replace(String, String) | Replaces the first string with second string | String |
| replaceAll(String, String) | Replaces the first string with second string at all occurrences. | String |
| substring(int, int) | Extracts a sub-string from specified start and end index values | String |
| toLowerCase() | Converts a string to lower case letters | String |
| toUpperCase() | Converts a string to upper case letters | String |
| trim() | Removes whitespace from both ends | String |
| toString(int) | Converts the value to a String object | String |
| split(String) | splits the string matching argument string | String[] |
| intern() | returns string from the pool | String |
| join(String, String, ...) | Joins all strings, first string as delimiter. | String |

String Handling Methods

```
public class Main {  
    public static void main(String[] args) {  
        String title = "Java Tutorials";  
        String Name = "www.javaprogramming.com";  
        System.out.println("Length of title: " + title.length());  
        System.out.println("Char at index 3: " + title.charAt(3));  
        System.out.println("Index of 'T': " + title.indexOf('T'));  
        System.out.println("Last index of 'a': " +  
            title.lastIndexOf('a'));  
        System.out.println("Empty: " + title.isEmpty());  
        System.out.println("Ends with '.com': " +  
            Name.endsWith(".com"));  
        System.out.println("Equals: " + Name.equals(title));  
        System.out.println("Sub-string: " + Name.substring(9, 14));  
        System.out.println("Upper case: " + Name.toUpperCase());  
        System.out.println("Upper case: " + title.toLowerCase());  
        String s=" Java ";  
        System.out.println("Before Trimming :" + s);  
        System.out.println("After trimming :" + s.trim());  
        System.out.println("Replacing a by z in " + title + " we get "  
            + title.replace("a","z"));  
    }  
}
```

```
Length of title: 14  
Char at index 3: a  
Index of 'T': 5  
Last index of 'a': 11  
Empty: false  
Ends with '.com': true  
Equals: false  
Sub-string: rogra  
Upper case: WWW.JAVAPROGRAMMING.COM  
Upper case: java tutorials  
Before Trimming : Java  
After trimming :Java  
Replacing a by z in Java Tutorials we get Jzvz Tutorizls
```

String Compare

- ▶ We can compare String in Java on the basis of content and reference.
- ▶ It is used in authentication (by equals() method), sorting (by compareTo() method), reference matching (by == operator) etc.
- ▶ There are three ways to compare String in Java:
 - ▶ By Using equals() Method
 - ▶ By Using == Operator
 - ▶ By compareTo() Method
- ▶ The String class **equals() method** compares the original **content** of the string. It compares values of string for equality. String class provides the following two methods:
 - ▶ public boolean equals(Object another) compares this string to the specified object.
 - ▶ public boolean equalsIgnoreCase(String another) compares this string to another string, ignoring case.

```
Eg 1: String s1="Java";
String s2="Java";
String s3=new String("Java");
String s4="Python";
System.out.println(s1.equals(s2));      // o/p will be true
System.out.println(s1.equals(s3));      // o/p will be true
System.out.println(s1.equals(s4));      // o/p will be false
```

```
Eg 2: String s1="Java";
String s2="JAVA";
System.out.println(s1.equals(s2)); //false
System.out.println(s1.equalsIgnoreCase(s2));
//true
```

String Compare

- The **== operator** compares **references** not values.

```
Eg 1: String s1="Java";
String s2="Java";
String s3=new String("Java");
System.out.println(s1==s2); // o/p will be true (because both refer to same instance)
System.out.println(s1==s3); // o/p will be false(because s3 refers to another instance created)
```

- The String class **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 objects. If:
 - s1 == s2 : The method returns 0.
 - s1 > s2 : The method returns a positive value.
 - s1 < s2 : The method returns a negative value.

```
Eg : String s1="Java";
String s2="Java";
String s3="Jp";
System.out.println(s1.compareTo(s2)); //0
System.out.println(s1.compareTo(s3)); // -15 (because s1<s3 and "a" is 15 times lower than "p")
System.out.println(s3.compareTo(s1)); // 15 (because s3 > s1 and "p" is 15 times greater than "a" )
```

String Concatenation

- ▶ In Java, String concatenation forms a new String that is the combination of multiple strings. There are two ways to concatenate strings in Java:
 - ▶ By + (String concatenation) operator
 - ▶ By concat() method

Eg 1:

```
String s="Java"+" Programming";
String s1=50+30+" JP "+40+40;
System.out.println(s); // Java Programming
System.out.println(s1); //80 JP 4040 Note: After a string literal, all the + will be treated as string concatenation
operator.
String s2=" by Oracle";
String s3=s.concat(s2);
System.out.println(s3); // Java Programming by Oracle
```

StringBuffer Class

- ▶ Java StringBuffer class is used to create mutable (modifiable) String objects.
- ▶ A String that can be modified or changed is known as mutable String. StringBuffer and StringBuilder classes are used for creating mutable strings.
- ▶ The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.
- ▶ Important Constructors of StringBuffer Class
 - ▶ StringBuffer() : It creates an empty String buffer with the initial capacity of 16. If the number of the character increases from its current capacity, it increases the capacity by $(oldcapacity*2)+2$.
 - ▶ StringBuffer(String str) : It creates a String buffer with the specified string.
 - ▶ StringBuffer(int capacity) : It creates an empty String buffer with the specified capacity as length.

► Important methods of StringBuffer class

| Method | Description |
|---|--|
| append(String s) | It is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc. |
| insert(int offset, String s) | It is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc. |
| replace(int startIndex, int endIndex, String str) | It is used to replace the string from specified startIndex and endIndex. |
| delete(int startIndex, int endIndex) | It is used to delete the string from specified startIndex and endIndex. |
| reverse() | is used to reverse the string. |
| capacity() | It is used to return the current capacity. |
| ensureCapacity(int minimumCapacity) | It is used to ensure the capacity at least equal to the given minimum. |
| charAt(int index) | It is used to return the character at the specified position. |
| length() | It is used to return the length of the string i.e. total number of characters. |
| substring(int beginIndex) | It is used to return the substring from the specified beginIndex. |
| substring(int beginIndex, int endIndex) | It is used to return the substring from the specified beginIndex and endIndex. |

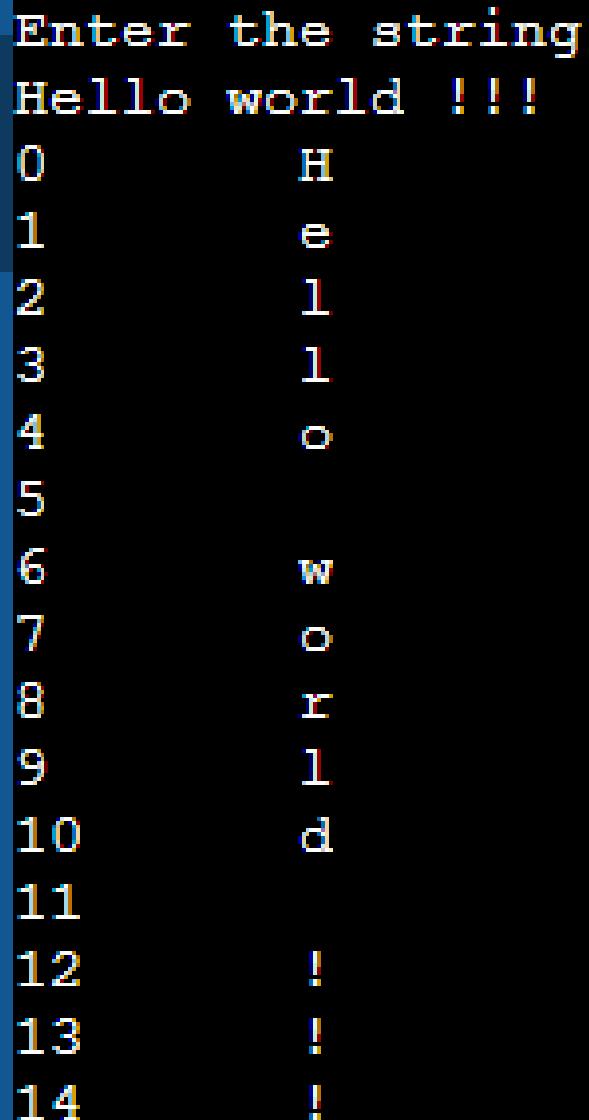
StringBuffer Class Example

```
public class Main {  
    public static void main(String[] args) {  
        StringBuffer sb=new StringBuffer("Java");  
        System.out.println(sb.append(" Tutorials")); //original string Java is changed to Java Tutorials  
        System.out.println(sb.insert(5, "Programming ")); // inserts the given String in original string at the given position  
        System.out.println(sb.replace(0,4, "HTML")); // replaces the given String from the specified beginIndex and  
        endIndex.  
        System.out.println(sb.delete(17,26)); //deletes the String from the specified beginIndex to endIndex.  
        System.out.println(sb.reverse()); // reverses the String  
    }  
}
```

```
Java Tutorials  
Java Programming Tutorials  
HTML Programming Tutorials  
HTML Programming  
gnimmargorP LMTH
```

- WAP to convert a string into an array of characters and display each character with its index

```
import java.util.*;
public class Main{
public static void main(String[] args) {
String str;
int i,n;
Scanner sc=new Scanner(System.in);
System.out.println("Enter the string");
str=sc.nextLine();
n=str.length();
char c[]=new char[n];
c=str.toCharArray();
for(i=0;i<=n-1;i++)
{
System.out.println(i+"\t"+c[i]);
}
}
}
```



```
Enter the string
Hello world !!!
0      H
1      e
2      l
3      l
4      o
5
6      w
7      o
8      r
9      l
10     d
11
12     !
13     !
14     !
```

► WAP to count number of vowels, consonants, digits and blank spaces in a string

```
import java.util.*;
public class Main{
public static void main(String[] args) {
String str;
int i,n,v=0,co=0,d=0,bs=0;
Scanner sc=new Scanner(System.in);
System.out.println("Enter the string");
str=sc.nextLine();
n=str.length();
char c[]=new char[n];
c=str.toCharArray();
for(i=0;i<=n-1;i++)
{
if(c[i]=='a' || c[i]=='e' || c[i]=='i' || c[i]=='o' || c[i]=='u' || c[i]=='A' ||
c[i]=='E' || c[i]=='I' || c[i]=='O' || c[i]=='U')
v++;
else if(c[i]==' ')
bs++;
else if(c[i]>='0' && c[i]<='9')
d++;
else if((c[i]>='a' && c[i]<='z') || (c[i]>='A' && c[i]<='Z'))
co++;
}
System.out.println("No. of vowels : " + v + "\nNo. of Consonants : " +
co + "\nNo. of digits : " + d + "\nNo of blank spaces : " + bs);
}}
```

```
Enter the string
Hello how r u 123
No. of vowels : 4
No. of Consonants : 6
No. of digits : 3
No of blank spaces : 4
```

Programming Practice Questions

- ▶ WAP in Java to perform the following String Operations.
 1. Create a string instance using String and String Buffer class each.
 2. Check the length and capacity of String and String Buffer objects
 3. Reverse the contents of a string and convert the resultant string in Upper Case.
 4. Append the second string to above resultant string.
 5. Extract a substring from resultant string.
- ▶ WAP in Java to implement run-length encoding.
- ▶ Write a Java method to extract all digits from a given string and return them as a new string.

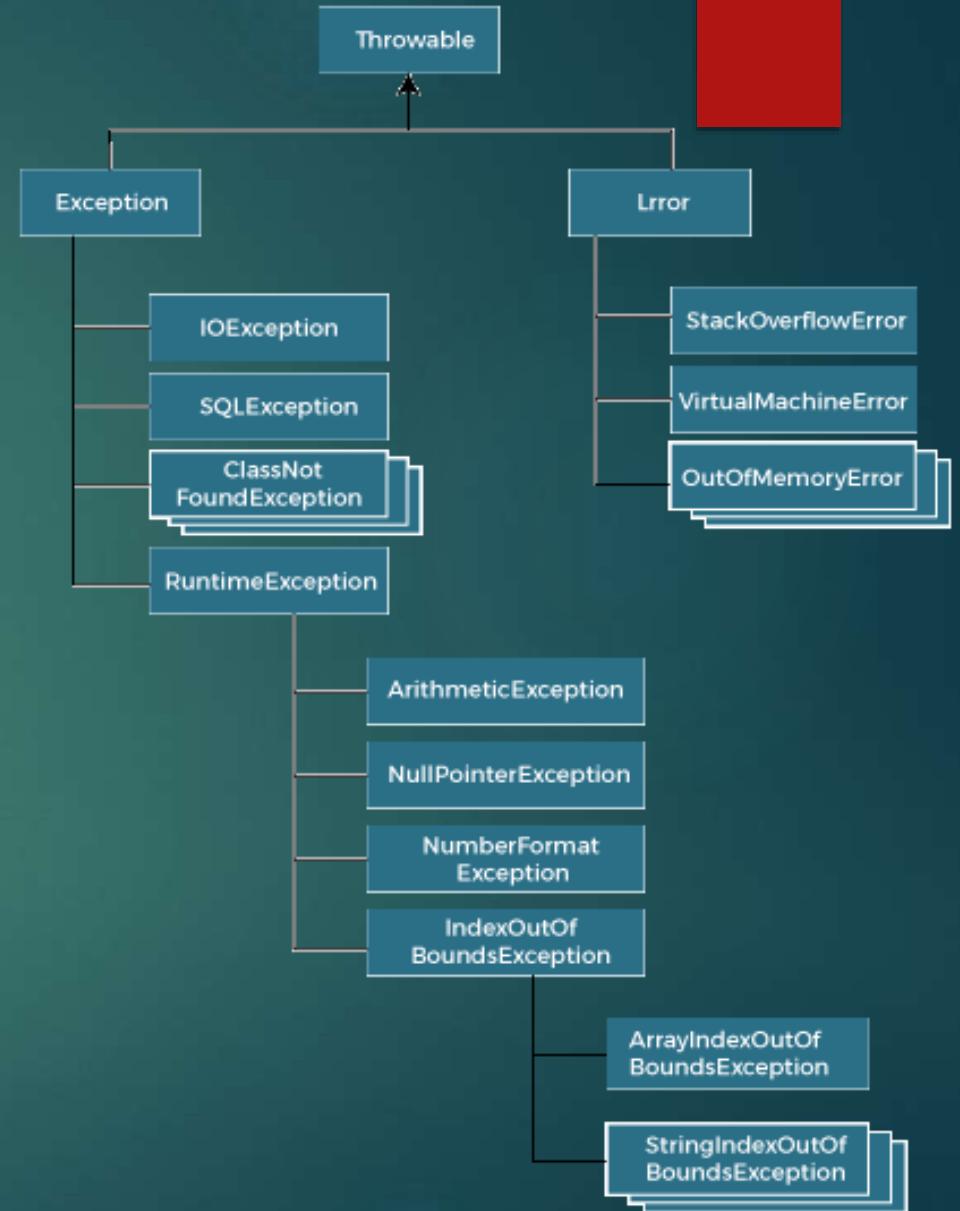
JAVA PROGRAMMING

Chap 6 : Exception Handling

- ▶ Exception handling refers to handling of abnormal or unexpected events.
- ▶ Some of these exceptions are known to the compiler while some other occur during **runtime** and are unknown to the compiler.
- ▶ The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc. so that the normal flow of the application can be maintained.
- ▶ Exceptions can be caught and handled by the program.
- ▶ When an exception occurs within a method, it creates an object.
- ▶ This object is called the exception object.
- ▶ It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred.
- ▶ Major reasons why an exception Occurs
 - ▶ Invalid user input
 - ▶ Device failure
 - ▶ Loss of network connection
 - ▶ Physical limitations (out of disk memory)
 - ▶ Code errors
 - ▶ Opening an unavailable file

- ▶ **Errors** represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc.
- ▶ Errors are usually beyond the control of the programmer, and we should not try to handle errors.
- ▶ When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.
- ▶ When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an exception (throw an error).
- ▶ “Exception” is a class and has subclasses according to the exceptions possible in Java.
- ▶ Eg: If an integer is expected while taking input and instead if a character is entered, then while parsing, error will occur. This error is called NumberFormatException. It is one of the subclass of the base class Exception.
- ▶ **Exceptions are classified as :**
- ▶ **Built-in Exceptions**
 - ▶ Checked Exception
 - ▶ Unchecked Exception
- ▶ **User-Defined Exceptions**

- ▶ The `java.lang.Throwable` class is the root class of Java Exception hierarchy inherited by two subclasses: `Exception` and `Error`.
- ▶ All exception and error types are subclasses of class **Throwable**, which is the base class of the hierarchy.
- ▶ One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. `NullPointerException` is an example of such an exception.
- ▶ Another branch, **Error** is used by the Java run-time system([JVM](#)) to indicate errors having to do with the run-time environment itself(JRE). `StackOverflowError` is an example of such an error.



Built-in Exceptions

- ▶ Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations.
- ▶ **Checked Exception :** The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.
- ▶ Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.
- ▶ Checked Exceptions make the programmers to handle the exception that may be thrown.
- ▶ There are two ways to deal with an exception :
 - ▶ Indicate that the method throws an exception
 - ▶ Method must catch the exception and take appropriate action using the try catch block
- ▶ **Unchecked Exception :** The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.
- ▶ The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time, but they are checked at runtime.
- ▶ In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error.
- ▶ Hence the programmers may not even know that such an exception would be thrown

Built-in Exceptions

- There are given some scenarios where unchecked exceptions may occur. They are as follows:
- If we divide any number by zero, there occurs an ArithmeticException.

```
int a=50/0;//ArithmeticException
```

- If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

```
String s=null;  
System.out.println(s.length());//NullPointerException
```

- If the formatting of any variable or number is mismatched, it may result into NumberFormatException. Suppose we have a string variable that has characters; converting this variable into digit will cause NumberFormatException.

```
String s="abc";  
int i=Integer.parseInt(s);//NumberFormatException
```

- When an array exceeds to its size, the ArrayIndexOutOfBoundsException occurs

```
int a[]=new int[5];  
a[10]=50; //ArrayIndexOutOfBoundsException
```

User defined Exceptions

- ▶ Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, users can also create exceptions, which are called 'user-defined Exceptions'.
- ▶ The advantages of Exception Handling in Java are as follows:
 - ▶ Provision to Complete Program Execution
 - ▶ Easy Identification of Program Code and Error-Handling Code
 - ▶ Propagation of Errors
 - ▶ Meaningful Error Reporting
 - ▶ Identifying Error Types

Java's Unchecked Exceptions

| Exception | Meaning |
|---------------------------------|---|
| ArithmaticException | Arithmatic error, such as divide-by-zero. |
| ArrayIndexOutOfBoundsException | Array index is out-of-bounds. |
| ArrayStoreException | Assignment to an array element of an incompatible type. |
| ClassCastException | Invalid cast. |
| EnumConstantNotPresentException | An attempt is made to use an undefined enumeration value. |
| IllegalArgumentException | Illegal argument used to invoke a method. |
| IllegalMonitorStateException | Illegal monitor operation, such as waiting on an unlocked thread. |
| IllegalStateException | Environment or application is in incorrect state. |
| IllegalThreadStateException | Requested operation not compatible with current thread state. |
| IndexOutOfBoundsException | Some type of index is out-of-bounds. |
| NegativeArraySizeException | Array created with a negative size. |
| NullPointerException | Invalid use of a null reference. |
| NumberFormatException | Invalid conversion of a string to a numeric format. |
| SecurityException | Attempt to violate security. |
| StringIndexOutOfBoundsException | Attempt to index outside the bounds of a string. |
| TypeNotFoundException | Type not found. |
| UnsupportedOperationException | An unsupported operation was encountered. |

Table 10-1 Java's Unchecked `RuntimeException` Subclasses Defined in `java.lang`

Java's Checked Exceptions

| Exception | Meaning |
|------------------------------|--|
| ClassNotFoundException | Class not found. |
| CloneNotSupportedException | Attempt to clone an object that does not implement the Cloneable interface. |
| IllegalAccessException | Access to a class is denied. |
| InstantiationException | Attempt to create an object of an abstract class or interface. |
| InterruptedException | One thread has been interrupted by another thread. |
| NoSuchFieldException | A requested field does not exist. |
| NoSuchMethodException | A requested method does not exist. |
| ReflectiveOperationException | Superclass of reflection-related exceptions. |

Table 10-2 Java's Checked Exceptions Defined in `java.lang`

WAP to perform division of two numbers accepted from the user to demonstrate ArithmeticException and NumberFormatException.

```
import java.io.*;
class Main{
    public static void main(String args[])throws IOException{
        int a,b,res;
        BufferedReader sc=new BufferedReader (new InputStreamReader (System.in));
        System.out.println("Enter 2 nos.");
        a=Integer.parseInt(sc.readLine());
        b=Integer.parseInt(sc.readLine());
        res=a/b;
        System.out.println("The Result is : "+res);
    }
}
```

ArithmaticException →

```
Enter 2 nos.
4
0
Exception in thread "main" java.lang.ArithmaticException: / by zero
        at Main.main(Main.java:9)
```

NumberFormatException →

```
Enter 2 nos.
5
d
Exception in thread "main" java.lang.NumberFormatException: For input string: "d"
        at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
        at java.base/java.lang.Integer.parseInt(Integer.java:652)
        at java.base/java.lang.Integer.parseInt(Integer.java:770)
        at Main.main(Main.java:11)
```

Java Exception Keywords

- ▶ Java provides five keywords that are used to handle the exception. The following table describes each.

| Keyword | Description |
|---------|---|
| try | The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. |
| finally | The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature. |

try-catch-finally

- ▶ **Try Block :**

- ▶ The statements that are prone to generate errors or exception are placed in try block.
- ▶ If an exception occurs then the catch block will be executed and then the finally block will be executed.
- ▶ Else if no exception occurs, then the control will directly go to the finally block i.e. the catch block will not be executed.
- ▶ The java code that you think can generate an exception is placed in the try catch block to handle the error.
- ▶ If an exception occurs but a matching catch block is not found then it reaches to the finally block.
- ▶ In any case, when the exception occurs in a particular statement of try block, the remaining statements of the try block after this statement are not executed i.e. no statements of the try block are executed after the exception generating statement.

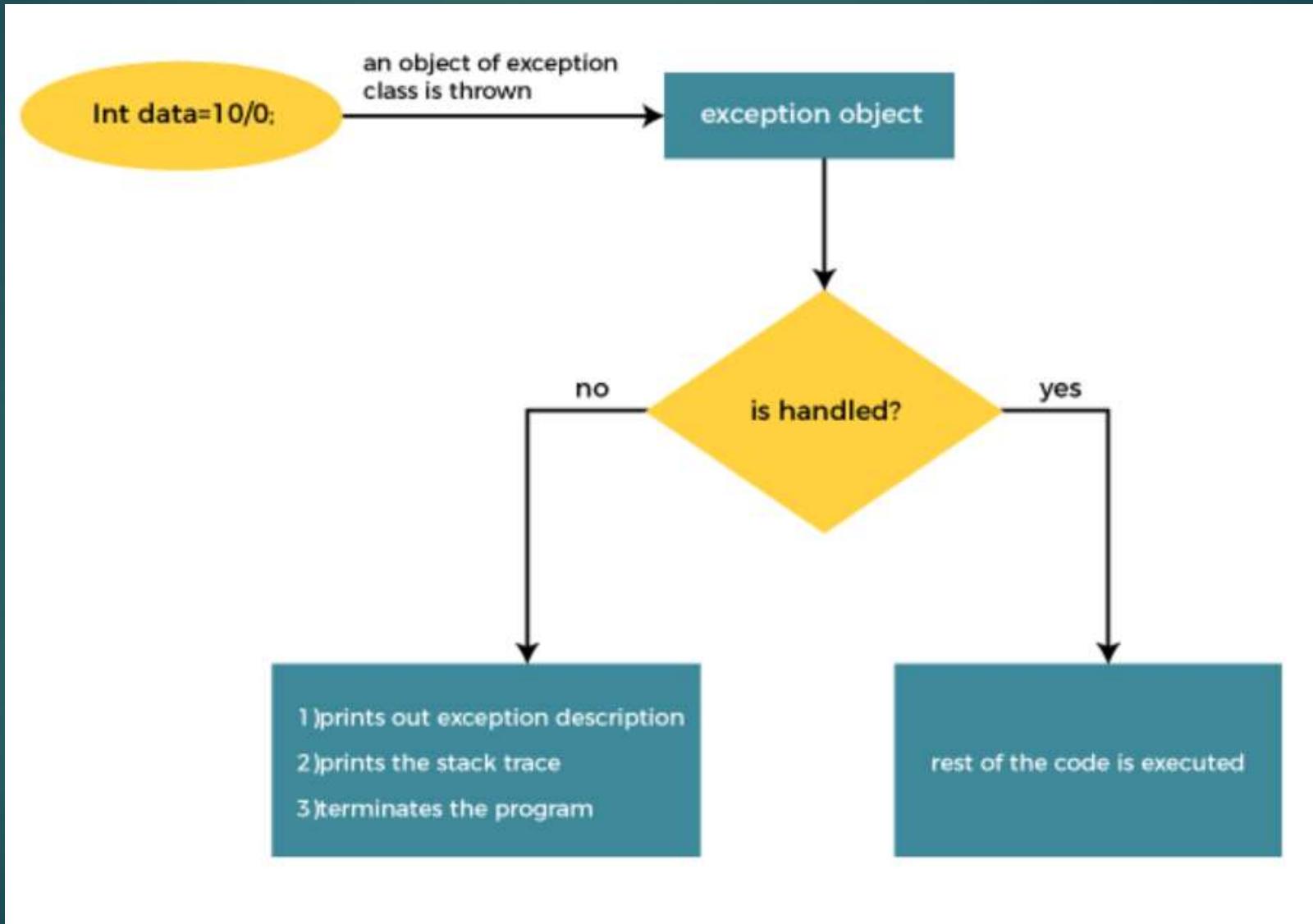
- ▶ **Catch Block :**

- ▶ The exception thrown by the try block are caught by the catch block.
- ▶ The type of the exception occurred must match with the exception mentioned in the brackets of the catch block.

- ▶ **Finally Block :**

- ▶ A finally block is always executed irrespective of whether the exception occurred or not
- ▶ It is an optional block. It is not necessary to have a finally block i.e. you may just have try catch blocks.

Internal Working of Java try-catch block



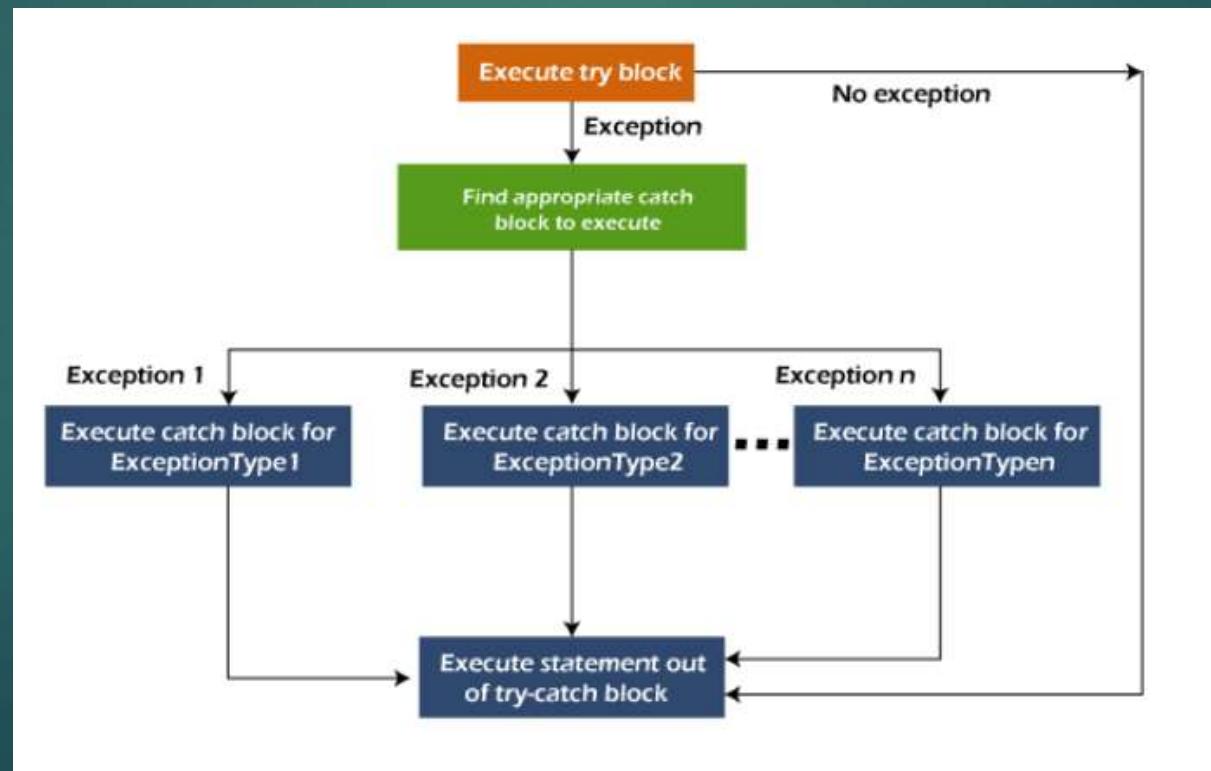
WAP to perform division of two numbers accepted from the user to demonstrate ArithmeticException using try catch block.

```
import java.io.*;
class Main{
    public static void main(String args[]) throws IOException{
        int a,b,res;
        BufferedReader sc=new BufferedReader (new InputStreamReader (System.in));
        System.out.println("Enter 2 nos.");
        a=Integer.parseInt(sc.readLine());
        b=Integer.parseInt(sc.readLine());
        try {
            res=a/b;
            System.out.println("The Result is : "+res);
        }
        catch (ArithmeticException ae)
        {System.out.println("Exception has occurred as divisor entered is zero");
        }
        System.out.println("Remaining code");
    }
}
```

```
Enter 2 nos.
5
0
Exception has occurred as divisor entered is zero
Remaining code
```

Multiple catch blocks

- ▶ A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.
- ▶ **Points to remember**
- ▶ At a time only one exception occurs and at a time only one catch block is executed.
- ▶ All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.



WAP to perform division of two numbers accepted from the user to demonstrate ArithmeticException and NumberFormatException using try catch block.

```
import java.io.*;
class Main{
    public static void main(String args[]) throws IOException{
        int a=0,b=0,res;
        BufferedReader sc=new BufferedReader (new
InputStreamReader (System.in));
        try {
            System.out.println("Enter 2 nos.");
            a=Integer.parseInt(sc.readLine());
            b=Integer.parseInt(sc.readLine());
            res=a/b;
            System.out.println("The Result is : "+res);
            String s=null;
            System.out.println(s.length());
        }
        catch (ArithmaticException ae) {
            System.out.println("Exception has occurred as divisor entered
is zero");
        }
        catch (NumberFormatException ne) {
            System.out.println("Invalid Input. Enter Integer Number.");
        }
        catch(Exception e) {
            System.out.println(e);
        }
        System.out.println("Remaining Code Continues");
    }
}
```

```
Enter 2 nos.
5
0
Exception has occurred as divisor entered is zero
Remaining Code Continues
```

```
Enter 2 nos.
5
d
Invalid Input. Enter Integer Number.
Remaining Code Continues
```

```
Enter 2 nos.
5
2
The Result is : 2
java.lang.NullPointerException
Remaining Code Continues
```

Nested try blocks

- 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.
- In Java, using a try block inside another try block is permitted. It is called as nested try block.
- Every statement that we enter a statement in try block, context of that exception is pushed onto the stack.
- For example, the inner try block can be used to handle `ArrayIndexOutOfBoundsException` while the outer try block can handle the `ArithemeticException` (division by zero).

```
....  
//main try block  
try  
{  
    statement 1;  
    statement 2;  
//try catch block within another try block  
    try  
    {  
        statement 3;  
        statement 4;  
//try catch block within nested try block  
        try  
        {  
            statement 5;  
            statement 6;  
        }  
        catch(Exception e2)  
        {  
            //exception message of 2nd nested try block  
        }  
        catch(Exception e1)  
        {  
            //exception message of 1st nested try block  
        }  
        catch(Exception e3)  
        {  
            //exception message of parent (outer) try block  
        }  
        ....  
    }
```

WAP to perform division of two numbers accepted from the user to demonstrate ArithmeticException and NumberFormatException using nested try catch block.

```
import java.io.*;
class Main{
    public static void main(String args[]) throws IOException{
        int a=0,b=0,res;
        BufferedReader sc=new BufferedReader (new
InputStreamReader (System.in));
        try {
            System.out.println("Enter 2 nos.");
            a=Integer.parseInt(sc.readLine());
            b=Integer.parseInt(sc.readLine());
            try {
                res=a/b;
                System.out.println("The Result is : "+res);
            }
            catch (ArithmaticException ae) {
                System.out.println("Exception has occurred as divisor
entered is zero");
            }
        }
        catch (NumberFormatException ne) {
            System.out.println("Invalid Input. Enter Integer Number.");
        }
        System.out.println("Remaining Code Continues");
    }
}
```

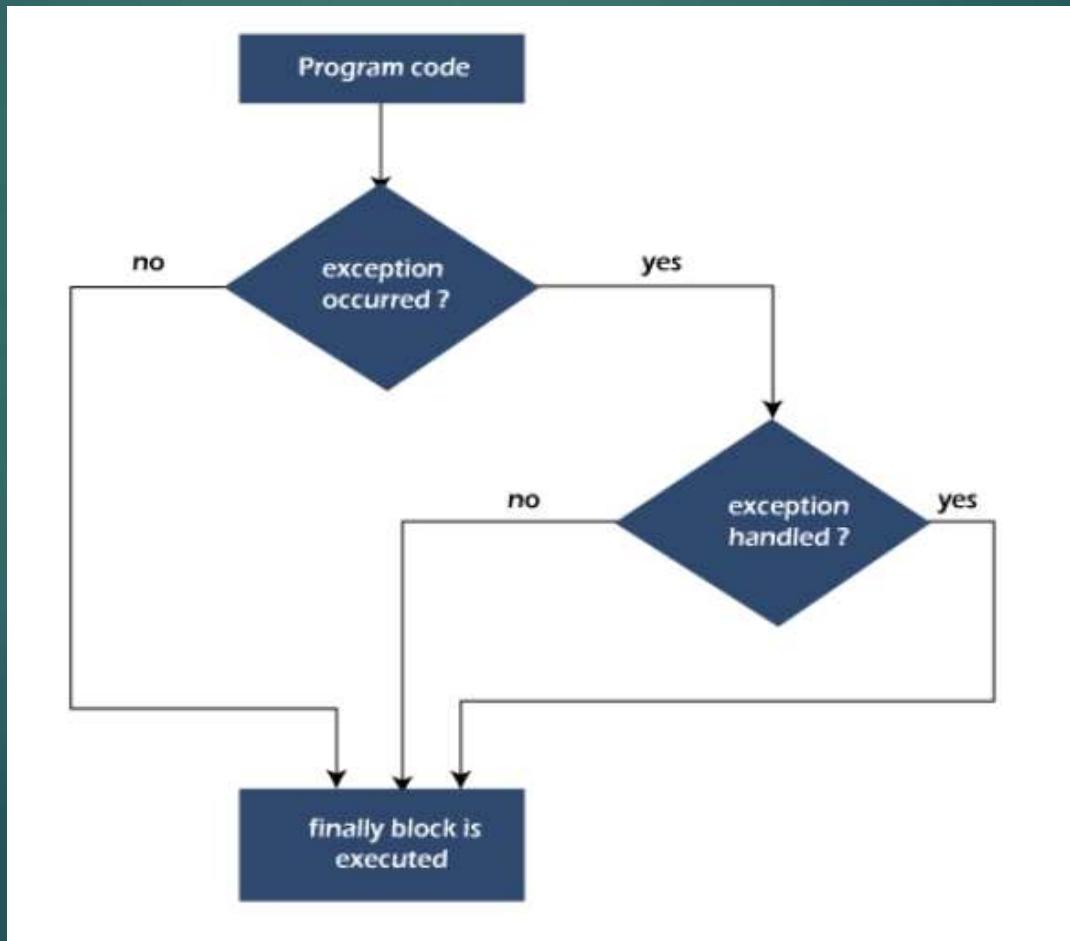
```
Enter 2 nos.
5
0
Exception has occurred as divisor entered is zero
Remaining Code Continues
```

```
Enter 2 nos.
5
d
Invalid Input. Enter Integer Number.
Remaining Code Continues
```

```
Enter 2 nos.
10
2
The Result is : 5
Remaining Code Continues
```

Internal Working of Java try-catch-finally block

- ▶ A finally block is always executed irrespective of whether the exception occurred or not
- ▶ It is an optional block. It is not necessary to have a finally block i.e. you may just have try catch blocks.
- ▶ finally block in Java can be used to put "cleanup" code such as closing a file, closing connection, etc.
- ▶ The important statements to be printed can be placed in the finally block.



WAP to perform division of two numbers accepted from the user to demonstrate ArithmeticException and NumberFormatException using try-catch-finally block.

```
import java.io.*;
class Main{
    public static void main(String args[]) throws IOException{
        int a=0,b=0,res;
        BufferedReader sc=new BufferedReader (new
InputStreamReader (System.in));
        try {
            System.out.println("Enter 2 nos.");
            a=Integer.parseInt(sc.readLine());
            b=Integer.parseInt(sc.readLine());
            try {
                res=a/b;
                System.out.println("The Result is : "+res);
            }
            catch (ArithmaticException ae)
            { System.out.println("Exception has occurred as divisor entered
is zero");
            }
            catch (NumberFormatException ne)
            { System.out.println("Invalid Input. Enter Integer Number.");
            }
            finally {
                System.out.println("Finally block executed");
            }
        }
```

```
Enter 2 nos.
5
0
Exception has occurred as divisor entered is zero
Finally block executed
```

```
Enter 2 nos.
5
d
Invalid Input. Enter Integer Number.
Finally block executed
```

```
Enter 2 nos.
10
2
The Result is : 5
Finally block executed
```

“throws” keyword

- ▶ The Java throws keyword is used to declare an exception.
- ▶ It gives an information to the programmer that there may occur an exception. So, it is better for the programmer to provide the exception handling code so that the normal flow of the program can be maintained.
- ▶ Exception Handling is mainly used to handle the checked exceptions.
- ▶ Syntax of Java throws

```
return_type method_name() throws exception_class_name{  
    //method code  
}
```

- ▶ throws keyword in Java is used in the signature of method to indicate that this method might throw one of the listed type (i.e. checked) exceptions. The caller to these methods has to handle the exception using a try-catch block.
- ▶ Important points to remember about throws keyword:
 - ▶ throws keyword is required only for checked exception and usage of throws keyword for unchecked exception is meaningless.
 - ▶ throws keyword is required only to convince compiler and usage of throws keyword does not prevent abnormal termination of program.
 - ▶ By the help of throws keyword we can provide information to the caller of the method about the exception.

WAP to demonstrate throws keyword.

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        int a;
        System.out.println("Enter age :");
        Scanner sc=new Scanner(System.in);
        a=sc.nextInt();
        checkAge(a);
    }
    static void checkAge(int age) throws ArithmeticException
    {
        if (age < 18) {
            throw new ArithmeticException("Access denied - You
must be at least 18 years old.");
        }
        else
        {
            System.out.println("Access granted - You are old
enough!");
        }
    }
}
```

```
Enter age :
20
Access granted - You are old enough!
```

```
Enter age :
15
Exception in thread "main" java.lang.ArithmetricException: Access denied - You must be at least 18 years old.
        at Main.checkAge(Main.java:13)
        at Main.main(Main.java:8)
```

“throw” keyword

- ▶ The Java throw keyword is used to throw an exception explicitly.
- ▶ We specify the exception object which is to be thrown. The Exception has some message with it that provides the error description. These exceptions may be related to user inputs, server, etc.
- ▶ We can throw either checked or unchecked exceptions in Java by throw keyword.
- ▶ It is mainly used to throw a custom exception i.e. you can make your own conditions to throw exceptions.
- ▶ It will not be a built in exception but it will be your user defined exception.

WAP to accept and display month number. Throw a NumberFormatException if month number exceeds 12 or is less than 0.

```
import java.util.*;
class Main{
    public static void main(String args[]){
        int m;
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter month number");
        m=sc.nextInt();
        try {
            if(m<1 || m>12)
                throw new NumberFormatException();
            System.out.println("The entered month number is :
"+m);
        }
        catch (NumberFormatException ne)
        { System.out.println("Invalid month number.");
        }
    }
}
```

```
Enter month number
25
Invalid month number.
```

```
Enter month number
5
The entered month number is : 5
```

WAP to accept and display month number. Throw an Exception if month number exceeds 12 or is less than 0. Make your own exception class to handle this exception.

```
import java.util.*;
class MonthNumberException extends Exception {
    MonthNumberException()
    {
        System.out.println("Invalid Month Number");
    }
}
class Main{
    public static void main(String args[]){
        int m;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter month number");
        m=sc.nextInt();
        try {
            if(m<1 || m>12)
                throw new MonthNumberException();
            System.out.println("The entered month number is :
"+m);
        }
        catch (MonthNumberException me)
        {
        }
    }
}
```

```
Enter month number
25
Invalid month number.
```

```
Enter month number
5
The entered month number is : 5
```

Programming Practice Questions

- ▶ Write a Java program that simulates a simple banking system. Implement nested try-catch blocks to handle:
 1. `ArithmaticException` when dividing by zero in calculating interest.
 2. `NumberFormatException` if the user inputs invalid account details.
- ▶ Create a custom exception class `InvalidAgeException` that extends `Exception`. Write a Java program that uses this custom exception to validate a user's age. If the user enters an age less than 0 or greater than 100, throw and handle `InvalidAgeException`.
- ▶ Create a Java application that prompts the user to enter a number and performs a calculation with it. Implement exception handling to manage invalid inputs and arithmetic errors, and ensure that any resources (e.g., scanners) used for input are closed in the finally block.
- ▶ Develop a program that prompts the user for two integers and performs division. Use multiple catch blocks to handle `InputMismatchException` (if the user inputs non-numeric values), `ArithmaticException` (if there is a division by zero), and `Exception` (for any other unexpected errors).
- ▶ Develop a library management system where you can borrow and return books. Define custom exceptions for scenarios such as when a book is already borrowed (`BookAlreadyBorrowedException`), when a book is not available in the library (`BookNotFoundException`), and when a user tries to return a book that they haven't borrowed (`InvalidReturnOperationException`).

JAVA PROGRAMMING

Chap 7 : Generics and
Collections

Generics

- ▶ Imagine, wouldn't it be nice if we could write a single sort method that could sort the elements in an Integer array, a String array, or an array of any type that supports ordering.
- ▶ Java Generic methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.
- ▶ Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time.
- ▶ Using Java Generic concept, we might write a generic method for sorting an array of objects, then invoke the generic method with Integer arrays, Double arrays, String arrays and so on, to sort the array elements.
- ▶ The idea is to allow type (Integer, String, ... etc., and user-defined types) to be a parameter to methods, classes, and interfaces.
- ▶ Using Generics, it is possible to create classes that work with different data types.
- ▶ An entity such as class, interface, or method that operates on a parameterized type is a generic entity.
- ▶ There are mainly 3 advantages of generics. They are as follows:
 - 1) **Type-safety:** We can hold only a single type of objects in generics. It doesn't allow to store other objects.
 - 2) **Type casting is not required:** There is no need to typecast the object.
 - 3) **Compile-Time Checking:** It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

Types of Java Generics

Generic Methods

- ▶ You can write a single generic method declaration that can be called with arguments of different types.
- ▶ Based on the types of the arguments passed to the generic method, the compiler handles each method call appropriately.
- ▶ Generic Java method takes a parameter and returns some value after performing a task.
- ▶ It is exactly like a normal function, however, a generic method has type parameters that are cited by actual type.
- ▶ This allows the generic method to be used in a more general way.
- ▶ The compiler takes care of the type of safety which enables programmers to code easily since they do not have to perform long, individual type castings.
- ▶ Following are the rules to define Generic Methods –
 - ▶ All generic method declarations have a type parameter section delimited by angle brackets (< and >) that precedes the method's return type.
 - ▶ Each type parameter section contains one or more type parameters separated by commas. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.
 - ▶ The type parameters can be used to declare the return type and act as placeholders for the types of the arguments passed to the generic method, which are known as actual type arguments.
 - ▶ A generic method's body is declared like that of any other method. **Note that type parameters can represent only reference types, not primitive types (like int, double and char).**

Types of Java Generics

Generic Classes

- ▶ A generic class is implemented exactly like a non-generic class.
- ▶ The only difference is that it contains a type parameter section.
- ▶ There can be more than one type of parameter, separated by a comma.
- ▶ The classes, which accept one or more parameters, are known as parameterized classes or parameterized types.
- ▶ we use the T type parameter to create the generic class of specific type.
- ▶ The type parameters naming conventions are important to learn generics thoroughly. The common type parameters are as follows:
 - ▶ T - Type
 - ▶ E - Element
 - ▶ K - Key
 - ▶ N - Number
 - ▶ V - Value

Generics Class Example

```
class GenericsClass<T> {          // create a Generic class
    private T data;                // variable of T type
    public GenericsClass(T data) {  // constructor of Generic class
        this.data = data;
    }
    public T getData() {           // method that return T type variable
        return this.data;
    }
}
```

```
Generic Class returns: 5
Generic Class returns: Java Programming
```

```
class Main {
    public static void main(String[] args) {
        GenericsClass<Integer> intObj = new GenericsClass<>(5);      // initialize generic class with Integer data
        System.out.println("Generic Class returns: " + intObj.getData());
    }
}
```

```
GenericsClass<String> stringObj = new GenericsClass<>("Java Programming"); // initialize generic class with String data
System.out.println("Generic Class returns: " + stringObj.getData());
}}
```

Generics Method Example

```
class DemoClass {  
    public <T> void genericsMethod(T data) {          // create a generics method  
        System.out.println("Data Passed: " + data);  
    }  
}  
  
class Main {  
    public static void main(String[] args) {  
        DemoClass demo = new DemoClass();  
        demo.<String>genericsMethod("Java Programming"); // generics method working with String  
        demo.genericsMethod(25);      // generics method working with integer without including the type parameter  
    }  
}
```

Data Passed: Java Programming
Data Passed: 25

Generics Class Example 2

```
class GenericsClass<T> {          // create a Generic class
    private T data, data1;           // variables of T type
    public GenericsClass(T data, T data1) { // constructor of Generic class
        this.data = data;
        this.data1 = data1;
    }
    public <T> void getData() {       // method that prints T type variable
        System.out.println(this.data);
        System.out.println(this.data1);
    }
}
class Main {
    public static void main(String[] args) {
        GenericsClass intObj = new GenericsClass(5,"10");      // initialize generic class with different data
        intObj.getData();
    }
}
GenericsClass stringObj = new GenericsClass("Java Programming",15);
stringObj.getData();
}}
```

```
5
10
Java Programming
15
```

Bounded Types

- ▶ In general, the type parameter can accept any data types.
- ▶ However, if we want to use generics for some specific types (such as accept data of number types) only, then we can use bounded types.
- ▶ In the case of bound types, we use the extends keyword.
- ▶ Syntax : <T extends A>
This means T can only accept data that are subtypes of A.
- ▶ Eg : class GenericsClass <T extends Number>
Here, GenericsClass is created with bounded type. This means GenericsClass can only work with data types that are subtypes of Number (Integer, Double, and so on).

Bounded Type Example

```
class GenericsClass <T extends Number> {  
    public void display() {  
        System.out.println("This is a bounded type generics class.");  
    }  
}  
  
class Main {  
    public static void main(String[] args) {  
        // create an object of GenericsClass  
        GenericsClass<Integer/Float/Double/Long/Short> obj = new GenericsClass<>();  
        obj.display();  
    }  
}
```

```
class GenericsClass <T extends Number> {  
    public void display() {  
        System.out.println("This is a bounded type generics class.");  
    }  
}  
  
class Main {  
    public static void main(String[] args) {  
        // create an object of GenericsClass  
        GenericsClass<String> obj = new GenericsClass<>();  
        obj.display();  
    }  
}
```

This is a bounded type generics class.

```
Main.java:12: error: type argument String is not within bounds of type-variable T  
    GenericsClass<String> obj = new GenericsClass<>();  
                           ^  
    where T is a type-variable:  
        T extends Number declared in class GenericsClass  
Main.java:12: error: incompatible types: cannot infer type arguments for GenericsClass<>  
    GenericsClass<String> obj = new GenericsClass<>();  
                           ^  
    reason: inference variable T has incompatible bounds  
        equality constraints: String  
        lower bounds: Number  
    where T is a type-variable:  
        T extends Number declared in class GenericsClass  
2 errors
```

Generic Restrictions

- ▶ You cannot use generics in certain ways and in certain scenarios as listed below –
 - ▶ You cannot use datatypes with generics.
 - ▶ You cannot instantiate the generic parameters.
 - ▶ The generic type parameter cannot be static.
 - ▶ You cannot cast parameterized type of one datatype to other.
 - ▶ You cannot create an array of generic type objects.
 - ▶ A generic type class cannot extend the throwable class therefore, you cannot catch or throw these objects.

Generic Restrictions

- ▶ You cannot use primitive datatypes with generics.

```
class Student<T>{  
    T age;  
    Student(T age){  
        this.age = age;  
    }  
}  
  
public class GenericsExample {  
    public static void main(String args[]){  
        Student<Float> std1 = new Student<Float>(25.5f);  
        Student<String> std2 = new Student<String>("25");  
        Student<int> std3 = new Student<int>(25);  
    }  
}
```

Int is a primitive datatype and hence cannot be used with generics.

We need to mention Integer in generics to work with integer data.

```
Main.java:11: error: unexpected type  
    Student<int> std3 = new Student<int>(25);  
                           ^  
    required: reference  
    found:     int  
  
Main.java:11: error: unexpected type  
    Student<int> std3 = new Student<int>(25);  
                           ^  
    required: reference  
    found:     int  
2 errors
```

Generic Restrictions

- You cannot instantiate the generic parameters.

```
class Student<T>{  
    T age;  
    Student(T age){  
        this.age = age;  
    }  
    public void display() {  
        System.out.println("Value of age: "+this.age);  
    }  
}  
  
public class Main {  
    public static void main(String args[]) {  
        Student<Float> std1 = new Student<Float>(25.5f);  
        std1.display();  
        T obj = new T();  
    }  
}
```

```
Main.java:14: error: cannot find symbol  
    T obj = new T();  
           ^  
      symbol:   class T  
      location: class Main  
Main.java:14: error: cannot find symbol  
    T obj = new T();  
           ^  
      symbol:   class T  
      location: class Main  
2 errors
```

Generic Restrictions

- The generic type parameter cannot be static.

```
class Student<T>{  
    static T age;  
    Student(T age){  
        this.age = age;  
    }  
    public void display() {  
        System.out.println("Value of age: "+this.age);  
    }  
}  
public class Main {  
    public static void main(String args[]) {  
        Student<Float> std1 = new Student<Float>(25.5f);  
        std1.display();  
    }  
}
```

```
Main.java:2: error: non-static type variable T cannot be referenced from a static context  
    static T age;  
           ^  
1 error
```

Generic Restrictions

- ▶ You cannot cast parameterized type of one datatype to other.

```
class Student<T>{  
    T age;  
    Student(T age){  
        this.age = age;  
    }  
    public void display() {  
        System.out.println("Value of age: "+this.age);  
    }  
}  
public class Main {  
    public static void main(String args[]) {  
        Student<Float> std1 = new Student<Float>(25.5f);  
        std1.display();  
        Student<Double> std2 = std1;  
        std2.display();  
    }  
}
```

```
Main.java:14: error: incompatible types: Student cannot be converted to Student  
        Student<Double> std2 = std1;  
                           ^  
1 error
```

Generic Restrictions

- ▶ You cannot create an array of generic type objects.
- ▶ Note : syntax for creating array of objects : Student s[] = new Student[n];

```
class Student<T>{  
    T age;  
    Student(T age){  
        this.age = age;  
    }  
    public void display() {  
        System.out.println("Value of age: "+this.age);  
    }  
}  
public class Main {  
    public static void main(String args[]) {  
        Student<Float> std1[ ] = new Student<Float>[5];  
    }  
}
```

```
Main.java:12: error: generic array creation  
        Student<Float> std1[ ] = new Student<Float>[5];  
                           ^
```

1 error

Generic Class Hierarchies

- ▶ Generic classes can be part of a class hierarchy in just the same way as a non-generic class.
- ▶ Thus, a generic class can act as a superclass or be a subclass.
- ▶ The key difference between generic and non-generic hierarchies is that in a generic hierarchy, any type arguments needed by a generic superclass must be passed up the hierarchy by all subclasses.
- ▶ A subclass can freely add its own type parameters, if necessary.

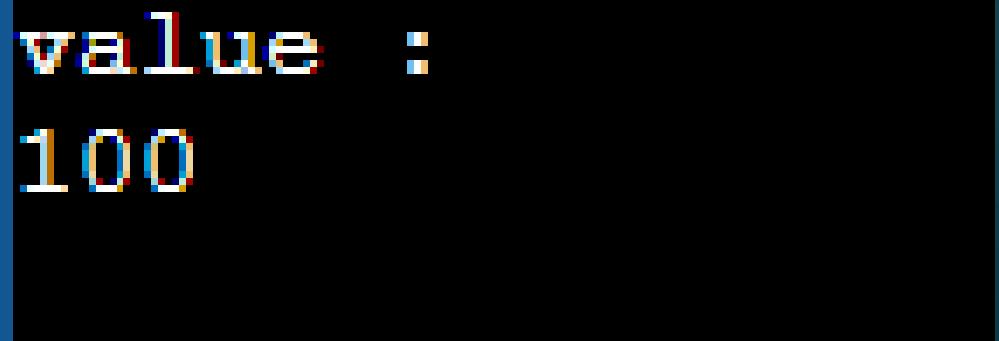
Generic Class Hierarchies

Eg 1 : Generic Super Class and Generic Sub Class

```
import java.io.*;  
class Base<T> {          // Class 1 - Parent class  
    T obj;                // Member variable of parent class  
    Base(T o1) {           // Constructor of parent class  
        obj = o1;  
    }  
    T getobj1() {          // Member function of parent class that returns an object  
        return obj;  
    } }  
  
class Child<T> extends Base<T> { // Class 2 - Child class  
    T obj2;                // Member variable of child class  
    Child(T o1, T o2) {     // Constructor of Child class  
        super(o1);          // Calling super class using super keyword  
        obj2 = o2;  
    }  
    T getobj2() {          // Member function of child class that returns an object  
        return obj2;  
    } }
```

Generic Class hierarchies

```
class Main {           // Class 3 - Main class
    public static void main(String[] args)
    {
        Child x = new Child("value : ",100);
        System.out.println(x.getobj1());
        System.out.println(x.getobj2());
    }
}
```



```
value :
100
```

Generic Class Hierarchies

Eg 1 : Non-Generic Super Class and Generic Sub Class

```
import java.io.*;  
  
class Base {          // non-generic super-class  
    int n;  
    Base(int i) {  
        n = i;  
    }  
    int getval() {  
        return n;  
    }  
  
    class Child<T> extends Base {    // generic sub-class  
        T obj;  
        Child(T o1, int i) {  
            super(i);  
            obj = o1;  
        }  
        T getobj() {  
            return obj;  
        }  
  
        class Main {  
            public static void main(String[] args)  
            {  
                Child c = new Child("Java Programming", 2023);  
                System.out.println(c.getobj() + " " + c.getval());  
            }  
        }  
    }  
}
```

Java Programming 2023

Java Collections

Collections

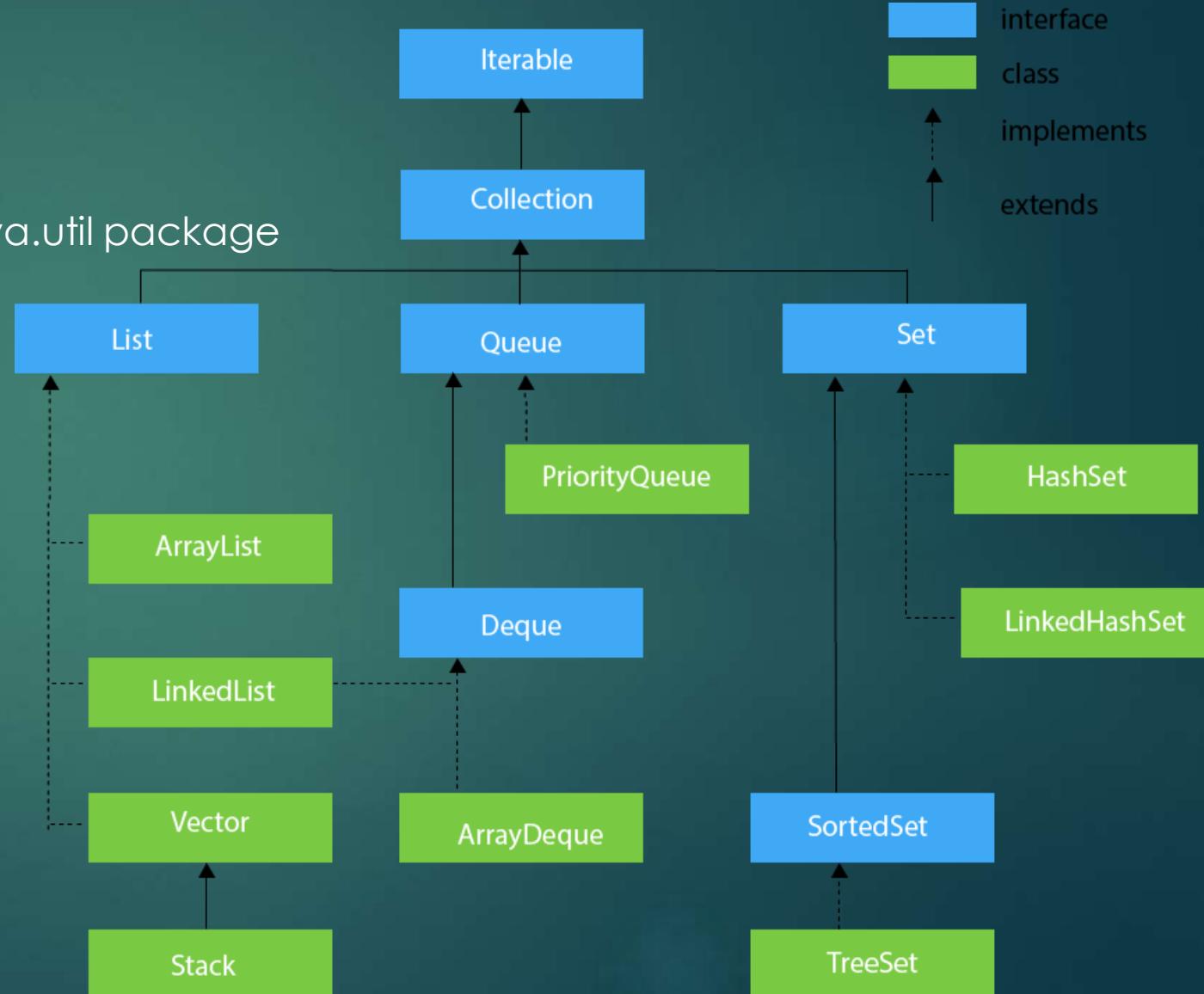
- ▶ The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects. A Collection represents a single unit of objects, i.e., a group.
- ▶ Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
- ▶ Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Why to use Java Collections

- ▶ There are several benefits of using Java collections such as:
 - ▶ Reducing the effort required to write the code by providing useful data structures and algorithms
 - ▶ Java collections provide high-performance and high-quality data structures and algorithms thereby increasing the speed and quality
 - ▶ Unrelated APIs can pass collection interfaces back and forth
 - ▶ Decreases extra effort required to learn, use, and design new API's
 - ▶ Supports reusability of standard data structures and algorithms

Collection Framework

- ▶ The Collection framework represents a unified architecture for storing and manipulating a group of objects.
It has:
 - ▶ Interfaces and its implementations
 - ▶ Classes
 - ▶ Algorithm
- ▶ The Collections framework is defined in the `java.util` package



Java Collections : Interface

Iterator interface

- ▶ Iterator is an interface that iterates the elements.
- ▶ It is used to traverse the list and modify the elements.
- ▶ Iterator interface has three methods which are mentioned below:
 - ▶ public boolean hasNext() – This method returns true if the iterator has more elements otherwise it returns false.
 - ▶ public object next() – It returns the element and moves the cursor pointer to the next element.
 - ▶ public void remove() – This method removes the last element returned by the iterator.
- ▶ There are three components that extend the collection interface i.e List, Queue and Sets.

Iterable interface

- ▶ The Iterable interface is the root interface for all the collection classes.
- ▶ The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.
- ▶ It contains only one abstract method. i.e., `Iterator<T> iterator()`
- ▶ It returns the iterator over the elements of type T.

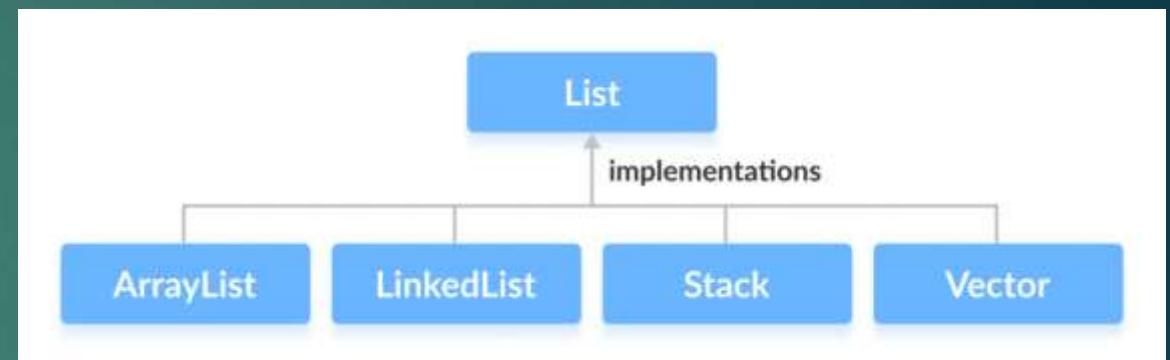
Java Collections : Interface

Collection interface

- ▶ The Collection interface is the root interface of the collections framework hierarchy.
- ▶ Java does not provide direct implementations of the Collection interface but provides implementations of its sub-interfaces like List, Set, and Queue
- ▶ The Collection interface is the interface which is implemented by all the classes in the collection framework.
- ▶ the Collection interface builds the foundation on which the collection framework depends.
- ▶ **Methods of Collection :** The Collection interface includes various methods that can be used to perform different operations on objects. These methods are available in all its sub interfaces.
 - ▶ add() - inserts the specified element to the collection
 - ▶ size() - returns the size of the collection
 - ▶ remove() - removes the specified element from the collection
 - ▶ iterator() - returns an iterator to access elements of the collection
 - ▶ addAll() - adds all the elements of a specified collection to the collection
 - ▶ removeAll() - removes all the elements of the specified collection from the collection
 - ▶ clear() - removes all the elements of the collection

Java Collections : List Interface

- ▶ A List is an ordered Collection of elements **which may contain duplicates.**
- ▶ It allows us to add and remove elements like an array
- ▶ It is an interface that extends the Collection interface.
- ▶ Since List is an interface, we cannot create objects from it.
- ▶ List interface is implemented by the following classes -
 - ▶ ArrayList
 - ▶ LinkedList
 - ▶ Vectors
 - ▶ Stack
- ▶ To instantiate the List interface, we must use :
 - ▶ `List <data-type> list1= new ArrayList();`
 - ▶ `List <data-type> list2 = new LinkedList();`
 - ▶ `List <data-type> list3 = new Vector();`
 - ▶ `List <data-type> list4 = new Stack();`



Java Collections : List Interface

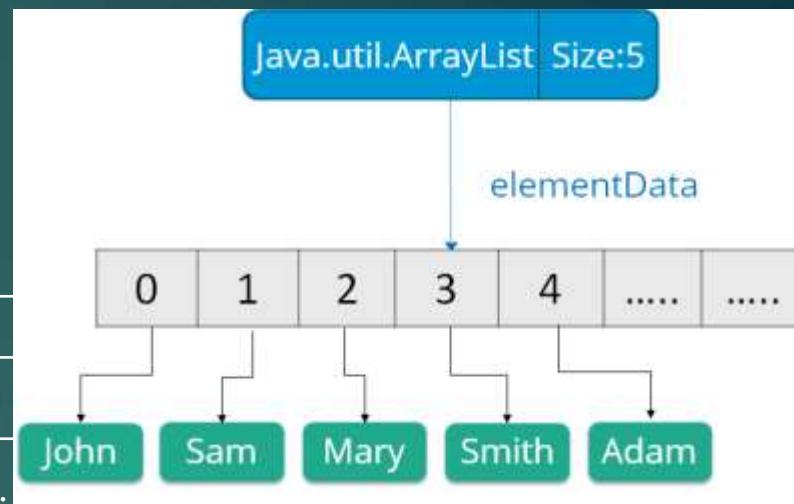
► **Methods of List** : The List interface includes all the methods of the Collection interface. Its because Collection is a super interface of List. Some of the commonly used methods of the Collection interface that's also available in the List interface are:

- ▶ add() - adds an element to a list
- ▶ addAll() - adds all elements of one list to another
- ▶ get() - helps to randomly access elements from lists
- ▶ iterator() - returns iterator object that can be used to sequentially access elements of lists
- ▶ set() - changes elements of lists
- ▶ remove() - removes an element from the list
- ▶ removeAll() - removes all the elements from the list
- ▶ clear() - removes all the elements from the list (more efficient than removeAll())
- ▶ size() - returns the length of lists
- ▶ toArray() - converts a list into an array
- ▶ contains() - returns true if a list contains specified element

ArrayList

- ▶ ArrayList is the implementation of List Interface where the elements can be dynamically added or removed from the list.
- ▶ It uses a dynamic array to store the duplicate element of different data types.
- ▶ Also, the size of the list is increased dynamically if the elements are added more than the initial size.
- ▶ The ArrayList class maintains the insertion order and is non-synchronized.
- ▶ The elements stored in the ArrayList class can be randomly accessed.
- ▶ Syntax: `ArrayList object = new ArrayList ()`;
- ▶ Some of the methods in array list are listed below:

| Method | Description |
|--|---|
| <code>boolean add(Collection c)</code> | Appends the specified element to the end of a list. |
| <code>void add(int index, Object element)</code> | Inserts the specified element at the specified position. |
| <code>void clear()</code> | Removes all the elements from this list. |
| <code>int lastIndexOf(Object o)</code> | Return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element. |
| <code>Object clone()</code> | Return a shallow copy of an ArrayList. |
| <code>Object[] toArray()</code> | Returns an array containing all the elements in the list. |
| <code>void trimToSize()</code> | Trims the capacity of this ArrayList instance to be the list's current size. |

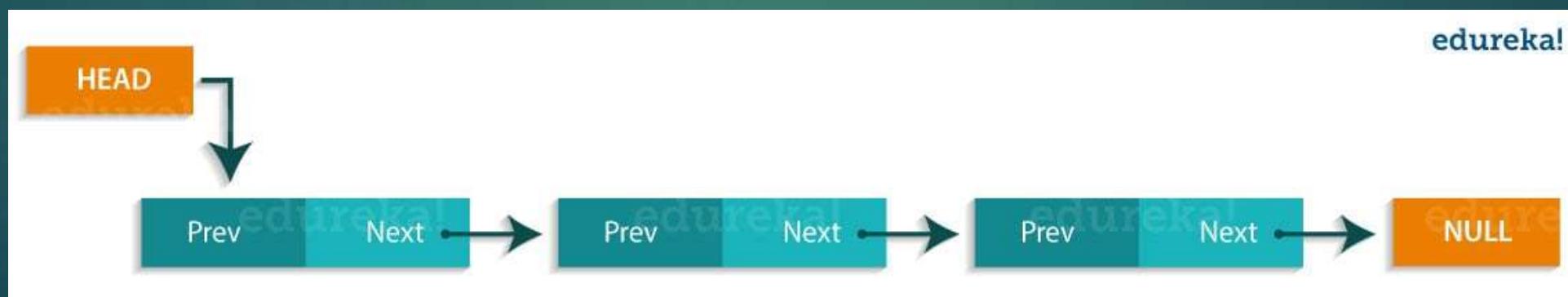


ArrayList

```
import java.util.*;  
class Main{  
    public static void main(String args[]){  
        ArrayList<String> animals=new ArrayList<String>(); // creating array list  
        animals.add("Dog");  
        animals.add("Cat");  
        animals.add("Horse");  
        Iterator itr=animals.iterator();  
        System.out.println("Array List : ");  
        while(itr.hasNext()){  
            System.out.println(itr.next());  
        }  
  
        String a1 = animals.get(2); // Access element from the list  
        System.out.println("Accessed Element: " + a1);  
        System.out.println(animals);  
  
        String r = animals.remove(1); // Remove element from the list  
        System.out.println("Removed Element: " + r);  
        System.out.println(animals);  
        animals.set(1,"Cow"); // Changing element in the list  
        System.out.println("Array List with Modified Element: " + animals);  
    } }  
  
Dog  
Cat  
Horse  
Accessed Element: Horse  
[Dog, Cat, Horse]  
Removed Element: Cat  
[Dog, Horse]  
Array List with Modified Element: [Dog, Cow]
```

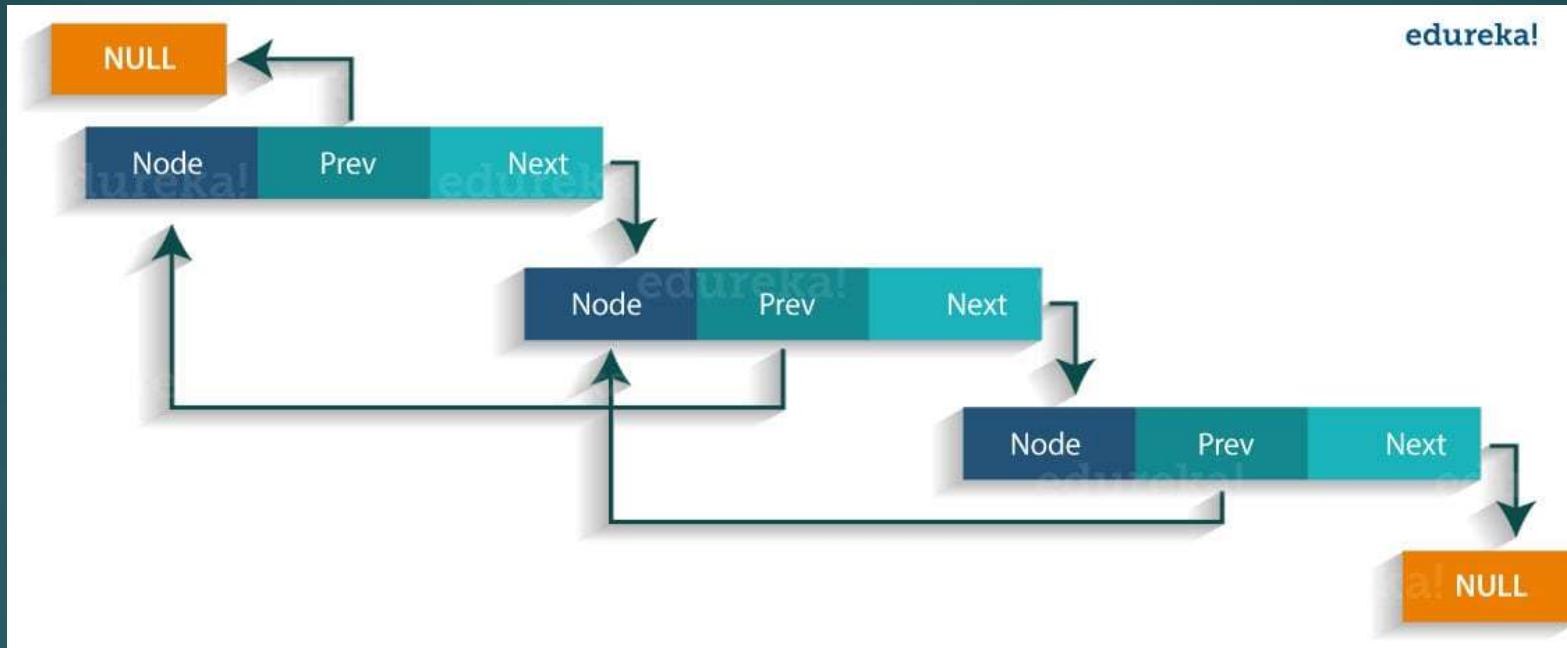
LinkedList

- ▶ Linked List is a sequence of links which contains items. Each link contains a connection to another link.
- ▶ Linked List implements the Collection interface.
- ▶ It uses a doubly linked list internally to store the elements.
- ▶ It can store the duplicate elements.
- ▶ It maintains the insertion order and is not synchronized.
- ▶ In Linked List, the manipulation is fast because no shifting is required.
- ▶ Syntax: `Linkedlist object = new Linkedlist();`
- ▶ Java Linked List class uses two types of Linked list to store the elements:
 1. Singly Linked List: In a singly Linked list each node in this list stores the data of the node and a pointer or reference to the next node in the list. Refer to the below image to get a better understanding of single Linked list.



LinkedList

2. Doubly Linked List: In a doubly Linked list, it has two references, one to the next node and another to previous node. You can refer to the below image to get a better understanding of doubly linked list.



LinkedList

- Some of the methods in the linked list are listed below:

| Method | Description |
|--------------------------------------|---|
| boolean add(Object o) | It is used to append the specified element to the end of the vector. |
| boolean contains(Object o) | Returns true if this list contains the specified element. |
| void add (int index, Object element) | Inserts the element at the specified element in the vector. |
| void addFirst(Object o) | It is used to insert the given element at the beginning. |
| void addLast(Object o) | It is used to append the given element to the end. |
| int size() | It is used to return the number of elements in a list |
| boolean remove(Object o) | Removes the first occurrence of the specified element from this list. |
| int indexOf(Object element) | Returns the index of the first occurrence of the specified element in this list, or -1. |
| int lastIndexOf(Object element) | Returns the index of the last occurrence of the specified element in this list, or -1. |

LinkedList

```
import java.util.*;  
  
class Main {  
  
    public static void main(String[] args) {  
        // Creating list using the LinkedList clas  
  
        List<Integer> numbers = new LinkedList<>();  
  
        // Add elements to the list  
  
        numbers.add(1);  
        numbers.add(2);  
        numbers.add(3);  
  
        System.out.println("List: " + numbers);  
  
        // Access element from the list  
  
        int number = numbers.get(2);  
  
        System.out.println("Accessed Element: " + number);  
  
        // Using the indexOf() method  
  
        int index = numbers.indexOf(2);  
  
        System.out.println("First occurrence of 2 is at index : " + index);  
    }  
}
```

```
// Remove element from the list  
  
int removedNumber = numbers.remove(1);  
  
System.out.println("Removed Element: " + removedNumber);  
  
Iterator itr = numbers.iterator();  
  
System.out.println("Updated List: ");  
  
while(itr.hasNext()){  
    System.out.println(itr.next());  
}  
}  
}  
}
```

```
List: [1, 2, 3]  
Accessed Element: 3  
First occurrence of 2 is at index : 1  
Removed Element: 2  
Updated List:  
1  
3
```

Vector

- ▶ Vectors are similar to arrays, where the elements of the vector object can be accessed via an index into the vector.
- ▶ Vector implements a dynamic array.
- ▶ Also, the vector is not limited to a specific size, it can shrink or grow automatically whenever required. It is similar to ArrayList, but with two differences :
 - ▶ Vector is synchronized.
 - ▶ Vector contains many legacy methods that are not part of the collections framework.
- ▶ Syntax: Vector object = new Vector(size,increment)
- ▶ Below are some of the methods of the Vector class:

| Method | Description |
|-------------------------------------|---|
| boolean add(Object o) | Appends the specified element to the end of the list. |
| void clear() | Removes all of the elements from this list. |
| void add(int index, Object element) | Inserts the specified element at the specified position. |
| boolean remove(Object o) | Removes the first occurrence of the specified element from this list. |
| boolean contains(Object element) | Returns true if this list contains the specified element. |
| int indexOfObject (Object element) | Returns the index of the first occurrence of the specified element in the list, or -1. |
| int size() | Returns the number of elements in this list. |
| int lastIndexOf(Object o) | Return the index of the last occurrence of the specified element in the list, or -1 if the list does not contain any element. |

Vector

- ▶ Vector is synchronized. This means whenever we want to perform some operation on vectors, the Vector class automatically applies a lock to that operation.
- ▶ It is because when one thread is accessing a vector, and at the same time another thread tries to access it, an exception called ConcurrentModificationException is generated. Hence, this continuous use of lock for each operation makes vectors less efficient.
- ▶ However, in array lists, methods are not synchronized. Instead, it uses the Collections.synchronizedList() method that synchronizes the list as a whole.
- ▶ Hence, It is recommended to use ArrayList in place of Vector because vectors less efficient.

```
import java.util.*;
```

```
class Main {
```

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

```
        Vector<String> mammals= new Vector<>();
```

```
        // Using the add() method
```

```
        mammals.add("Dog");
```

```
        mammals.add("Horse");
```

```
        mammals.add(2, "Cat"); // Using index number
```

```
        System.out.println("Vector: " + mammals);
```

```
        // Using addAll()
```

```
        Vector<String> animals = new Vector<>();
```

```
        animals.add("Crocodile");
```

```
        animals.addAll(mammals); // copy mammals vector to animals vector
```

```
        System.out.println("New Vector Animals: " + animals);
```

```
        String element = animals.get(2); // access elements from a vector
```

```
        System.out.println("Element at index 2: " + element);
```

Vector

```
        // Remove Element
```

```
        System.out.println("Removed Element: " + animals.remove(2));
```

```
        System.out.println("New Vector: " + animals);
```

```
        // Using iterator()
```

```
        Iterator<String> iterate = animals.iterator();
```

```
        System.out.print("Vector: ");
```

```
        while(iterate.hasNext()) {
```

```
            System.out.print(iterate.next() + " ");
```

```
}
```

```
        System.out.println();
```

```
        animals.clear(); // Using clear()
```

```
        System.out.println("Vector after clear(): " + animals);
```

```
    }
```

```
Vector: [Dog, Horse, Cat]
```

```
New Vector Animals: [Crocodile, Dog, Horse, Cat]
```

```
Element at index 2: Horse
```

```
Removed Element: Horse
```

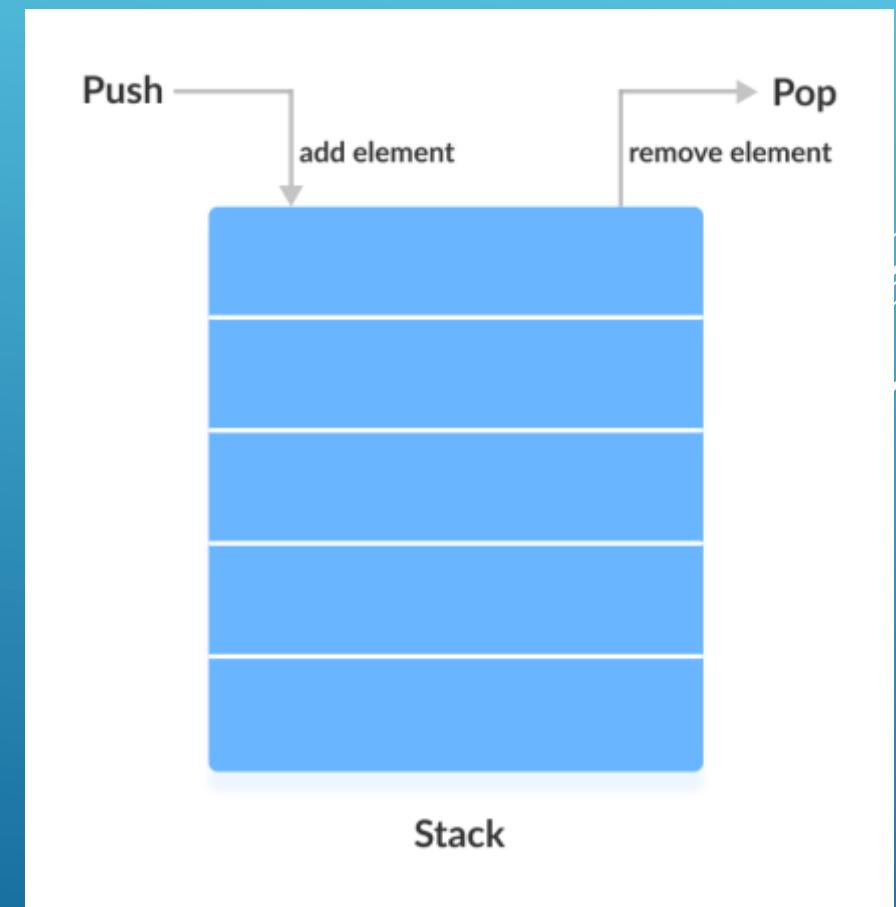
```
New Vector: [Crocodile, Dog, Cat]
```

```
Vector: Crocodile Dog Cat
```

```
Vector after clear(): []
```

Stack

- ▶ The stack is the subclass of Vector.
- ▶ It implements the last-in-first-out data structure i.e. elements are added to the top of the stack and removed from the top of the stack.
- ▶ The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.



Stack

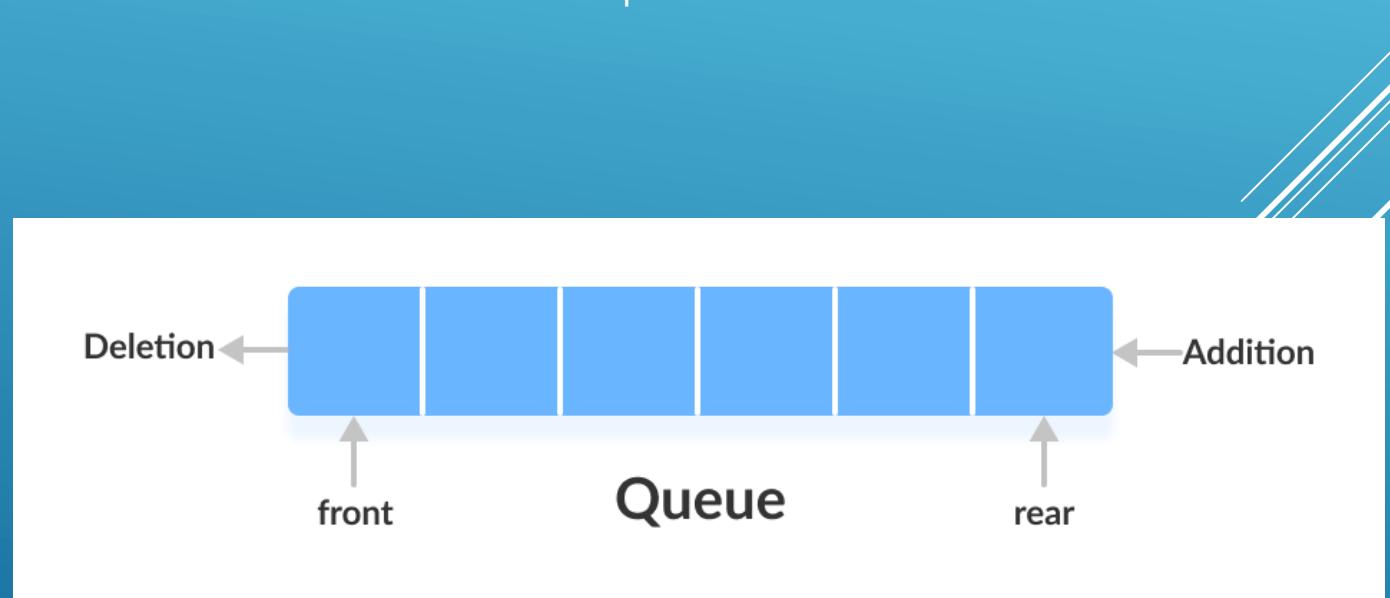
```
import java.util.*;  
  
class Main {  
  
    public static void main(String[] args) {  
        Stack<Integer> nos= new Stack<>();  
  
        // Add elements to Stack using push method  
  
        for(int i=1;i<=10;i++)  
        {  
            nos.push(i);  
        }  
  
        System.out.println("Stack: " + nos);  
  
        // Remove element from stack using pop method  
  
        System.out.println("Removed Element: " + nos.pop());  
  
        System.out.println("Stack after pop : " + nos);  
  
        // Access element from the top  
  
        System.out.println("Element at top: " + nos.peek());  
  
        // Searching an element in the stack  
        // search method returns the position of the element from the top of the stack.  
        // It returns position and not the index  
        int position = nos.search(7);  
        System.out.println("Position of 7 in stack: " + position);  
  
        // Check if stack is empty  
        System.out.println("Is the stack empty? " + nos.empty());  
  
        nos.clear(); // clear the stack  
        System.out.println("Stack : " + nos);  
        System.out.println("Is the stack empty? " + nos.empty());  
    }  
}
```

```
Stack: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
Removed Element: 10  
Stack after pop : [1, 2, 3, 4, 5, 6, 7, 8, 9]  
Element at top: 9  
Position of 7 in stack: 3  
Is the stack empty? false  
Stack : []  
Is the stack empty? true
```

Java Collections : Queue Interface

- ▶ Queue interface maintains the first-in-first-out order.
- ▶ It can be defined as an ordered list that is used to hold the elements which are about to be processed.
- ▶ There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.
- ▶ In a queue, the first element is removed first and last element is removed in the end.
- ▶ Since the Queue is an interface, we cannot provide the direct implementation of it.
- ▶ In order to use the functionalities of Queue, we need to use classes that implement it:
 - ▶ ArrayDeque
 - ▶ PriorityQueue
- ▶ Queue interface can be instantiated as:

```
Queue<String> q1 = new PriorityQueue();
Queue<String> q2 = new ArrayDeque();
```



Java Collections : Queue Interface

- ▶ Some of the commonly used methods of the Queue interface are:
 - ▶ add() - Inserts the specified element into the queue. If the task is successful, add() returns true, if not it throws an exception.
 - ▶ offer() - Inserts the specified element into the queue. If the task is successful, offer() returns true, if not it returns false.
 - ▶ element() - Returns the head of the queue. Throws an exception if the queue is empty.
 - ▶ peek() - Returns the head of the queue. Returns null if the queue is empty.
 - ▶ remove() - Returns and removes the head of the queue. Throws an exception if the queue is empty.
 - ▶ poll() - Returns and removes the head of the queue. Returns null if the queue is empty.

PriorityQueue Class

- ▶ The PriorityQueue class implements the Queue interface.
- ▶ It holds the elements or objects which are to be processed by their priorities.
- ▶ PriorityQueue doesn't allow null values to be stored in the queue.
- ▶ The elements of the priority queue are ordered according to their natural ordering, or by a Comparator provided at the queue construction time.
- ▶ The head of this queue is the least element with respect to the specified ordering.

Priority Queue

```
import java.util.*;  
  
class Main {  
  
    public static void main(String args[]){  
  
        // creating priority queue  
  
        PriorityQueue<String> queue=new PriorityQueue<String>();  
  
        // adding elements  
  
        queue.add("Cat"); // add() Inserts the specified element into the  
queue. If the task is successful, it returns true, if not it throws an  
exception.  
  
        queue.add("Dog");  
  
        queue.offer("Monkey"); // offer() Inserts the specified element into  
the queue. If the task is successful, it returns true, if not it returns false.  
  
        queue.offer("Horse");  
  
        System.out.println("head:"+queue.element()); //element() method  
returns head of the queue. It throws an exception if the queue is empty.  
  
        System.out.println("head:"+queue.peek()); // peek() method returns  
head of the queue. It returns null if the queue is empty.  
  
        System.out.println(queue);
```

queue.remove(); //remove() method returns and removes the head of the queue. Throws an exception if the queue is empty.

queue.poll(); // poll() method returns and removes the head of the queue. Returns null if the queue is empty.

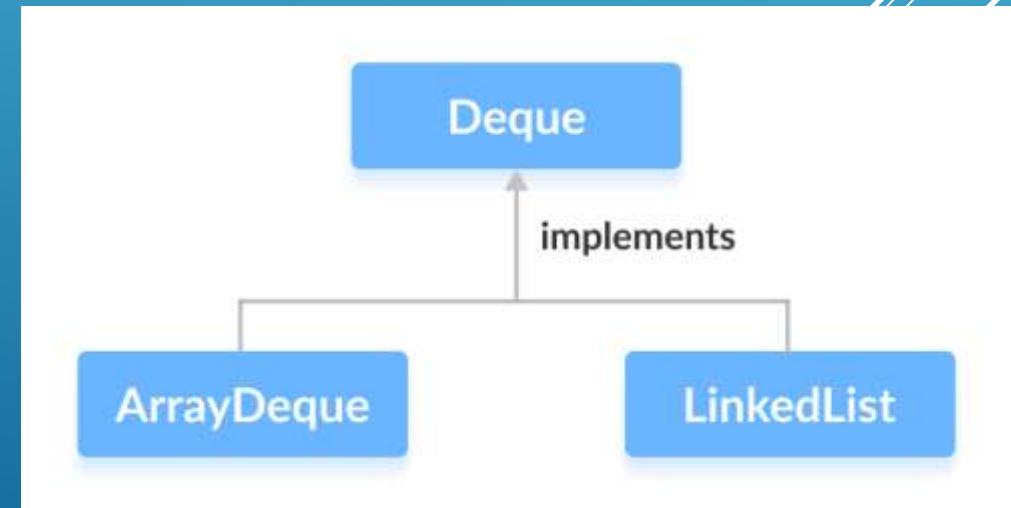
```
System.out.println("after removing two elements:");  
System.out.println(queue);
```

```
System.out.println("Accessed Element: " + queue.peek());  
System.out.println(queue);  
}}
```

```
head:Cat  
head:Cat  
[Cat, Dog, Monkey, Horse]  
after removing two elements:  
[Horse, Monkey]  
Accessed Element: Horse  
[Horse, Monkey]
```

Deque Interface (Double Ended Queue)

- ▶ The Deque interface of the Java collections framework provides the functionality of a double-ended queue.
- ▶ It extends the Queue interface.
- ▶ In a regular queue, elements are added from the rear and removed from the front.
- ▶ However, in a deque, we can insert and remove elements from both front and rear.
- ▶ In order to use the functionalities of the Deque interface, we need to use classes that implement it:
 - ▶ `ArrayDeque`
 - ▶ `LinkedList`
- ▶ `ArrayDeque` is faster than `ArrayList` and `Stack` and has no capacity restrictions.



Deque Interface (Double Ended Queue)

- ▶ **Methods of Deque :** Since Deque extends the Queue interface, it inherits all the methods of the Queue interface.
- ▶ Besides methods available in the Queue interface, the Deque interface also includes the following methods:
 - ▶ addFirst() - Adds the specified element at the beginning of the deque. Throws an exception if the deque is full.
 - ▶ addLast() - Adds the specified element at the end of the deque. Throws an exception if the deque is full.
 - ▶ offerFirst() - Adds the specified element at the beginning of the deque. Returns false if the deque is full.
 - ▶ offerLast() - Adds the specified element at the end of the deque. Returns false if the deque is full.
 - ▶ getFirst() - Returns the first element of the deque. Throws an exception if the deque is empty.
 - ▶ getLast() - Returns the last element of the deque. Throws an exception if the deque is empty.
 - ▶ peekFirst() - Returns the first element of the deque. Returns null if the deque is empty.
 - ▶ peekLast() - Returns the last element of the deque. Returns null if the deque is empty.
 - ▶ removeFirst() - Returns and removes the first element of the deque. Throws an exception if the deque is empty.
 - ▶ removeLast() - Returns and removes the last element of the deque. Throws an exception if the deque is empty.
 - ▶ pollFirst() - Returns and removes the first element of the deque. Returns null if the deque is empty.
 - ▶ pollLast() - Returns and removes the last element of the deque. Returns null if the deque is empty.

```
import java.util.*;
```

```
class Main {
```

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

```
        // Creating Deque using the ArrayDeque class
```

```
        Deque<Integer> numbers = new ArrayDeque<>();
```

```
        // add elements to the Deque
```

```
        numbers.offer(1);
```

```
        numbers.offerLast(2);
```

```
        numbers.offerFirst(3);
```

```
        System.out.println("Deque: " + numbers);
```

```
        // Access elements of the Deque
```

```
        System.out.println("First Element: " + numbers.peekFirst());
```

```
        System.out.println("Last Element: " + numbers.peekLast());
```

```
        // Remove elements from the Deque
```

```
        System.out.println("Removed First Element: " + numbers.pollFirst());
```

```
        System.out.println("Updated Deque: " + numbers);
```

```
}
```

```
}
```

```
Deque: [3, 1, 2]
First Element: 3
Last Element: 2
Removed First Element: 3
Removed Last Element: 2
Updated Deque: [1]
```

Java Collections : Set Interface

- ▶ The Set interface of the Java Collections framework provides the features of the mathematical set in Java.
- ▶ It extends the Collection interface.
- ▶ Unlike the List interface, sets cannot contain duplicate elements.
- ▶ Since Set is an interface, we cannot create objects from it.
- ▶ In order to use functionalities of the Set interface, we can use these classes:
 - ▶ HashSet
 - ▶ LinkedHashSet
 - ▶ EnumSet
 - ▶ TreeSet
- ▶ These classes are defined in the Collections framework and implement the Set interface.
- ▶ Syntax : `Set<String> abc = new HashSet<>();`
- ▶ **Set Operations** : The Java Set interface allows us to perform basic mathematical set operations like -
 - ▶ Union - to get the union of two sets x and y, we can use `x.addAll(y)`
 - ▶ Intersection - to get the intersection of two sets x and y, we can use `x.retainAll(y)`
 - ▶ Subset - to check if x is a subset of y, we can use `y.containsAll(x)`

Java Collections : Set Interface

► Methods of Set

- The Set interface includes all the methods of the Collection interface. It's because Collection is a super interface of Set.
- Some of the commonly used methods of the Collection interface that's also available in the Set interface are:
 - add() - adds the specified element to the set
 - addAll() - adds all the elements of the specified collection to the set
 - iterator() - returns an iterator that can be used to access elements of the set sequentially
 - remove() - removes the specified element from the set
 - removeAll() - removes all the elements from the set that is present in another specified set
 - retainAll() - retains all the elements in the set that are also present in another specified set
 - clear() - removes all the elements from the set
 - size() - returns the length (number of elements) of the set
 - toArray() - returns an array containing all the elements of the set
 - contains() - returns true if the set contains the specified element
 - containsAll() - returns true if the set contains all the elements of the specified collection
 - hashCode() - returns a hash code value (address of the element in the set)

HashSet Class

- ▶ Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.
- ▶ The important points about Java HashSet class are:
 - ▶ HashSet stores the elements by using a mechanism called hashing.
 - ▶ HashSet contains unique elements only.
 - ▶ HashSet allows null value.
 - ▶ HashSet class is non synchronized.
 - ▶ HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashCode.
 - ▶ HashSet is the best approach for search operations.
 - ▶ The initial default capacity of HashSet is 16, and the load factor is 0.75.
- ▶ Syntax : HashSet<Integer> numbers = new HashSet<>(8, 0.75);
- ▶ The above syntax creates HashSet with 8 capacity and 0.75 load factor
- ▶ HashSet is commonly used if we have to access elements randomly. It is because elements in a hash table are accessed using hash codes.
- ▶ The hashCode of an element is a unique identity that helps to identify the element in a hash table.
- ▶ HashSet cannot contain duplicate elements. Hence, each hash set element has a unique hashCode.



```
class Main {  
  
    public static void main(String[] args) {  
  
        HashSet<Integer> evenNumber = new HashSet<>();  
  
        // Using add() method  
  
        evenNumber.add(2);  
  
        evenNumber.add(4);  
  
        evenNumber.add(6);  
  
        System.out.println("HashSet evenNumber: " + evenNumber);  
  
        // Using addAll() method  
  
        HashSet<Integer> numbers = new HashSet<>();  
  
        numbers.addAll(evenNumber); // aka Union operation. Union of  
        numbers and evenNumber sets  
  
        numbers.add(5);  
  
        System.out.println("New HashSet numbers: " + numbers);  
  
        //to check if a set is a subset of another set or not, we can use the  
        containsAll() method.  
  
        boolean result = numbers.containsAll(evenNumber);  
  
        //using remove() method  
        numbers.remove(4);  
  
        System.out.println("HashSet numbers after removing an  
        element : " + numbers);  
  
        //intersection of two sets  
        numbers.retainAll(evenNumber);  
  
        System.out.println("Intersection is: " + numbers);  
  
        numbers.add(5);  
  
        //difference of two sets  
        numbers.removeAll(evenNumber);  
  
        System.out.println("Difference is: " + numbers);  
  
        //using removeAll() method  
        numbers.removeAll(numbers);  
  
        System.out.println("HashSet after removing all the elements  
        : " + numbers);  
    }  
}
```

HashSet Class

HashSet Class

```
HashSet evenNumber: [2, 4, 6]
New HashSet numbers: [2, 4, 5, 6]
Is evenNumber subset of numbers? true
HashSet numbers after removing an element : [2, 5, 6]
Intersection is: [2, 6]
Difference is: [5]
HashSet after removing all the elements : []
```

LinkedHashSet Class

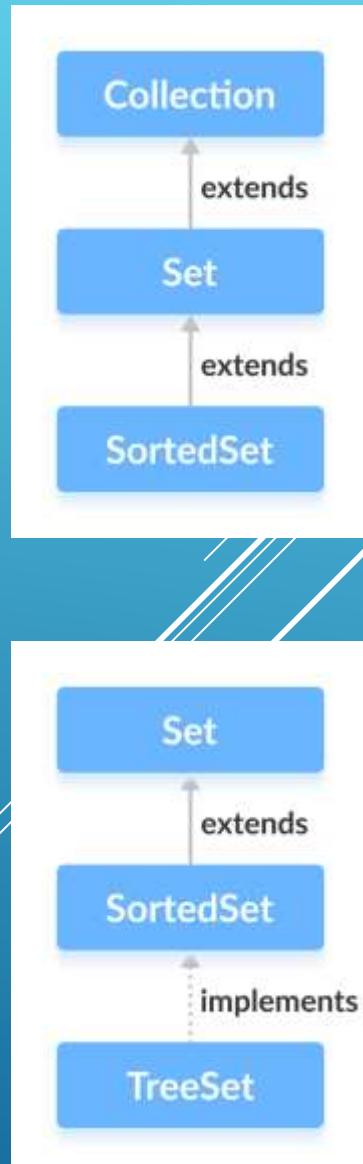
- ▶ Java LinkedHashSet class is a Hashtable and Linked list implementation of the Set interface. It inherits the HashSet class and implements the Set interface.
- ▶ The important points about the Java LinkedHashSet class are:
 - ▶ Java LinkedHashSet class contains unique elements only like HashSet.
 - ▶ Java LinkedHashSet class provides all optional set operations and permits null elements.
 - ▶ Java LinkedHashSet class is non-synchronized.
 - ▶ Java LinkedHashSet class maintains insertion order.
- ▶ linked hash sets maintain a doubly-linked list internally for all of its elements. The linked list defines the order in which elements are inserted in hash tables.

```
class Main {  
  
    public static void main(String[] args) {  
  
        // Creating an arrayList of even numbers  
  
        ArrayList<Integer> evenNumbers = new ArrayList<>();  
  
        evenNumbers.add(2);  
  
        evenNumbers.add(4);  
  
        System.out.println("ArrayList: " + evenNumbers);  
  
        // Creating a LinkedHashSet from an ArrayList  
  
        LinkedHashSet<Integer> numbers = new  
        LinkedHashSet<>(evenNumbers);  
  
        numbers.add(5);  
  
        numbers.add(6);  
  
        numbers.add(3);  
  
        System.out.println("LinkedHashSet: " + numbers);  
  
        numbers.remove(5);  
  
        System.out.println("LinkedHashSet numbers after deleting an  
        element : " + numbers);  
  
        evenNumbers.retainAll(numbers);  
  
        System.out.println("Intersection is : " + evenNumbers);  
  
        numbers.removeAll(evenNumbers);  
  
        System.out.println("Difference : " + numbers);  
  
        boolean result = numbers.containsAll(evenNumbers);  
  
        System.out.println("Is LinkedHashSet2 subset of LinkedHashSet1? " +  
        result);  
    }  
}
```

ArrayList: [2, 4]
LinkedHashSet: [2, 4, 5, 6, 3]
LinkedHashSet numbers after deleting an element : [2, 4, 6, 3]
Intersection is : [2, 4]
Difference : [6, 3]
Is LinkedHashSet2 subset of LinkedHashSet1? false

SortedSet Interface

- The SortedSet interface of the Java Collections framework is used to store elements with some order in a set.
- It extends the Set interface.
- In order to use the functionalities of the SortedSet interface, we need to use the TreeSet class that implements it.
- Syntax : `SortedSet<String> abc=new TreeSet<>();`
- Here we have used no arguments to create a sorted set. Hence the set will be sorted naturally.
- **Methods of SortedSet**
- The SortedSet interface includes all the methods of the Set interface. It's because Set is a super interface of SortedSet. Besides it, the SortedSet interface also includes these methods:
 - `comparator()` - returns a comparator that can be used to order elements in the set
 - `first()` - returns the first element of the set
 - `last()` - returns the last element of the set
 - `headSet(element)` - returns all the elements of the set before the specified element
 - `tailSet(element)` - returns all the elements of the set after the specified element including the specified element
 - `subSet(element1, element2)` - returns all the elements between the element1 and element2 including element1



TreeSet Class

- ▶ Java TreeSet class implements the Set interface that uses a tree for storage. The objects of the TreeSet class are stored in ascending order.
- ▶ The important points about the Java TreeSet class are:
 - ▶ Java TreeSet class contains unique elements only like HashSet.
 - ▶ Java TreeSet class access and retrieval times are quite fast.
 - ▶ Java TreeSet class doesn't allow null element.
 - ▶ Java TreeSet class is non synchronized.
 - ▶ Java TreeSet class maintains ascending order.
 - ▶ The TreeSet can only allow those generic types that are comparable.
- ▶ Syntax : TreeSet<Integer> abc=new TreeSet<>();
- ▶ TreeSet is being implemented using a binary search tree

TreeSet Class

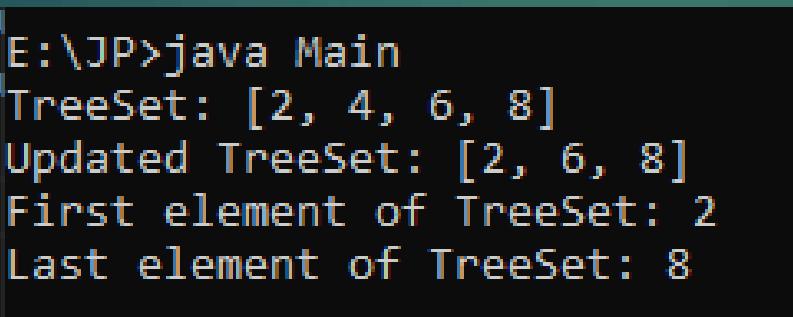
- ▶ **Methods of TreeSet :** Apart from the methods of SortedSet, Set and Collections interface, the TreeSet interface also includes these methods:
 - ▶ `higher(element)` - Returns the lowest element among those elements that are strictly greater than the specified element or else returns NULL
 - ▶ `lower(element)` - Returns the greatest element among those elements that are strictly less than the specified element or else returns NULL
 - ▶ `ceiling(element)` - Returns the lowest element among those elements that are greater than or equal to the specified element. If the element passed exists in a tree set, it returns the element passed as an argument or else returns NULL.
 - ▶ `floor(element)` - Returns the greatest element among those elements that are less than or equal to the specified element. If the element passed exists in a tree set, it returns the element passed as an argument or else returns NULL.
 - ▶ `pollFirst()` - returns and removes the first element from the set
 - ▶ `pollLast()` - returns and removes the last element from the set

TreeSet Class

- ▶ headSet(element, booleanValue) - The headSet() method returns all the elements of a tree set before the specified element (which is passed as an argument). The booleanValue parameter is optional. Its default value is false. If true is passed as a booleanValue, the method returns all the elements before the specified element **including the specified element**.
- ▶ tailSet(element, booleanValue) - The tailSet() method returns all the elements of a tree set after the specified element (which is passed as a parameter) including the specified element. The booleanValue parameter is optional. Its default value is true. If false is passed as a booleanValue, the method returns all the elements after the specified element **without including the specified element**.
- ▶ subSet(e1, bv1, e2, bv2) - The subSet() method returns all the elements between e1 and e2 including e1. The bv1 and bv2 are optional parameters. The default value of bv1 is true, and the default value of bv2 is false. If false is passed as bv1, the method returns all the elements between e1 and e2 without including e1. If true is passed as bv2, the method returns all the elements between e1 and e2, including e1.

TreeSet Class

```
import java.util.*;  
class Main {  
    public static void main(String[] args) {  
        TreeSet<Integer> evenNumbers = new TreeSet<>();  
        // Using the add() method  
        evenNumbers.add(4);  
        evenNumbers.add(2);  
        evenNumbers.add(6);  
        evenNumbers.add(6);  
        evenNumbers.add(8);  
        System.out.println("TreeSet: " + evenNumbers);  
        evenNumbers.remove(4);  
        System.out.println("Updated TreeSet: " + evenNumbers);  
        System.out.println("First element of TreeSet: " +  
evenNumbers.first());  
        System.out.println("Last element of TreeSet: " +  
evenNumbers.last());  
        // Using higher()  
        System.out.println("Using higher: " + evenNumbers.higher(4));  
        // Using lower()  
        System.out.println("Using lower: " + evenNumbers.lower(4));  
        // Using ceiling()  
        System.out.println("Using ceiling: " + evenNumbers.ceiling(4));  
        // Using floor()  
        System.out.println("Using floor: " + evenNumbers.floor(3));  
        // Using pollFirst()  
        System.out.println("Removed First Element: " +  
evenNumbers.pollFirst());  
        // Using pollLast()  
        System.out.println("Removed Last Element: " +  
evenNumbers.pollLast());  
        System.out.println("TreeSet after Polling: " + evenNumbers);  
        evenNumbers.add(10);  
        evenNumbers.add(12);  
        evenNumbers.add(14);  
        System.out.println("New TreeSet: " + evenNumbers);  
    }  
}
```



(code further continued on slide 58)

TreeSet Class

```
Using higher: 6
Using lower: 2
Using ceiling: 6
Using floor: 2
Removed First Element: 2
Removed Last Element: 8
TreeSet after Polling: [6]
New TreeSet: [6, 10, 12, 14]
```

TreeSet Class

```
System.out.println("New TreeSet: " + evenNumbers);  
//using subSet()  
  
// using headSet()  
  
// Using headSet() with default boolean value  
System.out.println("Using headSet without boolean value: " +  
evenNumbers.headSet(12));  
  
// Using headSet() with specified boolean value  
System.out.println("Using headSet with boolean value: " +  
evenNumbers.headSet(12, true));  
  
// using tailSet()  
  
// Using tailSet() with default boolean value  
System.out.println("Using tailSet without boolean value: " +  
evenNumbers.tailSet(12));  
  
// Using tailSet() with specified boolean value  
System.out.println("Using tailSet with boolean value: " +  
evenNumbers.tailSet(12, false));  
  
// Using subSet() with default boolean value  
System.out.println("Using subSet without boolean value: " +  
evenNumbers.subSet(6, 12));  
  
// Using subSet() with specified boolean value  
System.out.println("Using subSet with boolean value: " +  
evenNumbers.subSet(6, false, 12, true));  
}  
}
```

(code further continued on slide 59)

```
New TreeSet: [6, 10, 12, 14]  
Using headSet without boolean value: [6, 10]  
Using headSet with boolean value: [6, 10, 12]  
Using tailSet without boolean value: [12, 14]  
Using tailSet with boolean value: [14]  
Using subSet without boolean value: [6, 10]  
Using subSet with boolean value: [10, 12]
```

TreeSet Class

```
// Union of two sets  
TreeSet<Integer> numbers = new TreeSet<>();  
numbers.addAll(evenNumbers);  
System.out.println("Union is: " + numbers);  
numbers.add(3);  
numbers.add(5);  
System.out.println("Updated numbers TreeSet is: " + numbers);  
  
// Intersection of two sets  
numbers.retainAll(evenNumbers);  
System.out.println("Intersection is: " + numbers);  
numbers.add(3);  
numbers.add(5);  
  
// Difference between two sets  
numbers.removeAll(evenNumbers);  
System.out.println("Difference is: " + numbers);  
}  
}
```

```
Union is: [6, 10, 12, 14]  
Updated numbers TreeSet is: [3, 5, 6, 10, 12, 14]  
Intersection is: [6, 10, 12, 14]  
Difference is: [3, 5]
```

TreeSet Class

- By default, tree set elements are sorted naturally i.e in ascending order. However, we can also traverse the set in descending order

```
import java.util.*;  
class Main{  
    public static void main(String args[]){  
        //Creating and adding elements  
        TreeSet<String> animals=new TreeSet<String>();  
        animals.add("Dog");  
        animals.add("Cat");  
        animals.add("Monkey");  
        animals.add("Elephant");  
        animals.add("Cow");  
  
        //Traversing elements in ascending order  
        Iterator<String> itr=animals.iterator();  
        System.out.println("Ascending Order :");  
        while(itr.hasNext()){  
            System.out.print(itr.next() + " ");  
        }  
        System.out.println("\n");  
  
        //Traversing elements in descending order  
        Iterator<String> des=animals.descendingIterator();  
        System.out.println("Descending Order :");  
        while(des.hasNext()){  
            System.out.print(des.next() + " ");  
        }  
    }  
}
```

Ascending Order :
Cat Cow Dog Elephant Monkey

Descending Order :
Monkey Elephant Dog Cow Cat

LinkedHashSet vs HashSet

- ▶ Both LinkedHashSet and HashSet implements the Set interface. However, there exist some differences between them.
 - ▶ LinkedHashSet maintains a linked list internally. Due to this, it maintains the insertion order of its elements.
 - ▶ The LinkedHashSet class requires more storage than HashSet. This is because LinkedHashSet maintains linked lists internally.
 - ▶ The performance of LinkedHashSet is slower than HashSet. It is because of linked lists present in LinkedHashSet.

LinkedHashSet Vs. TreeSet

- ▶ Here are the major differences between LinkedHashSet and TreeSet:
 - ▶ The TreeSet class implements the SortedSet interface. That's why elements in a tree set are sorted. However, the LinkedHashSet class only maintains the insertion order of its elements.
 - ▶ A TreeSet is usually slower than a LinkedHashSet. It is because whenever an element is added to a TreeSet, it has to perform the sorting operation.
 - ▶ LinkedHashSet allows the insertion of null values. However, we cannot insert a null value to TreeSet.

TreeSet vs HashSet

- ▶ Both the TreeSet as well as the HashSet implements the Set interface. However, there exist some differences between them.
- ▶ Unlike HashSet, elements in TreeSet are stored in some order. It is because TreeSet implements the SortedSet interface as well.
- ▶ TreeSet provides some methods for easy navigation. For example, first(), last(), headSet(), tailSet(), etc. It is because TreeSet also implements the NavigableSet interface.
- ▶ HashSet is faster than the TreeSet for basic operations like add, remove, contains and size.

Programming Practice Questions

- ▶ Implement a generic stack class GenericStack<T> that supports the following operations:
 - ▶ push(T item): Add an item to the stack.
 - ▶ pop(): Remove and return the top item from the stack.
 - ▶ peek(): Return the top item without removing it.
 - ▶ isEmpty(): Check if the stack is empty.
- ▶ Extend the above program to implement a generic stack class BoundedStack that can only hold numbers.
- ▶ Write a program that creates a List of integers. Populate the list with the first 10 positive integers, then:
 - ▶ Print the list.
 - ▶ Remove the third element from the list.
 - ▶ Add the number 100 at the end of the list.
 - ▶ Print the updated list.
- ▶ Write a program that accepts a List<Double> of floating-point numbers and sorts the list in descending order. Print the sorted list.
- ▶ Write a method that merges two List<Integer> objects into a single list. The resulting list should contain all elements from both lists, but without any duplicates.
- ▶ Write a program that creates a Set<Integer> and adds the first 10 positive integers. Then, attempt to add a duplicate integer. Print the set to show that duplicates are not allowed.
- ▶ WAP that creates two Set<Integer> objects and
 - ▶ returns a new set that contains the union of both sets.
 - ▶ returns a new set containing only the elements that are present in both sets.
 - ▶ returns a new set containing the difference of both sets.
 - ▶ check if one Set is a subset of another Set. Return true if it is, and false otherwise.

JAVA PROGRAMMING

Chap 8 : GUI Design using
Java FX

Java FX

- ▶ JavaFX is a Java library and a GUI toolkit designed to develop and facilitate Rich Internet applications, web applications, and desktop applications.
- ▶ Rich Internet Applications are those web applications that allow alike characteristics and expertise as that of desktop applications. These applications contribute more satisfying visual experience to the users when compared to the standard web applications.
- ▶ The most significant perk of using JavaFX is that the applications written using this library can run on multiple operating systems like Windows, Linux, iOS, Android, and several platforms like Desktops, Web, Mobile Phones, TVs, Tablets, etc.
- ▶ This characteristic of the JavaFX library makes it very versatile across operating systems and different platforms.
- ▶ After the arrival of JavaFX, Java developers and programmers were able to develop the GUI applications more effectively and more productively.
- ▶ JavaFX was introduced to supersede the Java Swing GUI framework.
- ▶ Nevertheless, the JavaFX provides more enhanced functionalities and features than the Java Swing.

Steps to configure Java FX on Netbeans & Eclipse :

https://www.tutorialspoint.com/javafx/javafx_environment.htm

Features of Java FX

Java Library – JavaFX is a Java library, which allows the user to gain the support of all the Java characteristics such as multithreading, generics, lambda expressions, and many more. The user can also use any of the Java editors or IDE's of their choice, such as Eclipse, NetBeans, to write, compile, run, debug, and package their JavaFX application.

Platform Independent – The rich internet applications made using JavaFX are platform-independent. The JavaFX library is open for all the scripting languages that can be administered on a JVM, which comprise – Java, Groovy, Scala, and JRuby.

FXML – JavaFX emphasizes an HTML-like declarative markup language known as FXML. FXML is based on extensible markup language (XML). The sole objective of this markup language is to specify a user interface (UI). In FXML, the programming can be done to accommodate the user with an improved GUI.

Scene Builder – JavaFX also implements a tool named Scene Builder, which is a visual editor for FXML. Scene Builder generates FXML mark-ups that can be transmitted to the IDE's like Eclipse and NetBeans, which further helps the user to combine the business logic to their applications. The users can also use the drag and drop design interface, which is used to design FXML applications.

Hardware-accelerated Graphics Pipeline – The graphics of the JavaFX applications are based on the hardware-accelerated graphics rendering pipeline, commonly known as Prism. The Prism engine offers smooth JavaFX graphics that can be rendered quickly when utilized with a supported graphics card or graphics processing unit (GPU). In the case where the system does not hold the graphic cards, then the prism engine defaults to the software rendering stack.

Swing Interoperability – In a JavaFX application, you can embed Swing content using the Swing Node class. Similarly, you can also update the existing Swing applications with JavaFX features.

Features of Java FX

WebView – JavaFX applications can also insert web pages. To embed web pages, Web View of JavaFX uses a new HTML rendering engine technology known as WebKitHTML. WebView is used to make it possible to insert web pages within a JavaFX application. JavaScript running in WebView can call Java APIs and vice-versa.

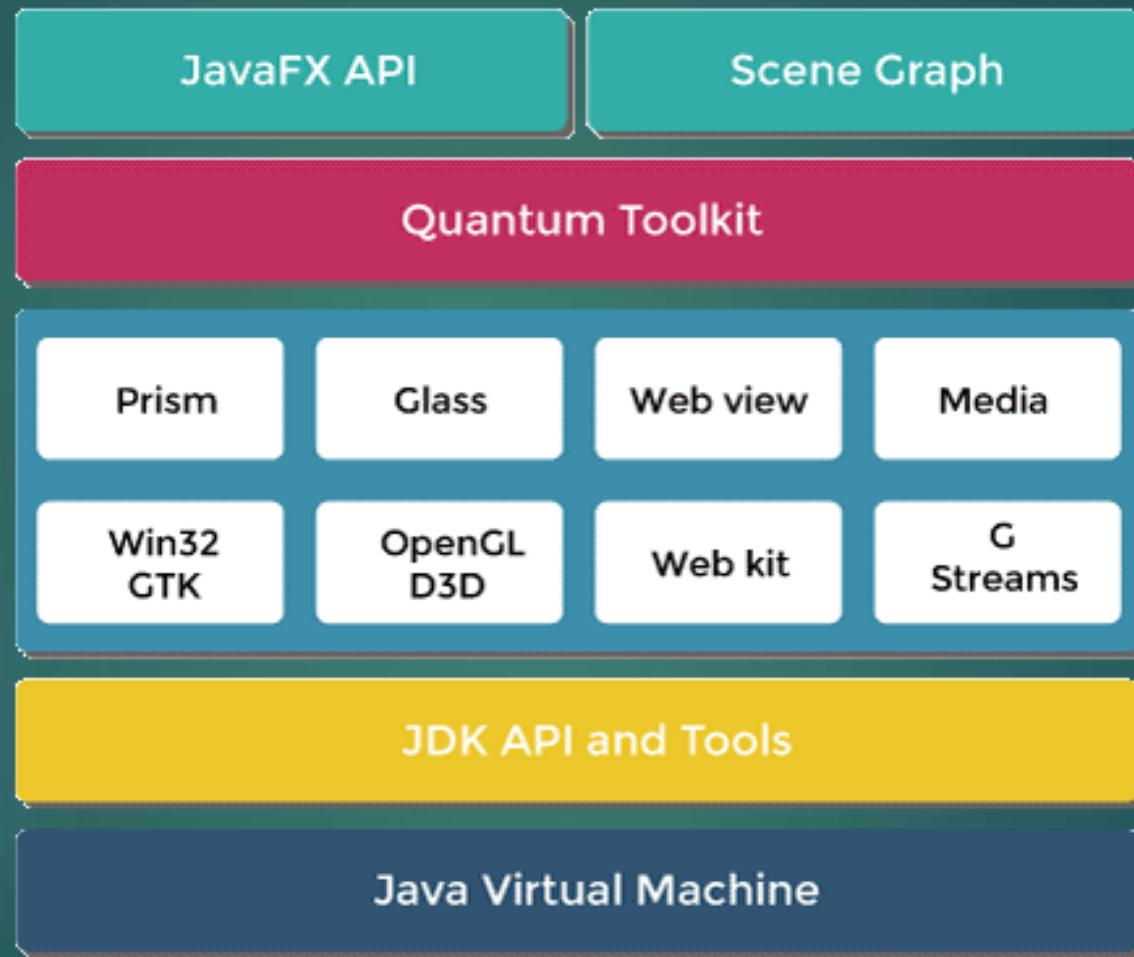
Built-in UI controls – JavaFX comprises all the major built-in UI controls that help in developing well-featured applications. These built-in UI components are not operating system-dependent. In simple words, these controls do not depend on any of the Operating systems like Windows, iOS, Android, etc. These built-in controls are single-handedly ample to perform a whole implementation of the applications.

CSS Styling – Just like websites use CSS for styling, JavaFX also provides the feature to integrate the application with CSS styling. The users can enhance the styling of their applications and can also improve the outlook of their implementation by having simple knowledge and understanding of CSS styling.

Rich set of APIs – JavaFX library also presents a valuable collection of APIs that helps in developing GUI applications, 2D and 3D graphics, and many more. This collection of APIs also includes all the characteristics of the Java platform. Hence, working with this API, a user can access the specialties of Java languages such as Generics, Annotations, Multithreading, and Lambda Expressions, and many other features as well. In JavaFX, the popular Java Collections library was also improved, and notions like lists and maps were introduced. Using these APIs, the users can witness the variations in the data models.

High-Performance media engine – Like the graphics pipeline, JavaFX also possesses a media pipeline that advances stable internet multimedia playback at low latency. This high-performance media engine or media pipeline is based on a multimedia framework known as Gstreamer.

Architecture of Java FX



Architecture of Java FX

- ▶ **JavaFX API** – The topmost layer of JavaFX architecture holds a JavaFX public API that implements all the required classes that are capable of producing a full-featured JavaFX application with rich graphics. The list of all the important packages of this API is as follows.
 - ▶ javafx.animation: It includes classes that are used to combine transition-based animations such as fill, fade, rotate, scale and translation, to the JavaFX nodes (collection of nodes makes a scene graph).
 - ▶ javafx.css – It comprises classes that are used to append CSS-like styling to the JavaFX GUI applications.
 - ▶ javafx.geometry – It contains classes that are used to represent 2D figures and execute methods on them.
 - ▶ javafx.scene – This package of JavaFX API implements classes and interfaces to establish the scene graph. In extension, it also renders sub-packages such as canvas, chart, control, effect, image, input, layout, media, paint, shape, text, transform, web, etc. These are the diverse elements that sustain this precious API of JavaFX.
 - ▶ javafx.application – This package includes a collection of classes that are responsible for the life cycle of the JavaFX application.
 - ▶ javafx.event – It includes classes and interfaces that are used to perform and manage JavaFX events.
 - ▶ javafx.stage – This package of JavaFX API accommodates the top-level container classes used for the JavaFX application.
- ▶ **Scene Graph** – A Scene Graph is the starting point of the development of any of the GUI Applications. In JavaFX, all the GUI Applications are made using a Scene Graph only. The Scene Graph includes the primitives of the rich internet applications that are known as nodes. In simple words, **a single component in a scene graph is known as a node**. In general, a scene graph is made up of a collection of nodes. All these nodes are organized in the form of a hierarchical tree that describes all of the visual components of the application's user interface (UI). A node instance can be appended to a scene graph only once.

Architecture of Java FX

- ▶ A node is a visual/graphical object and it may include –
 - ▶ Geometrical (Graphical) objects – (2D and 3D) such as circle, rectangle, polygon, etc.
 - ▶ UI controls – such as Button, Checkbox, Choice box, Text Area, etc.
 - ▶ Containers – (layout panes) such as Border Pane, Grid Pane, Flow Pane, etc.
 - ▶ Media elements – such as audio, video and image objects.
- ▶ The nodes are of three general types.
 - ▶ Root Node – A root node is a node that does not have any node as its parent.
 - ▶ Leaf Node – A leaf node is a node that does not contain any node as its children.
 - ▶ Branch Node – A branch node is a node that contains a node as its parent and also has a node as its children.
- ▶ **Quantum Toolkit** – Quantum Toolkit is used to connect prism and glass windowing tool kits collectively and prepares them for the above layers in the stack. In simple words, it ties Prism and GWT together and makes them available to JavaFX.
- ▶ **Prism** – The graphics of the JavaFX applications are based on the hardware-accelerated graphics rendering pipeline, commonly known as Prism. The Prism engine supports smooth JavaFX graphics that can be executed swiftly when utilized with a backed graphics card or graphics processing unit (GPU). In the situation where the system does not contain the graphic cards, then the prism engine defaults to the software rendering stack. To render graphics, a Prism uses –
 - ▶ DirectX 9 on Windows XP and Vista.
 - ▶ DirectX 11 on Windows 7.
 - ▶ OpenGL on Mac and Linux, Embedded Systems.

Architecture of Java FX

- ▶ **Glass Windowing Toolkit** – Glass Windowing Toolkit or simply Glass is a platform-dependent layer that assists in connecting the JavaFX platform to the primary operating system (OS). Glass Windowing Toolkit is very useful as it provides services such as controlling the windows, events, timers, and surfaces to the native operating system.
- ▶ **WebView** – JavaFX applications can also insert web pages. To embed web pages, Web View of JavaFX uses a new HTML rendering engine technology known as WebKitHTML. WebView is used to make it possible to insert web pages within a JavaFX application. JavaScript appearing in WebView can call Java APIs and vice-versa. This element promotes different web technologies like HTML5, CSS, JavaScript, DOM, and SVG. Using web view, we can execute the HTML content from the JavaFX application and can also implement some CSS styles to the user interface (UI) part of the application. Using WebView, you can –
 - ▶ Render HTML content from local or remote URL.
 - ▶ Support history and provide Back and Forward navigation.
 - ▶ Reload the content.
 - ▶ Apply effects to the web component.
 - ▶ Edit the HTML content.
 - ▶ Execute JavaScript commands.
 - ▶ Handle events.

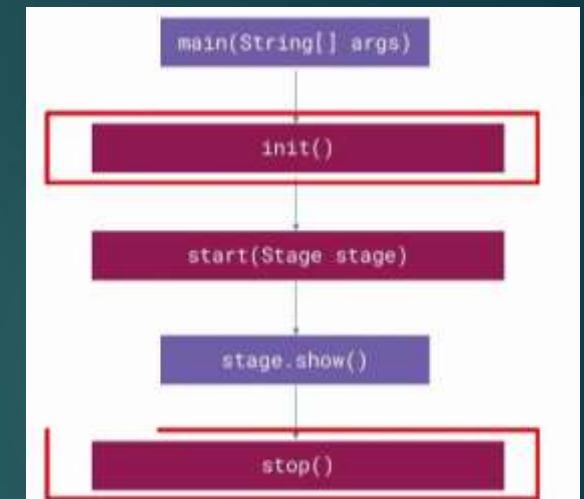
Media Engine – Like the graphics pipeline, JavaFX also possesses a media pipeline that advances stable internet multimedia playback at low latency. This high-performance media engine or media pipeline is based on a multimedia framework known as Gstreamer. By applying the Media engine, the JavaFX application can support the playback of audio and video media files. The package `javafx.scene.media` covers all the classes and interfaces that can provide media functionalities to JavaFX applications. It is provided in the form of three components, which are –
Media Object – This represents a media file, Media Player – To play media content, Media View – To display media.

LifeCycle of Java FX Application

- We need to import `javafx.application.Application` class in every JavaFX application.
- This provides the following life cycle methods for JavaFX application.

```
public void init()  
public abstract void start(Stage primaryStage)  
public void stop()
```

- **init()** – The `init()` method is an empty method that can be overridden. In this method, the user cannot create a stage or a scene.
- **start()** – The `start()` method is the entry point method of the JavaFX application where all the graphics code of JavaFX is to be written.
- **stop()** – The `stop()` method is an empty method that can also be overridden, just like the `init()` method. In this method, the user can write the code to halt the application.
- Other than these methods, the JavaFX application also implements a static method known as **launch()**. This `launch()` method is used to launch the JavaFX application. As stated earlier, the `launch()` method is static, the user should call it from a static method only. Generally, that static method, which calls the `launch()` method, is the `main()` method only.



LifeCycle of Java FX Application

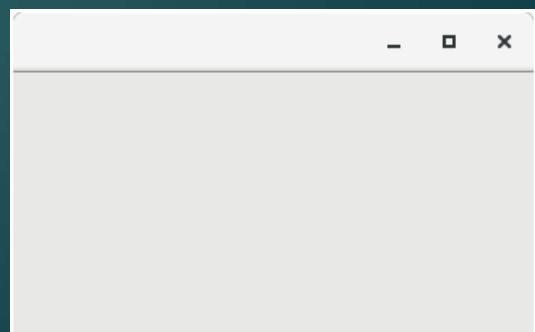
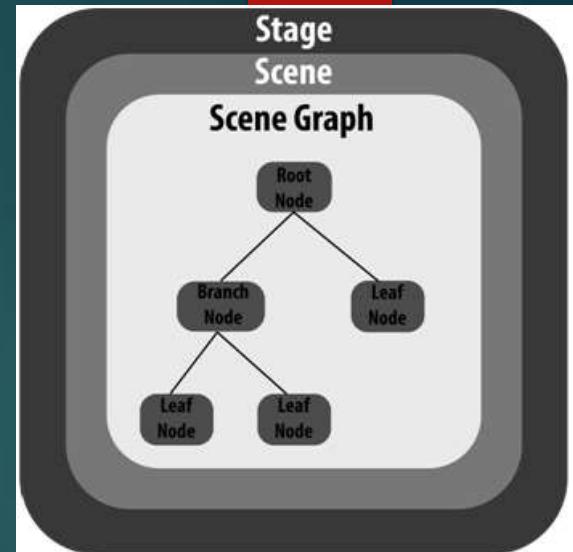
- ▶ Whenever a user launches a JavaFX application, there are few actions that are carried out in a particular manner only.
- ▶ The following is the order given in which a JavaFX application is launched.
 - ▶ Firstly, an instance of the application class is created.
 - ▶ After that, the init() method is called.
 - ▶ After the init() method, the start() method is called.
 - ▶ After calling the start() method, the launcher waits for the JavaFX application to end and then calls the stop() method.
- ▶ in order to create a basic JavaFX application, we need to:
 - Import javafx.application.Application into our code.
 - Inherit Application into our class.
 - Override start() method of Application class.

Application Structure of Java FX

- ▶ JavaFX application is divided hierarchically into three main components known as Stage, Scene and nodes.

Stage

- ▶ Stage in a JavaFX application is similar to the Frame in a Swing Application.
- ▶ It acts like a container for all the JavaFX objects.
- ▶ Primary Stage is created internally by the platform.
- ▶ Other stages can further be created by the application.
- ▶ The object of primary stage is passed to start method.
- ▶ We need to call show method on the primary stage object in order to show our primary stage.
- ▶ Initially, the primary Stage looks like following.
- ▶ However, we can add various objects to this primary stage.
- ▶ The objects can only be added in a hierarchical way i.e. first, scene graph will be added to this primaryStage and then that scene graph may contain the nodes.
- ▶ A node may be any object of the user's interface like text area, buttons, shapes, media, etc.
- ▶ It is represented by Stage class of the package javafx.stage.
- ▶ A stage has two parameters determining its position namely Width and Height. It is divided as Content Area and Decorations (Title Bar and Borders).
- ▶ You have to call the show() method to display the contents of a stage.



Application Structure of Java FX

Scene

- ▶ A scene represents the physical contents of a JavaFX application.
- ▶ It contains all the contents of a scene graph.
- ▶ The class Scene of the package javafx.scene represents the scene object. **JavaFx.Scene.Scene** class provides all the methods to deal with a scene object.
- ▶ At an instance, the scene object is added to only one stage.
- ▶ You can create a scene by instantiating the Scene Class.
- ▶ You can opt for the size of the scene by passing its dimensions (height and width) along with the root node to its constructor.
- ▶ Creating scene is necessary in order to visualize the contents on the stage.
- ▶ In order to implement Scene in our JavaFX application, we must import javafx.scene package in our code.

Application Structure of Java FX

Scene Graph

- ▶ A scene graph is a tree-like data structure (hierarchical) representing the contents of a scene.
- ▶ In contrast, a node is a visual/graphical object of a scene graph. A node is the element which is visualized on the stage. It can be any button, text box, layout, image, radio button, check box, etc.
- ▶ A node may include –
 - ▶ Geometrical (Graphical) objects (2D and 3D) such as – Circle, Rectangle, Polygon, etc.
 - ▶ UI Controls such as – Button, Checkbox, Choice Box, Text Area, etc.
 - ▶ Containers (Layout Panes) such as Border Pane, Grid Pane, Flow Pane, etc.
 - ▶ Media elements such as Audio, Video and Image Objects. It can be seen as the collection of various nodes.
- ▶ The nodes are implemented in a tree kind of structure.
- ▶ There is always one root in the scene graph. This will act as a parent node for all the other nodes present in the scene graph. However, this node may be any of the layouts available in the JavaFX system.
- ▶ The leaf nodes exist at the lowest level in the tree hierarchy.
- ▶ Each of the node present in the scene graphs represents classes of `javafx.scene` package therefore we need to import the package into our application in order to create a full featured javafx application.

Application Structure of Java FX

Scene Graph

- ▶ As discussed earlier a node is of three types –
- ▶ **Root Node** – The first Scene Graph is known as the Root node.
- ▶ **Branch Node/Parent Node** – The node with child nodes are known as branch/parent nodes. The abstract class named Parent of the package javafx.scene is the base class of all the parent nodes, and those parent nodes will be of the following types –
 - ▶ Group – A group node is a collective node that contains a list of children nodes. Whenever the group node is rendered, all its child nodes are rendered in order. Any transformation, effect state applied on the group will be applied to all the child nodes.
 - ▶ Region – It is the base class of all the JavaFX Node based UI Controls, such as Chart, Pane and Control.
 - ▶ WebView – This node manages the web engine and displays its contents.
- ▶ Leaf Node – The node without child nodes is known as the leaf node. For example, Rectangle, Ellipse, Box, ImageView, MediaView are examples of leaf nodes.
- ▶ It is mandatory to pass the root node to the scene graph.

First JavaFX Application

Create a simple JavaFX application which prints **hello world** on the console on clicking the button shown on the stage.

Step-1 : Extend javafx.application.Application and override start()

- As we have studied earlier that start() method is the starting point of constructing a JavaFX application therefore we need to first override start method of javafx.application.Application class.
- Object of the class javafx.stage.Stage is passed into the start() method therefore import this class and pass its object into start method.
- JavaFX.application.Application needs to be imported in order to override start method.

```
package application;           // creating source package. Created while making project in netbeans
import javafx.application.Application; // importing Application class
import javafx.stage.Stage;
public class Hello_World extends Application{ // inheriting Application class

    @Override
    public void start(Stage primaryStage) throws Exception { // primaryStage is created. Object of primaryStage is
passed to abstract method start() of Application class

        // JavaFX objects can be added here
    }
}
```

First JavaFX Application

Step-2 : Create a Button

- ▶ A button can be created by instantiating the javafx.scene.control.Button class.
- ▶ For this, we have to import this class into our code.
- ▶ Pass the button label text in Button class constructor.
- ▶ The code will look like following.

```
package application;                                // creating source package. Created while making project in netbeans
import javafx.application.Application;
Import javafx.scene.control.Button;
import javafx.stage.Stage;
public class Hello_World extends Application{    // inheriting Application class

    @Override
    public void start(Stage primaryStage) throws Exception {    // Object of the Stage class is passed to abstract
method start() of Application class

    Button btn1=new Button("Hello World");    // create a button and pass the caption on the button as parameter to
the constructor
}
}
```

First JavaFX Application

Step 3: Create a layout and add button to it

- ▶ JavaFX provides the number of layouts.
- ▶ We need to implement one of them in order to visualize the widgets properly.
- ▶ It exists at the top level of the scene graph and can be seen as a root node.
- ▶ All the other nodes (buttons, texts, etc.) need to be added to this layout.
- ▶ In this application, we have implemented StackPane layout.
- ▶ It can be implemented by instantiating javafx.scene.layout.StackPane class.
- ▶ The code will now look like following.

```
package application;                                // creating source package. Created while making project in netbeans
import javafx.application.Application;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class Hello_World extends Application{    // inheriting Application class
    @Override
    public void start(Stage primaryStage) throws Exception {    // Object of the Stage class is passed to abstract
method start() of Application class
        Button btn1=new Button("Hello World");    // create a button and pass the caption on the button as parameter to
the constructor
        StackPane root=new StackPane();    // create layout object
        root.getChildren().add(btn1);
    }
}
```

First JavaFX Application

Step 4: Create a Scene

- ▶ The layout needs to be added to a scene.
- ▶ Scene remains at the higher level in the hierarchy of application structure.
- ▶ It can be created by instantiating javafx.scene.Scene class.
- ▶ We need to pass the layout object to the scene class constructor. we can also pass the width and height of the required stage for the scene in the Scene class constructor.
- ▶ Our application code will now look like following.

```
package application;                                // creating source package. Created while making project in netbeans
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class Hello_World extends Application{    // inheriting Application class
    @Override
    public void start(Stage primaryStage) throws Exception {    // Object of the Stage class is passed to abstract method
        start() of Application class
        Button btn1=new Button("Hello World");    // create a button and pass the caption on the button as parameter to the
        constructor
        StackPane root=new StackPane();
        root.getChildren().add(btn1);
        Scene scene=new Scene(root, 300, 250);    // pass layout object, width and height of the stage to Scene class constructor
    }
}
```

First JavaFX Application

Step 5: Prepare the Stage

- ▶ javafx.stage.Stage class provides some important methods which are required to be called to set some attributes for the stage.
- ▶ We can set the title of the stage.
- ▶ We also need to call show() method without which, the stage won't be shown.
- ▶ Lets look at the code which describes how can we prepare the stage for the application.

```
package application;                                // creating source package. Created while making project in netbeans
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class Hello_World extends Application{    // inheriting Application class
    @Override
    public void start(Stage primaryStage) throws Exception {    // Object of the Stage class is passed to abstract method
        start() of Application class
        Button btn1=new Button("Hello World");    // create a button and pass the caption on the button as parameter to the
        constructor
        StackPane root=new StackPane();
        root.getChildren().add(btn1);
        Scene scene=new Scene(root, 300, 250);    // pass layout object, width and height of the stage to Scene class constructor
        primaryStage.setScene(scene);
        primaryStage.setTitle("First JavaFX Application");
        primaryStage.show();    } }
```

First JavaFX Application

Step 6: Create an event for the button

- As our application prints hello world for an event on the button, we need to create an event for the button.
- For this purpose, call setOnAction() on the button and define a anonymous class Event Handler as a parameter to the method.
- Inside this anonymous class, define a method handle() which contains the code for how the event is handled.
- In our case, it is printing hello world on the console.

```
package application;  
import javafx.application.Application;  
import javafx.event.ActionEvent;  
import javafx.event.EventHandler;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.stage.Stage;  
import javafx.scene.layout.StackPane;  
public class Hello_World extends Application{  
@Override  
    public void start(Stage primaryStage) throws Exception {  
        // TODO Auto-generated method stub  
        Button btn1=new Button("Hello World");  
        btn1.setOnAction(new EventHandler<ActionEvent>() {  
            @Override
```

```
                public void handle(ActionEvent arg0) {  
                    // TODO Auto-generated method stub  
                    System.out.println("Hello World !!!");  
                }  
            });  
            StackPane root=new StackPane();  
            root.getChildren().add(btn1);  
            Scene scene=new Scene(root,600,400);  
            primaryStage.setScene(scene);  
            primaryStage.setTitle("First JavaFX Application");  
            primaryStage.show();  
        }  
    }
```

First JavaFX Application

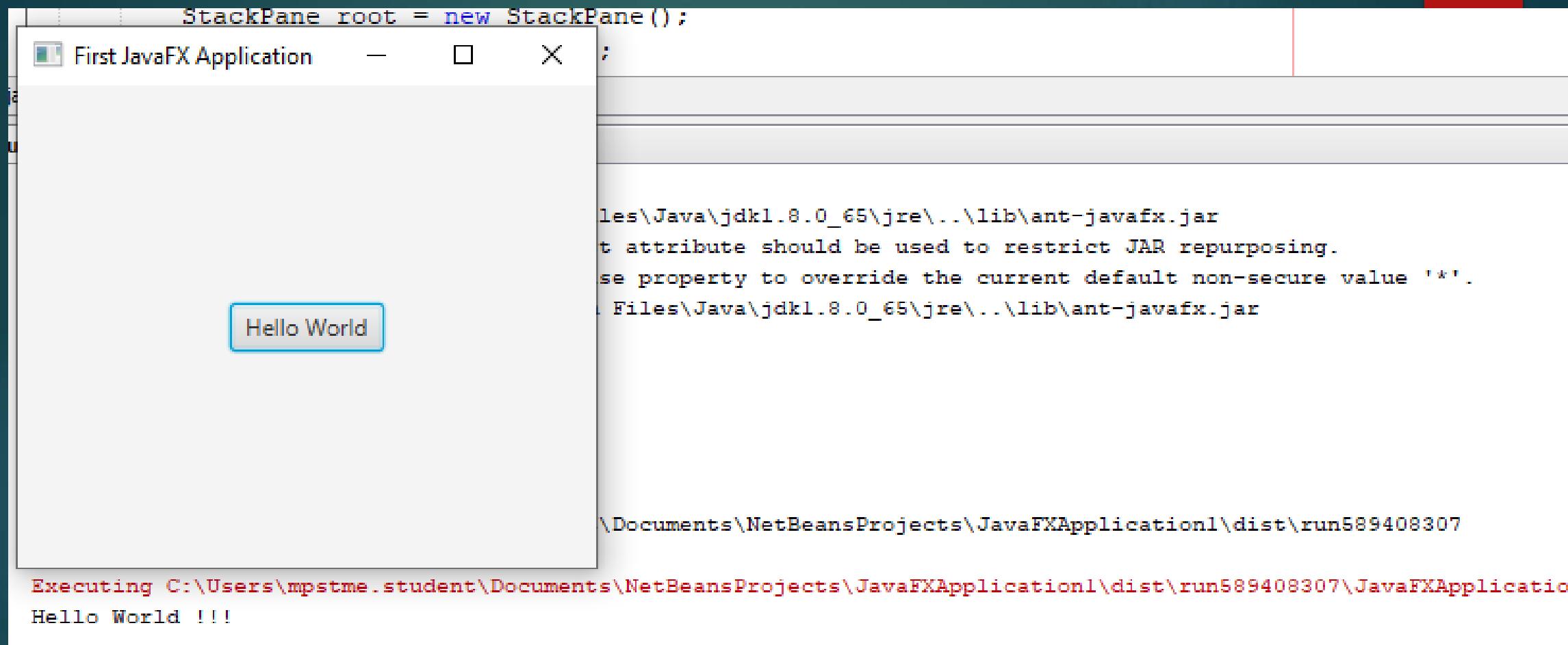
Step 7: Create a main method

- ▶ Till now, we have configured all the necessary things which are required to develop a basic JavaFX application but this application is still incomplete.
- ▶ We have not created main method yet.
- ▶ Hence, at the last, we need to create a main method in which we will launch the application i.e. will call launch() method and pass the command line arguments (args) to it.
- ▶ The code will now look like following.

```
package application;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class Hello_World extends Application{
@Override
    public void start(Stage primaryStage) throws Exception {
        // TODO Auto-generated method stub
        Button btn1=new Button("Hello World");
        btn1.setOnAction(new EventHandler<ActionEvent>() {
            @Override
        }
```

```
        public void handle(ActionEvent arg0) {
            // TODO Auto-generated method stub
            System.out.println("Hello World !!!");
        }
    });
    StackPane root=new StackPane();
    root.getChildren().add(btn1);
    Scene scene=new Scene(root,600,400);
    primaryStage.setScene(scene);
    primaryStage.setTitle("First JavaFX Application");
    primaryStage.show();
}

public static void main (String[] args)
{
    launch(args);
}
```



2D SHAPES



2D Shapes

- ▶ In some of the applications, we need to show two dimensional shapes to the user.
- ▶ However, JavaFX provides the flexibility to create our own 2D shapes on the screen .
- ▶ There are various classes which can be used to implement 2D shapes in our application.
- ▶ All these classes resides in **javafx.scene.shape package**.
- ▶ This package contains the classes which represents different types of 2D shapes.
- ▶ There are several methods in the classes which deals with the coordinates regarding 2D shape creation.
- ▶ **What are 2D shapes?**
- ▶ In general, a two dimensional shape can be defined as the geometrical figure that can be drawn on the coordinate system consist of X and Y planes.
- ▶ Using JavaFX, we can create 2D shapes such as Line, Rectangle, Circle, Ellipse, Polygon, Cubic Curve, quad curve, Arc, etc. The class **javafx.scene.shape.Shape is the base class for all the shape classes**.
- ▶ Using the JavaFX library and Shape Class, you can draw –
 - ▶ Predefined shapes such as Line, Rectangle, Circle, Ellipse, Polygon, Polyline, Cubic Curve, Quad Curve, Arc.
 - ▶ Path elements such as MoveTO Path Element, Line, Horizontal Line, Vertical Line, Cubic Curve, Quadratic Curve, Arc.
 - ▶ In addition to these, you can also draw a 2D shape by parsing SVG path.
- ▶ To create a shape, you need to –
 - ▶ Instantiate the respective class of the required shape.
 - ▶ Set the properties of the shape.
 - ▶ Add the shape object to the group.

2D Shapes

- ▶ Eg : Drawing a rectangle

```
Rectangle rect = new Rectangle();      //instantiate the rectangle class  
// setting properties of rectangle like x and y coordinates, width and height using respective methods  
rect.setX(10);  
rect.setY(20);  
rect.setWidth(100);  
rect.setHeight(100);  
Group root = new Group(rect);          // adding object of the shape to the group
```

2D Shapes

| Shape | Description |
|-------------|--|
| Line | In general, Line is the geometrical figure which joins two (X,Y) points on 2D coordinate system. In JavaFX, javafx.scene.shape.Line class needs to be instantiated in order to create lines. |
| Rectangle | In general, Rectangle is the geometrical figure with two pairs of two equal sides and four right angles at their joint. In JavaFX, javafx.scene.shape.Rectangle class needs to be instantiated in order to create Rectangles. |
| Ellipse | In general, ellipse can be defined as a curve with two focal points. The sum of the distances to the focal points are constant from each point of the ellipse. In JavaFX, javafx.scene.shape.Ellipse class needs to be instantiated in order to create Ellipse. |
| Arc | Arc can be defined as the part of the circumference of the circle or ellipse. In JavaFX, javafx.scene.shape.Arc class needs to be instantiated in order to create Arcs. |
| Circle | A circle is the special type of Ellipse having both the focal points at the same location. In JavaFX, Circle can be created by instantiating javafx.scene.shape.Circle class. |
| Polygon | Polygon is a geometrical figure that can be created by joining the multiple Co-planner line segments. In JavaFX, javafx.scene.shape.Polygon class needs to be instantiated in order to create polygon. |
| Cubic Curve | A Cubic curve is a curve of degree 3 in the XY plane. In JavaFX, javafx.scene.shape.CubicCurve class needs to be instantiated in order to create Cubic Curves. |
| Quad Curve | A Quad Curve is a curve of degree 2 in the XY plane. In JavaFX, javafx.scene.shape.QuadCurve class needs to be instantiated in order to create QuadCurve. |

Line

- ▶ Line can be defined as the geometrical structure which joins two points (X1,Y1) and (X2,Y2) in a X-Y coordinate plane.
- ▶ JavaFX allows the developers to create the line on the GUI of a JavaFX application.
- ▶ JavaFX library provides the class Line which is the part of javafx.scene.shape package.
- ▶ **How to create a Line?**
 - ▶ Instantiate the class javafx.scene.shape.Line.
 - ▶ set the required properties of the class object.
 - ▶ Add class object to the group
- ▶ Properties : Line class contains various properties described below.

| Property | Description | Setter Methods |
|----------|--|-------------------|
| endX | The X coordinate of the end point of the line | setEndX(Double) |
| endY | The y coordinate of the end point of the line | setEndY(Double) |
| startX | The x coordinate of the starting point of the line | setStartX(Double) |
| startY | The y coordinate of the starting point of the line | setStartY(Double) |

Line

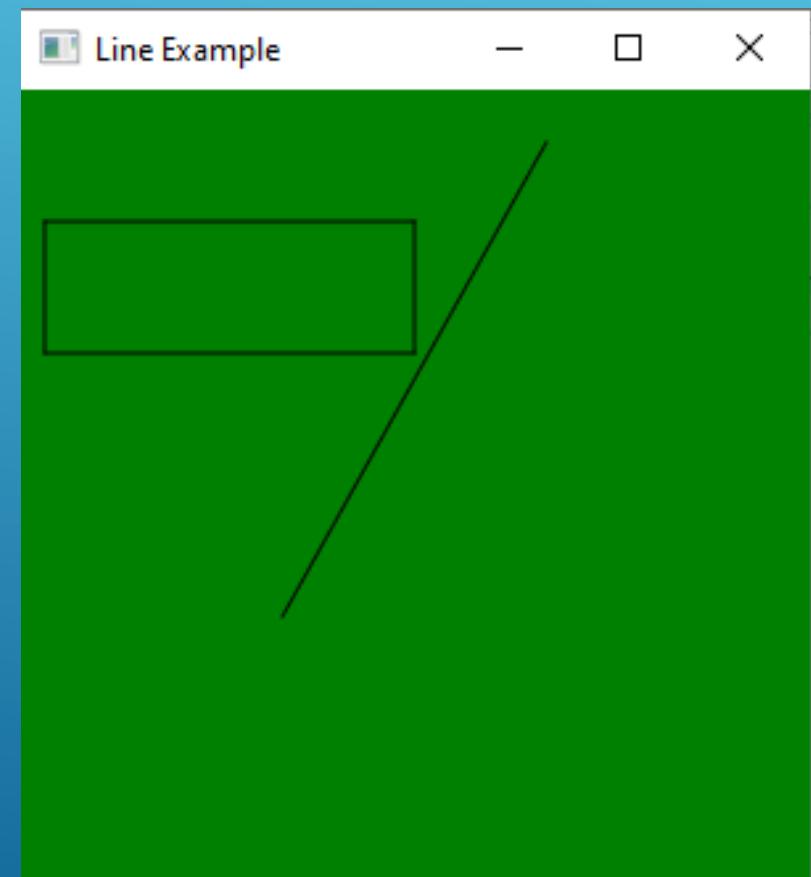
```
package javafxapplication1;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.shape.Line;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
public class DrawLine extends Application{

    @Override
    public void start(Stage primaryStage) throws Exception {
        // TODO Auto-generated method stub
        Line line = new Line(); //instantiating Line class
        line.setStartX(200); //setting starting X point of Line
        line.setStartY(20); //setting starting Y point of Line
        line.setEndX(100); //setting ending X point of Line
        line.setEndY(200); //setting ending Y point of Line
        //adding multiple lines
        Line line1 = new Line(10,50,150,50);
        Line line2 = new Line(10,100,150,100);
        Line line3 = new Line(10,50,10,100);
        Line line4 = new Line(150,50,150,100);
        Group root = new Group(line, line1, line2, line3,
        line4); //Creating a Group and adding line object to the
        group
    }
}
```

```
Scene scene = new Scene(root,300,300,Color.GREEN); // set background color
primaryStage.setScene(scene);
primaryStage.setTitle("Line Example");
primaryStage.show();
}

public static void main(String[] args) {
    launch(args);
} }
```



Rectangle

- ▶ In general, Rectangles can be defined as the geometrical figure consists of four sides, out of which, the opposite sides are always equal and the angle between the two adjacent sides is 90 degree.
- ▶ A Rectangle with four equal sides is called square.
- ▶ JavaFX library allows the developers to create a rectangle by instantiating `javafx.scene.shape.Rectangle` class
- ▶ Properties of Rectangle Class -

| Property | Description | Setter Method |
|-----------|---|--|
| ArcHeight | Vertical diameter of the arc at the four corners of rectangle | <code>setArcHeight(Double height)</code> |
| ArcWidth | Horizontal diameter of the arc at the four corners of the rectangle | <code>setArcWidth(Double Width)</code> |
| Height | Defines the height of the rectangle | <code>setHeight(Double height)</code> |
| Width | Defines the width of the rectangle | <code>setWidth(Double width)</code> |
| X | X coordinate of the upper left corner | <code>setX(Double X-value)</code> |
| Y | Y coordinate of the upper left corner | <code>setY(Double(Y-value)</code> |

Rectangle

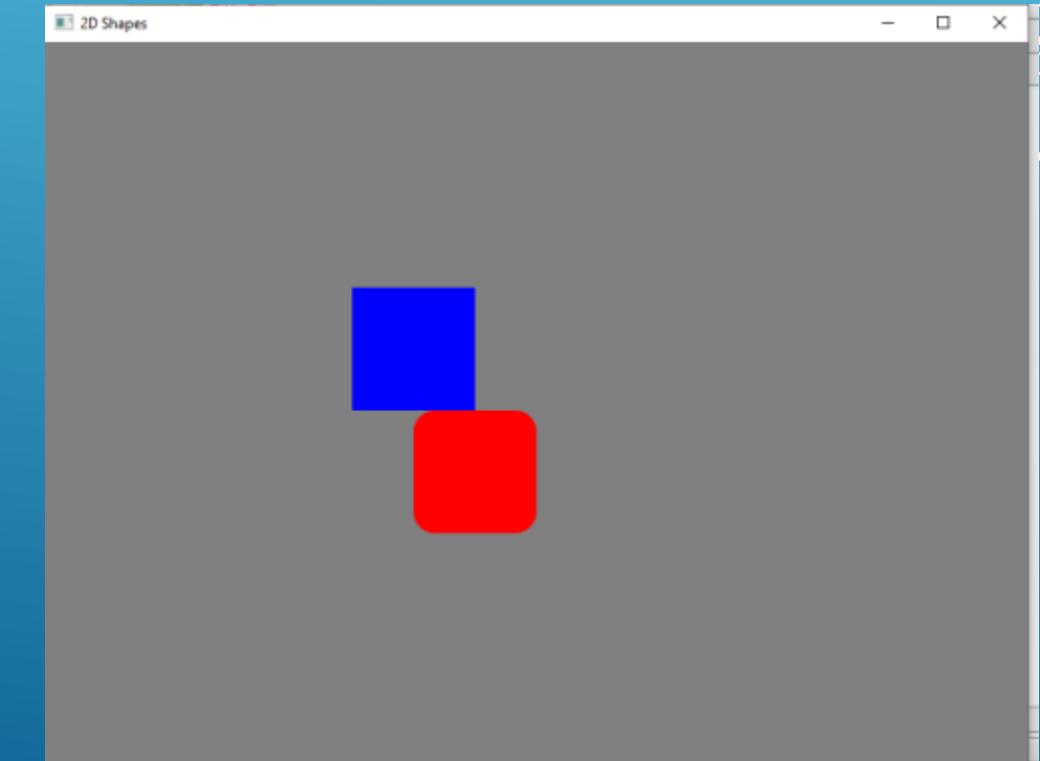
```
package javafxapplication1;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
public class Drawrect extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Rectangle rect=new Rectangle();
        rect.setX(250);
        rect.setY(200);
        rect.setWidth(100);
        rect.setHeight(100);
        rect.setFill(Color.BLUE);
        // Rounded Rectangle
        Rectangle rect1=new Rectangle();
        rect1.setX(300);
        rect1.setY(300);
        rect1.setWidth(100);
        rect1.setHeight(100);
        rect1.setArcHeight(35.0);
        rect1.setArcWidth(35.0);
        rect1.setFill(Color.RED);
```

```
        Group root = new Group(rect, rect1);
        Scene scene = new Scene(root,800,800,Color.GRAY);

        primaryStage.setTitle("2D Shapes");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```



Ellipse

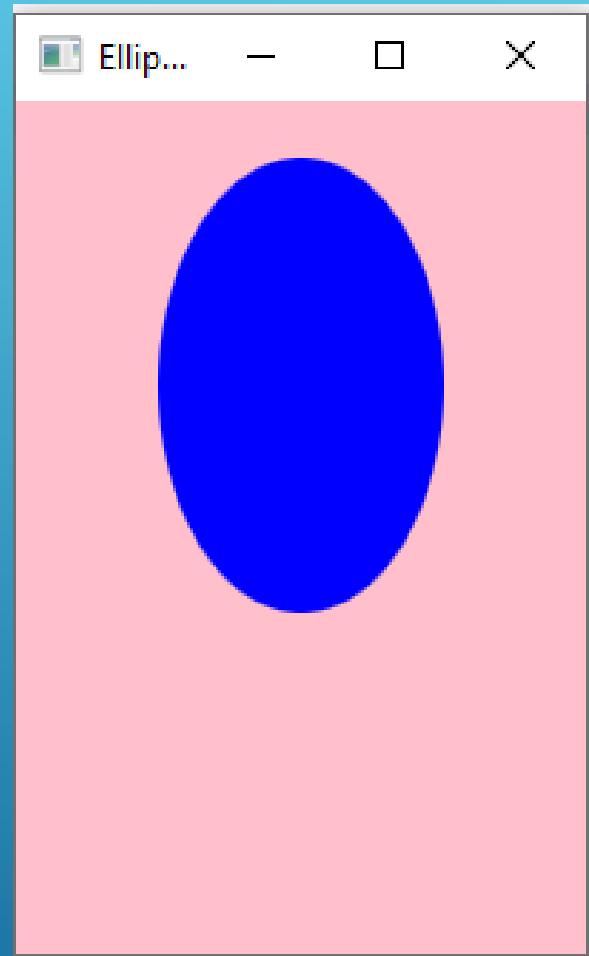
- ▶ In general, ellipse can be defined as the geometrical structure with the two focal points.
- ▶ The focal points in the ellipse are chosen so that the sum of the distance to the focal points is constant from every point of the ellipse.
- ▶ In JavaFX, the class `javafx.scene.shape.Ellipse` represents Ellipse.
- ▶ This class needs to be instantiated in order to create ellipse.
- ▶ This class contains various properties which needs to be set in order to render ellipse on a XY place.

| Property | Description | Setter Methods |
|----------|--|--|
| CenterX | Horizontal position of the centre of ellipse | <code>setCenterX(Double X-value)</code> |
| CenterY | Vertical position of the centre of ellipse | <code>setCenterY(Double Y-value)</code> |
| RadiusX | Width of Eclipse | <code>setRadiusX(Double X-Radius Value)</code> |
| RadiusY | Height of Eclipse | <code>setRadiusY(Double Y-Radius Value)</code> |

Ellipse

```
package javafxapplication1;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.shape.Ellipse;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
public class DrawEllipse extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Ellipse e = new Ellipse();
        e.setCenterX(100);
        e.setCenterY(100);
        e.setRadiusX(50);
        e.setRadiusY(80);
        e.setFill(Color.BLUE);
        Group root = new Group(e);
        Scene scene = new Scene(root, 200, 300, Color.PINK);
        primaryStage.setTitle("Ellipse Example");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```



Arc

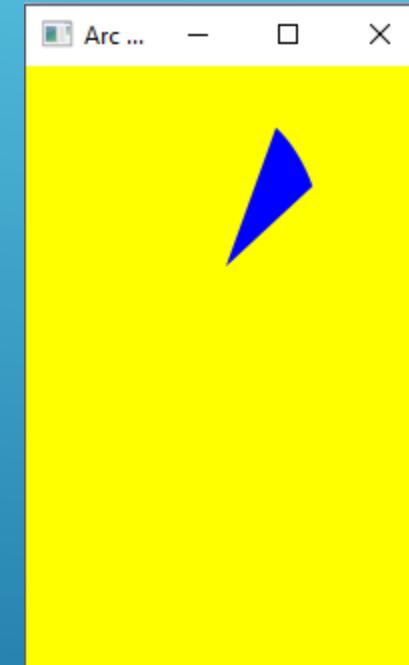
- ▶ In general, Arc is the part of the circumference of a circle or ellipse.
- ▶ It needs to be created in some of the JavaFX applications wherever required.
- ▶ JavaFX allows us to create the Arc on GUI by just instantiating `javafx.scene.shape.Arc` class.
- ▶ Just set the properties of the class to the appropriate values to show arc as required by the Application.

| Property | Description | Method |
|------------|--|--|
| CenterX | X coordinate of the centre point | <code>setCenterX(Double value)</code> |
| CenterY | Y coordinate of the centre point | <code>setCenterY(Double value)</code> |
| Length | Angular extent of the arc in degrees | <code>setLength(Double value)</code> |
| RadiusX | Full width of the ellipse of which, Arc is a part. | <code>setRadiusX(Double value)</code> |
| RadiusY | Full height of the ellipse of which, Arc is a part | <code>setRadiusY(Double value)</code> |
| StartAngle | Angle of the arc in degrees | <code>setStartAngle(Double value)</code> |
| type | Type of Arc : OPEN, CHORD, ROUND | <code>setType(Double value)</code> |

Arc

```
package javafxapplication1;  
import javafx.application.Application;  
import javafx.scene.Group;  
import javafx.scene.Scene;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Arc;  
import javafx.scene.shape.ArcType;  
import javafx.stage.Stage;  
public class DrawArc extends Application{  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        Arc arc = new Arc();  
        arc.setCenterX(100);  
        arc.setCenterY(100);  
        arc.setRadiusX(50);  
        arc.setRadiusY(80);  
        arc.setStartAngle(30);  
        arc.setLength(30);  
        arc.setType(ArcType.ROUND);  
        arc.setFill(Color.BLUE);  
        Group root = new Group(arc);  
        Scene scene = new Scene(root,200,300,Color.YELLOW);  
        primaryStage.setTitle("Arc Example");  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
}
```

```
public static void main(String[] args) {  
    launch(args);  
}
```



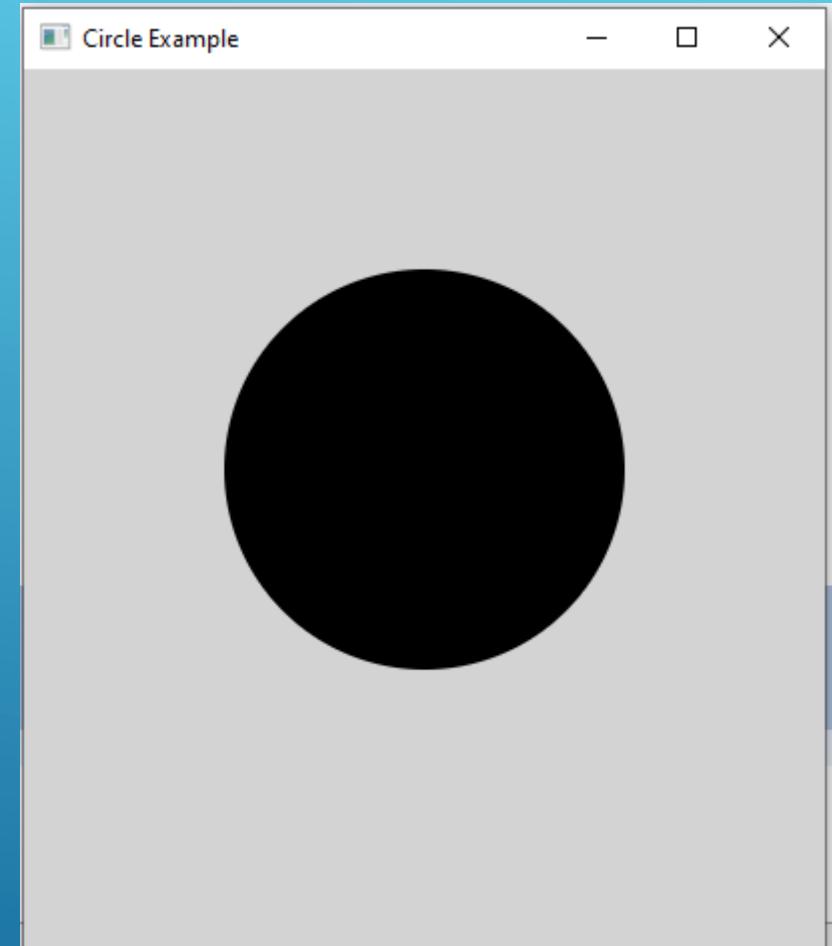
Circle

- ▶ A circle is a special type of ellipse with both of the focal points at the same position.
- ▶ Its horizontal radius is equal to its vertical radius.
- ▶ JavaFX allows us to create Circle on the GUI of any application by just instantiating `javafx.scene.shape.Circle` class.
- ▶ Just set the class properties by using the instance setter methods and add the class object to the Group.

| Property | Description | Setter Methods |
|----------------------|--------------------------------------|---------------------------------------|
| <code>centerX</code> | X coordinate of the centre of circle | <code>setCenterX(Double value)</code> |
| <code>centerY</code> | Y coordinate of the centre of circle | <code>setCenterY(Double value)</code> |
| <code>radius</code> | Radius of the circle | <code>setRadius(Double value)</code> |

Circle

```
package javafxapplication1;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Arc;
import javafx.scene.shape.ArcType;
import javafx.stage.Stage;
public class DrawCircle extends Application{
    @Override
    public void start(Stage primaryStage) throws Exception {
        Circle c = new Circle();
        c.setCenterX(200);
        c.setCenterY(200);
        c.setRadius(100);
        Group root = new Group(c);
        Scene scene = new Scene(root,400,500,Color.LIGHTGRAY);
        primaryStage.setTitle("Circle Example");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```



Polygons

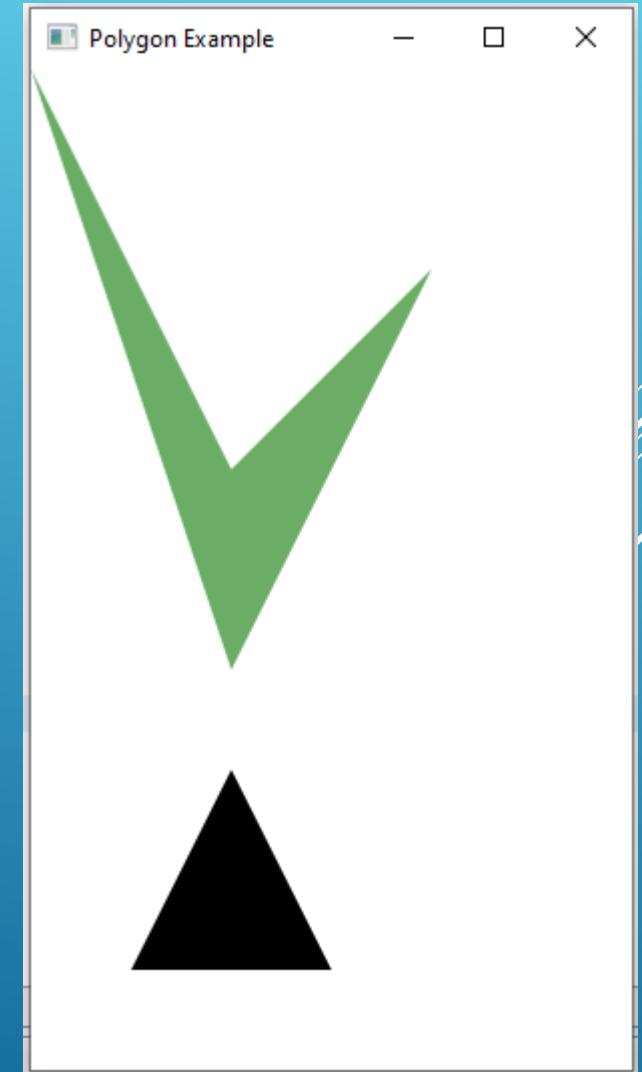
- ▶ Polygon can be defined as a plain figure with at least three straight sides forming a loop.
- ▶ In the case of polygons, we mainly consider the length of its sides and the interior angles.
- ▶ Triangles, squares, Pentagons, Hexagons, etc are all polygons.
- ▶ In JavaFX, Polygon can be created by instantiating `javafx.scene.shape.Polygon` class.
- ▶ We need to pass a Double array into the class constructor representing X-Y coordinates of all the points of the polygon.
- ▶ The syntax is given below.

```
Polygon poly = new Polygon(DoubleArray);
```

Polygons

```
package javafxapplication1;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.shape.Polygon;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
public class DrawPolygon extends Application {
    @Override
    public void start(Stage primaryStage) {
        Polygon poly = new Polygon(0.0,0.0,100.0, 200.0, 200.0, 100.0, 100.0,300.0);
        Polygon poly1 =new Polygon(100.0, 350.0, 50.0, 450.0, 150.0, 450.0);
        // setting color based on rgb values
        int red=20, green=125, blue=10;
        poly.setFill(Color.rgb(red, green, blue,0.63));
        Group root = new Group(poly, poly1);
        Scene scene = new Scene(root,500,500);
        primaryStage.setTitle("Polygon Example");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```



Text

- ▶ In some of the cases, we need to provide the text based information on the interface of our application.
- ▶ JavaFX library provides a class named javafx.scene.text.Text for this purpose.
- ▶ This class provides various methods to alter various properties of the text.
- ▶ We just need to instantiate this class to implement text in our application.
- ▶ Use the setter method setText(string) to set the string as a text for the text class object.
- ▶ Follow the syntax given below to instantiate the Text class.

```
Text <text_Object> = new Text();
text.setText(<string-text>);
```

- ▶ By default, the text will be displayed to the center of the screen.

Font and position of the Text

- ▶ JavaFX enables us to apply various fonts to the text nodes.
- ▶ We just need to set the property font of the Text class by using the setter method setFont().
- ▶ This method accepts the object of Font class.
- ▶ The class Font belongs to the package javafx.scene.text.
- ▶ It contains a static method named font().
- ▶ This returns an object of Font type which will be passed as an argument into the setFont() method of Text class.

Text

- ▶ The method `Font.font()` accepts the following parameters.
- ▶ **Family:** it represents the family of the font. It is of string type and should be an appropriate font family present in the system.
- ▶ **Weight:** this `Font` class property is for the weight of the font. There are 9 values which can be used as the font weight. The values are `FontWeight.BLACK`, `BOLD`, `EXTRA_BOLD`, `EXTRA_LIGHT`, `LIGHT`, `MEDIUM`, `NORMAL`, `SEMI_BOLD`, `THIN`.
- ▶ **Posture:** this `Font` class property represents the posture of the font. It can be either `FontPosture.ITALIC` or `FontPosture.REGULAR`.
- ▶ **Size:** this is a double type property. It is used to set the size of the font.
- ▶ The Syntax of the method `setFont()` is given below.

```
<text_object>.setFont(Font.font(<String font_family>, <FontWeight>, <FontPosture>, <FontSize>))
```

- ▶ **Applying Stroke and Color to Text**
- ▶ Stroke means the padding at the boundary of the text.
- ▶ JavaFX allows us to apply stroke and colors to the text.
- ▶ `javafx.scene.text.Text` class provides a method named `setStroke()` which accepts the `Paint` class object as an argument. Just pass the color which will be painted on the stroke.
- ▶ We can also set the width of the stroke by passing a width value of double type into `setStrokeWidth()` method.
- ▶ To set the color of the Text, `javafx.scene.text.Text` class provides another method named `setFill()`. We just need to pass the color which is to be filled in the text.

Text

- ▶ **Text Decoration**

- ▶ We can apply the decorations to the text by setting the properties `strikeThrough` and `underline` of `javafx.scene.text.Text` class.
- ▶ The syntax of both the methods is given below.

`<TextObject>.setStrikeThrough(Boolean value) //pass true to put a line across the text`

`<TextObject>.setUnderLine(Boolean value) //pass true to underline the text`

Text

| Property | Description | Setter Methods |
|-------------------|---|---|
| boundstype | This property is of object type. It determines the way in which the bounds of the text is being calculated. | setBoundsType(TextBoundsType value) |
| font | Font of the text. | setFont(Font value) |
| fontsmoothingType | Defines the requested smoothing type for the font. | setFontSmoothingType(FontSmoothingType value) |
| linespacing | Vertical space in pixels between the lines. It is double type property. | setLineSpacing(double spacing) |
| strikethrough | This is a boolean type property. We can put a line through the text by setting this property to true. | setStrikeThrough(boolean value) |
| textalignment | Horizontal Text alignment | setTextAlignment(TextAlignment value) |
| textorigin | Origin of text coordinate system in local coordinate system. | setTextOrigin(VPos value) |
| text | It is a string type property. It defines the text string which is to be displayed. | setText(String value) |
| underline | It is a boolean type property. We can underline the text by setting this property to true. | setUnderLine(boolean value) |
| wrappingwidth | Width limit for the text from where the text is to be wrapped. It is a double type property. | setWrappingWidth(double value) |
| x | X coordinate of the text | setX(double value) |
| y | Y coordinate of the text | setY(double value) |

Text

```
package javafxapplication1;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
import javafx.scene.text.TextAlignment;
//import javafx.scene.layout.StackPane;
public class JavaFXText extends Application{
@Override
public void start(Stage primaryStage) throws Exception {
    Text text = new Text("Hello !! Welcome to Java FX");
    text.setX(50);
    text.setY(100);

    text.setFont(Font.font("Arial",FontWeight.EXTRA_BOLD,Font
Posture.ITALIC,50)); // font formatting
    text.setFill(Color.YELLOW); // setting colour of the text
    text.setStroke(Color.BLACK); // setting the border for the
text
    text.setStrokeWidth(3); // setting border width to 3
    text.setUnderline(true); // underline the text
}
```

Text text1 = new Text("The line spacing property of the javafx.scene.text.Text class specifies the line spacing between the lines of the text (node) vertically. You can set the value to this property using the setLineSpacing() method. This method accepts a boolean value as a parameter and sets the specified space between the lines (vertically).");

```
text1.setX(50);
text1.setY(300);
text1.setStrikethrough(true); // strike the text
text1.setTextAlignment(TextAlignment.CENTER);
// Aligning text
text1.setLineSpacing(10); // Line Spacing
text1.setWrappingWidth(300); // text wrapping
Group root = new Group(text, text1);
Scene scene = new Scene(root,800,400);
primaryStage.setScene(scene);
primaryStage.setTitle("Text Example");
primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
} }
```

Text



Java FX Layouts

Java FX Layouts

- ▶ Layouts are the top level container classes that define the UI styles for scene graph objects.
- ▶ Layout can be seen as the parent node to all the other nodes.
- ▶ JavaFX provides various layout panes that support different styles of layouts.
- ▶ In JavaFX, Layout defines the way in which the components are to be seen on the stage.
- ▶ It basically organizes the scene-graph nodes.
- ▶ We have several built-in layout panes in JavaFX that are HBox, VBox, StackPane, FlowBox, AnchorPane, etc.
- ▶ Each Built-in layout is represented by a separate class which needs to be instantiated in order to implement that particular layout pane.
- ▶ All these classes belong to javafx.scene.layout package. javafx.scene.layout.Pane class is the base class for all the built-in layout classes in JavaFX..
- ▶ **Steps to create layout**
- ▶ In order to create the layouts, we need to follow the following steps.
 - ▶ Instantiate the respective layout class, for example, HBox root = new HBox();
 - ▶ Setting the properties for the layout, for example, root.setSpacing(20);
 - ▶ Adding nodes to the layout object, for example, root.getChildren().addAll(<NodeObjects>);

Java FX Layouts

| Class | Description |
|------------|---|
| BorderPane | Organizes nodes in top, left, right, centre and the bottom of the screen. |
| FlowPane | Organizes the nodes in the horizontal rows according to the available horizontal spaces. Wraps the nodes to the next line if the horizontal space is less than the total width of the nodes |
| GridPane | Organizes the nodes in the form of rows and columns. |
| HBox | Organizes the nodes in a single row. |
| Pane | It is the base class for all the layout classes. |
| StackPane | Organizes nodes in the form of a stack i.e. one onto another |
| VBox | Organizes nodes in a vertical column. |

BorderPane

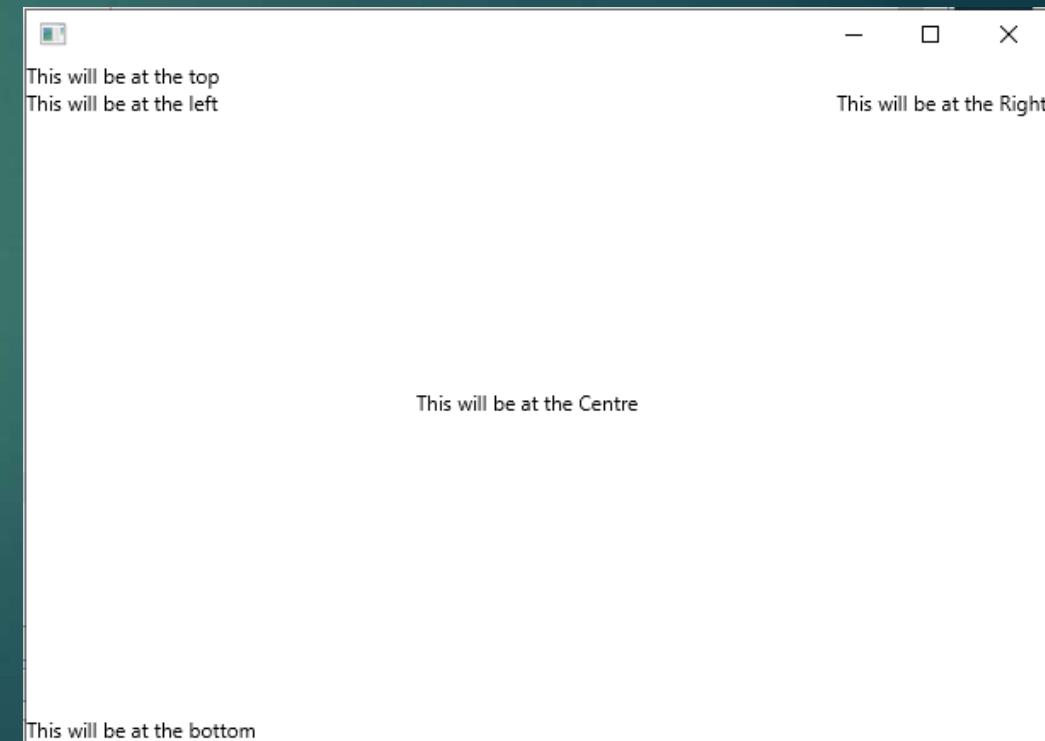
- ▶ BorderPane arranges the nodes at the left, right, centre, top and bottom of the screen.
- ▶ It is represented by `javafx.scene.layout.BorderPane` class.
- ▶ This class provides various methods like `setRight()`, `setLeft()`, `setCenter()`, `setBottom()` and `setTop()` which are used to set the position for the specified nodes.
- ▶ We need to instantiate BorderPane class to create the BorderPane layout.

| Type | Property | Setter Methods | Description |
|------|----------|--------------------------|--|
| Node | Bottom | <code>setBottom()</code> | Add the node to the bottom of the screen |
| Node | Centre | <code>setCenter()</code> | Add the node to the center of the screen |
| Node | Left | <code>setLeft()</code> | Add the node to the left of the screen |
| Node | Right | <code>setRight()</code> | Add the node to the right of the screen |
| Node | Top | <code>setTop()</code> | Add the node to the top of the screen |

- ▶ There are the following constructors in the class.
 - ▶ `BorderPane()` : create the empty layout
 - ▶ `BorderPane(Node Center)` : create the layout with the center node
 - ▶ `BorderPane(Node Center, Node top, Node right, Node bottom, Node left)` : create the layout with all the nodes

BorderPane

```
package javafxapplication1.UIControls;  
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.Label;  
import javafx.scene.layout.*;  
import javafx.stage.Stage;  
import javafx.scene.text.Text;  
public class BorderPaneDemo extends Application {  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        BorderPane BPane = new BorderPane();  
        BPane.setTop(new Text("This will be at the top"));  
        BPane.setLeft(new Text("This will be at the left"));  
        BPane.setRight(new Text("This will be at the Right"));  
        BPane.setCenter(new Text("This will be at the Centre"));  
        BPane.setBottom(new Text("This will be at the bottom"));  
        Scene scene = new Scene(BPane,600,400);  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
    public static void main(String[] args) {  
        launch(args);  
    } }
```



HBox

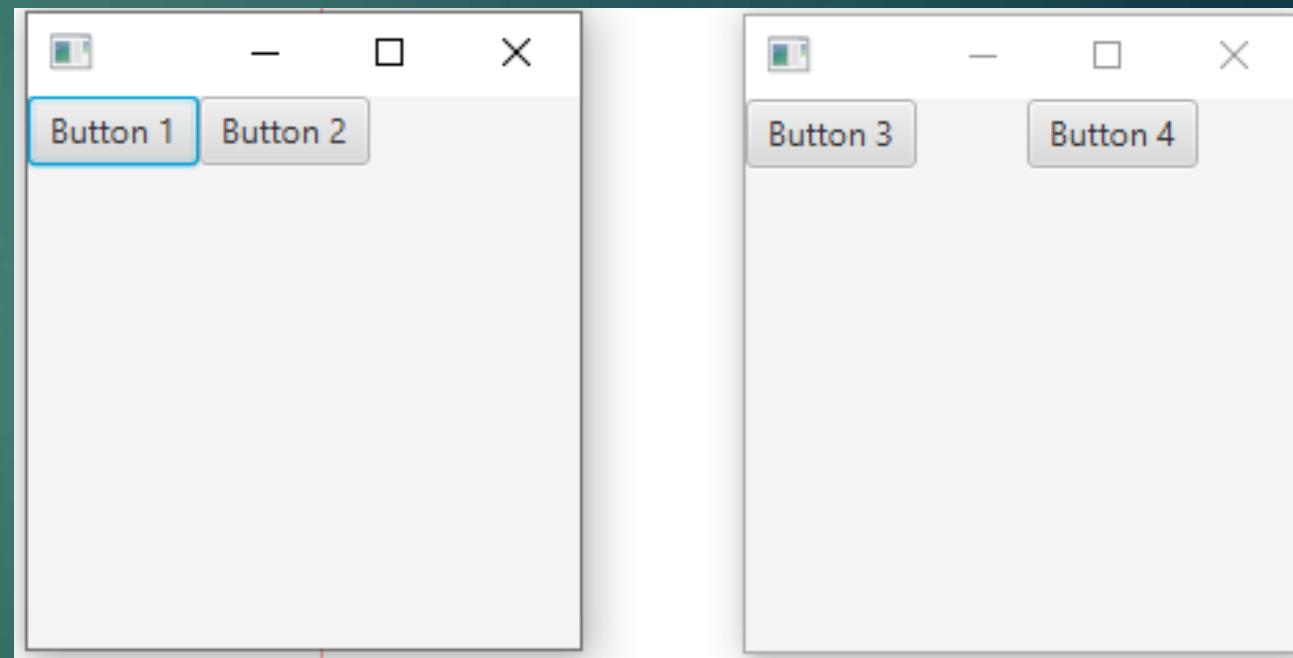
- ▶ HBox layout pane arranges the nodes in a single row.
- ▶ It is represented by javafx.scene.layout.HBox class.
- ▶ We just need to instantiate HBox class in order to create HBox layout.

| Property | Description | Setter Methods |
|------------|---|-----------------------|
| alignment | This represents the alignment of the nodes. | setAlignment(Double) |
| fillHeight | This is a boolean property. If you set this property to true the height of the nodes will become equal to the height of the HBox. | setFillHeight(Double) |
| spacing | This represents the space between the nodes in the HBox. It is of double type. | setSpacing(Double) |

- ▶ The HBox class contains two constructors that are given below.
 - ▶ new HBox() : create HBox layout with 0 spacing
 - ▶ new HBox(Double spacing) : create HBox layout with a spacing value

HBox

```
package javafxapplication1.UIControls;  
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.layout.HBox;  
import javafx.stage.Stage;  
public class HBoxDemo extends Application {  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        Button btn1 = new Button("Button 1");  
        Button btn2 = new Button("Button 2");  
        Button btn3 = new Button("Button 3");  
        Button btn4 = new Button("Button 4");  
        HBox root = new HBox(btn1,btn2);  
        Scene scene = new Scene(root,200,200);  
        primaryStage.setScene(scene);  
        primaryStage.show();  
  
        HBox root1 = new HBox(btn3,btn4);  
        root1.setSpacing(40);  
        Scene scene1 = new Scene(root1,200,200);  
        Stage newStage=new Stage();  
        newStage.setScene(scene1);  
        newStage.show();  
    }  
    public static void main(String[] args) {  
        launch(args);  
    } }
```



VBox

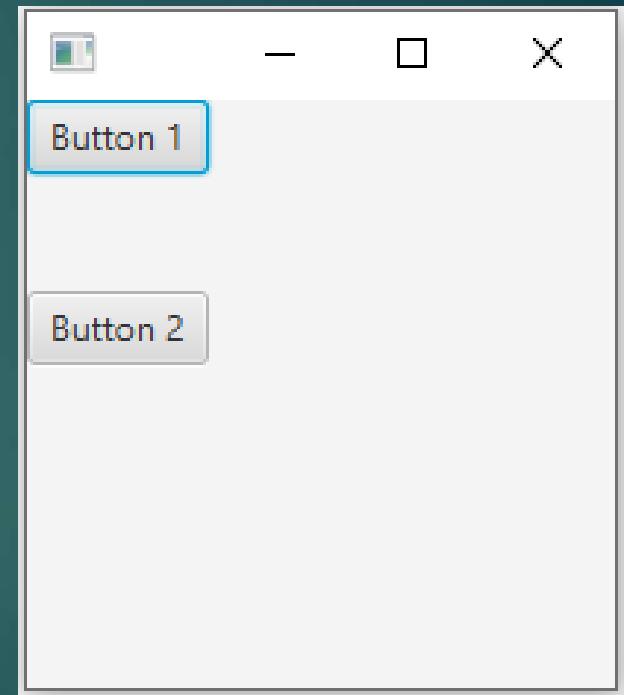
- ▶ Instead of arranging the nodes in horizontal row, Vbox Layout Pane arranges the nodes in a single vertical column.
- ▶ It is represented by `javafx.scene.layout.VBox` class which provides all the methods to deal with the styling and the distance among the nodes.
- ▶ This class needs to be instantiated in order to implement VBox layout in our application.

| Property | Description | Setter Methods |
|-----------|--|------------------------------------|
| Alignment | This property is for the alignment of the nodes. | <code>setAlignment(Double)</code> |
| FillWidth | This property is of the boolean type. The Width of resizeable nodes can be made equal to the Width of the VBox by setting this property to true. | <code>setFillWidth(boolean)</code> |
| Spacing | This property is to set some spacing among the nodes of VBox. | <code>setSpacing(Double)</code> |

- ▶ The VBox class contains constructors that are given below.
 - ▶ `VBox()` : creates layout with 0 spacing
 - ▶ `Vbox(Double spacing)` : creates layout with a spacing value of double type
 - ▶ `Vbox(Double spacing, Node? children)` : creates a layout with the specified spacing among the specified child nodes
 - ▶ `Vbox(Node? children)` : creates a layout with the specified nodes having 0 spacing among them

VBox

```
package javafxapplication1.UIControls;  
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.layout.VBox;  
import javafx.stage.Stage;  
public class VBoxDemo extends Application {  
  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        Button btn1 = new Button("Button 1");  
        Button btn2 = new Button("Button 2");  
        VBox root = new VBox(btn1,btn2);  
        root.setSpacing(40);  
        Scene scene = new Scene(root,200,200);  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
    public static void main(String[] args) {  
        launch(args);  
    } }
```



StackPane

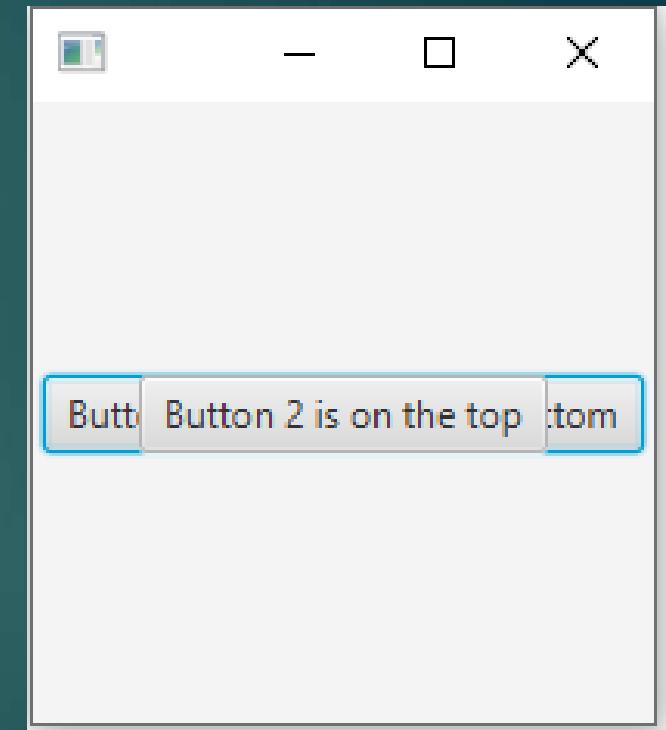
- ▶ The StackPane layout pane places all the nodes into a single stack where every new node gets placed on the top of the previous node.
- ▶ It is represented by `javafx.scene.layout.StackPane` class.
- ▶ We just need to instantiate this class to implement StackPane layout into our application.

| Property | Description | Setter Method |
|-----------|---|--|
| alignment | It represents the default alignment of children within the StackPane's width and height | <code>setAlignment(Node child, Pos value)</code> <code>setAlignment(Pos value)</code> |

- ▶ The class contains two constructors that are given below.
 - ▶ `StackPane()`
 - ▶ `StackPane(Node? Children)`

StackPane

```
package javafxapplication1.UIControls;  
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.layout.StackPane;  
import javafx.stage.Stage;  
public class StackPaneDemo extends Application {  
  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        Button btn1 = new Button("Button 1 is stacked to the bottom");  
        Button btn2 = new Button("Button 2 is on the top");  
        StackPane root = new StackPane(btn1,btn2);  
        Scene scene = new Scene(root,200,200);  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
    public static void main(String[] args) {  
        launch(args);  
    } }
```



GridPane

- ▶ GridPane Layout pane allows us to add the multiple nodes in multiple rows and columns.
- ▶ It is seen as a flexible grid of rows and columns where nodes can be placed in any cell of the grid.
- ▶ It is represented by `javafx.scence.layout.GridPane` class. We just need to instantiate this class to implement GridPane.

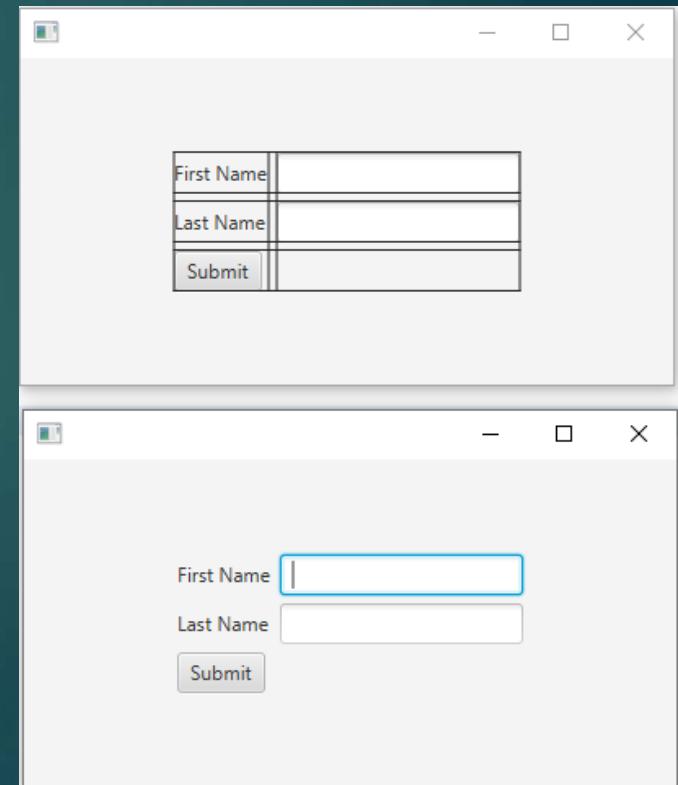
| Property | Description | Setter Methods |
|------------------|---|---|
| alignment | Represents the alignment of the grid within the GridPane. | <code>setAlignment(Pos value)</code> |
| gridLinesVisible | This property is intended for debugging. Lines can be displayed to show the gridpane's rows and columns by setting this property to true. | <code>setGridLinesVisible(Boolean value)</code> |
| hgap | Horizontal gaps among the columns | <code>setHgap(Double value)</code> |
| vgap | Vertical gaps among the rows | <code>setVgap(Double value)</code> |

- ▶ The class contains only one constructor that is given below.
 - ▶ Public `GridPane():` creates a gridpane with 0 hgap/vgap.

GridPane

```
package javafxapplication1.UIControls;  
import javafx.application.Application;  
import javafx.geometry.Pos;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.scene.layout.GridPane;  
import javafx.stage.Stage;  
public class GridPaneDemo extends Application {  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        Label first_name=new Label("First Name");  
        Label last_name=new Label("Last Name");  
        TextField tf1=new TextField();  
        TextField tf2=new TextField();  
        Button Submit=new Button ("Submit");  
        GridPane root=new GridPane();  
        Scene scene = new Scene(root,400,200);  
        root.add(first_name,0,0);  
        root.add(tf1,1,0);  
        root.addRow(1, last_name,tf2);  
        root.addRow(2, Submit);  
        root.setAlignment(Pos.CENTER);  
    }  
}
```

```
//root.setGridLinesVisible(false);  
root.setGridLinesVisible(true);  
root.setHgap(5);  
root.setVgap(5);  
primaryStage.setScene(scene);  
primaryStage.show();  
}  
public static void main(String[] args) {  
    launch(args);  
} }
```



FlowPane

- ▶ FlowPane layout pane organizes the nodes in a flow that are wrapped at the flowpane's boundary.
- ▶ The horizontal flowpane arranges the nodes in a row and wrap them according to the flowpane's width.
- ▶ The vertical flowpane arranges the nodes in a column and wrap them according to the flowpane's height.
- ▶ FlowPane layout is represented by `javafx.scene.layout.FlowPane` class.

| Property | Description | Setter Methods |
|------------------|---|--|
| alignment | The overall alignment of the flowpane's content. | <code>setAlignment(Pos value)</code> |
| columnHalignment | The horizontal alignment of nodes within the columns. | <code>setColumnHalignment(HPos Value)</code> |
| hgap | Horizontal gap between the columns. | <code>setHgap(Double value)</code> |
| orientation | Orientation of the flowpane | <code>setOrientation(Orientation value)</code> |
| prefWrapLength | The preferred height or width where content should wrap in the horizontal or vertical flowpane. | <code>setPrefWrapLength(double value)</code> |
| rowValignment | The vertical alignment of the nodes within the rows. | <code>setRowValignment(VPos value)</code> |
| vgap | The vertical gap among the rows | <code>setVgap(Double value)</code> |

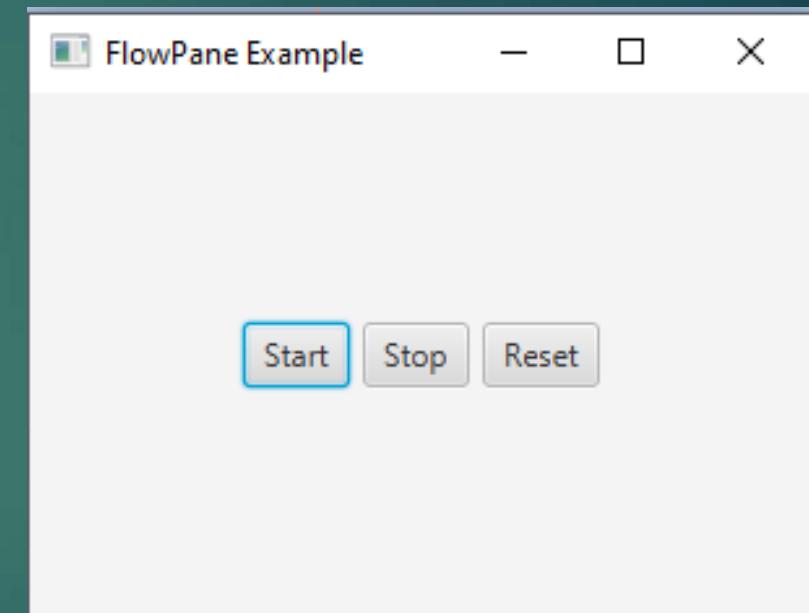
FlowPane

- ▶ There are 8 constructors in the class that are given below.
 - ▶ FlowPane()
 - ▶ FlowPane(Double Hgap, Double Vgap)
 - ▶ FlowPane(Double Hgap, Double Vgap, Node? children)
 - ▶ FlowPane(Node... Children)
 - ▶ FlowPane(Orientation orientation)
 - ▶ FlowPane(Orientation orientation, double Hgap, Double Vgap)
 - ▶ FlowPane(Orientation orientation, double Hgap, Double Vgap, Node? children)
 - ▶ FlowPane(Orientation orientation, Node... Children)

FlowPane

```
package javafxapplication1.UIControls;  
import javafx.application.Application;  
import javafx.geometry.Pos;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.layout.FlowPane;  
import javafx.stage.Stage;  
public class FlowPaneDemo extends Application {  
  
    @Override  
    public void start(Stage primaryStage) {  
        Button btn1=new Button("Start");  
        Button btn2=new Button("Stop");  
        Button btn3=new Button("Reset");  
        FlowPane root = new FlowPane(btn1, btn2, btn3);  
        root.setVgap(6);  
        root.setHgap(5);  
        root.setPrefWrapLength(250);  
        root.setAlignment(Pos.CENTER);  
        Scene scene = new Scene(root,300,200);  
        primaryStage.setTitle("FlowPane Example");  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
}
```

```
        public static void main(String[] args) {  
            launch(args);  
        }  
    }
```



UI Controls

Java FX UI Controls

- ▶ The graphical user interface of every desktop application mainly considers UI elements, layouts and behaviour.
- ▶ The UI elements are the one which are actually shown to the user for interaction or information exchange.
- ▶ Layout defines the organization of the UI elements on the screen.
- ▶ Behaviour is the reaction of the UI element when some event is occurred on it.
- ▶ However, the package `javafx.scene.control` provides all the necessary classes for the UI components like `Button`, `Label`, etc.
- ▶ Every class represents a specific UI control and defines some methods for their styling.

Java FX UI Controls

| SN | Control | Description |
|----|-------------------|--|
| 1 | Label | Label is a component that is used to define a simple text on the screen. Typically, a label is placed with the node, it describes. |
| 2 | Button | Button is a component that controls the function of the application. Button class is used to create a labelled button. |
| 3 | RadioButton | The Radio Button is used to provide various options to the user. The user can only choose one option among all. A radio button is either selected or deselected. |
| 4 | CheckBox | Check Box is used to get the kind of information from the user which contains various choices. User marked the checkbox either on (true) or off (false). |
| 5 | TextField | Text Field is basically used to get the input from the user in the form of text. <code>javafx.scene.control.TextField</code> represents TextField |
| 6 | PasswordField | PasswordField is used to get the user's password. Whatever is typed in the password field is not shown on the screen to anyone. |
| 7 | HyperLink | HyperLink are used to refer any of the webpage through your application. It is represented by the class <code>javafx.scene.control.HyperLink</code> |
| 8 | Slider | Slider is used to provide a pane of options to the user in a graphical form where the user needs to move a slider over the range of values to select one of them. |
| 9 | ProgressBar | Progress Bar is used to show the work progress to the user. It is represented by the class <code>javafx.scene.control.ProgressBar</code> . |
| 10 | ProgressIndicator | Instead of showing the analogue progress to the user, it shows the digital progress so that the user may know the amount of work done in percentage. |
| 11 | ScrollBar | JavaFX Scroll Bar is used to provide a scroll bar to the user so that the user can scroll down the application pages. |
| 12 | Menu | JavaFX provides a Menu class to implement menus. Menu is the main component of any application. |
| 13 | ToolTip | JavaFX ToolTip is used to provide hint to the user about any component. It is mainly used to provide hints about the text fields or password fields being used in the application. |

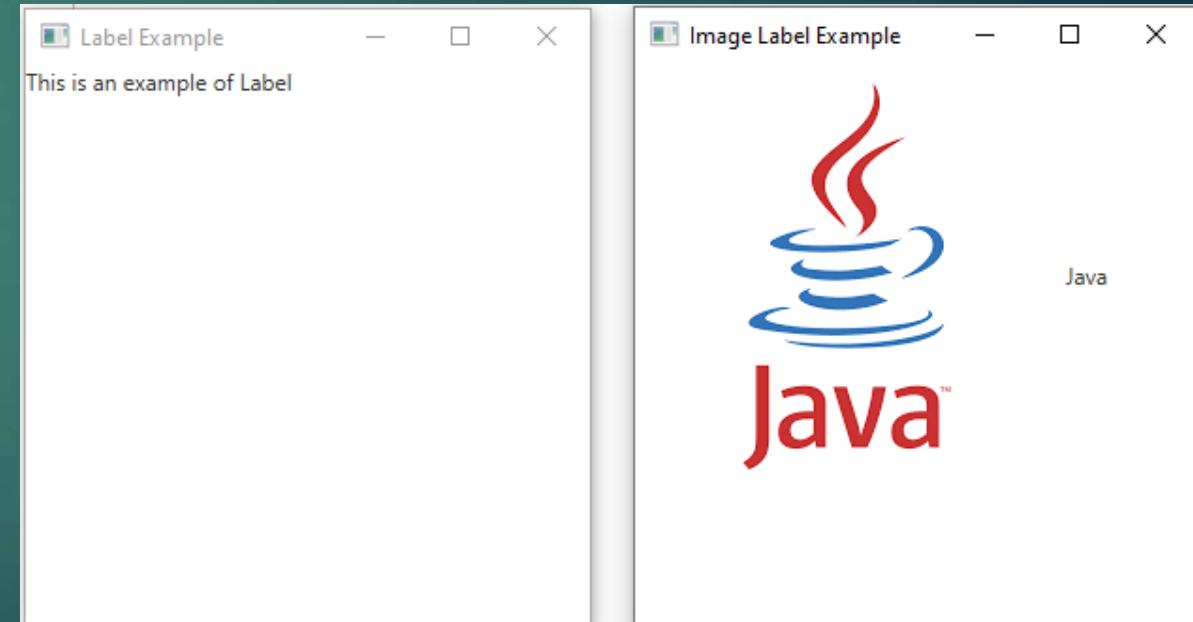
Label

- ▶ javafx.scene.control.Label class represents label control.
- ▶ As the name suggests, the label is the component that is used to place any text information on the screen.
- ▶ It is mainly used to describe the purpose of the other components to the user.
- ▶ You can not set a focus on the label using the Tab key.
- ▶ Package: javafx.scene.control
- ▶ Constructors:
 - ▶ Label(): creates an empty Label
 - ▶ Label(String text): creates Label with the supplied text
 - ▶ Label(String text, Node graphics): creates Label with the supplied text and graphics

Label

```
package javafxapplication1.UIControls;  
import java.io.InputStream;  
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.Label;  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
import javafx.stage.Stage;  
import javafx.scene.Group;  
public class LabelDemo extends Application {  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        // TODO Auto-generated method stub  
        Label my_label=new Label("This is an example of Label");  
        FileInputStream input= new  
FileInputStream("C:/Users/mpstme.student/Documents/NetBeansPr  
ojects/JavaFXApplication1/src/javafxapplication1/UIControls/javai  
mg.png");  
        Image img = new Image(input);  
        ImageView imageview=new ImageView(img);  
        Label my_label1=new Label("Java",imageview);  
        Group root = new Group(my_label);  
        Group root1 = new Group(my_label1);  
        Scene scene=new Scene(root,300,300);  
        Scene scene1=new Scene(root1,300,300);  
        primaryStage.setScene(scene);  
        primaryStage.setTitle("Label Example");  
        primaryStage.show();
```

```
// creating another stage to add another scene  
Stage newstage = new Stage();  
newstage.setScene(scene1);  
newstage.setTitle("Image Label Example");  
newstage.show();  
}  
public static void main(String[] args) {  
    launch(args);  
} }
```



Button

- ▶ JavaFX button control is represented by `javafx.scene.control.Button` class. A button is a component that can control the behaviour of the Application. An event is generated whenever the button gets clicked.
- ▶ Button can be created by instantiating `Button` class. Use the following line to create button object.

```
Button btn = new Button("My Button");
```

- ▶ **Button Action**

- ▶ `Button` class provides `setOnAction()` method which is used to set the action for the button click event.
- ▶ An object of the anonymous class implementing the `handle()` method, is passed in this method as a parameter.
- ▶ We can also pass lambda expressions to handle the events.

Button

```
package javafxapplication1.UIControls;
import java.io.FileInputStream;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;
import javafx.scene.layout.HBox;
public class ButtonDemo extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception
    {
        Button btn=new Button("This is a button");
        // Image Button
        FileInputStream input=new
FileInputStream("C:/Users/mpstme.student/Documents/
NetBeansProjects/JavaFXApplication1/src/javafxapplication1/UIControls/javaimg.png");
        Image image = new Image(input);
        ImageView img=new ImageView(image);
        Button btn1=new Button();
        btn1.setGraphic(img);
        //button event handling
        btn1.setOnAction(new EventHandler<ActionEvent>()
```

```
    @Override
    public void handle(ActionEvent args) {
        // TODO Auto-generated method stub
        System.out.println("Button clicked");
    }
});
HBox root = new HBox(btn,btn1);
root.setSpacing(40);
Scene scene=new Scene(root,300,300);
primaryStage.setScene(scene);
primaryStage.setTitle("Button Class Example");
primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
}
```

Button



Output - JavaFXApplication1 (run-single) X

```
ant -f C:\\\\Users\\\\mpstme.student\\\\Documents\\\\NetBeansProjects\\\\JavaFXApplication1\ninit:\nDeleting: C:\\\\Users\\\\mpstme.student\\\\Documents\\\\NetBeansProjects\\\\JavaFXApplication1\\\\bu\ndeps-jar:\nUpdating property file: C:\\\\Users\\\\mpstme.student\\\\Documents\\\\NetBeansProjects\\\\JavaFXA\nCompiling 1 source file to C:\\\\Users\\\\mpstme.student\\\\Documents\\\\NetBeansProjects\\\\JavaF\ncompile-single:\nrun-single:\nButton clicked
```

Java -Dnb.

A screenshot of the same JavaFX application window from the previous image, but with a blue border drawn around the Java logo area. This indicates that the Java logo is being selected or highlighted. The window title is "Button Class Example" and it contains a button labeled "This is a button". The Java logo is centered in the window.

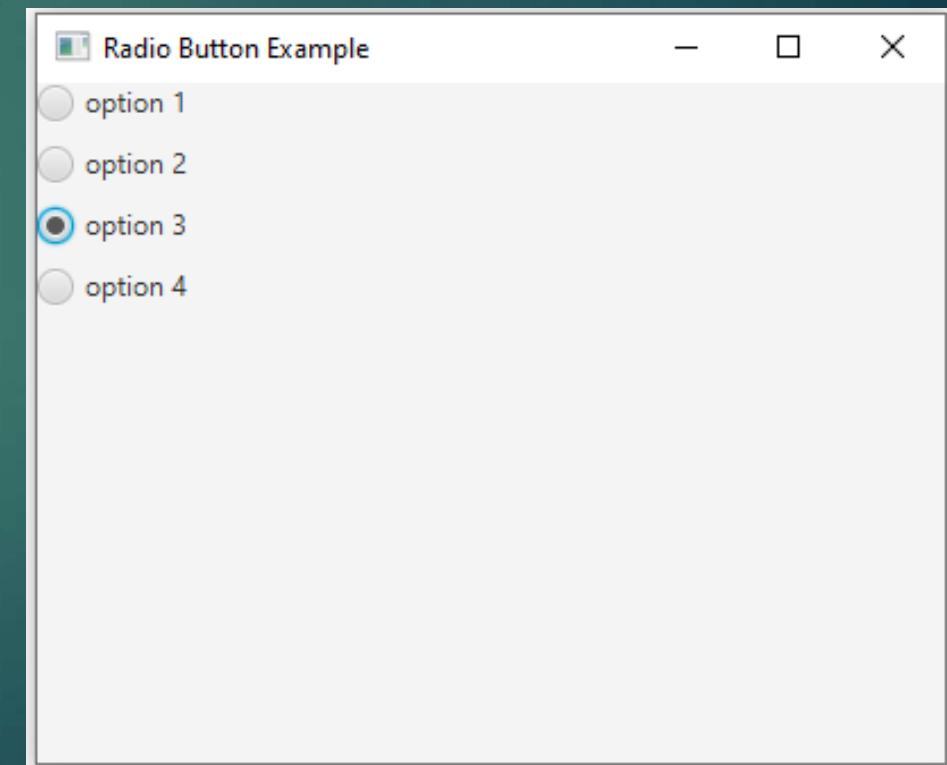
Radio Button

- The Radio Button is used to provide various options to the user.
- The user can only choose one option among all.
- A radio button is either selected or deselected.
- It can be used in a scenario of multiple choice questions in the quiz where only one option needs to be chosen by the student.

Radio Button

```
package javafxapplication1.UIControls;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
public class RadioButtonDemo extends Application {
@Override
public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub
    //ToggleGroup is used to define group of radiobuttons.
    Only one radiobutton in each group can be selected at a
    time.
    ToggleGroup group = new ToggleGroup();
    RadioButton button1 = new RadioButton("option 1");
    RadioButton button2 = new RadioButton("option 2");
    RadioButton button3 = new RadioButton("option 3");
    RadioButton button4 = new RadioButton("option 4");
    button1.setToggleGroup(group);
    button2.setToggleGroup(group);
    button3.setToggleGroup(group);
    button4.setToggleGroup(group);
    VBox root=new VBox(button1,button2,button3,button4);
    root.setSpacing(10);
```

```
Scene scene=new Scene(root,400,300);
primaryStage.setScene(scene);
primaryStage.setTitle("Radio Button Example");
primaryStage.show();
}
public static void main(String[] args) {
launch(args);
} }
```



Checkbox

- ▶ The Check Box is used to provide more than one choices to the user.
- ▶ It can be used in a scenario where the user is prompted to select more than one option or the user wants to select multiple options.
- ▶ It is different from the radio button in the sense that, we can select more than one checkboxes in a scenerio.
- ▶ Instantiate javafx.scene.control.CheckBox class to implement CheckBox.

```
CheckBox checkbox = new CheckBox();
```

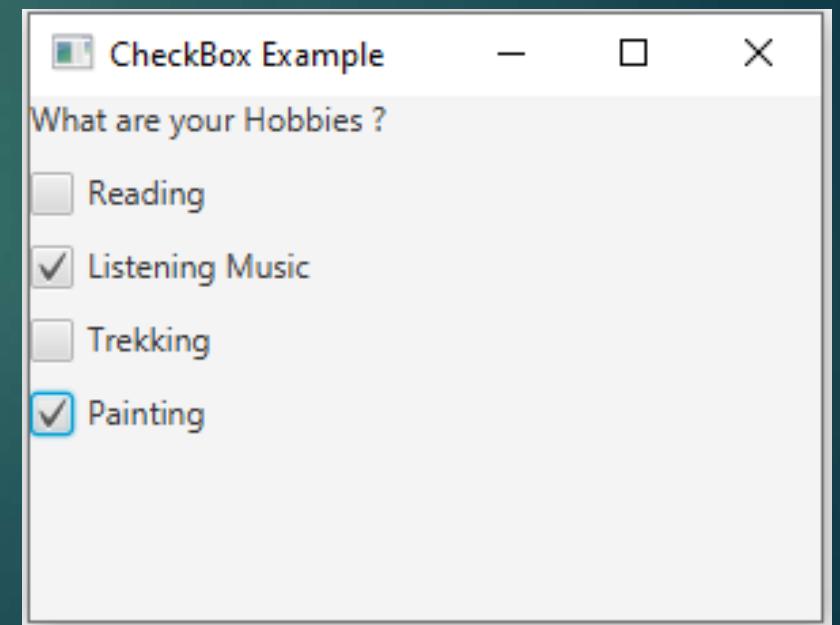
- ▶ Use the following line to attach a label with the checkbox.

```
CheckBox checkbox = new CheckBox("Label Name");
```

- ▶ We can change the CheckBox Label at any time by calling an instance method setText("text").
- ▶ We can make it selected by calling setSelected("true")

Checkbox

```
package javafxapplication1.UIControls;  
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.CheckBox;  
import javafx.scene.control.Label;  
import javafx.scene.layout.VBox;  
import javafx.stage.Stage;  
public class CheckboxDemo extends Application {  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        // TODO Auto-generated method stub  
        Label lbl = new Label("What are your Hobbies ?");  
        CheckBox c1 = new CheckBox("Reading");  
        CheckBox c2 = new CheckBox("Listening Music");  
        CheckBox c3 = new CheckBox("Trekking");  
        CheckBox c4 = new CheckBox("Painting");  
        VBox root = new VBox(lbl,c1,c2,c3,c4);  
        root.setSpacing(10);  
        Scene scene=new Scene(root,800,200);  
        primaryStage.setScene(scene);  
        primaryStage.setTitle("CheckBox Example");  
        primaryStage.show();  
    }  
    public static void main(String[] args) {  
        launch(args);    } }
```



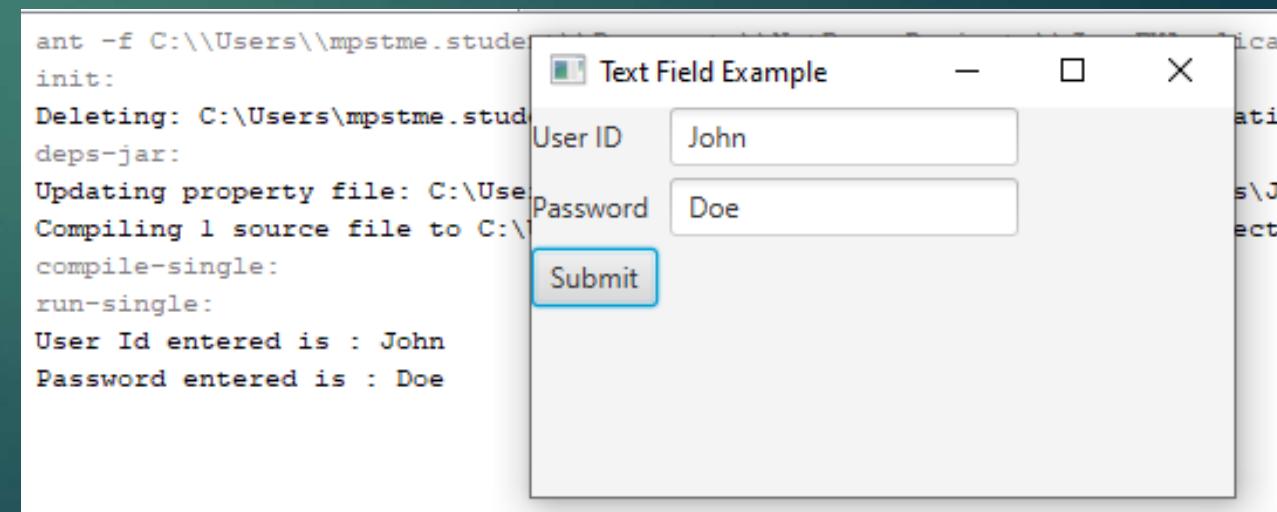
Text Field

- ▶ Text Field is basically used to get the input from the user in the form of text.
- ▶ `javafx.scene.control.TextField` represents `TextField`.
- ▶ It provides various methods to deal with textfields in JavaFX.
- ▶ `TextField` can be created by instantiating `TextField` class.
- ▶ `TextField` class provides an instance method `getText()` to retrieve the textfield data.
- ▶ It returns `String` object which can be used to save the user details in database.

Text Field

```
package javafxapplication1.UIControls;  
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.scene.layout.GridPane;  
import javafx.stage.Stage;  
import javafx.event.ActionEvent;  
import javafx.event.EventHandler;  
public class TextFieldDemo extends Application {  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        // TODO Auto-generated method stub  
        Label user_id=new Label("User ID");  
        Label password = new Label("Password");  
        TextField tf1=new TextField();  
        TextField tf2=new TextField();  
        Button b = new Button("Submit");  
        GridPane root = new GridPane();  
        root.addRow(0, user_id, tf1);  
        root.addRow(1, password, tf2);  
        root.addRow(2, b);  
        root.setHgap(5);  
        root.setVgap(5);  
        b.setOnAction(new EventHandler<ActionEvent>() {
```

```
        @Override  
        public void handle(ActionEvent args) {  
            // TODO Auto-generated method stub  
            System.out.println("User Id entered is : " + tf1.getText());  
            System.out.println("Password entered is : " + tf2.getText());  
        }  
    } );  
    Scene scene=new Scene(root,300,300);  
    primaryStage.setScene(scene);  
    primaryStage.setTitle("Text Field Example");  
    primaryStage.show();  
}  
    public static void main(String[] args) {  
        launch(args);  
    } }
```



Password Field

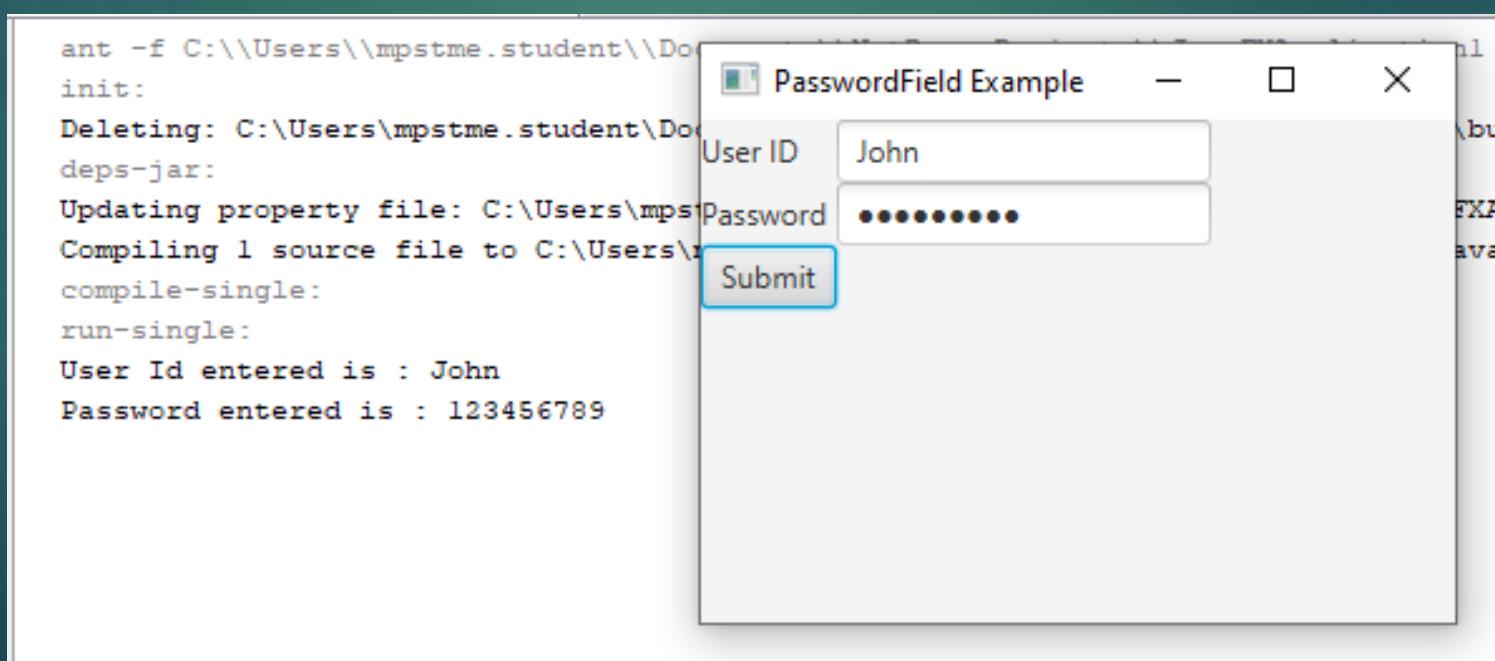
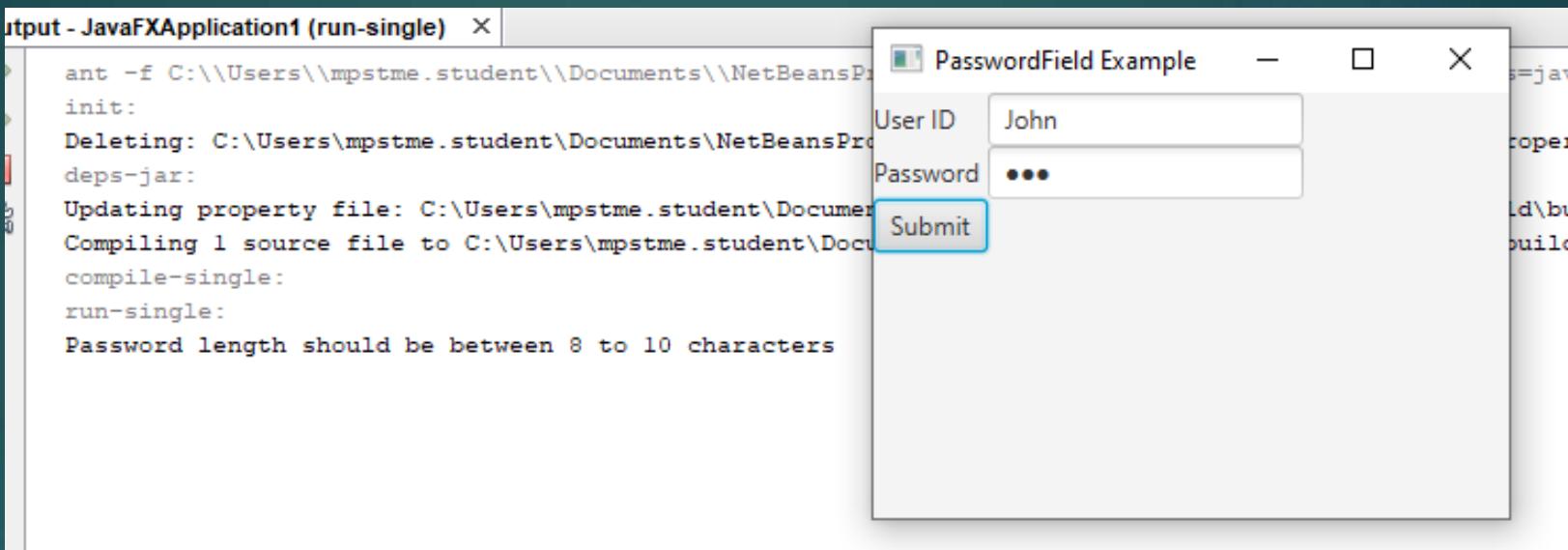
- ▶ Typing a password in a text field is not secure for the user.
- ▶ The Application must use a specific component to get the password from the user.
- ▶ Password Field can be created by instantiating javafx.scene.control.PasswordField class.
- ▶ PasswordField class contains a method named as setPromptText() for showing a prompt text to the user in password field.
- ▶ The data written in the password field is retrieved by getText() method.

Password Field

```
package javafxapplication1.UIControls;  
import javafx.application.Application;  
import javafx.event.ActionEvent;  
import javafx.event.EventHandler;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.PasswordField;  
import javafx.scene.control.TextField;  
import javafx.scene.layout.GridPane;  
import javafx.stage.Stage;  
public class PasswordFieldDemo extends Application {  
    public void start(Stage primaryStage) throws Exception {  
        Label user_id=new Label("User ID");  
        Label password = new Label("Password");  
        TextField tf=new TextField();  
        PasswordField pf=new PasswordField();  
        tf.setPromptText("Enter U_Id");  
        pf.setPromptText("Enter Password");  
        Button b = new Button("Submit");  
        GridPane root = new GridPane();  
        root.addRow(0, user_id, tf);  
        root.addRow(1, password, pf);  
        root.addRow(5, b);  
        b.setOnAction(new EventHandler<ActionEvent>() {
```

```
        @Override  
        public void handle(ActionEvent args) {  
            String pwd=pf.getText();  
            if(pwd.length()<8 || pwd.length()>10)  
            {  
                System.out.println("Password length should be between  
8 to 10 characters");  
            }  
            else  
            {  
                System.out.println("User Id entered is : " + tf.getText());  
                System.out.println("Password entered is : " + pwd);  
            }  
        } );  
  
        Scene scene=new Scene(root,300,200);  
        primaryStage.setScene(scene);  
        primaryStage.setTitle("PasswordField Example");  
        primaryStage.show();  
    }  
    public static void main(String[] args) {  
        launch(args);  
    } }
```

Password Field



File Chooser

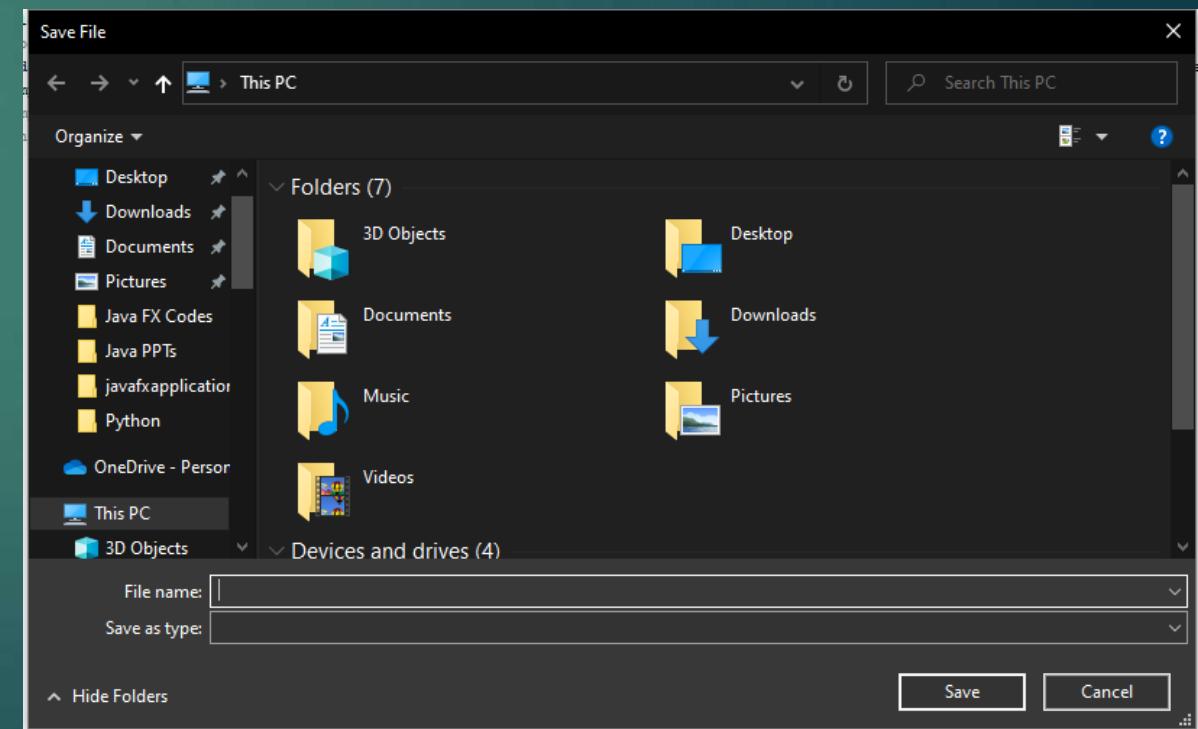
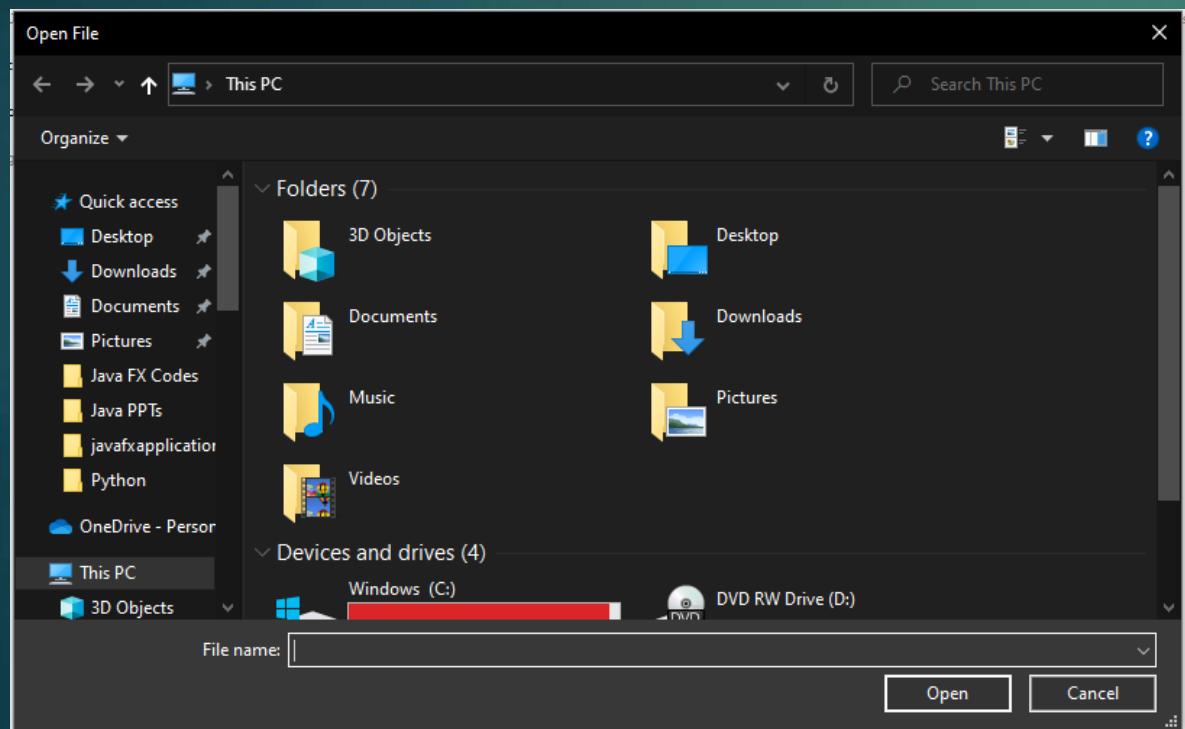
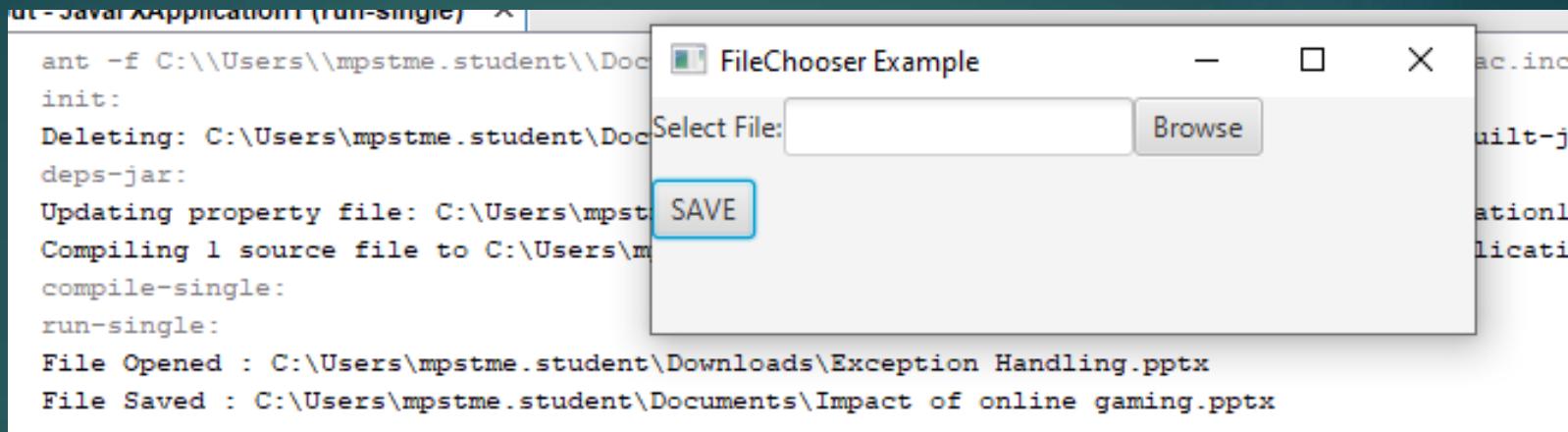
- ▶ JavaFX File chooser enables users to browse the files from the file system.
- ▶ `javafx.stage.FileChooser` class represents `FileChooser`.
- ▶ It can be created by instantiating `FileChooser` class.
- ▶ As we see in the modern day applications, there are two types of dialogues shown to the user, one is for opening the file and the other is for saving the files.
- ▶ In each case, the user needs to browse a location for the file and give the name to the file.
- ▶ The `FileChooser` class provides two types of methods,
 - ▶ `showOpenDialog()`
 - ▶ `showSaveDialog()`

File Chooser

```
package javafxapplication1.UIControls;  
import java.io.File;  
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.*;  
import javafx.scene.layout.GridPane;  
import javafx.stage.FileChooser;  
import javafx.stage.Stage;  
public class FileChooserDemo extends Application{  
  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        // TODO Auto-generated method stub  
        // showOpenDialog() Demo  
        Label label = new Label("Select File:");  
        TextField tf= new TextField();  
        Button btn = new Button("Browse");  
        btn.setOnAction(e->  
        {  
            FileChooser file = new FileChooser();  
            file.setTitle("Open File");  
            File file3=file.showOpenDialog(primaryStage);  
            System.out.println("File Opened : "+file3);  
       });  
    }  
}
```

```
// showSaveDialog() Demo  
Button btn1 = new Button("SAVE");  
btn1.setOnAction(e->  
{  
    FileChooser file1 = new FileChooser();  
    file1.setTitle("Save File");  
    File file2 = file1.showSaveDialog(primaryStage);  
    System.out.println("File Saved : "+file2);  
});  
GridPane root = new GridPane();  
root.addRow(0,label,tf,btn);  
root.addRow(1,btn1);  
root.setVgap(10);  
Scene scene = new Scene(root,350,100);  
primaryStage.setScene(scene);  
primaryStage.setTitle("FileChooser Example");  
primaryStage.show();  
}  
public static void main(String[] args) {  
    launch(args);  
} }
```

File Chooser



Menu

- ▶ JavaFX provides a Menu class to implement menus.
- ▶ Menu is the main component of any application.
- ▶ In JavaFX, `javafx.scene.control.Menu` class provides all the methods to deal with menus.
- ▶ This class needs to be instantiated to create a Menu.

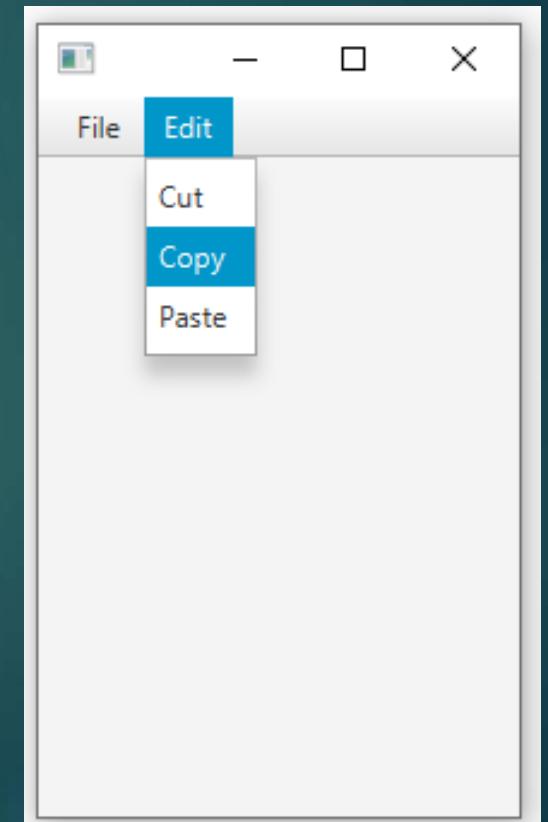
Menu

```
package javafxapplication1.UIControls;  
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.*;  
import javafx.scene.layout.BorderPane;  
import javafx.stage.Stage;  
public class MenuDemo extends Application {
```

```
    @Override  
    public void start(Stage primaryStage) throws Exception {
```

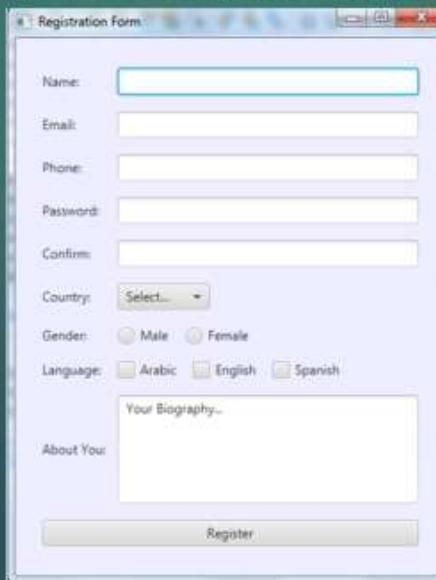
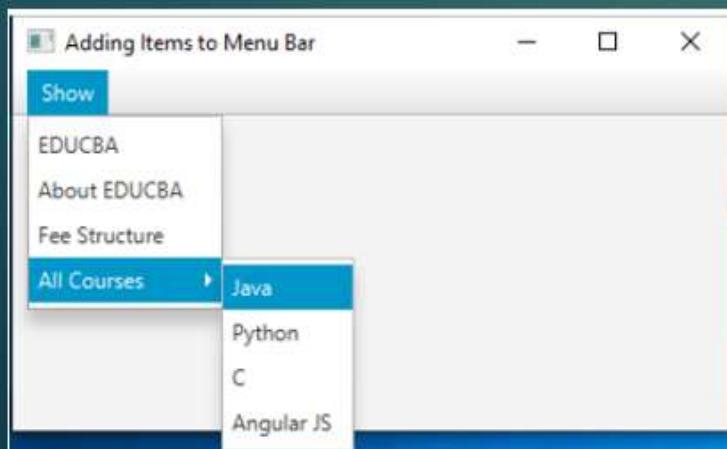
```
        BorderPane root = new BorderPane();  
        Scene scene = new Scene(root,200,300);  
        MenuBar menubar = new MenuBar();  
        Menu FileMenu = new Menu("File");  
        MenuItem fileMenuItem1=new MenuItem("New");  
        MenuItem fileMenuItem2=new MenuItem("Save");  
        MenuItem fileMenuItem3=new MenuItem("Exit");  
        Menu EditMenu=new Menu("Edit");  
        MenuItem editMenuItem1=new MenuItem("Cut");  
        MenuItem editMenuItem2=new MenuItem("Copy");  
        MenuItem editMenuItem3=new MenuItem("Paste");  
        EditMenu.getItems().addAll(editMenuItem1,editMenuItem2,editMenuItem3);  
        FileMenu.getItems().addAll(fileMenuItem1,fileMenuItem2,fileMenuItem3);  
        menubar.getMenus().addAll(FileMenu,EditMenu);  
        root.setTop(menubar);
```

```
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
    public static void main(String[] args) {  
        launch(args);  
    } }
```



Programming Practice Questions

- ▶ Create a simple GUI-based calculator using JavaFX. The calculator should perform basic operations like addition, subtraction, multiplication, and division. The GUI should include buttons for digits (0-9) and operations, as well as a display area to show the input and result.
- ▶ Create a JavaFX application with a button and a label. Each time the button is clicked, the label should update to show the current count of clicks.
- ▶ Build a simple JavaFX application with a text field and a button. When the button is clicked, display "Hello, [input text] !" in a label, where [input text] is what the user entered. Also format the text using various `javafx` text formatting classes and methods.
- ▶ Create a basic quiz application that presents a single multiple-choice question. When the user selects an answer and clicks a button, display whether their answer was correct or incorrect.
- ▶ Create a number guessing game where the application randomly selects a number between 1 and 100. The user should input their guess, and upon clicking a button, the app will indicate whether the guess is too high, too low, or correct.
- ▶ Design the following UI's using Java FX -



JAVA PROGRAMMING

Chap 9 : Event Handling using
Java FX

Introduction to Java FX Event Handling

- ▶ JavaFX provides us the flexibility to create various types of applications such as Desktop Applications, Web applications and graphical applications.
- ▶ In the modern day applications, the users play a vital role in the proper execution of the application. The user need to interact with the application in most of the cases.
- ▶ In JavaFX, An event is occurred whenever the user interacts with the application nodes.
- ▶ There are various sources by using which, the user can generate the event.
- ▶ For example, User can make the use of mouse or it can press any button on the keyboard or it can scroll any page of the application in order to generate an event.
- ▶ Hence, we can say that the events are basically the notifications that tell us that something has occurred at the user's end.
- ▶ A perfect Application is the one which takes the least amount of time in handling the events.

Types of Events

- ▶ The events can be broadly classified into the following two categories –
- ▶ Foreground Events – These events require the direct interaction of a user. They are generated as consequences of a person interacting with the graphical components in a GUI. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page, etc.
- ▶ Background Events – These events that don't require the interaction of end-user. The operating system interruptions, hardware or software failure, timer expiry, operation completion are the example of background events.

Java FX Events

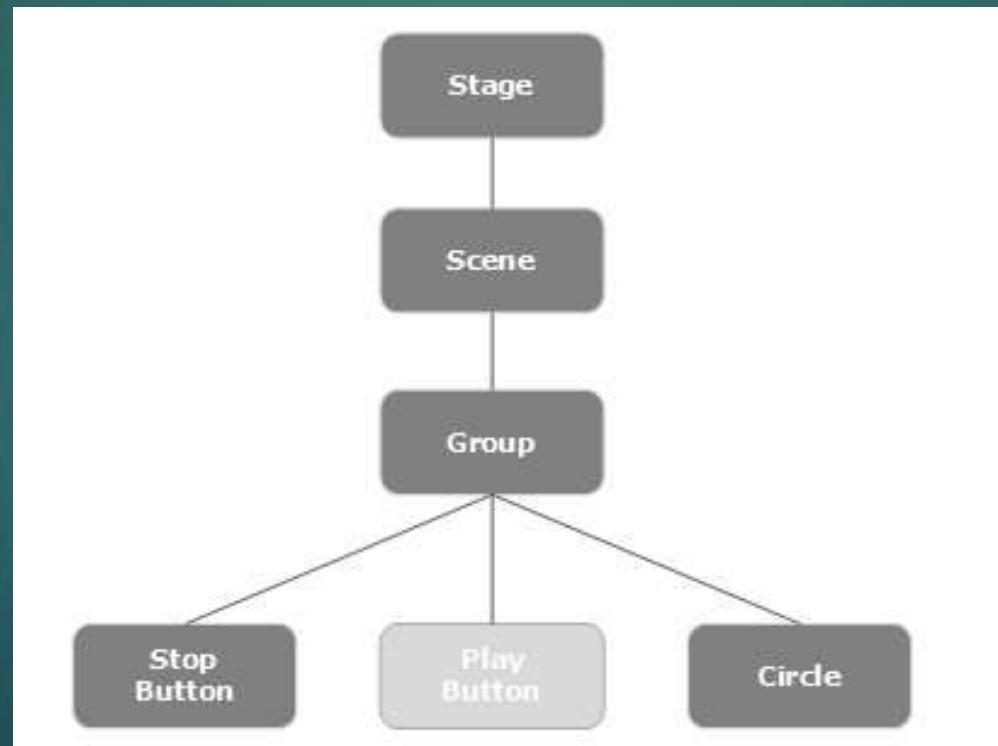
- ▶ Events are basically used to notify the application about the actions taken by the user.
- ▶ JavaFX provides the mechanism to capture the events, route the event to its target and letting the application handle the events.
- ▶ JavaFX provides support to handle a wide varieties of events.
- ▶ The class named Event of the package javafx.event is the base class for an event.
- ▶ An instance of any of its subclass is an event.
- ▶ JavaFX provides a wide variety of events. Some of them are listed below.
- ▶ **Mouse Event** – This is an input event that occurs when a mouse is clicked. It is represented by the class named MouseEvent. It includes actions like mouse clicked, mouse pressed, mouse released, mouse moved, mouse entered target, mouse exited target, etc.
- ▶ **Key Event** – This is an input event that indicates the key stroke occurred on a node. It is represented by the class named KeyEvent. This event includes actions like key pressed, key released and key typed.
- ▶ **Drag Event** – This is an input event which occurs when the mouse is dragged. It is represented by the class named DragEvent. It includes actions like drag entered, drag dropped, drag entered target, drag exited target, drag over, etc.
- ▶ **Window Event** – This is an event related to window showing/hiding actions. It is represented by the class named WindowEvent. It includes actions like window hiding, window shown, window hidden, window showing, etc.

Java FX Event Handling

- ▶ Event Handling is the mechanism that controls the event and decides what should happen, if an event occurs.
- ▶ This mechanism has the code which is known as an event handler that is executed when an event occurs.
- ▶ JavaFX provides handlers and filters to handle events.
- ▶ In JavaFX every event has –
 - ▶ Target – The node on which an event occurred. A target can be a window, scene, and a node.
 - ▶ Source – The source from which the event is generated will be the source of the event. In the above scenario, mouse is the source of the event.
 - ▶ Type – Type of the occurred event; in case of mouse event – mouse pressed, mouse released are the type of events.

Phases of Event Handling in JavaFX

- ▶ Also known as **Event Delivery Process**
- ▶ Whenever an event is generated, JavaFX undergoes the following phases in order to handle the events.
- ▶ **Route Construction :** Whenever an event is generated, the default/initial route of the event is determined by construction of an Event Dispatch chain.
- ▶ It is the path from the stage to the source Node.
- ▶ An event dispatch chain is created in the following image for the event generated on one of the scene graph node.



Phases of Event Handling in JavaFX

- ▶ **Event Capturing Phase** : Once the Event Dispatch Chain is created, the event is dispatched from the source node of the event.
- ▶ All the nodes are traversed by the event from top to bottom.
- ▶ If the event filter is registered with any of these nodes, then it will be executed.
- ▶ If any of the nodes are not registered with the event filter then the event is transferred to the target node.
- ▶ The target node processes the event in that case.
- ▶ **Event Bubbling Phase** : In the event bubbling phase, the event is travelled from the target node to the stage node (bottom to top).
- ▶ If any of the nodes in the event dispatch chain has a handler registered for the generated event, it will be executed.
- ▶ If none of these nodes have handlers to handle the event, then the event reaches the root node and finally the process will be completed.
- ▶ **Event Handlers and Filters** : Event Handlers and filters contains application logic to process an event.
- ▶ A node can be registered to more than one Event Filters.
- ▶ The interface `javafx.event.EventHandler` must be implemented by all the event handlers and filters.
- ▶ In case of parent-child nodes, you can provide a common filter/handler to the parents, which is processed as default for all the child nodes.
- ▶ during the event processing phase, a filter is executed and during the event bubbling phase, a handler is executed.

Adding and Removing Event Filter

- ▶ To add an event filter to a node, you need to register this filter using the method `addEventFilter()` of the `Node` class.

```
//Creating the mouse event handler
EventHandler<MouseEvent> eventHandler = new EventHandler<MouseEvent>() {
    @Override
    public void handle(MouseEvent e) {
        System.out.println("Hello World");
        circle.setFill(Color.DARKSLATEBLUE);
    }
};
//Adding event Filter
Circle.addEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);
```

- ▶ In the same way, you can remove a filter using the method `removeEventFilter()` as shown below

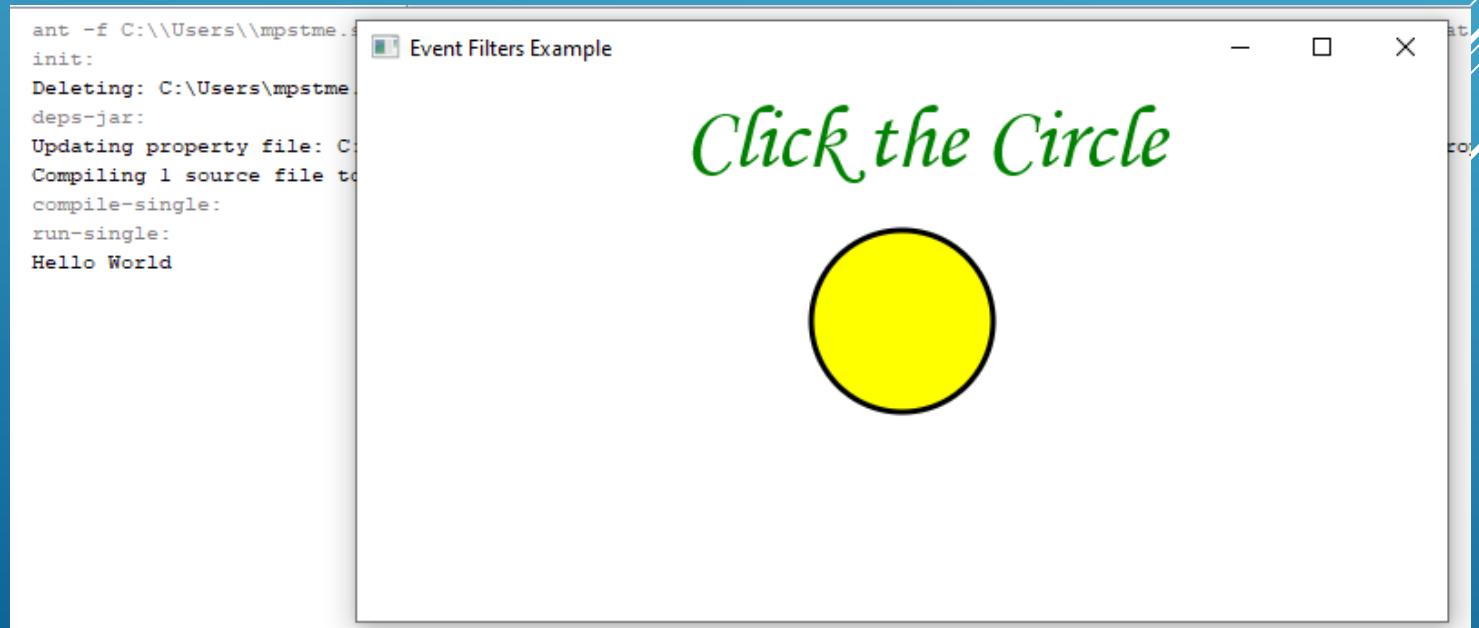
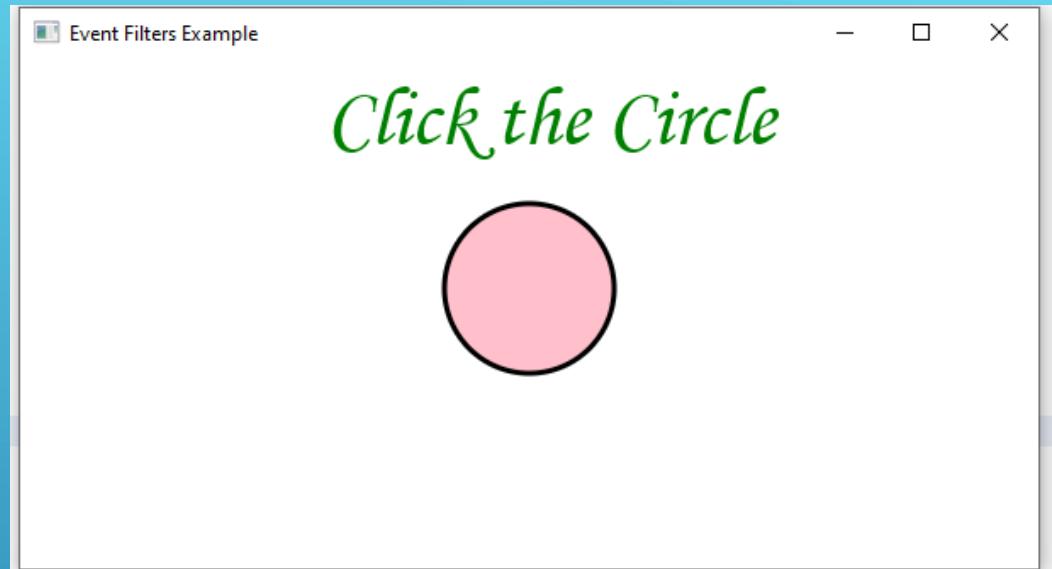
```
circle.removeEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);
```

Adding and Removing Event Filter

```
package javafxapplication1.UIControls;  
import javafx.application.Application;  
import javafx.event.EventHandler;  
import javafx.scene.Group;  
import javafx.scene.Scene;  
import javafx.scene.input.MouseEvent;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Circle;  
import javafx.scene.text.Font;  
import javafx.scene.text.FontWeight;  
import javafx.scene.text.Text;  
import javafx.stage.Stage;  
  
public class EventHandling extends Application {  
    @Override  
    public void start(Stage stage) {  
        Circle circle = new Circle();  
        circle.setCenterX(300.0f);  
        circle.setCenterY(135.0f);  
        circle.setRadius(50);  
        circle.setFill(Color.PINK);  
  
        circle.setStrokeWidth(3);  
        circle.setStroke(Color.BLACK);  
        Text text = new Text("Click the Circle");  
        text.setFont(Font.font("Monotype Corsiva", FontWeight.BOLD, 50));  
        text.setFill(Color.GREEN);  
        text.setX(180);  
        text.setY(50);  
  
        //Creating the mouse event handler  
        EventHandler<MouseEvent> eventHandler = new  
        EventHandler<MouseEvent>() {  
            @Override  
            public void handle(MouseEvent e) {  
                System.out.println("Hello World");  
                circle.setFill(Color.YELLOW);  
            } };  
  
        //Registering the event filter  
        circle.addEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);  
    }  
}
```

Adding and Removing Event Filter

```
Group root = new Group(circle, text);
Scene scene = new Scene(root, 600, 300);
stage.setTitle("Event Filters Example");
stage.setScene(scene);
stage.show();
}
public static void main(String args[]){
    launch(args);
}
}
```



Convenience Methods

- ▶ Some of the classes in JavaFX define event handler properties.
- ▶ By setting the values to these properties using their respective setter methods, you can register to an event handler. These methods are known as convenience methods.
- ▶ Most of these methods exist in the classes like Node, Scene, Window, etc., and they are available to all their sub classes.
- ▶ Some EventHandler properties along with their setter methods (convenience methods) are described in the following table.

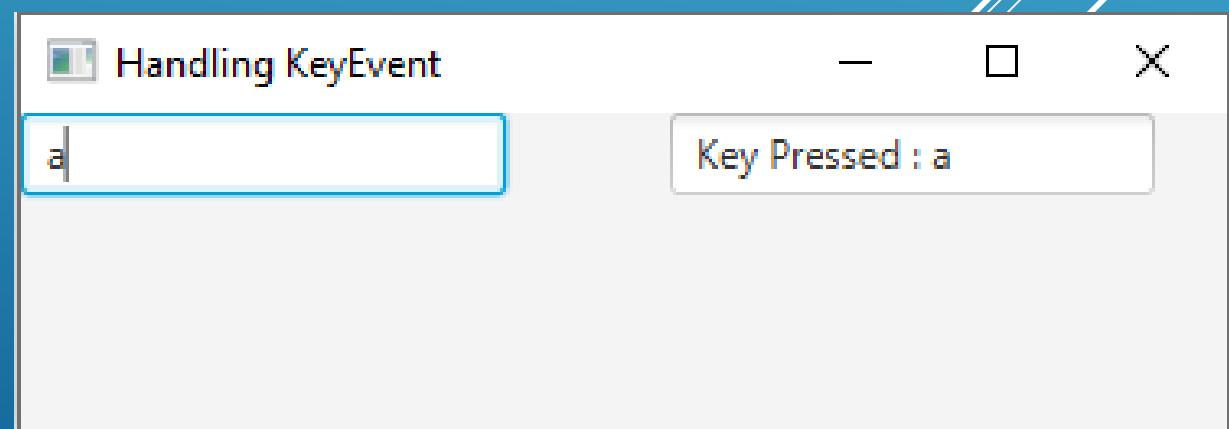
| EventHandler Property | Description | Setter Methods |
|--------------------------|---|---|
| onDragDetected | This is of the type EventHandler of MouseEvent. This indicates a function which is to be called when the drag gesture is detected. | setOnDragDetected(EventHandler value) |
| onDragDone | This is of the type EventHandler of DragEvent. | setOnDragDone(EventHandler value) |
| onDragDropped | This is of the type EventHandler of DragEvent. This is assigned to a function which is to be called when the mouse is released during a drag and drop operation. | setOnDragDropped(EventHandler value) |
| onInputMethodTextChanged | This is of the type EventHandler of InputMethodEvent. This is assigned to a function which is to be called when the Node has focus and the input method text has changed. | setOnInputMethodTextChanged(Event Handler value) |
| onKeyPressed | This is of the type EventHandler of KeyEvent. This is assigned to a function which is to be called when the Node has focus and key is pressed. | setOnKeyPressed(EventHandler value) |
| onKeyReleased | This is of the type EventHandler of KeyEvent. This is assigned to a function which is to be called when the Node has focus and key is released. | setOnKeyReleased(EventHandler value) |
| onKeyTyped | This is of the type EventHandler of KeyEvent. This is assigned to a function which is to be called when the Node has focus and key is typed. | setOnKeyTyped(EventHandler value) |

Convenience Methods

| EventHandler Property | Description | Setter Methods |
|-----------------------|---|---|
| onMouseClicked | This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse button is clicked on the node. | setOnMouseClicked(EventHandler value) |
| onMouseDragEntered | This is of the type EventHandler of MouseDragEvent. This is assigned to a function which is to be called when a full press drag release gesture enters the node. | setOnMouseDragEntered(EventHandler value) |
| onMouseDragExited | This is of the type EventHandler of MouseDragEvent. This is assigned to a function which is to be called when a full press drag release gesture exits the node. | setOnMouseDragExited(Event Handler value) |
| onMouseDragged | This is of the type EventHandler of MouseDragEvent. This is assigned to a function which is to be called when the mouse button is pressed and dragged on the node. | setOnMouseDragged(EventHandler value) |
| onMouseDragOver | This is of the type EventHandler of MouseDragEvent. This is assigned to a function which is to be called when a full press drag release gesture progresses within the node. | setOnMouseDragOver(EventHandler value) |
| onMouseDragReleased | This is of the type EventHandler of MouseDragEvent. This is assigned to a function which is to be called when a full press drag release gesture ends within the node. | setOnMouseDragReleased(EventHandler value) |
| onMouseEntered | This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse enters the node. | setOnMouseEntered(EventHandler value) |
| onMouseExited | This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse exits the node. | setOnMouseExited(EventHandler value) |
| onMouseMoved | This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse moves within the node and no button has been pushed. | setOnMouseMoved(EventHandler value) |
| onMousePressed | This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse button is pressed on the node. | setOnMousePressed(EventHandler value) |
| onMouseReleased | This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse button is released on the node. | setOnMouseReleased(EventH andler value) |

Convenience Methods Example (setOnKeyPressed)

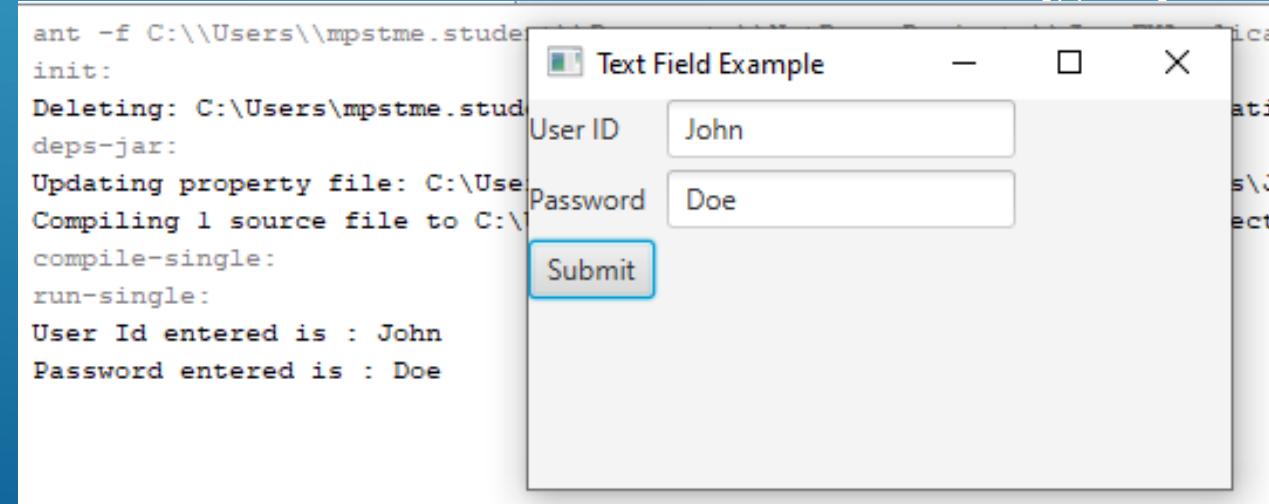
```
package javafxapplication1.UIControls;  
import javafx.application.Application;  
import javafx.event.EventHandler;  
import javafx.scene.Group;  
import javafx.scene.Scene;  
import javafx.scene.control.TextField;  
import javafx.scene.input.KeyEvent;  
import javafx.stage.Stage;  
import javafx.scene.layout.GridPane;  
public class ConvenienceMethodsDemo extends Application{  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        TextField tf1 = new TextField();  
        TextField tf2 = new TextField();  
        //Handling KeyEvent for textfield 1  
        tf1.setOnKeyPressed(new EventHandler<KeyEvent>() {  
            @Override  
            public void handle(KeyEvent k) {  
                // TODO Auto-generated method stub  
                tf2.setText("Key Pressed :"+" "+k.getText());  
            }  
        });  
        //setting group and scene  
        GridPane root = new GridPane();  
        root.addRow(0, tf1, tf2);  
        root.setHgap(50);  
        Scene scene = new Scene(root,500,200);  
        primaryStage.setScene(scene);  
        primaryStage.setTitle("Handling KeyEvent");  
        primaryStage.show();  
    }  
    public static void main(String[] args) {  
        launch(args);  
    } }
```



Convenience Methods Example (setOnAction)

```
package javafxapplication1.UIControls;  
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextField;  
import javafx.scene.layout.GridPane;  
import javafx.stage.Stage;  
import javafx.event.ActionEvent;  
import javafx.event.EventHandler;  
public class TextFieldDemo extends Application {  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        // TODO Auto-generated method stub  
        Label user_id=new Label("User ID");  
        Label password = new Label("Password");  
        TextField tf1=new TextField();  
        TextField tf2=new TextField();  
        Button b = new Button("Submit");  
        GridPane root = new GridPane();  
        root.addRow(0, user_id, tf1);  
        root.addRow(1, password, tf2);  
        root.addRow(2, b);  
        root.setHgap(5);  
        root.setVgap(5);  
        b.setOnAction(new EventHandler<ActionEvent>() {
```

```
        @Override  
        public void handle(ActionEvent args) {  
            // TODO Auto-generated method stub  
            System.out.println("User Id entered is : " + tf1.getText());  
            System.out.println("Password entered is : " + tf2.getText());  
        }  
    } );  
    Scene scene=new Scene(root,300,300);  
    primaryStage.setScene(scene);  
    primaryStage.setTitle("Text Field Example");  
    primaryStage.show();  
}  
public static void main(String[] args) {  
    launch(args);  
} }
```



Event Handlers

- ▶ JavaFX facilitates us to use the Event Handlers to handle the events generated by Keyboard Actions, Mouse Actions, and many more source nodes.
- ▶ Event Handlers are used to handle the events in the Event bubbling phase. There can be more than one Event handlers for a single node.
- ▶ We can also use single handler for more than one node and more than one event type. In this part of the tutorial, we will discuss, how the Event Handlers can be used for processing events.

Adding an Event Handler

- ▶ Event Handler must be registered for a node in order to process the events in the event bubbling phase.
- ▶ Event handler is the implementation of the EventHandler interface.
- ▶ The handle() method of the interface contains the logic which is executed when the event is triggered.
- ▶ To register the EventHandler, addEventHandler() is used.
- ▶ In this method, two arguments are passed. One is event type and the other is EventHandler object.
- ▶ The syntax of addEventHandler() is given below.

```
node.addEventHandler(<EventType>,new EventHandler<Event-Type>()
{
    public void handle(<EventType> e)
    {
        //handling code
    }
});
```

Event Handlers

- ▶ Removing EventHandler
- ▶ when we no longer need an EventHandler to process the events for a node or event types, we can remove the EventHandler by using the method removeEventHandler() method.
- ▶ This method takes two arguments, event type and EventHandler Object.
- ▶ `node.removeEventHandler(<EventType>,handler);`

JAVA PROGRAMMING

Chap 10 : Java and Database
Programming

Need for Database Programming with Java

- ▶ Database programming is an integral part of Java programming because it allows applications to store, retrieve, manipulate, and manage data efficiently. Here are some reasons why database programming is essential in Java:
 - ▶ **Data Storage:** Many applications require persistent data storage, where data can be saved and retrieved even after the application is closed or restarted. Databases provide a structured and efficient way to store data.
 - ▶ **Data Retrieval:** Java applications often need to retrieve data from databases to present it to users or perform operations on it. Databases allow developers to query and retrieve specific data easily.
 - ▶ **Data Manipulation:** Databases enable developers to add, update, and delete data, which is crucial for maintaining the integrity of data and performing operations like CRUD (Create, Read, Update, Delete).
 - ▶ **Scalability:** Databases are designed to handle large amounts of data efficiently. Java applications can scale effectively by using databases to store and manage data.
 - ▶ **Data Security:** Databases provide security features such as authentication, authorization, and encryption to protect sensitive data. This is crucial for applications that deal with user information or other confidential data.
 - ▶ **Data Consistency:** Databases enforce data consistency through constraints and transactions, ensuring that data remains accurate and reliable.
 - ▶ **Data Sharing:** Databases enable multiple users or applications to access and share data concurrently while maintaining data integrity.
 - ▶ **Reporting and Analysis:** Java applications often need to generate reports or perform complex data analysis. Databases can store and structure data in a way that makes these tasks more efficient.
 - ▶ **Persistence Layer:** In many Java applications, a database serves as the persistence layer, separating the application's logic from the data storage concerns. This separation allows for more maintainable and scalable code.

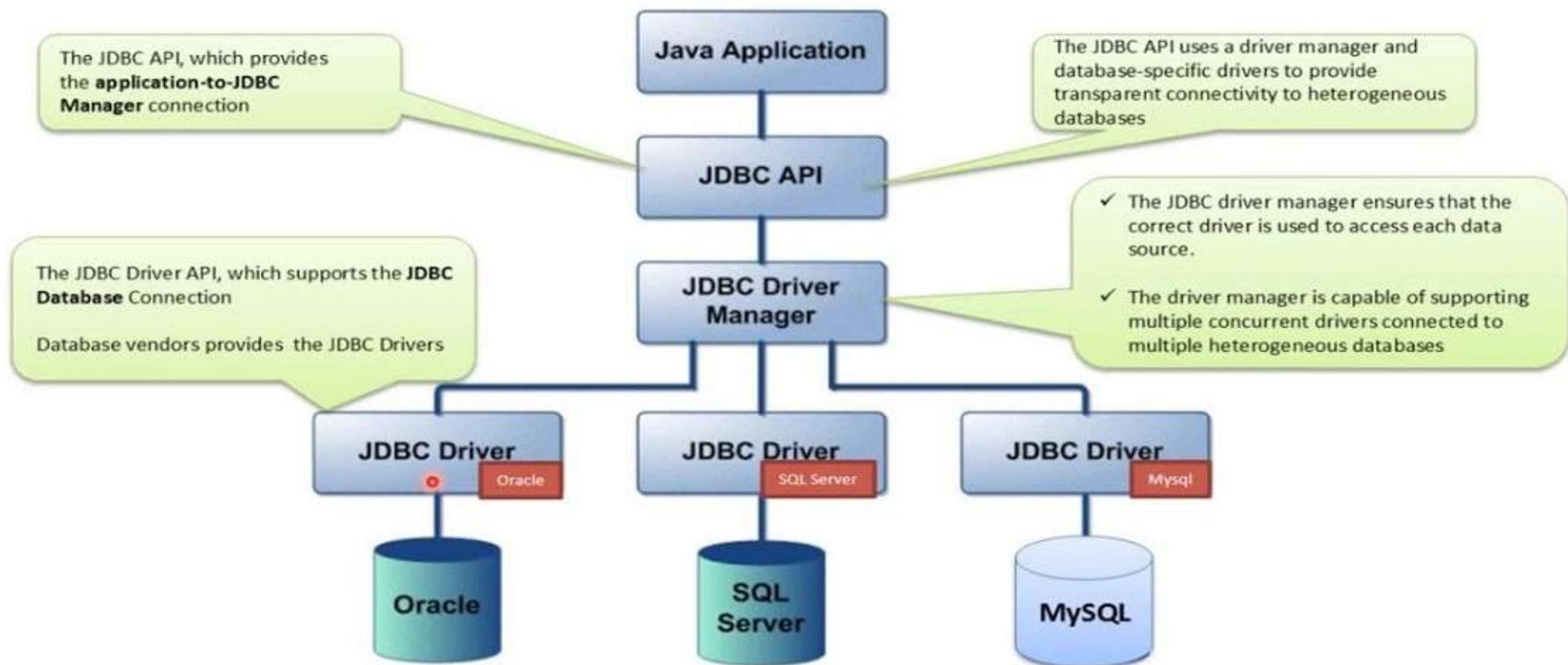
Need for Database Programming with Java

- ▶ **Integration:** Java applications can easily integrate with various database management systems (DBMS) such as MySQL, PostgreSQL, Oracle, MongoDB, and others. This flexibility allows developers to choose the database system that best suits their application's requirements.
- ▶ To interact with databases in Java, developers typically use Database Connectivity APIs like JDBC (Java Database Connectivity) or Object-Relational Mapping (ORM) frameworks like Hibernate. These tools provide the necessary abstractions and methods to connect to databases, execute queries, and handle database-related tasks seamlessly within Java applications.

JDBC

- ▶ JDBC stands for Java Database Connectivity is a Java API(Application Programming Interface) used to interact with databases.
- ▶ JDBC is a specification from Sun Microsystems and it is used by Java applications to communicate with relational databases.
- ▶ We can use JDBC to execute queries on various databases and perform operations like SELECT, INSERT, UPDATE and DELETE.
- ▶ JDBC API helps Java applications interact with different databases like MSSQL, ORACLE, MYSQL, etc.
- ▶ It consists of classes and interfaces of JDBC that allow the applications to access databases and send requests made by users to the specified database.
- ▶ **Purpose of JDBC**
 - ▶ Java programming language is used to develop enterprise applications.
 - ▶ These applications are developed with intention of solving real-life problems and need to interact with databases to store required data and perform operations on it.
 - ▶ Hence, to interact with databases there is a need for efficient database connectivity, ODBC(Open Database Connectivity) driver is used for the same.
 - ▶ ODBC is an API introduced by Microsoft and used to interact with databases.
 - ▶ It can be used by only the windows platform and can be used for any language like C, C++, Java, etc. ODBC is procedural.
 - ▶ JDBC is an API with classes and interfaces used to interact with databases. It is used only for Java languages and can be used for any platform. JDBC is highly recommended for Java applications as there are no performance or platform dependency problems.

JDBC Architecture



JDBC Architecture

- ▶ **Application :** Applications in JDBC architecture are java applications like applets or servlet that communicates with databases.
- ▶ **JDBC API :** The JDBC API is an Application Programming Interface used to create Databases. JDBC API uses classes and interfaces to connect with databases. Some of the important classes and interfaces defined in JDBC architecture in java are the DriverManager class, Connection Interface, etc. The primary packages in JDBC are java.sql and javax.sql, which contain classes and interfaces for core database operations.
- ▶ **DriverManager :** DriverManager class in the JDBC architecture is used to establish a connection between Java applications and databases. Using the getConnection method of this class a connection is established between the Java application and data sources. It loads the appropriate JDBC driver based on the provided database URL and establishes a connection to the database.
- ▶ **JDBC Drivers :** JDBC drivers are used to connecting with data sources. All databases like Oracle, MSSQL, MYSQL, etc. have their drivers, to connect with these databases we need to load their specific drivers. Class is a java class used to load drivers. Class.forName() method is used to load drivers in JDBC architecture. There are four types of JDBC drivers:
 - a. Type-1: JDBC-ODBC Bridge Driver (now deprecated)
 - b. Type-2: Native-API Driver
 - c. Type-3: Network Protocol Driver
 - d. Type-4: Thin Driver (also known as the Direct-to-Database Pure Java Driver)
- ▶ Each type of driver uses a different mechanism to connect to the database, and the choice of driver depends on the specific database and deployment requirements.

JDBC Architecture

- ▶ **Data Sources :** Data Sources in the JDBC architecture are the databases that we can connect using this API. These are the sources where data is stored and used by Java applications. JDBC API helps to connect various databases like Oracle, MYSQL, MSSQL, PostgreSQL, etc.

Types of Drivers

- ▶ JDBC (Java Database Connectivity) drivers are platform-specific or database-specific implementations that allow Java applications to connect to and interact with relational databases. There are four main types of JDBC drivers, known as JDBC driver types 1, 2, 3, and 4. Each type has its own characteristics and is suited for different scenarios:
- ▶ **Type-1 JDBC Driver (JDBC-ODBC Bridge Driver):**
 - ▶ Type-1 drivers are also known as JDBC-ODBC Bridge drivers.
 - ▶ They use the JDBC-ODBC bridge to connect to the database.
 - ▶ The bridge relies on the ODBC (Open Database Connectivity) driver installed on the system to communicate with the database.
 - ▶ Type-1 drivers are considered legacy and have been deprecated in favor of other driver types. They are not recommended for new development.
- ▶ **Type-2 JDBC Driver (Native-API Driver):**
 - ▶ Type-2 drivers are known as Native-API drivers.
 - ▶ They are platform-specific drivers that use a database-specific native API to communicate with the database.
 - ▶ These drivers provide better performance than Type-1 drivers because they communicate directly with the database without the ODBC layer.
 - ▶ However, they are still somewhat dependent on the platform and database, which can limit portability.

Types of Drivers

- ▶ **Type-3 JDBC Driver (Network Protocol Driver):**
- ▶ Type-3 drivers are also called Network Protocol drivers.
- ▶ They use a middleware or network protocol to connect to the database server.
- ▶ The client-side of the driver translates JDBC calls into a database-independent protocol, which is then transmitted over the network to a server-side component that interacts with the database.
- ▶ Type-3 drivers are generally platform-independent and can work with different databases as long as there is a compatible server-side component.

- ▶ **Type-4 JDBC Driver (Thin Driver or Direct-to-Database Pure Java Driver):**
- ▶ Type-4 drivers are known as Thin drivers.
- ▶ They are pure Java drivers that communicate directly with the database using a database-specific protocol.
- ▶ Type-4 drivers are highly portable because they don't rely on platform-specific libraries or middleware.
- ▶ They are typically the preferred choice for modern Java applications as they offer good performance and platform independence.
- ▶ Examples of Type-4 drivers include Oracle Thin Driver, PostgreSQL JDBC Driver, and MySQL Connector/J.

- ▶ The choice of JDBC driver type depends on various factors, including the specific database being used, the application's portability requirements, and performance considerations. In general, Type-4 drivers are recommended for new Java applications because they offer good performance and are platform-independent. However, in some legacy or specialized scenarios, other driver types may still be used.

JDBC Components

- ▶ **JDBC API :** The JDBC API is an Application Programming Interface used to create Databases. JDBC API uses classes and interfaces to connect with databases. Some of the important classes and interfaces defined in JDBC architecture in java are the DriverManager class, Connection Interface, etc. The primary packages in JDBC are java.sql and javax.sql, which contain classes and interfaces for core database operations.
- ▶ **DriverManager :** DriverManager class in the JDBC architecture is used to establish a connection between Java applications and databases. Using the getConnection method of this class a connection is established between the Java application and data sources. It loads the appropriate JDBC driver based on the provided database URL and establishes a connection to the database.
- ▶ **JDBC Test Suite :** JDBC Test Suite is used to test operations being performed on databases using JDBC drivers. JDBC Test Suite tests operations such as insertion, updating, and deletion.
- ▶ **JDBC-ODBC Bridge Drivers :** As the name suggests JDBC-ODBC Bridge Drivers are used to translate JDBC methods to ODBC function calls. JDBC-ODBC Bridge Drivers are used to connect database drivers to the database. Even after using JDBC for Java Enterprise Applications, we need an ODBC connection for connecting with databases.
JDBC-ODBC Bridge Drivers are used to bridge the same gap between JDBC and ODBC drivers. The bridge translates the object-oriented JDBC method call to the procedural ODBC function call. It makes use of the sun.jdbc.odbc package. This package includes a native library to access ODBC characteristics.

JDBC Interfaces

- ▶ JDBC API uses various interfaces to establish connections between applications and data sources. Some of the popular interfaces in JDBC API are as follows:
- ▶ **Driver interface** - The JDBC Driver interface provided implementations of the abstract classes such as java.sql.Connection, Statement, PreparedStatement, Driver, etc. provided by the JDBC API.
- ▶ **Connection interface** - The connection interface is used to create a connection with the database. getConnection() method of DriverManager class of the Connection interface is used to get a Connection object.
- ▶ **Statement interface** - The Statement interface provides methods to execute SQL queries on the database. executeQuery(), executeUpdate() methods of the Statement interface are used to run SQL queries on the database.
- ▶ **ResultSet interface** - ResultSet interface is used to store and display the result obtained by executing a SQL query on the database. executeQuery() method of statement interface returns a resultset object.
- ▶ **RowSet interface** - RowSet interface is a component of Java Bean. It is a wrapper of ResultSet and is used to keep data in tabular form.
- ▶ **ResultSetMetaData interface** - Metadata means data about data. ResultSetMetaData interface is used to get information about the resultset interface. The object of the ResultSetMetaData interface provides metadata of resultset like number of columns, column name, total records, etc.
- ▶ **DatabaseMetaData interface** - DatabaseMetaData interface is used to get information about the database vendor being used. It provides metadata like database product name, database product version, total tables/views in the database, the driver used to connect to the database, etc.

JDBC Classes

- ▶ JDBC API uses various classes that implement the above interfaces. Methods of these classes in JDBC API are used to create connections and execute queries on databases. A list of most commonly used class in JDBC API are as follows:
- ▶ **DriverManager class** - DriverManager class is a member of the java.sql package. It is used to establish a connection between the database and its driver.
- ▶ **Blob class** - A java.sql.Blob is a binary large object that can store large amounts of binary data, such as images or other types of files. Fields defined as TEXT also hold large amounts of data.
- ▶ **Clob class** - The java.sql.Clob interface of the JDBC API represents the CLOB datatype. Since the Clob object in JDBC is implemented using an SQL locator, it holds a logical pointer to the SQL CLOB (not the data).
- ▶ **Types class** - Type class defined and store constants that are used to identify generic SQL types also known as JDBC types.

Steps for querying the database with JDBC

- ▶ Java applications need to be programmed for interacting with data sources. JDBC Drivers for specific databases are to be loaded in a java application for JDBC support which can be done dynamically at run time. These JDBC drivers communicate with the respective data source.
- ▶ Steps to connect a Java program using JDBC API.

1. Import JDBC Packages: Import the necessary JDBC packages at the beginning of your Java program. You'll typically need to import classes from the `java.sql` package.

2. Load Driver: Load JDBC Driver for specific databases using `forName()` method of class `Class`. Syntax: `Class.forName("com.mysql.jdbc.Driver")`

3. Create Connection: Create a connection with a database using `DriverManager` class. Database credentials are to be passed while establishing the connection. Syntax: `DriverManager.getConnection()`

4. Create SQL Query: To manipulate the database we need to create a query using commands like `INSERT`, `UPDATE`, `DELETE`, etc. These queries are created and stored in string format. Syntax: `String sql_query = "INSERT INTO Student(name, roll_no) values('ABC','XYZ')"`

5. Create SQL Statement: The query we have created is in form of a string. To perform the operations in the string on a database we need to fire that query on the database. To achieve this we need to convert a string object into SQL statements. This can be done using `createStatement()` and `prepareStatement()` interfaces.

Syntax: `Statement St = con.createStatement();`

Steps for querying the database with JDBC

6. Execute Statement: To execute SQL statements on the database we can use two methods depending on which type of query we are executing.

Execute Update: Execute update method is used to execute queries like insert, update, delete, etc. Return type of executeUpdate() method is int. Syntax: int check = st.executeUpdate(sql);

Execute Query: Execute query method is used to execute queries used to display data from the database, such as select. Return type of executeQuery() method is result set. Syntax: Resultset = st.executeQuery(sql);

7. Closing Statement: After performing operations on the database, it is better to close every interface object to avoid further conflicts. Syntax: con.close();