3140705
Object Oriented Programming - I

Unit-1 Introduction to Java & Elementary Programming

Prof. Hardik A. Doshi

9978911553

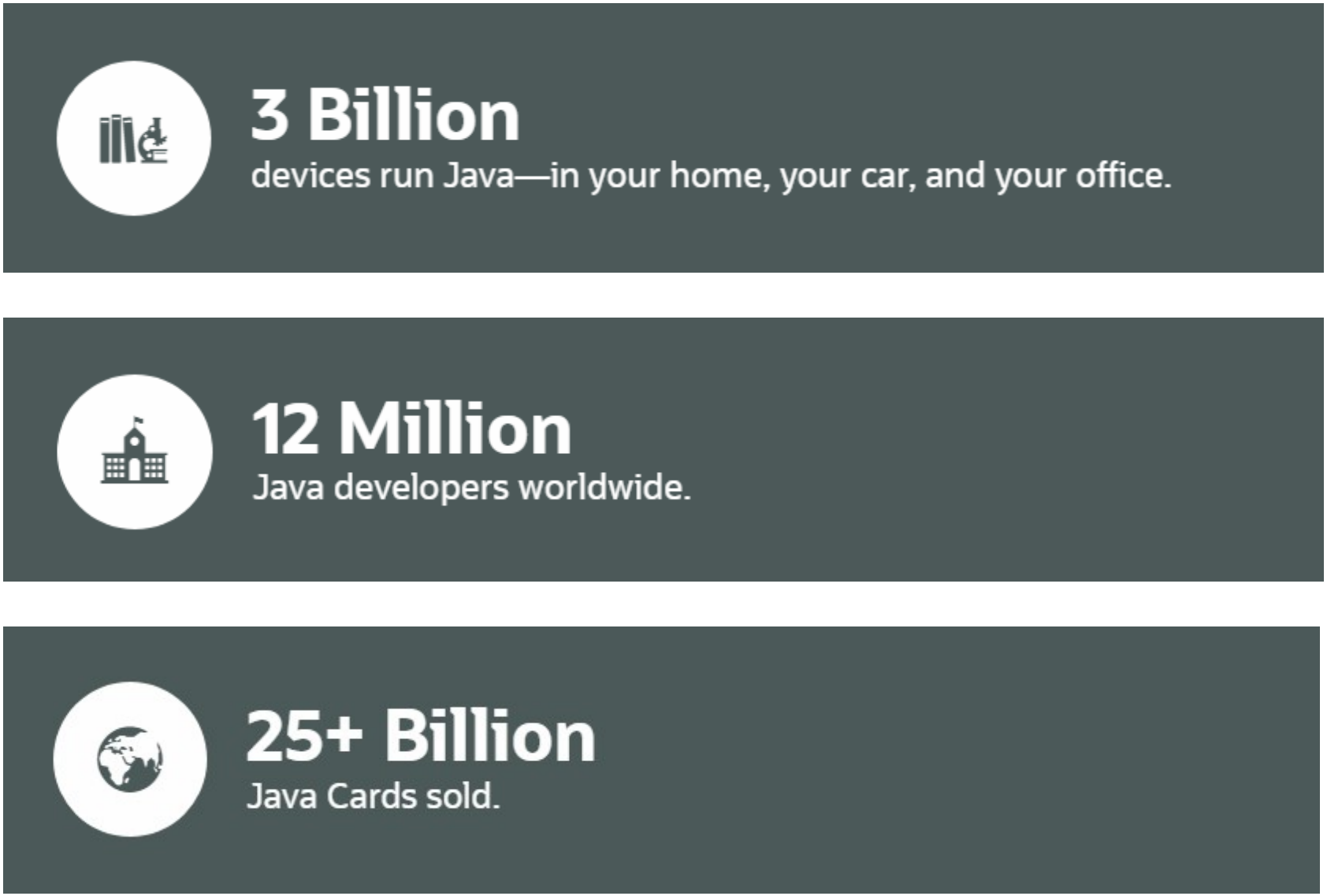
hardik.doshi@darshan.ac.in

Dedicated Faculty, Committed Education

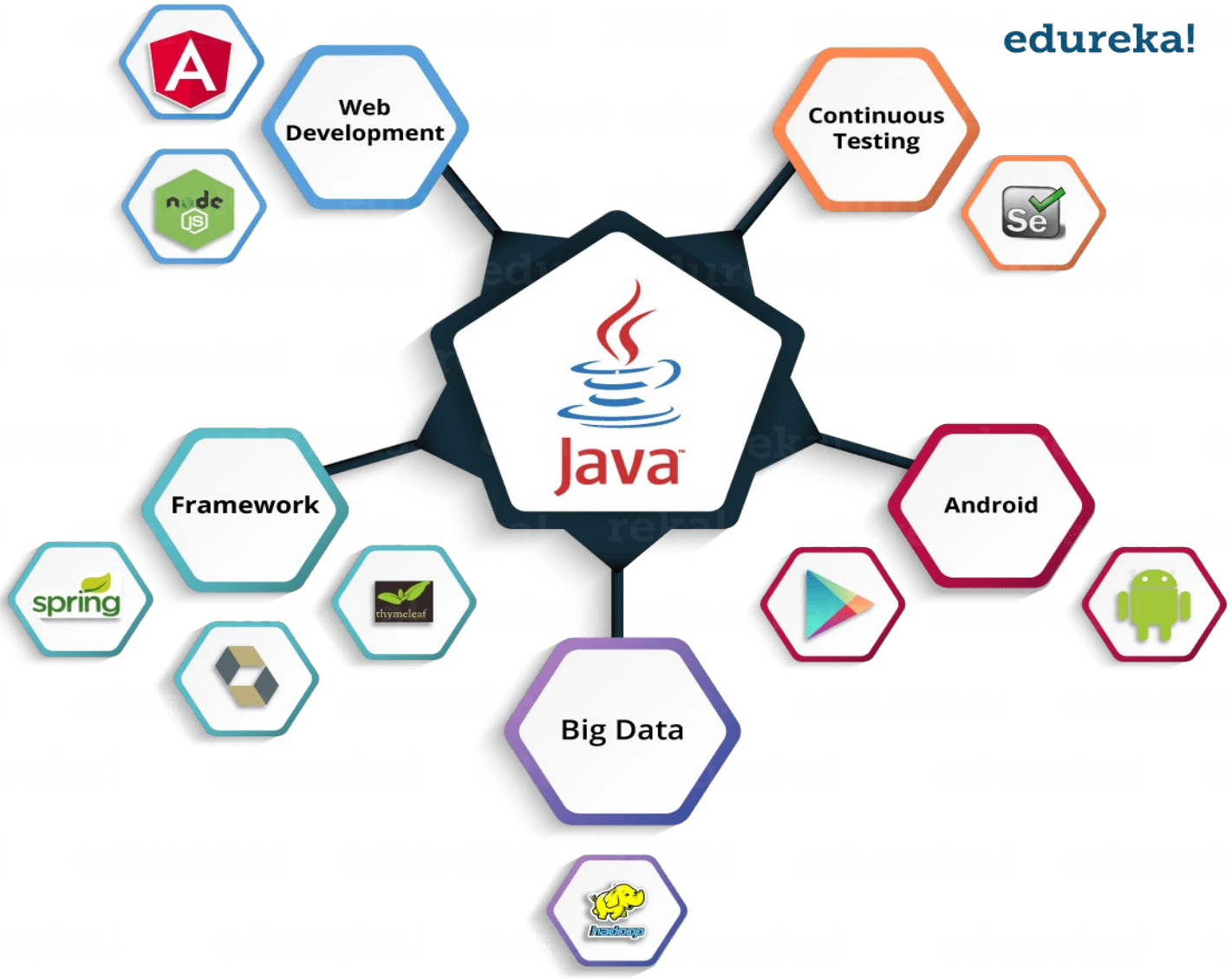
Darshan

Institute of Engineering & Technology

Java Around the World



What Java is used for?

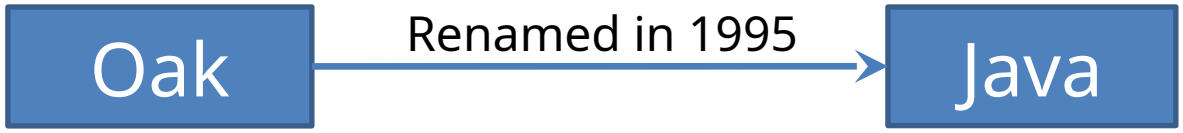


What Java is used for?

- **Banking:** To deal with transaction management.
- **Retail:** Billing applications that you see in a store/restaurant are completely written in Java.
- **Information Technology:** Java is designed to solve implementation dependencies.
- **Android:** Applications are either written in Java or use Java API.
- **Financial services:** It is used in server-side applications.
- **Stock market:** To write algorithms as to which company they should invest in.
- **Big Data:** Hadoop MapReduce framework is written using Java.
- **Scientific and Research Community:** To deal with huge amount of data.

History of Java

- James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan (Green Team) at Sun Microsystems, Inc. conceived Java in 1991.
- Initially named as Oak language.



What is Java?

Concurrent

- executes many statements instead of sequentially executing it

Object Oriented

- supports encapsulation, inheritance and polymorphism

Independent

- follows the logic of “Write once, Run anywhere”

Java Buzzwords

Simple: Java inherits C/C++ syntax and many object-oriented features of C++.

Object Oriented: “Everything is an object” paradigm, which possess some state, behavior and all the operations are performed using these objects.

Robust: Java has a strong memory management system. It helps in eliminating error as it checks the code during compile and runtime.

Multithreaded: Java supports multiple threads of execution, including a set of synchronization primitives. This makes programming with threads much easier.

Java Buzzwords

Architectural Neutral: Java is platform independent which means that any application written on one platform can be easily ported to another platform.

Interpreted: Java is compiled to bytecodes, which are interpreted by a Java run-time environment.

High Performance: Java achieves high performance through the use of bytecode which can be easily translated into native machine code. With the use of JIT (Just-In-Time) compilers, Java enables high performance.

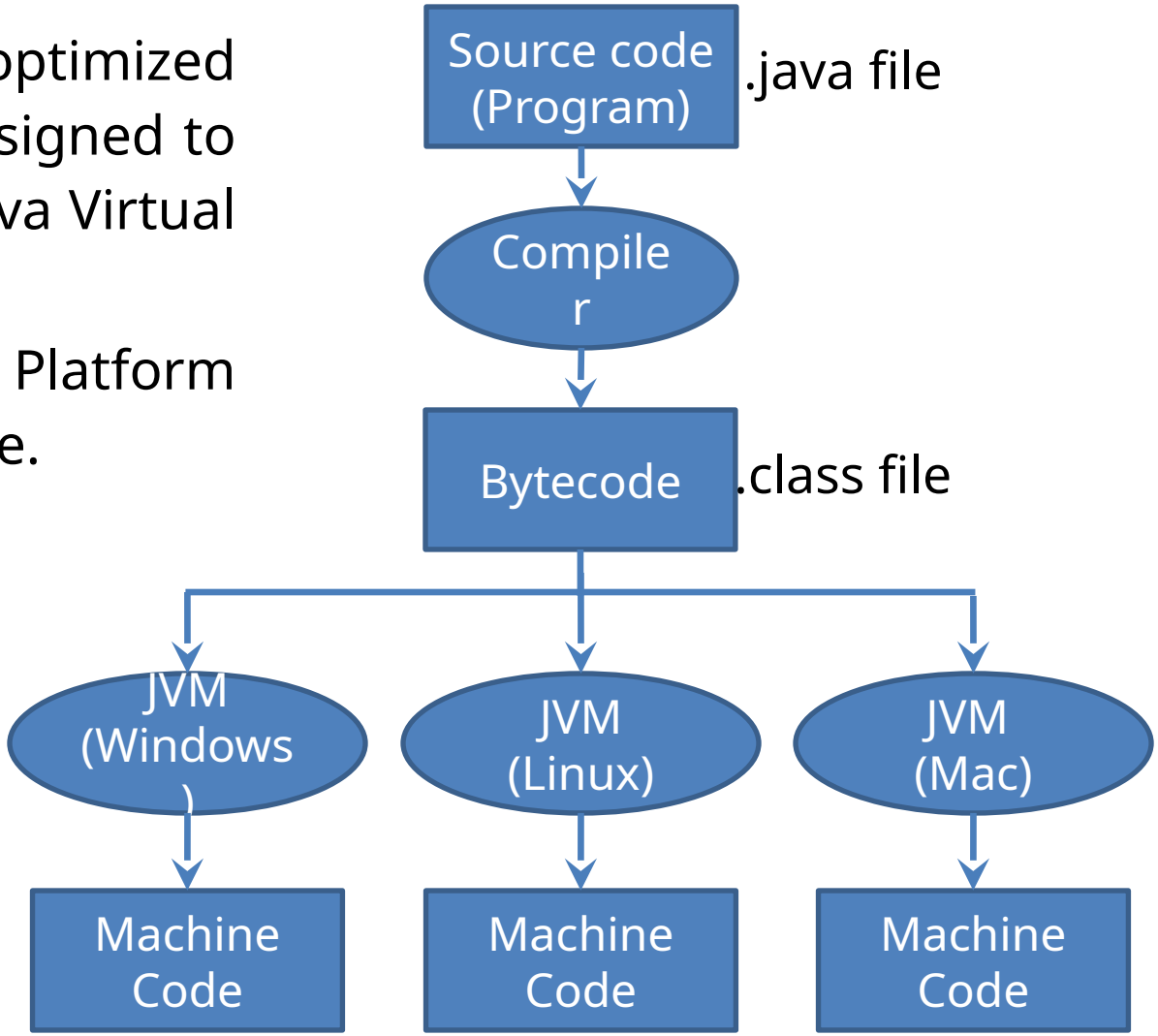
Java Buzzwords

Distributed: Java provides a feature which helps to create distributed applications. Using Remote Method Invocation (RMI), a program can invoke a method of another program across a network and get the output. You can access files by calling the methods from any machine on the internet.

Dynamic: Java has ability to adapt to an evolving environment which supports dynamic memory allocation due to which memory wastage is reduced and performance of the application is increased.

Java's Magic : Bytecode

- Bytecode is a highly optimized set of instructions designed to be executed by the Java Virtual Machine(JVM).
- Bytecode makes Java Platform independent language.



Components of Java

- Java Virtual Machine (JVM)
- Java Runtime Environment (JRE)
- Java Development Kit (JDK) [Current Version: JDK 13.0.1]

Java Virtual Machine (JVM)

- JVM is
  - An abstract machine
  - Provides a run-time environment to execute bytecode
- It follows 3 notations
  1. Specification
  2. Implementation
  3. Runtime Instance
- It performs following operation:
  - Loads code
  - Verifies code
  - Executes code
  - Provides runtime environment

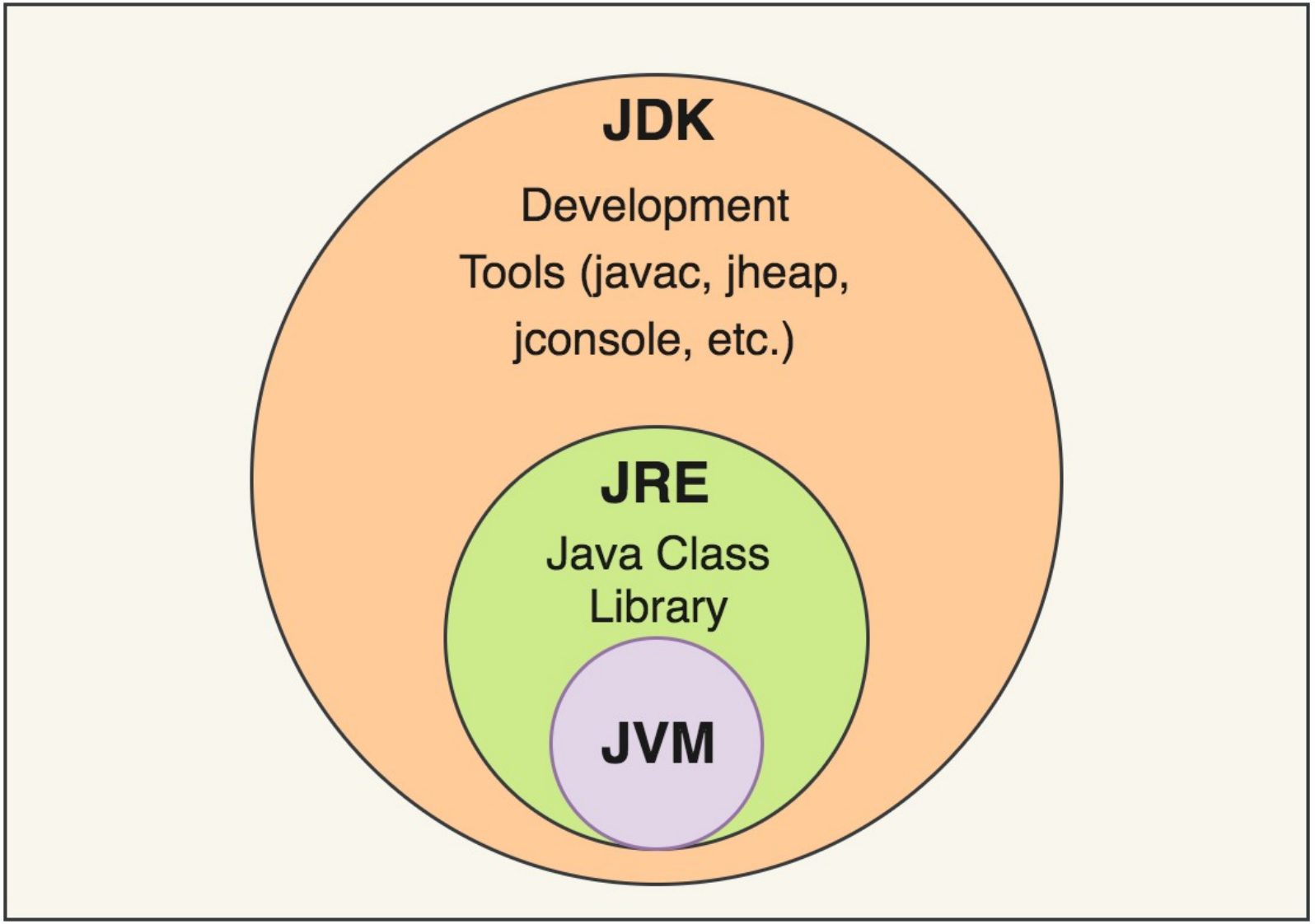
Java Runtime Environment (JRE)

- JRE refers to a runtime environment in which Java bytecode can be executed.
- It implements the JVM (Java Virtual Machine) and provides all the class libraries and other support files that JVM uses at runtime.
- Basically, it is an implementation of the JVM which physically exists.

Java Development Kit (JDK)

- It is the tool necessary to
  - Compile
  - Document
  - Package Java programs
- The JDK completely includes JRE which contains tools for Java programmers.
- Along with JRE, it includes an
  - compiler (javac)
  - application launcher (java / appletviewer)
  - interpreter/loader
  - archiver (jar)
  - documentation generator (javadoc)
  - other tools needed in Java development.
- In short, it contains JRE + development tools.
- The Java Development Kit is freeware available on https://www.oracle.com/technetwork/java/javase/downloads/index.html

JVM vs JRE vs JDK



JDK Version History

Version	Release Date
JDK Beta	1995
JDK 1.0	January 1996
JDK 1.1	February 1997
J2SE 1.2	December 1998
J2SE 1.3	May 2000
J2SE 1.4	February 2002
J2SE 5.0	September 2004
Java SE 6	December 2006

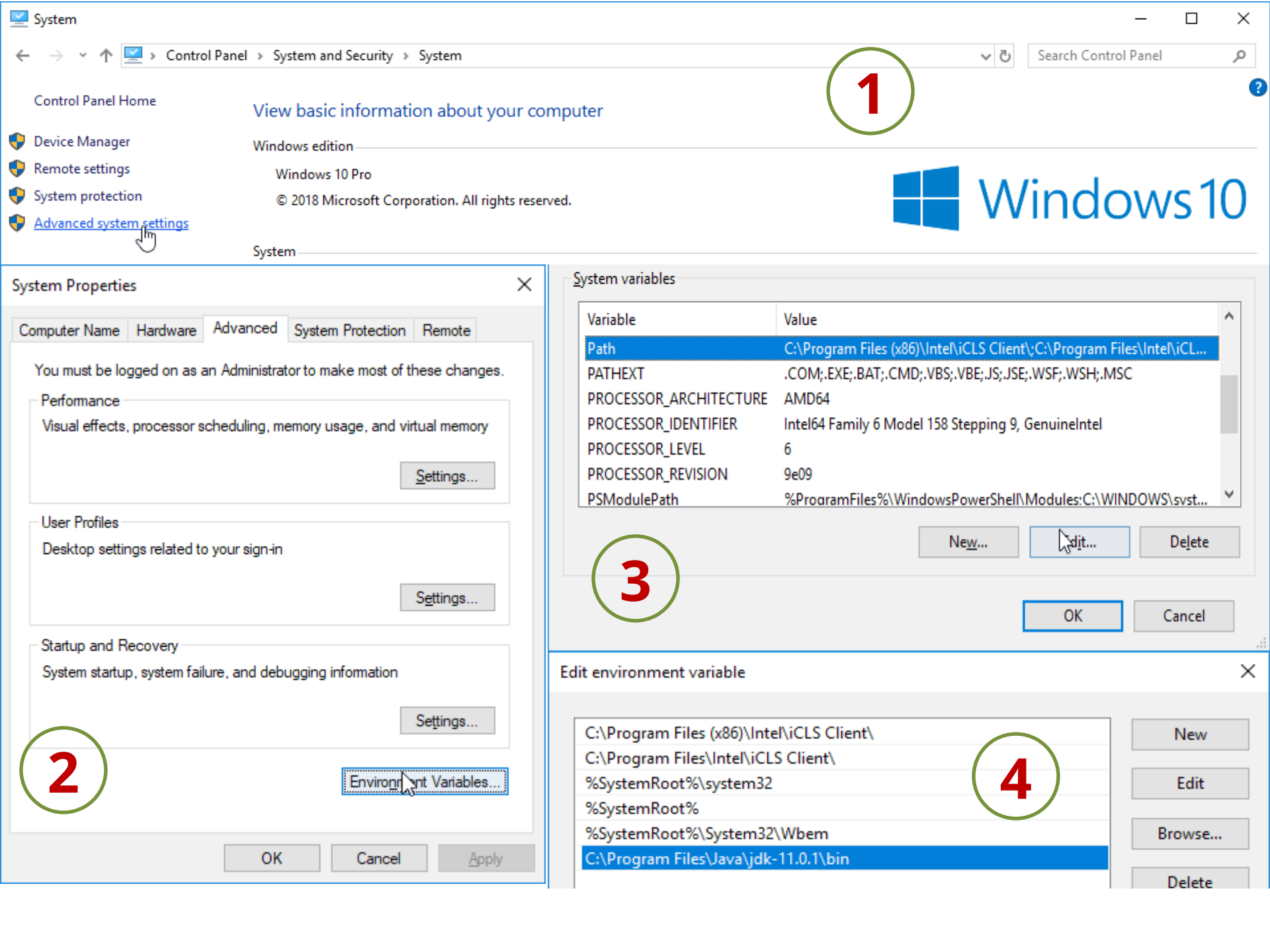
Version	Release Date
Java SE 7	July 2011
Java SE 8 (LTS)	March 2014
Java SE 9	September 2017
Java SE 10	March 2018
Java SE 11 (LTS)	September 2018
Java SE 12	March 2019
Java SE 13	September 2019

Installing JDK

- Download JDK for Windows platform (.exe) from

  https://www.oracle.com/technetwork/java/javase/downloads/index.html

- Install the executable of JDK
- Set the path variable of System variables by performing following steps
  - Go to "System Properties" (Right click This PC → Properties → Advanced System Settings)
  - Click on the "Environment variables" button under the "Advanced" tab
  - Then, select the "Path" variable in System variables and click on the "Edit" button
  - Click on the "New" button and add the path where Java is installed, followed by \ bin. By default, Java is installed in C:\Program Files\Java\jdk-13.0.1 (If nothing else was specified when you installed it). In that case, You will have to add a new path with: C:\Program Files\Java\jdk-11.0.1\bin
  - Then, click "OK", and save the settings
  - At last, open Command Prompt (cmd.exe) and type java -version to see if Java is running on your machine


System
Control Panel › System and Security › System
Search Control Panel
Control Panel Home
Device Manager
Remote settings
System protection
Advanced system settings
View basic information about your computer
1
Windows edition
Windows 10 Pro
© 2018 Microsoft Corporation. All rights reserved.
Windows 10
System
System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your sign-in
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
2
Environment Variables...
OK
Cancel
Apply
System variables
Variable
Value
Path
C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCL...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE
AMD64
PROCESSOR_IDENTIFIER
Intel64 Family 6 Model 158 Stepping 9, GenuineIntel
PROCESSOR_LEVEL
6
PROCESSOR_REVISION
9e09
PSModulePath
%ProgramFiles%\WindowsPowerShell\Modules;C:\WINDOWS\syst...
New...
Edit...
Delete
3
OK
Cancel
Edit environment variable
C:\Program Files (x86)\Intel\iCLS Client\
C:\Program Files\Intel\iCLS Client\
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
C:\Program Files\Java\jdk-11.0.1\bin
4
New
Edit
Browse...
Delete

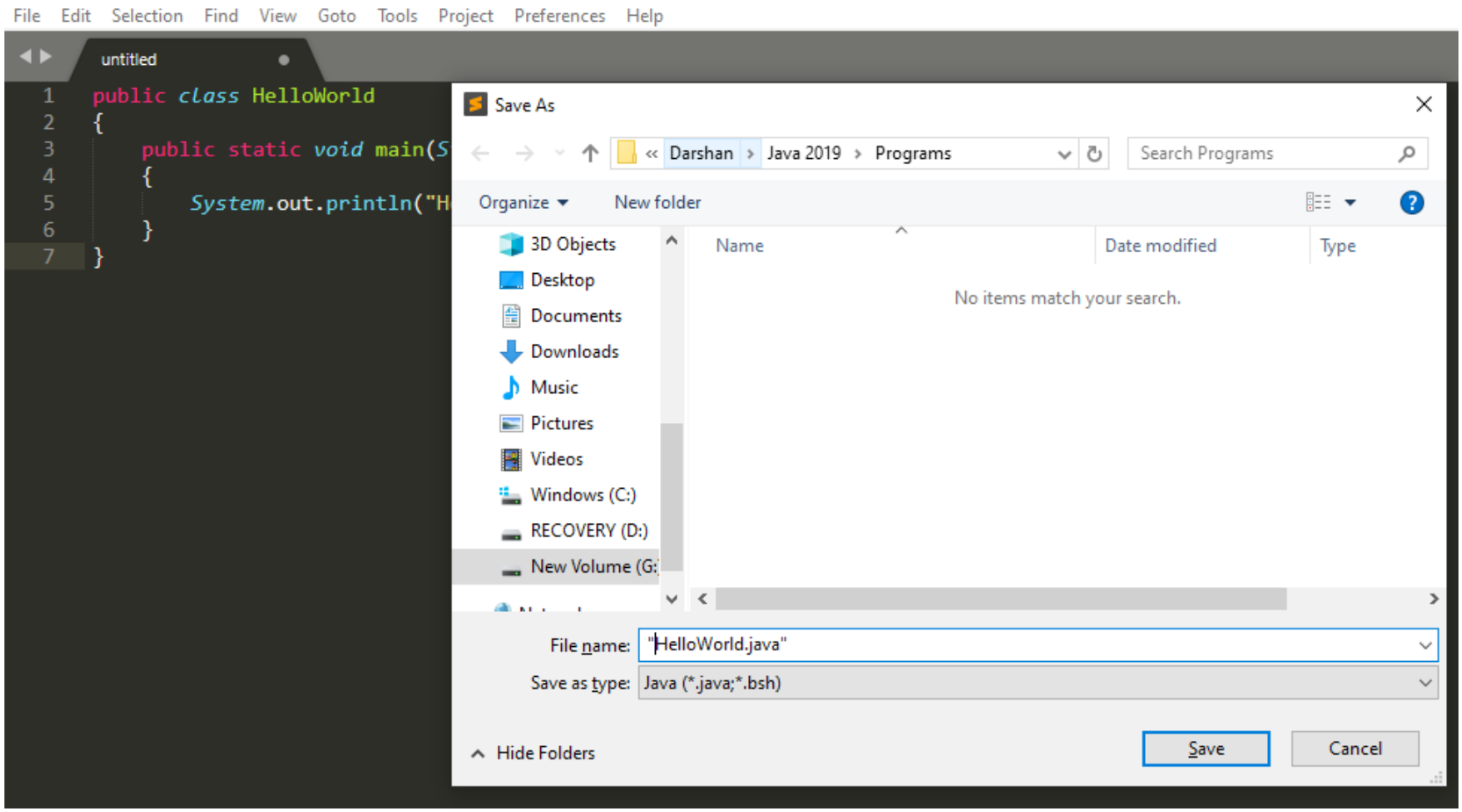
First Simple Program

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

- The program prints “Hello World” on the console.

How to execute Java Program?

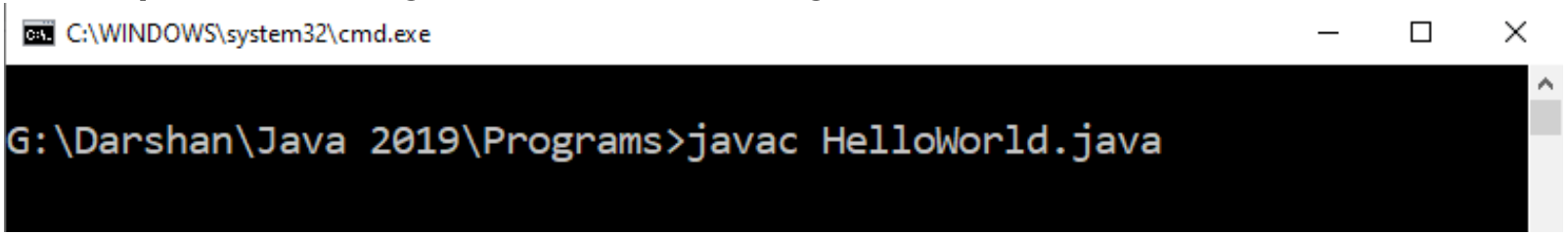
1. Save the program with the same name that of class which contains **`public static void main(String[]`**



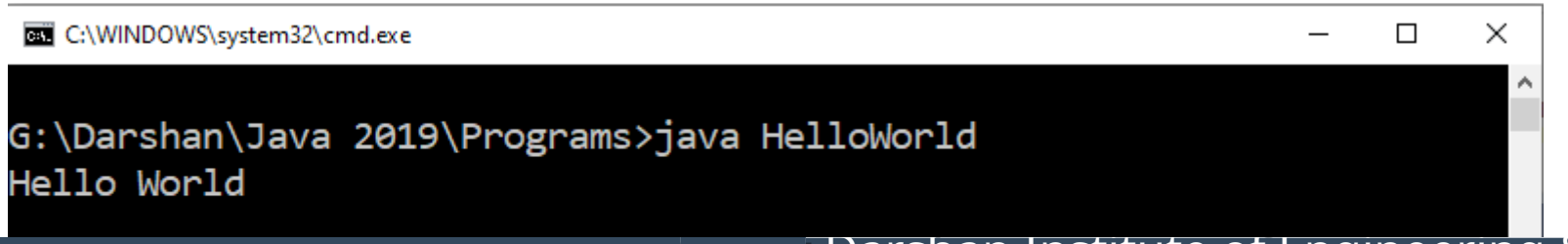
How to execute Java Program?

2. Open command prompt (cmd) / terminal & navigate to desired directory / folder.

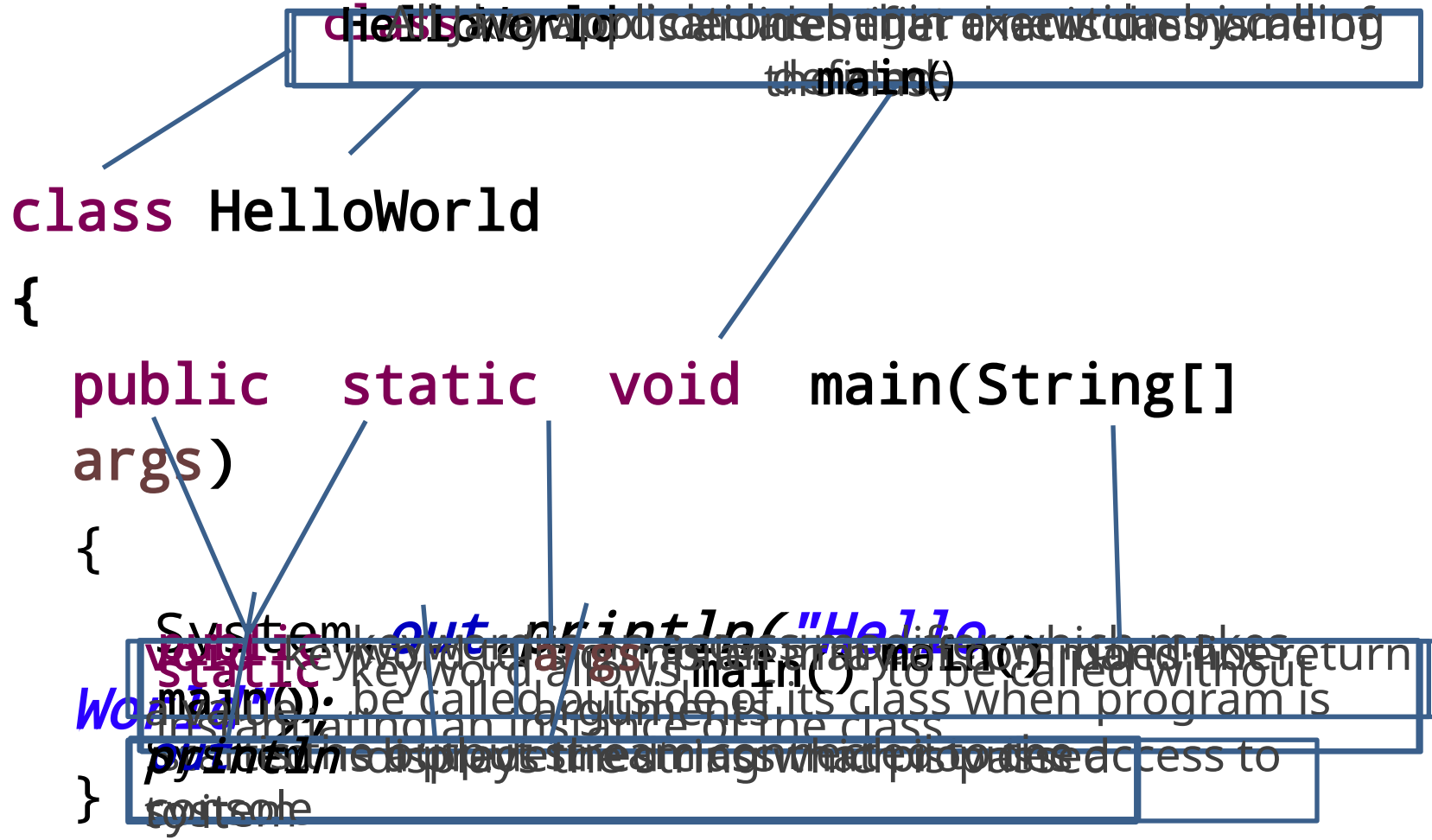
3. Compile the ".java" file with javac command.



4. Execute the ".class" file with java command without extension.



Closer look at First Sample Program



Lexical Issues

- Whitespace
  - It is a space, tab or newline
- Identifiers
  - They are used for class names, method names and variable names.
  - An identifier may be any descriptive sequence of
    - uppercase(A...Z) and lowercase(a..z) letters
    - Numbers(0..9)
    - Underscore(_) and dollar-sign($) characters

AvgTemp	count	a4	$test	this_is_ok

2count	high-temp	Not/ok

Lexical Issues

- Literals
  - Constant value in Java is created using literal representation.

100	98.6	‘X’	“This is a test”

- Comments
  - There are 3 types of comment in Java
    1. Single-line → //
    2. Multiline → starts with /* and ends with */
    3. Documentation → starts with /** and ends with */

Lexical Issues

- Separators

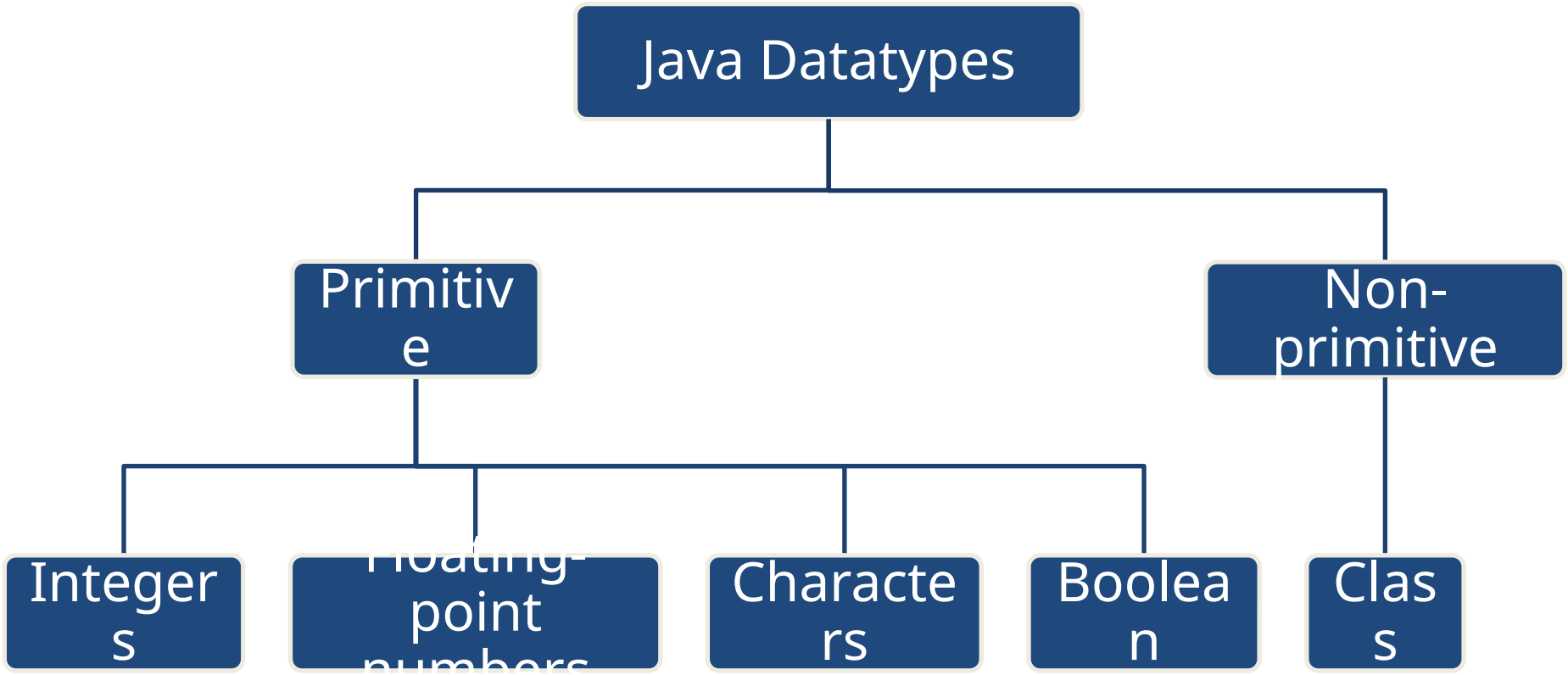
( )	Parenthesis	– Used to contain list of parameters in method definition and invocation. – Used for defining precedence in expressions, containing expressions in control statements and surrounding cast types.
{ }	Braces	– Used to contain values of automatically initialized arrays. – Used to define block of code for classes, methods and local scopes.
[ ]	Brackets	– Used to declare array types.
;	Semicolon	– Terminates statements.
,	Comma	– Separates consecutive identifiers in declaration.
.	Period	– Used to separate package names from subpackages and classes. – Used to separate variable or method from a reference variable.

Lexical Issues

- Java Keywords
  - 50 keywords
  - Cannnot be used as names for variables, class or method

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

Data Types


Java Datatypes
Primitive
Non-primitive
Integers
Floating-point numbers
Characters
Boolean
Class

Primitive Data Types

Data Type	Size	Range	Example
byte	1 Byte	-128 to 127	byte a = 10;
short	2 Bytes	-32,768 to 32,767	short a = 200;
int	4 Bytes	-2,147,483,648 to 2,147,483,647	int a = 50000;
long	8 Bytes	-9,223,372,036,854,775,808	long a = 20;
float	4 Bytes	1.4e-045 to 3.4e+038	float a = 10.2f;
double	8 Bytes	4.9e-324 to 1.8e+308	double a = 10.2;
char	2 Bytes	0 to 65536 (Stores ASCII of character)	char a = ‘a’;
boolean	Not defined	true or false	boolean a = true;

Escape Sequences

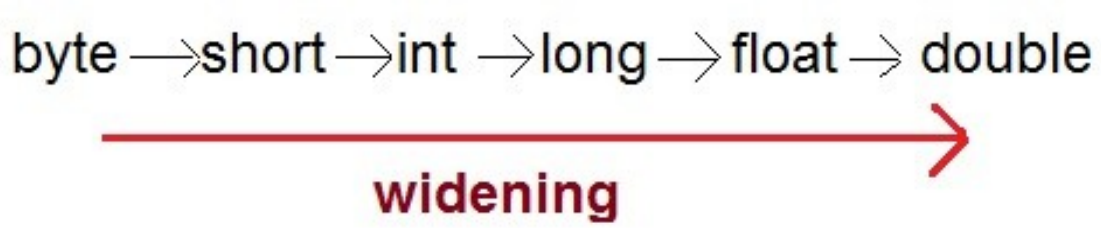
- Escape sequences in general are used to signal an alternative interpretation of a series of characters.
- For example, if you want to put quotes within quotes you must use the escape sequence, \", on the interior quotes.

```
System.out.println("Good Morning \"Jack\"");
```

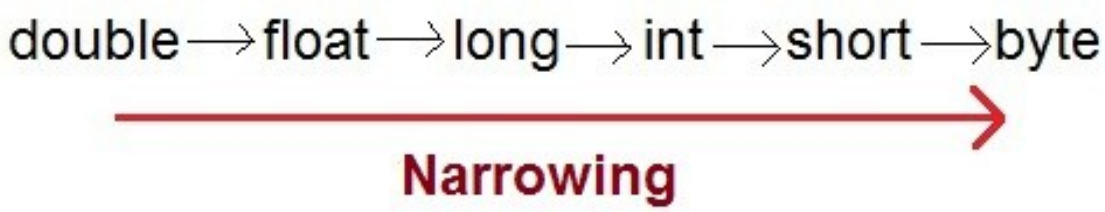
Escape Sequence	Description
\’	Single quote
\”	Double quote
\\	Backslash
\r	Carriage return
\n	New Line
\t	Tab

Type Casting

- Assigning a value of one type to a variable of another type is known as Type Casting.
- In Java, type casting is classified into two types,
  - Widening/Automatic Type Casting (Implicit)



  - Narrowing Type Casting(Explicitly done)



Automatic Type Casting

- When one type of data is assigned to other type of variable , an *automatic type conversion* will take place if the following two conditions are satisfied:
  - The two types are compatible
  - The destination type is larger than the source type
- Such type of casting is called "*widening conversion*".
- Example:

  int can always hold values of byte and short

```
public static void main(String[] args) {
    byte b = 5;
    // √ this is correct
    int a = b;
}
```

Casting Incompatible Types

- To create a conversion between two incompatible types, you must use a *cast*
- A *cast* is an explicit type conversion.
- Such type is called "*narrowing conversion*".
- Syntax:

  (target-type) value
- Example:

```
public static void main(String[] args) {
    int a = 5;
    // × this is not correct
    byte b = a;
    // √ this is correct
    byte b = (byte)a ;
}
```

Operators

1. Arithmetic Operators
2. Relational Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Conditional / Ternary Operator
7. Instance of Operator

Arithmetic Operator

Note : A = 10 & B = 20

Operator	Description	Example
+	Addition	A + B = 30
-	Subtraction	A - B = -10
*	Multiplication	A * B = 200
/	Division	B / A = 2
%	Modulus	B % A = 0
++	Increment	B++ = 21
--	Decrement	B-- = 19

Relational Operators

Note : A = 10 & B = 20

Operator	Description	Example
==	Equals	(A == B) is not true.
!=	Not Equals	(A != B) is true.
>	Greater than	(A > B) is not true.
<	Less than	(A < B) is true.
>=	Greater than equals	(A >= B) is not true.
<=	Less than equals	(A <= B) is true.
==	Equals	(A == B) is not true.

Bitwise Operators

Note : A = 60 & B = 13

Operator	Description	Example
&	Binary AND Operator	A & B = 12 which is 0000 1100
|	Binary OR Operator	A | B = 61 which is 0011 1101
^	Binary XOR Operator	A ^ B = 49 which is 0011 0001
~	Binary Ones Complement Operator	~A = -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator	A << 2 = 240 which is 1111 0000
>>	Binary Right Shift Operator.	A >> 2 = 15 which is 1111
>>>	Shift right zero fill operator.	A >>>2 = 15 which is 0000 1111

Logical Operators

Note : A = true & B = false

Operator	Description	Example
&&	Logical AND operator	(A && B) is false.
||	Called Logical OR Operator	(A || B) is true.
!	Called Logical NOT Operator	!(A && B) is true.

Assignment Operators

Operator	Description	Example
=	Simple assignment operator	C = A + B will assign value of A + B into C
+=	Add AND assignment operator	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2

Conditional Operator (Ternary)

- Conditional Operator (? :)
  - Syntax:

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

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

instanceof Operator

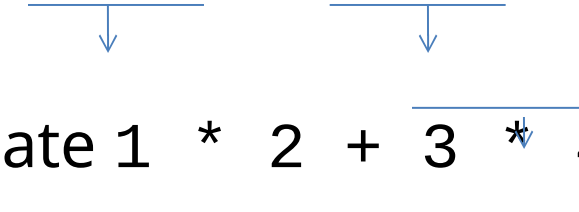
- `instanceof` Operator
  - Syntax:

    (Object reference variable) instanceof (class/interface type)
  - Example:

    boolean result = name instanceof String;

Operator Precedence & Associativity

- How does java evaluate `1 + 10 * 9 ?`
  - `(1 + 10 ) * 9 = 99` **OR** `1 + (10 * 9) = 91`
- To get the correct answer for the given problem Java came up with Operator precedence. (multiplication have higher precedence than addition so correct answer will be **91** in this case)
- For Operator, associativity means that when the same operator appears in a row, then to which direction the expression will be evaluated. (It would be from **Left to Right**)
- How does java evaluate `1 * 2 + 3 * 4 / 5 ???`

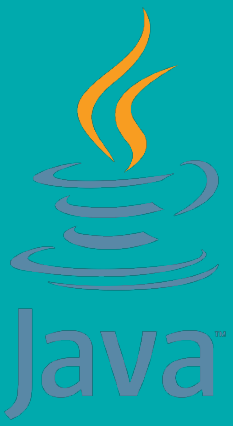


Precedence of Java Operators

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

3140705
Object Oriented Programming - I

Unit – 7
JavaFX and Event-driven programming and animations



Prof. Hardik A. Doshi

9978911553

hardik.doshi@darshan.ac.in

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

What is JavaFX?

- *JavaFX* is a Java library used to build Rich Internet Applications (RIA).
- The applications developed using JavaFX can run on various devices such as Desktop Computers, Mobile Phones, TVs, Tablets, etc.
- To develop GUI Applications using Java programming language, the programmers rely on libraries such as Advanced Windowing Toolkit (AWT) and Swing. After the advent of JavaFX, these Java programmers can now develop GUI applications effectively with rich content.

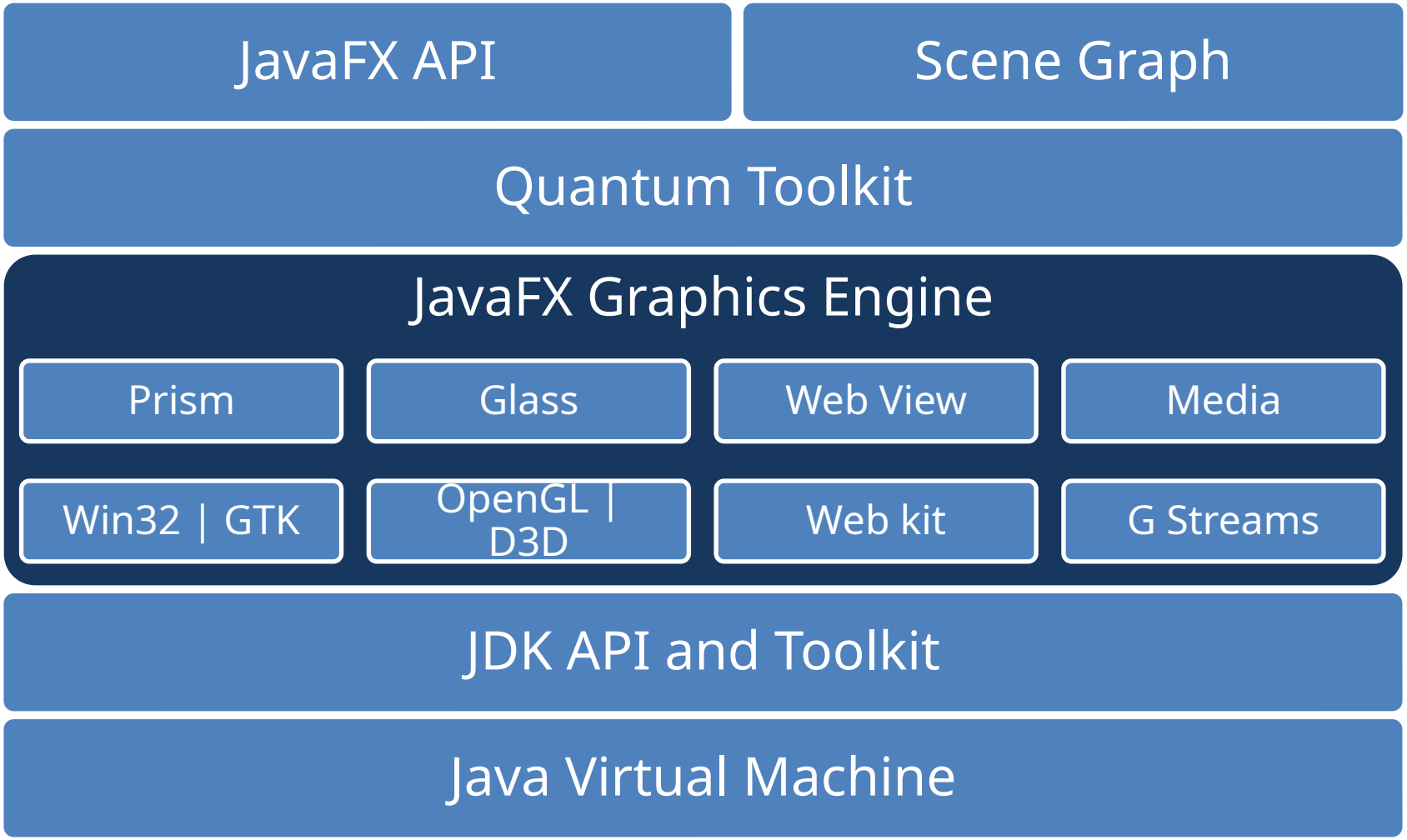
Need for JavaFX

- To develop Client Side Applications with rich features, the programmers used to depend on various libraries to add features such as Media, UI controls, Web, 2D and 3D, etc.
- JavaFX provides a rich set of graphics and media API's and it leverages the modern Graphical Processing Unit through hardware accelerated graphics.
- One can use JavaFX with JVM based technologies such as Java, Groovy and JRuby. If developers opt for JavaFX, there is no need to learn additional technologies.

Features of JavaFX

- Written in Java
- FXML
- Scene Builder
- Swing Interoperability
- Built-in UI controls
- CSS like Styling
- Canvas and Printing API
- Rich set of API’s
- Integrated Graphics library
- Graphics pipeline

Architecture of JavaFX API



Architecture of JavaFX API

- Scene Graph
  - A Scene Graph is the starting point of the construction of the GUI Application. It holds the (GUI) application primitives that are termed as nodes.
  - A node is a visual/graphical object and it may include
    - Geometrical (Graphical) objects
    - UI controls
    - Containers
    - Media elements
- Prism
  - Prism is a high performance hardware–accelerated graphical pipeline that is used to render the graphics in JavaFX. It can render both 2-D and 3-D graphics.

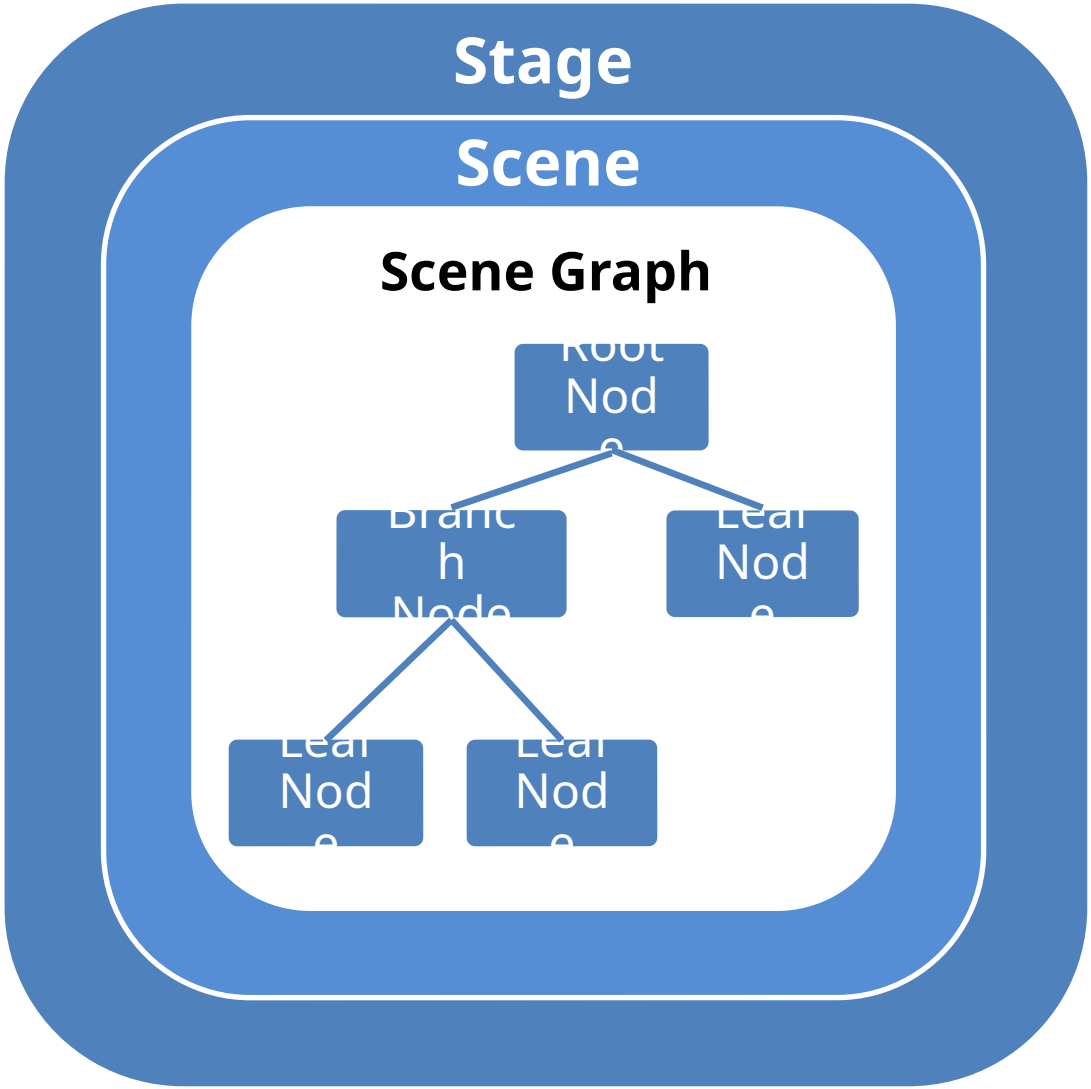
Architecture of JavaFX API

- GWT (Glass Windowing Toolkit)
  - GWT provides services to manage Windows, Timers, Surfaces and Event Queues.
  - GWT connects the JavaFX Platform to the Native Operating System.
- Quantum Toolkit
  - It is an abstraction over the low-level components of Prism, Glass, Media Engine, and Web Engine. It ties Prism and GWT together and makes them available to JavaFX.
- WebView
  - WebView is the component of JavaFX which is used to process HTML content. It uses a technology called Web Kit, which is an internal open-source web browser engine. This component supports different web technologies like HTML5, CSS, JavaScript, DOM and SVG.

Architecture of JavaFX API

- Media Engine
  - The JavaFX media engine is based on an open-source engine known as a Streamer. This media engine supports the playback of video and audio content.

JavaFX Application Structure



Stage

- A stage (a window) contains all the objects of a JavaFX application.
- It is represented by `Stage` class of the package `javafx.stage`.
- The primary stage is created by the platform itself. The created stage object is passed as an argument to the `start()` method of the `Application` class.
- A stage has two parameters determining its position namely Width and Height.
- There are five types of stages available
  - Decorated
  - Undecorated
  - Transparent
  - Unified
  - Utility
- You have to call the `show()` method to display the contents of a stage.

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. 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.

Scene Graph and Nodes

- 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 may include
  - Geometrical (Graphical) objects (2D and 3D) such as – Circle, Rectangle, Polygon, etc.
  - UI Controls – Button, Checkbox, Choice Box, Text Area, etc.
  - Containers (Layout Panes) – Border Pane, Grid Pane, Flow Pane, etc.
  - Media elements – Audio, Video and Image Objects.

Scene Graph and Nodes

- The `Node` class of the package `javafx.scene` represents a node in JavaFX, this class is the super class of all the nodes.

  Root Node – The first Scene Graph is known as the Root node. It is mandatory to pass the root node to the scene graph.

  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.
  - 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.

Steps to create JavaFX application

- Prepare a scene graph with the required nodes.
- Prepare a Scene with the required dimensions and add the scene graph (root node of the scene graph) to it.
- Prepare a stage and add the scene to the stage and display the contents of the stage.

Prepare a Scene graph

- Since the root node is the first node, you need to create a root node and it can be chosen from the *Group, Region or WebView*.
- Group

  A Group node is represented by the class named `Group` which belongs to the package `javafx.scene`, you can create a Group node by instantiating this class as shown below.

```
Group root = new Group();
Group root = new Group(NodeObject);
```

Prepare a Scene graph

- Region

  It is the Base class of all the JavaFX Node-based UI Controls, such as –

  - `Chart` – This class is the base class of all the charts and it belongs to the package `javafx.scene.chart` which embeds charts in application.
  - `Pane` – A Pane is the base class of all the layout panes such as AnchorPane, BorderPane, DialogPane, etc. This class belong to a package that is called as – `javafx.scene.layout` which inserts predefined layouts in your application.
  - `Control` – It is the base class of the User Interface controls such as Accordion, ButtonBar, ChoiceBox, ComboBoxBase, HTMLEditor, etc. This class belongs to the package javafx.scene.control.

- WebView

  This node manages the web engine and displays its contents.

Preparing the Scene

- A JavaFX scene is represented by the `Scene` class of the package `javafx.scene`.

```
Scene scene = new Scene(root,width,heigth);
```

- While instantiating, it is mandatory to pass the root object to the constructor of the `Scene` class whereas width and height of the scene are optional parameters to the constructor.

Preparing the Stage

- Stage is the container of any JavaFX application and it provides a window for the application. It is represented by the `Stage` class of the package `javafx.stage`.
- An object of this class is passed as a parameter of the `start()` method of the Application class.
- Using this object, various operations on the stage can be performed like
  - *Set the title* for the stage using the method `setTitle()`.

```
primaryStage.setTitle("Sample application");
```

  - *Attach the scene* object to the stage using the `setScene()` method.

```
primaryStage.setScene(scene);
```

  - *Display the contents* of the scene using the `show()` method as shown below.

```
primaryStage.show();
```

Lifecycle of JavaFX Application

- The JavaFX Application class has three life cycle methods.

  `start()` – The entry point method where the JavaFX graphics code is to be written.

  `stop()` – An empty method which can be overridden, here the logic to stop the application is written.

  `init()` – An empty method which can be overridden, stage or scene cannot be created in this method.

- It also provides a static method named `launch()` to launch JavaFX application. This method is called from static content only mainly main method.

Lifecycle of JavaFX Application

- Whenever a JavaFX application is launched, the following actions will be carried out (in the same order).
  - An instance of the application class is created.
  - `init()` method is called.
  - `start()` method is called.
  - The launcher waits for the application to finish and calls the `stop()` method.

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
public class JavafxSample extends Application {
  @Override
  public void start(Stage primaryStage) throws Exception {
     Group root = new Group();
     Scene scene = new Scene(root ,600, 300);
     scene.setFill(Color.BROWN);
     primaryStage.setTitle("Sample Application");
     primaryStage.setScene(scene);
     primaryStage.show();
   }
   public static void main(String args[]){
      launch(args);
   }
}
```

Create Scene Graph using Group, Region or WebView

Create Scene by adding Group (root) to it along with its width and height

Set the scene to the stage object (primaryStage) which is passed as an argument to start() using setScene() method

2D Shape

- 2D shape is a geometrical figure that can be drawn on the XY plane like Line, Rectangle, Circle, etc.
- Using the JavaFX library, you can draw –
  - Predefined shapes – Line, Rectangle, Circle, Ellipse, Polygon, Polyline, Cubic Curve, Quad Curve, Arc.
  - Path elements – MoveTo Path Element, Line, Horizontal Line, Vertical Line, Cubic Curve, Quadratic Curve, Arc.
  - 2D shape by parsing SVG path.
- Each of the above mentioned 2D shape is represented by a class which belongs to the package `javafx.scene.shape`. The class named Shape is the base class of all the 2-Dimensional shapes in JavaFX.

Classes for Shape (`javafx.scene.shape`)

Shape	Class	Example

Line	`Line`	`Line line = new Line();`
		`line.setStartX(100.0);`
		`line.setStartY(150.0);`
		`line.setEndX(500.0);`
		`line.setEndY(150.0);`

Rectangle & Rounded Rectangle	`Rectangle`	`Rectangle rectangle = new Rectangle();`
		`rectangle.setX(150.0f);`
		`rectangle.setY(75.0f);`
		`rectangle.setWidth(300.0f);`
		`rectangle.setHeight(150.0f);`
		`rectangle.setArcWidth(30.0);`
		`rectangle.setArcHeight(20.0);`

Circle	`Circle`	`Circle circle = new Circle();`
		`circle.setCenterX(300.0f);`
		`circle.setCenterY(135.0f);`
		`circle.setRadius(100.0f);`

Classes for Shape (`javafx.scene.shape`)

Shape	Class	Example

Ellipse	`Ellipse`	`Ellipse ellipse = new Ellipse();`
		`ellipse.setCenterX(300.0f);`
		`ellipse.setCenterY(150.0f);`
		`ellipse.setRadiusX(150.0f);`
		`ellipse.setRadiusY(75.0f);`

Polygon	`Polygon`	`Polygon polygon = new Polygon();`
		
		`polygon.getPoints().addAll(new Double[]{`
		`300.0, 50.0,`
		`450.0, 150.0,`
		`300.0, 250.0,`
		`150.0, 150.0,`
		`});`

Classes for Shape (`javafx.scene.shape`)

Shape	Class	Example

Polyline	Polyline	`Polyline polyline = new Polyline();`
		`polyline.getPoints().addAll(new Double[]{`
		`200.0, 50.0,`
		`400.0, 50.0,`
		`450.0, 150.0,`
		`400.0, 250.0,`
		`200.0, 250.0,`

Cubic Curve	CubicCurve	`CubicCurve cubicCurve = new CubicCurve();`
		`cubicCurve.setStartX(100.0f);`
		`cubicCurve.setStartY(150.0f);`
		`cubicCurve.setControlX1(400.0f);`
		`cubicCurve.setControlY1(40.0f);`
		`cubicCurve.setControlX2(175.0f);`
		`cubicCurve.setControlY2(250.0f);`
		`cubicCurve.setEndX(500.0f);`
		`cubicCurve.setEndY(150.0f);`

Classes for Shape (`javafx.scene.shape`)

Shape	Class	Description

Quad Curve	QuadCurve	`QuadCurve quadCurve = new QuadCurve();`
		`quadCurve.setStartX(100.0);`
		`quadCurve.setStartY(220.0f);`
		`quadCurve.setEndX(500.0f);`
		`quadCurve.setEndY(220.0f);`
		`quadCurve.setControlX(250.0f);`

Arc	Arc	`Arc arc = new Arc();`
		`arc.setCenterX(100.0);`
		`arc.setCenterY(100.0);`
		`arc.setRadiusX(100.0);`
		`arc.setRadiusY(100.0);`
		`arc.setStartAngle(0.0);`
		`arc.setLength(100.0);`

JavaFX - Colors

- `javafx.scene.paint` package provides various classes to apply colors to an application. This package contains an abstract class named `Paint` and it is the base class of all the classes that are used to apply colors.
- Using these classes, you can apply colors in the following patterns
  - Uniform – color is applied uniformly throughout node.
  - Image Pattern – fills the region of the node with an image pattern.
  - Gradient – the color applied to the node varies from one point to the other. It has two kinds of gradients namely Linear Gradient and Radial Gradient.

Creating instance of Color

- Instance of Color class can be created by providing Red, Green, Blue and Opacity value ranging from 0 to 1 in double.

```
Color color = new Color(double red, double green,
              double blue, double opacity);
```

- Example

```
Color color = new Color(0.0,0.3,0.2,1.0);
```

- Instance of Color class can be created using following methods also

```
Color c = Color.rgb(0,0,255);      //passing RGB values
Color c = Color.hsb(270,1.0,1.0);  //passing HSB values
Color c = Color.web("0x0000FF",1.0);//passing hex code
                                              for web
```

Applying Color to the Nodes

- `setFill(Color)` method is used to apply color to nodes such as Shape, Text, etc.
- `setStroke(Color)` method is used to apply strokes to the nodes.

```
//Setting color to the text
Color color = new Color.BEIGE
text.setFill(color);

//Setting color to the stroke
Color color = new Color.DARKSLATEBLUE
circle.setStroke(color);
```

```
import javafx.application.Application;
import javafx.scene.*;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.shape.Circle;
public class ColorExample extends Application {
    @Override
    public void start(Stage stage) {
        Circle circle = new Circle();
        circle.setCenterX(300.0f);
        circle.setCenterY(180.0f);
        circle.setRadius(90.0f);
        circle.setFill(Color.DARKRED);
        circle.setStrokeWidth(3);
        circle.setStroke(Color.DARKSLATEBLUE);
        Group root = new Group(circle);
        Scene scene = new Scene(root, 600, 300);
        stage.setTitle("Color Example");
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String args[]){
        launch(args);
    }
}
```

JavaFX – Image

- You can load and modify images using the classes provided by JavaFX in the package `javafx.scene.image`.
- JavaFX supports the image formats like Bmp, Gif, Jpeg, Png.

Loading an Image

- Class Image of `javafx.scene.image` package is used to load an image
- Any of the following argument is required to the constructor of the class
  - An InputStream object of the image to be loaded or

```
FileInputStream inputstream = new FileInputStream ("C:\\image.jpg");
Image image = new Image(inputstream);
```

  - A string variable holding the URL for the image.

```
Image image = new Image("http://sample.com/res/flower.png");
```

- After loading image in Image object, view is set to load the image using `ImageView` class

```
ImageView imageView = new ImageView(image);
```

```java
import java.io.FileInputStream;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;

public class ImageExample extends Application {
   @Override
   public void start(Stage stage) throws Exception {
        Image image = new Image(new FileInputStream("C://image.jpeg"));
        ImageView imageView = new ImageView(image);
        imageView.setX(50);
        imageView.setY(25);
        imageView.setFitHeight(455);
        imageView.setFitWidth(500);
        imageView.setPreserveRatio(true);
        Group root = new Group(imageView);
        Scene scene = new Scene(root, 600, 500);
        stage.setTitle("Loading an image");
        stage.setScene(scene);
        stage.show();
   }
   public static void main(String args[]) {
    launch(args);
   }
}
```

Layout Panes

- After constructing all the required nodes in a scene, we will generally arrange them in order.
- This arrangement of the components within the container is called the Layout of the container.
- JavaFX provides several predefined layouts such as HBox, VBox, Border Pane, Stack Pane, Text Flow, Anchor Pane, Title Pane, Grid Pane, Flow Panel, etc.
- Each of the above mentioned layout is represented by a class and all these classes belongs to the package `javafx.layout`. The class named `Pane` is the base class of all the layouts in JavaFX.

Layout Panes (`javafx.scene.layout`)

Sr.	Shape & Description
1	HBox • The HBox layout arranges all the nodes in our application in a single horizontal row. • The class named HBox of the
2	VBox • The VBox layout arranges all the nodes in our application in a single vertical column. • The class named VBox of the
3	BorderPane • The Border Pane layout arranges the nodes in our application in top, left, right, bottom and center positions. • The class named BorderPane of the package javafx.scene.layout represents the border pane layout.

Layout Panes (`javafx.scene.layout`)

Sr.	Shape & Description
4	StackPane • The stack pane layout arranges the nodes in our application on top of another just like in a stack. The node added first is placed at the bottom of the stack and the next node is placed on top of it.
5	TextFlow • The Text Flow layout arranges multiple text nodes in a single flow. • The class named TextFlow of the package javafx.scene.layout
6	AnchorPane • The Anchor pane layout anchors the nodes in our application at a particular distance from the pane. • The class named AnchorPane of the package javafx.scene.layout represents the Anchor Pane layout.

Layout Panes (`javafx.scene.layout`)

Sr.	Shape & Description
7	TilePane • The Tile Pane layout adds all the nodes of application in the form of uniformly sized tiles. • The class named TilePane of the package javafx.scene.layout
8	GridPane • The Grid Pane layout arranges the nodes in our application as a grid of rows and columns. This layout comes handy while creating forms. • The class named GridPane of the package javafx.scene.layout
9	FlowPane • The flow pane layout wraps all the nodes in a flow. A horizontal flow pane wraps the elements of the pane at its height, while a vertical flow pane wraps the elements at its width. • The class named FlowPane of the package javafx.scene.layout

Creating a Layout

- To create a layout, you need to –
  - Create node.
  - Instantiate the respective class of the required layout.
  - Set the properties of the layout.
  - Add all the created nodes to the layout.

```java
import javafx.application.Application;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class HBoxExample extends Application {
    @Override
    public void start(Stage stage) {
        TextField textField = new TextField();
        Button playButton = new Button("Play");
        Button stopButton = new Button("stop");
        HBox hbox = new HBox();
        hbox.setSpacing(10);
        hbox.setMargin(textField, new Insets(20, 20, 20, 20));
        hbox.setMargin(playButton, new Insets(20, 20, 20, 20));
        hbox.setMargin(stopButton, new Insets(20, 20, 20, 20));
        ObservableList<Node> list = hbox.getChildren();
        list.addAll(textField, playButton, stopButton);
        Scene scene = new Scene(hbox);
        stage.setTitle("Hbox Example");
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String args[]){
        launch(args);
    }
```

JavaFX - Events

- In GUI applications, web applications and graphical applications, whenever a user interacts with the application (nodes), an event is said to have been occurred.
- For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.

JavaFX - 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.
- JavaFX provides a wide variety of events. Some of them are as follows:

1. Mouse Event – occurs when a mouse is clicked.
   Class - `MouseEvent`
   Actions - mouse clicked, mouse pressed, mouse released, mouse moved, mouse entered target, mouse exited target, etc.
2. Key Event – indicates the key stroke occurred on a node.
   Class – `KeyEvent`
   Actions - key pressed, key released and key typed.
3. Drag Event – occurs when the mouse is dragged.
   Class - `DragEvent`.
   Actions - drag entered, drag dropped, drag entered target, drag exited target, drag over, etc.
4. Window Event – occurs when window showing/hiding takes place.
   Class - `WindowEvent`
   Actions – window hiding, window shown, window hidden, window showing, etc.

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.
  - *Type* – Type of the occurred event; in case of mouse event – mouse pressed, mouse released are the type of events.

Phases of Event Handling

- Target selection
- Route Construction
- Event Capturing Phase
- Event Bubbling Phase

Target selection

- When an action occurs, the system determines which node is the target based on internal rules:
- Key events - the target is the node that has focus.
- Mouse events - the target is the node at the location of the cursor.
- Gesture events - the target is the node at the center point of all touches at the beginning of the gesture.
- Swipe events - the target is the node at the center of the entire path of all of the fingers.
- Touch events - the target for each touch point is the node at the location first pressed.

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.

Event Capturing Phase

- After the construction of the event dispatch chain, the root node of the application dispatches the event.
- This event travels to all nodes in the dispatch chain (from top to bottom).
- If any of these nodes has a filter registered for the generated event, it will be executed.
- If none of the nodes in the dispatch chain has a filter for the event generated, then it is passed to the target node and finally the target node processes the event.

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 filters and handlers are those which contains application logic to process an event.
- A node can register to more than one handler/filter. 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 capturing phase, a filter is executed and during the event bubbling phase, a handler is executed.
- All the handlers and filters implement the interface `EventHandler` of the package `javafx.event`.

Handling Mouse Event

```
import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.*;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;

public class JavafxSample extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage primaryStage) {
   Group root = new Group();
   Scene scene = new Scene(root, 300, 250);
   scene.setOnMouseClicked(mouseHandler);
       scene.setOnMouseDragged(mouseHandler);
       scene.setOnMouseEntered(mouseHandler);
       scene.setOnMouseExited(mouseHandler);
       scene.setOnMouseMoved(mouseHandler);
       scene.setOnMousePressed(mouseHandler);
       scene.setOnMouseReleased(mouseHandler);

       primaryStage.setScene(scene);
       primaryStage.show();
    }
```

```
EventHandler<MouseEvent> mouseHandler = new
EventHandler<MouseEvent>()
    {
          @Override
      public void handle(MouseEvent mouseEvent) {
        System.out.println(mouseEvent.getEventType() + "\n" +
    "X : Y - "
    + mouseEvent.getX() + " : " + mouseEvent.getY() + "\n" +
    "SceneX : SceneY - "
    + mouseEvent.getSceneX()+" : "+mouseEvent.getSceneY() +
    "\n" + "ScreenX : ScreenY - "
    + mouseEvent.getScreenX()+" : "+mouseEvent.getScreenY());
    }
    };
}
```

Handling Key Event

```
import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.input.KeyEvent;
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class JavafxSample extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage primaryStage) {
        Text text = new Text();
        text.setX(10.0);
        text.setY(100.0);
        text.setFont(Font.font("verdana", FontWeight.BOLD, FontPosture.REGULAR,
15));
        Group root = new Group(text);
        Scene scene = new Scene(root, 300, 250);
        scene.setOnKeyPressed(new EventHandler<KeyEvent>() {
            public void handle(KeyEvent ke) {
                text.setText("Key Pressed: " + ke.getCode().toString());
            }
        });
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```

Inner Class / Nested Class

- Inner classes are class within Class.
- Inner class instance has special relationship with Outer class. This special relationship gives inner class access to member of outer class as if they are the part of outer class.
- Additionally, it can access all the members of outer class including private data members and methods.
- Syntax

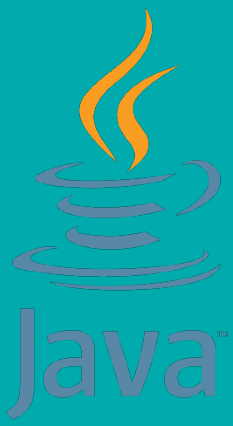
```
//outer class
class OuterClass
{
//inner class
class InnerClass
{
}
}
```

Inner Class Example

```
class Outer {
    int outer_x = 100;
    void test() {
        Inner inner = new Inner();
        inner.display();
    }
    class Inner {
        void display() {
        System.out.println("Display : outer_x-" +
        outer_x);
        }
    }
}
public class InnerClassDemo {
    public static void main(String[] args) {
    Outer outer = new Outer();
    outer.test();
    }
}
```

3140705
Object Oriented Programming - I

Unit – 8 JavaFX UI Controls & Multimedia



Prof. Hardik A. Doshi

9978911553

hardik.doshi@darshan.ac.in

Dedicated Faculty, Committed Education

Darshan

Institute of Engineering & Technology

Label

- Label is used to display a short text or an image, it is a non-editable text control.
- It is useful for displaying text that is required to fit within a specific space.
- Label can only display text or image and it cannot get focus.
- Constructor for the Label class are:

Constructor	Description
`Label()`	Creates an empty Label
`Label(String text)`	Creates Label with supplied text
`Label(String text, Node graphics)`	Creates a Label with the supplied text and graphic.

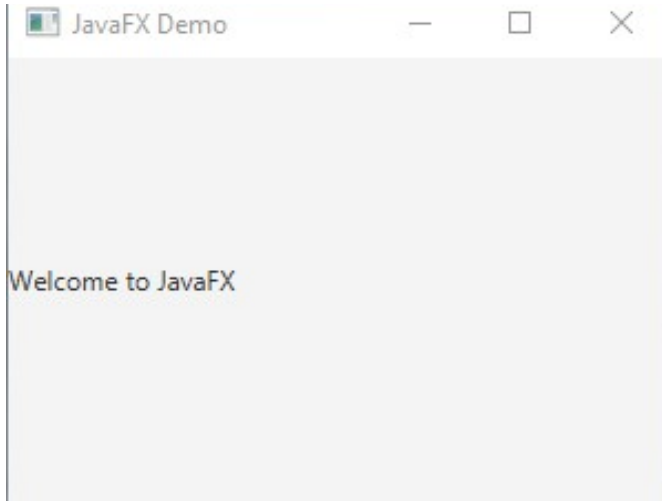
Label Example

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.stage.Stage;

public class LabelExample extends Application  {
    @Override
    public void start(Stage primaryStage) throws Exception {

        Label label = new Label("Welcome to JavaFX");

        Scene scene = new Scene(label, 200, 200);
    primaryStage.setTitle("JavaFX Demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```



Button

- Button control enables an application to have some action executed when the application user clicks the button.
- The button control can contain text and/or a graphic.
- When a button is pressed and released a ActionEvent is sent. Some action can be performed based on this event by implementing an EventHandler to process the ActionEvent.
- Buttons can also respond to mouse events by implementing an EventHandler to process the MouseEvent.

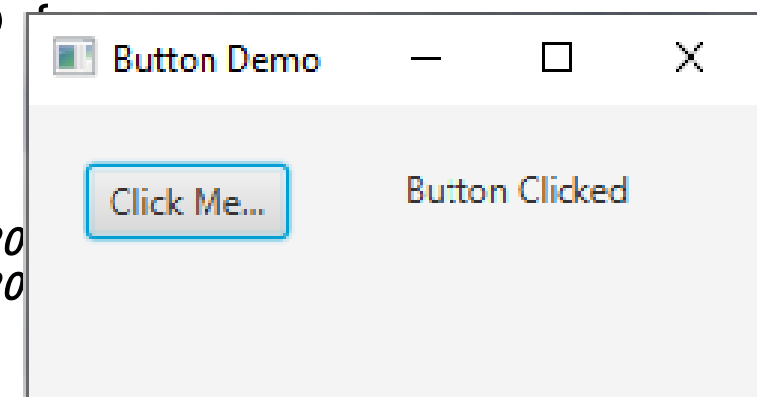
Button

- Constructor for the Button class are:

Constructor	Description
`Button()`	Creates a button with an empty string for its label.
`Button(String text)`	Creates a button with the specified text as its label.
`Button(String text, Node graphic)`	Creates a button with the specified text and icon for its label.

Button Example

```
import javafx.application.Application;
import javafx.event.*;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class ButtonDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        Label lbl = new Label();
        btn.setText("Click Me...");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                lbl.setText("Button Clicked");
            }
        });
        HBox root = new HBox();
        root.setMargin(btn, new Insets(20,20,20,20
        root.setMargin(lbl, new Insets(20,20,20,20
        root.getChildren().add(btn);
        root.getChildren().add(lbl);
        primaryStage.setTitle("Button Demo");
        primaryStage.setScene(new Scene(root, 250, 100));
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

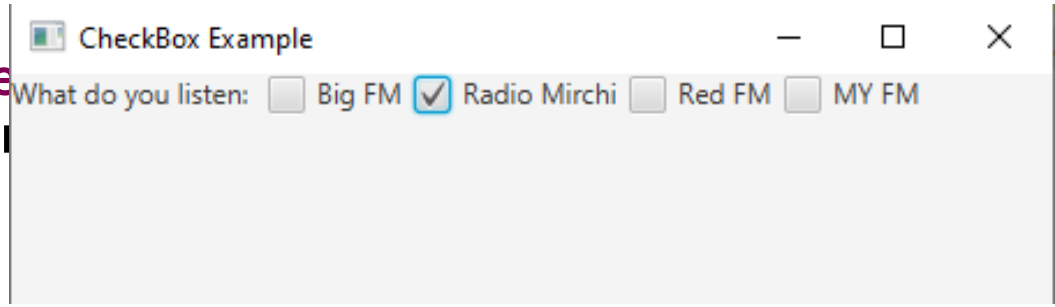


Checkbox

- 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.
- It is different from the radiobutton in the sense that, we can select more than one checkboxes in a scenerio.
- Constructor for the Checkbox class are:

Constructor	Description
`CheckBox()`	Creates a check box with an empty string for its label.
`CheckBox(String text)`	Creates a check box with the specified text as its label.

Checkbox Example



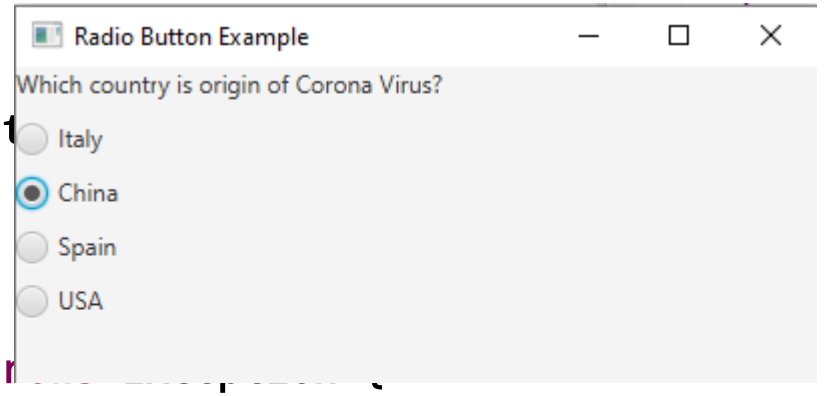

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class CheckboxDemo exte
    public static void main(Str
        launch(args);
    }
    @Override
    public void start(Stage primaryStage) throws Exception {
        Label l = new Label("What do you listen: ");
        CheckBox c1 = new CheckBox("Big FM");
        CheckBox c2 = new CheckBox("Radio Mirchi");
        CheckBox c3 = new CheckBox("Red FM");
        CheckBox c4 = new CheckBox("MY FM");
        HBox root = new HBox();
        root.getChildren().addAll(l,c1,c2,c3,c4);
        root.setSpacing(5);
        Scene scene=new Scene(root,450,100);
        primaryStage.setScene(scene);
        primaryStage.setTitle("CheckBox Example");
        primaryStage.show();
    }
}
```

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.
- 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.

Constructor	Description
`RadioButton()`	Creates a radio button with an empty string for its label.
`RadioButton(String text)`	Creates a radio button with the specified text as its label.

RadioButton Example




```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
public class RadioButtonDemo extends Applicat
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage primaryStage) thr
        Label lbl = new Label("Which country is origin of Corona Virus?");
        ToggleGroup group = new ToggleGroup();
        RadioButton button1 = new RadioButton("Italy");
        RadioButton button2 = new RadioButton("China");
        RadioButton button3 = new RadioButton("Spain");
        RadioButton button4 = new RadioButton("USA");
        button1.setToggleGroup(group);
        button2.setToggleGroup(group);
        button3.setToggleGroup(group);
        button4.setToggleGroup(group);
        VBox root=new VBox();
        root.setSpacing(10);
        root.getChildren().addAll(lbl,button1,button2,button3,button4);
        Scene scene=new Scene(root,400,300);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Radio Button Example");
        primaryStage.show();
    }
}
```

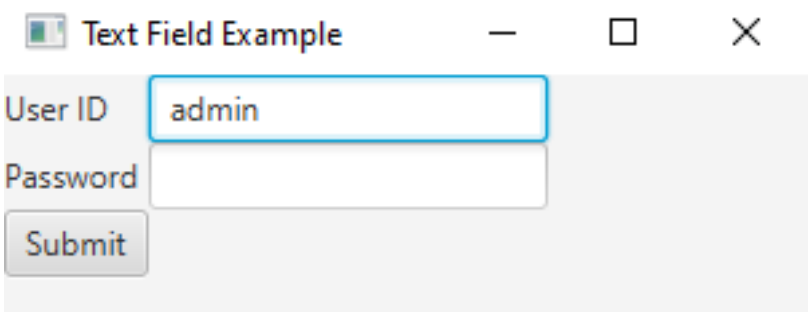
TextField

- Text input component that allows a user to enter a single line of unformatted text.
- Constructor for the TextField class are:

Constructor	Description
`TextField()`	Creates a TextField with empty text content.
`TextField(String text)`	Creates a TextField with initial text content.

TextField Example

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
public class TextFieldDemo extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage primaryStage) throws Exception {
        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);
        Scene scene=new Scene(root,300,200);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Text Field Example");
        primaryStage.show();
    }
}
```



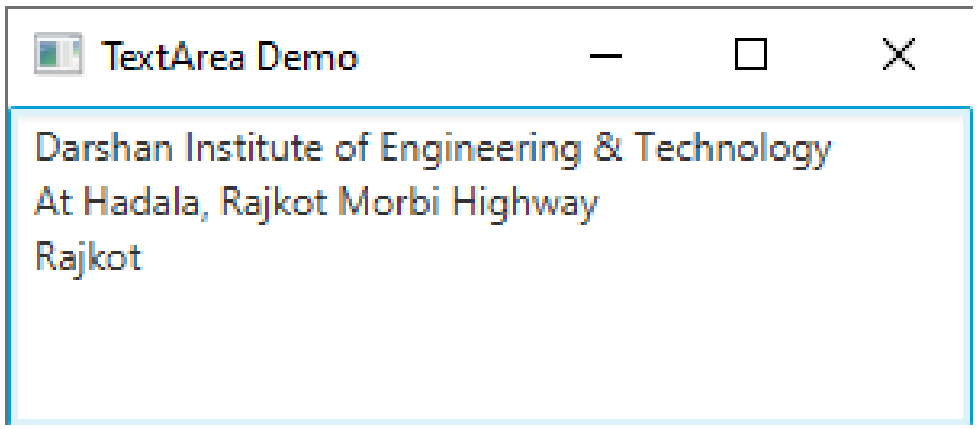
TextArea

- Text input component that allows a user to enter multiple lines of plain text.
- Constructor for the TextArea class are:

Constructor	Description
`TextArea()`	Creates a TextArea with empty text content.
`TextArea(String text)`	Creates a TextArea with initial text content.

TextArea Example

```
public class TextAreaDemo extends Application  {
public static void main(String[] args) {
        Application.launch(args);
    }
    @Override
    public void start(Stage primaryStage) throws Exception {
   TextArea textArea = new TextArea();
   VBox vbox = new VBox(textArea);
   Scene scene = new Scene(vbox, 300, 100);
   primaryStage.setTitle("TextArea Demo");
   primaryStage.setScene(scene);
   primaryStage.show();
    }
}
```



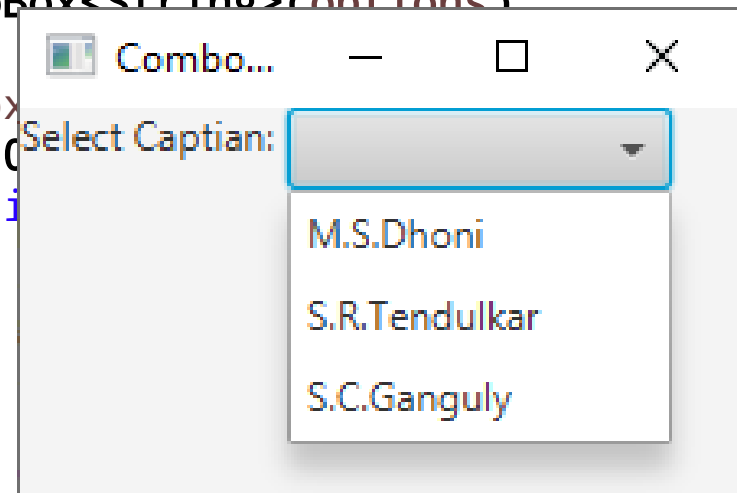
ComboBox

- A combo box is a typical element of a user interface that enables users to choose one of several options.
- A combo box is helpful when the number of items to show exceeds some limit, because it can add scrolling to the drop down list.
- Constructor for the ComboBox class are:

Constructor	Description
`ComboBox()`	Creates a default ComboBox instance with an empty items list and default selection model.
`ComboBox(ObservableList<T> items)`	ComboBox(ObservableList<T> items) Creates a default ComboBox instance with the provided items list and a default selection model.

ComboBox Example

```
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
public class ComboBoxDemo extends Application  {
    @Override
    public void start(Stage primaryStage) throws Exception {
        Label lbl = new Label("Select Captian: ");
       ObservableList<String> options =
FXCollections.observableArrayList(
                "M.S.Dhoni",
                "S.R.Tendulkar",
                "S.C.Ganguly"
            );
        ComboBox<String> comboBox = new ComboBox<String>(options);
        HBox hbox = new HBox();
        hbox.getChildren().addAll(lbl,comboBox
        Scene scene = new Scene(hbox, 200, 120
        primaryStage.setTitle("ComboBox Experi
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        Application.launch(args);
    }
```



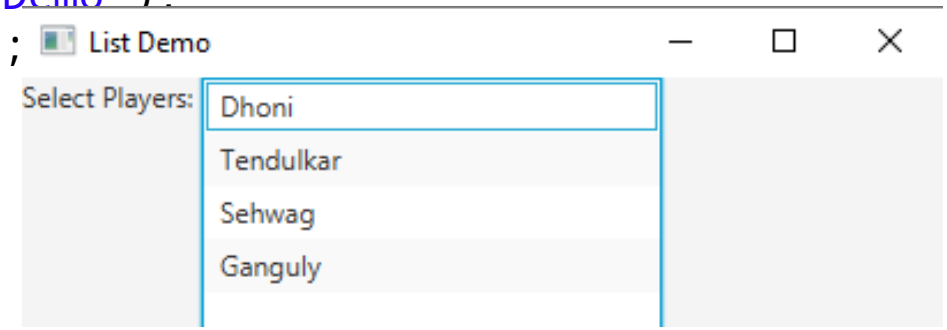
ListView

- A ListView displays a horizontal or vertical list of items from which the user may select, or with which the user may interact.
-

Constructor	Description
`ListView()`	Creates a default ListView which will display contents stacked vertically.
`ListView(ObservableList<T> items)`	Creates a default ListView which will stack the contents retrieved from the provided ObservableList vertically.

ListView Example

```
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class ListViewDemo extends Application  {
    @Override
    public void start(Stage primaryStage) throws Exception {
        Label lbl = new Label("Select Players: ");
        ObservableList<String> options =
                FXCollections.observableArrayList(
                    "Dhoni","Tendulkar","Sehwag","Ganguly"
                );
        ListView<String> list = new ListView<String>(options);
        list.setPrefSize(200, 50);
        HBox hbox = new HBox();
        hbox.getChildren().addAll(lbl,list);
        Scene scene = new Scene(hbox, 400, 300);
        primaryStage.setTitle("List Demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[]
        Application.launch(args);
    }
}
```



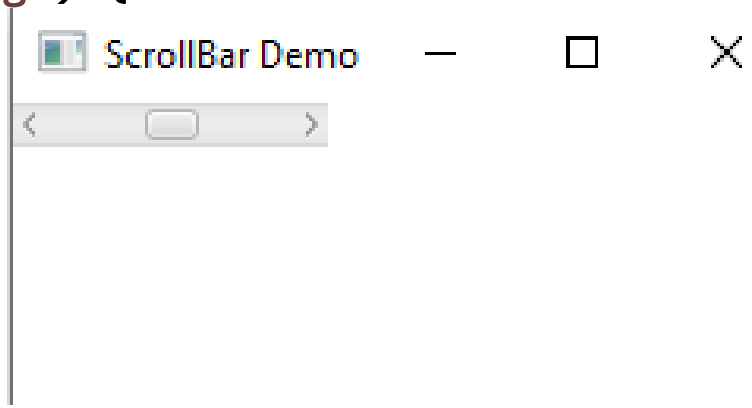
ScrollBar

- JavaFX Scroll Bar is used to provide a scroll bar to the user so that the user can scroll down the application pages.
- Either a horizontal or vertical bar with increment and decrement buttons and a "thumb" with which the user can interact. Typically not used alone but used for building up more complicated controls such as the
-

Constructor	Description
`ScrollBar()`	Creates a new horizontal ScrollBar.

ScrollBar Example

```
import javafx.application.Application;
import javafx.scene.*;
import javafx.scene.control.ScrollBar;
import javafx.stage.Stage;
public class ScrollBarDemo extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage stage) {
        ScrollBar sc = new ScrollBar();
        sc.setMin(0);
        sc.setMax(100);
        sc.setValue(50);
        Group root = new Group();
        Scene scene = new Scene(root, 250, 100);
        stage.setScene(scene);
        root.getChildren().add(sc);
        stage.setTitle("ScrollBar Demo");
        stage.show();
    }
}
```



Slider

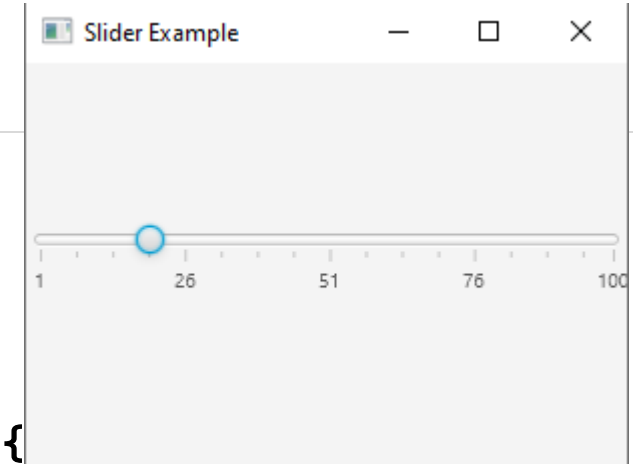
- The Slider Control is used to display a continuous or discrete range of valid numeric choices and allows the user to interact with the control.
- It is typically represented visually as having a "track" and a "knob" or "thumb" which is dragged within the track. The Slider can optionally show tick marks and labels indicating the different slider position values.
- The three fundamental variables of the slider are min, max, and value. The value should always be a number within the range defined by min and max. min should always be less than or equal to max. min defaults to 0, whereas max defaults to 100.

Slider

- Constructor for the Slider class are:

Constructor	Description
`Slider()`	Creates a default Slider instance.
`Slider(double min, double max, double value)`	Constructs a Slider control with the specified slider min, max and current value values.

Slider Example




```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Slider;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class SliderDemo extends Application{
    @Override
    public void start(Stage primaryStage) throws Exception {
        Slider slider = new Slider(1,100,20);
        slider.setShowTickLabels(true);
        slider.setShowTickMarks(true);
        StackPane root = new StackPane();
        root.getChildren().add(slider);
        Scene scene = new Scene(root,300,200);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Slider Example");
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Video

- In the case of playing video, we need to use the MediaView node to display the video onto the scene.
- For this purpose, we need to instantiate the MediaView class by passing the Mediaplayer object into its constructor. Due to the fact that, MediaView is a JavaFX node, we will be able to apply effects to it.

Video Example

```
import java.io.File;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
import javafx.stage.Stage;
public class JavaFXVideoDemo extends Application
{
    public void start(Stage primaryStage) throws Exception {
        String path = "G:\\demo.mp4";
        Media media = new Media(new File(path).toURI().toString());
        MediaPlayer mediaPlayer = new MediaPlayer(media);
        MediaView mediaView = new MediaView(mediaPlayer);
        mediaPlayer.setAutoPlay(true);
        Group root = new Group();
        root.getChildren().add(mediaView);
        Scene scene = new Scene(root,1366,768);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Playing video");
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Audio

- We can load the audio files with extensions like .mp3,.wav and .aifff by using JavaFX Media API. We can also play the audio in HTTP live streaming format.
- Instantiate javafx.scene.media.Media class by passing the audio file path in its constructor to play the audio files.

Audio Example

```
import java.io.File;
import javafx.application.Application;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.stage.Stage;
public class JavaFXAudioDemo extends Application
{
    public void start (Stage primaryStage) throws Exception {
        String path = "G://demo.mp3";
        Media media = new Media(new
File(path).toURI().toString());
        MediaPlayer mediaPlayer = new MediaPlayer(media);
        mediaPlayer.setAutoPlay(true);
        primaryStage.setTitle("Playing Audio");
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

2150704
Object Oriented Programming with JAVA

Unit – 9
IO Programming

Prof. Hardik A. Doshi

9978911553

hardik.doshi@darshan.ac.in

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Stream

- The **java.io** package contains all the classes required for input-output operations.
- All streams represent an input source and an output destination.
- The stream in the java.io package supports all the datatype including primitive.
- A stream can be defined as a sequence of data.
- There are two kinds of Streams
  - **Byte Stream**
  - **Character Stream**

Byte Streams

- Byte streams provide a convenient means for handling input and output of bytes.
- Byte streams are used, for example, when reading or writing binary data.

FileOutputStream

- Java `FileOutputStream` is an output stream for writing data to a file.
- `FileOutputStream` will create the file before opening it for output.
- On opening a read only file, it will throw an exception.

FileOutputSteam – Methods

Sr.	Method
1	**void write(byte[] b)** This method writes b.length bytes from the specified byte array to this file
2	**void write(byte[] b, int off, int len)** This method writes len bytes from the specified byte array starting at offset off to this file output stream.
3	**void write(int b)** This method writes the specified byte to this file output stream.
4	**void close()** This method closes this file output stream and releases any system resources associated with this stream.

FileOutputStream - Example

```
class FileOutDemo {
    public static void main(String args[]) {
        try {
            FileOutputStream fout = new
            FileOutputStream("abc.txt");
            String s = "Sourav Ganguly is my favorite player";
            byte b[] = s.getBytes();
            fout.write(b);
            fout.close();
            System.out.println("Success...");
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

FileInputStream

- `FileInputStream` class is used to read bytes from a file.
- It should be used to read byte-oriented data for example to read image, audio, video etc.

FileInputSteam – Methods

Sr.	Method
1	**public int read()**
2	**public int read(byte[] b)** b - the buffer into which the data is read. Returns: the total number of bytes read into the buffer, or -1
3	**public int read(byte[] b, int off, int len)** b - the buffer into which the data is read. off - the start offset in the destination array b len - the maximum number of bytes read. Returns: the total number of bytes read into the buffer, or -1
4	**public long skip(long n)** n - the number of bytes to be skipped. Returns: the actual number of bytes skipped.
5	**public int available()** an estimate of the number of remaining bytes that can be read
6	**public void close()** Closes this file input stream and releases any system resources associated.

Example (FileInputStream)

```
class SimpleRead {
    public static void main(String args[]) {
        try {
            FileInputStream fin = new
            FileInputStream("abc.txt");
            int i = 0;
            while ((i = fin.read()) != -1) {
                System.out.println((char) i);
            }
            fin.close();
        } catch (Exception e) {
             System.out.println(e);
        }
    }
}
```

Example of Byte Steams

```
import java.io.*;
public class CopyFile {
    public static void main(String args[]) throws
    IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("input.txt");
            out = new FileOutputStream("output.txt");
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

Character Streams

- Character Streams provide a convenient means for handling input and output of characters.
- Internationalization is possible as it uses Unicode.

Reader

- The Java `Reader` class is the base class of all Reader's in the I-O API.
- Subclasses include a `FileReader`, `FileWriter`, `BufferedReader`, `BufferedWriter`, `InputStreamReader`, `StringReader` and several others.
- Here is a simple Java IO Reader example:

```
Reader reader = new FileReader("c:\\data\\
myfile.txt");
int data = reader.read();
while (data != -1) {
    char dataChar = (char) data;
    data = reader.read();
}
```

- Combining Readers with InputStream

```
Reader reader = new InputStreamReader("c:\\data\\
myfile.txt");
```

Writer

- The Java `Writer` class is the base class of all Writers in the I-O API.
- Subclasses include `BufferedWriter`, `PrintWriter`, `StringWriter` and several others.
- Here is a simple Java IO Writer example:

```
Writer writer = new FileWriter("c:\\data\\file-
output.txt");
writer.write("Hello World Writer");
writer.close();
```

- Combining Readers With OutputStreams

```
Writer writer =
    new OutputStreamWriter("c:\\data\\file-output.txt");
```

FileWriter

- FileWriter is useful to create a file writing characters into it.
- This class inherits from the OutputStreamWriter class.
- Constructors of FileWriter are as follows

Sr.	Constructor
1	**FileWriter(File file)** Constructs a FileWriter object given a File object.
2	**FileWriter (File file, boolean append)** Constructs a FileWriter object given a File object, it will append if second parameter is true.
3	**FileWriter(String file)** Constructs a FileWriter object from the path given in parameter.
4	**FileWriter (String file, boolean append)** Constructs a FileWriter object from the path given, it will append if second parameter is true.

FileWriter – Methods

Sr.	Methods
1	**public void write (int c) throws IOException** Writes a single character.
2	**public void write (char [] str) throws IOException** Writes an array of characters.
3	**public void write(String str)throws IOException** Writes a string
4	**public void write(String str,int off,int len)throws IOException** Writes a portion of a string. Here off is offset from which to start writing characters and len is number of character to write.

FileReader

- FileReader is useful to read data in the form of characters from a text file.
- This class inherit from the InputStreamReader Class.
- Constructors of FileReader class are as follows:

Sr.	Constructor
1	**FileReader(File file)** Creates a FileReader , given the File to read from.
2	**FileReader(String fileName)** Creates a new FileReader , given the name of the file to read from.
3	**FileReader(FileDescripter fd)** Creates a new FileReader , given the FileDescripter to read from.

FileReader (Cont.)

Sr.	Methods
1	**public int read () throws IOException** Reads a single character. This method will block until a character is available, an I/O error occurs, or the end of the stream is reached.
2	**public int read(char[] cbuff) throws IOException** Reads characters into an array. This method will block until some input is available, an I/O error occurs, or the end of the stream is reached.
3	**public abstract int read(char[] buff, int off, int len) throws IOException** Reads characters into a portion of an array. This method will block until some input is available, an I/O error occurs, or the end of the stream is reached. Parameters: cbuf – Destination buffer off – Offset at which to start storing characters len – Maximum number of characters to read
4	**public long skip(long n) throws IOException**

Example FileReader/FileWriter

```
import java.io.*;

public class CopyFile {
    public static void main(String args[]) throws
    IOException {
        FileReader in = null;
        FileWriter out = null;
        try {
            in = new FileReader("input.txt");
            out = new FileWriter("output.txt");
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

BufferedReader

- The `java.io.BufferedReader` class reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.
- Following are the important points about `BufferedReader`:
  - The buffer size may be specified, or the default size may be used.

Sr.	Constructor
1	**BufferedReader(Reader in)** This creates a buffering character-input stream that uses a default-sized input buffer.
2	**BufferedReader(Reader in, int sz)** This creates a buffering character-input stream that uses an input buffer of the specified size.

BufferedReader – Example

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

class BufferedReaderDemo {
    public static void main(String[] args) throws IOException
    {
        FileReader fr = new FileReader("input.txt");
        BufferedReader br = new BufferedReader(fr);
        char c[] = new char[20];
        br.skip(8);
        if (br.ready()) {
            System.out.println(br.readLine());
            br.read(c);
            for (int i = 0; i < 20; i++) {
                System.out.print(c[i]);
            }
        }
    }
}
```

```
G:\Darshan\Java 2019\PPTs
ava.io.BufferedReader;
import java.io.FileR
```

BufferedReader (Cont.)

Sr.	Methods
1	**void close()** This method closes the stream and releases any system resources associated with it.
2	**int read()** This method reads a single character.
3	**int read(char[] cbuf, int off, int len)** This method reads characters into a portion of an array.
4	**String readLine()** This method reads a line of text.
5	**void reset()** This method resets the stream.
6	**long skip(long n)** This method skips characters.

File class

- Java **`File`** class represents the files and directory pathnames in an **abstract manner**. This class is used for creation of **files** and **directories**, **file searching**, **file deletion** etc.
- The File object represents the actual file/directory on the

Sr.	Constructor
1	**File(String pathname)** Creates a new File instance by converting the given pathname string into an abstract pathname.
2	**File(String parent, String child)** Creates a new File instance from a parent pathname string and a child pathname string.
3	**File(URI uri)** Creates a new File instance by converting the given file: URI into an abstract pathname.

```java
import java.io.File;
class FileDemo {
    public static void main(String args[]) {
        File f1 = new File("FileDemo.java");
        System.out.println("File Name: " + f1.getName());
        System.out.println("Path: " + f1.getPath());
        System.out.println("Abs Path: " + f1.getAbsolutePath());
        System.out.println("Parent: " + f1.getParent());
        System.out.println(f1.exists() ? "exists" : "does not
        exist");
        System.out.println(f1.canWrite() ? "is writeable" : "is
        not writeable");
        System.out.println (f1.canRead () ? "is readable" : "is
        not readable");
        System.out.println ("is " + (f1.isDirectory() ? "" : "not"
        + " a directory"));
        System.out.println(f1.isFile() ? "is normal file" : "might
        be
        Sys                                                    " : "is
        not
        Sys
        f1.
        Sys                                                    "
        Byt
    }
```

```
G:\Darshan\Java 2019\PPTs\HAD\Programs>java FileDemo
File Name: FileDemo.java
Path: FileDemo.java
Abs Path: G:\Darshan\Java 2019\PPTs\HAD\Programs\FileDemo.java
Parent: null
exists
is writeable
is readable
is not a directory
is normal file
is not absolute
File last modified: 1587202242054
File size: 951 Bytes
```

Methods of File Class

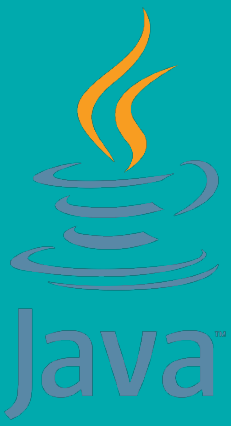
Sr.	Method
1	**public boolean isAbsolute()** Tests whether this abstract pathname is absolute. Returns true if this abstract pathname is absolute, false otherwise
2	**public String getAbsolutePath()** Returns the absolute pathname string of this abstract pathname.
3	**public boolean canRead()** Tests whether the application can read the file denoted by this abstract pathname. Returns true if and only if the file specified by this abstract pathname exists and can be read by the application; false otherwise.
4	**public boolean canWrite()** Tests whether the application can modify to the file denoted by this abstract pathname. Returns true if and only if the file system actually contains a file denoted by this abstract pathname and the application is allowed to write to the file; false otherwise.
5	**public boolean exists()**

Methods of File Class (Cont.)

Sr.	Method
6	**public boolean isDirectory()** Tests whether the file denoted by this abstract pathname is a directory. Returns true if and only if the file denoted by this abstract pathname exists and is a directory; false otherwise.
7	**public boolean isFile()** Tests whether the file denoted by this abstract pathname is a normal file. A file is normal if it is not a directory and, in addition, satisfies other system-dependent criteria
8	**public long lastModified()** Returns the time that the file denoted by this abstract pathname was last modified. Returns a long value representing the time the file was last modified, measured in milliseconds since the epoch (00:00:00 GMT, January 1, 1970).
9	**public long length()** Returns the length of the file denoted by this abstract pathname.
10	**public boolean delete()** Deletes the file or directory.

3140705
Object Oriented Programming - I

Unit – 10 Collection Framework



Prof. Hardik A. Doshi

9978911553

hardik.doshi@darshan.ac.in

Dedicated Faculty, Committed Education

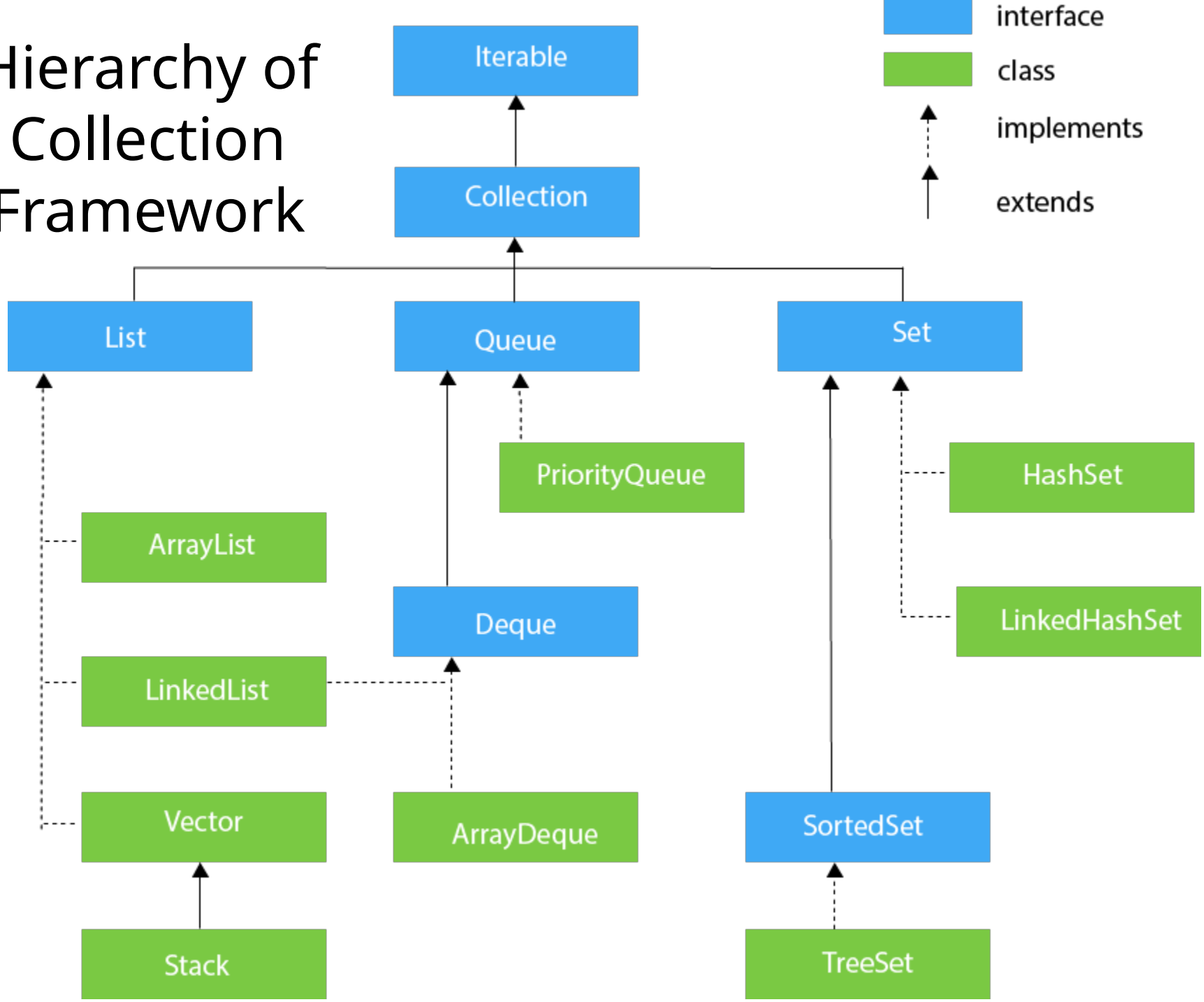
Darshan

Institute of Engineering & Technology

Collection

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

Hierarchy of Collection Framework



Collection Interface - Methods

Sr.	Method & Description
1	`boolean add(E e)` It is used to insert an element in this collection.
2	`boolean addAll(Collection<? extends E> c)` It is used to insert the specified collection elements in the invoking collection.
3	`void clear()` It removes the total number of elements from the collection.
4	`booelan contains(Object element)` It is used to search an element.
5	`boolean containsAll(Collection<?> c)` It is used to search the specified collection in the collection.
6	`boolean equals(Object obj)` Returns true if invoking collection and obj are equal. Otherwise returns false.
7	`int hashCode()` Returns the hashcode for the invoking collection.

Collection Interface - Methods

Sr.	Method & Description
8	`boolean isEmpty()` Returns true if the invoking collection is empty. Otherwise returns false.
9	`Iterator iterator()` It returns an iterator.
10	`boolean remove(Object obj)` Removes one instance of obj from the invoking collection. Returns true if the element was removed. Otherwise, returns false.
11	`boolean removeAll(Collection<?> c)` It is used to delete all the elements of the specified collection from the invoking collection.
12	`boolean retainAll(Collection<?> c)` It is used to delete all the elements of invoking collection except the specified collection.
13	`int size()` It returns the total number of elements in the collection.

Collection Interface - Methods

Sr.	Method & Description
14	`Object[] toArray()` Returns an array that contains all the elements stored in the invoking collection. The array elements are copies of the collection elements.

List Interface

- The **List** interface extends **Collection** and declares the behavior of a collection that stores a sequence of elements.
- Elements can be inserted or accessed by their position in the list, using a zero-based index.
- A list may contain duplicate elements.
- List is a generic interface with following declaration

```
interface List<E>
```

where E specifies the type of object.

List Interface (example)

```
import java.util.*;
public class CollectionsDemo {
    public static void main(String[] args) {
        List a1 = new ArrayList();
        a1.add("Sachin");
        a1.add("Sourav");
        a1.add("Shami");
        System.out.println("ArrayList Elements");
        System.out.print("\t" + a1);

        List l1 = new LinkedList();
        l1.add("Mumbai");
        l1.add("Kolkata");
        l1.add("Vadodara");
        System.out.println();
        System.out.println("Linke
        System.out.print("\t" + l
    }
}
```

Here **ArrayList** & **LinkedList** implements **List** Interface

```
G:\Darshan\Java 2019\PPTs\HAD\Programs>ja
ArrayList Elements
        [Sachin, Sourav, Shami]
LinkedList Elements
        [Mumbai, Kolkata, Vadodara]
G:\Darshan\Java 2019\PPTs\HAD\Programs>
```

List Interface - Methods

Sr.	Method & Description
1	`void add(int index, Object obj)` Inserts `obj` into the invoking list at the index passed in `index`. Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten.
2	`boolean addAll(int index, Collection c)` Inserts all elements of `c` into the invoking list at the index passed in `index`. Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten. Returns **true** if the invoking list changes and returns **false** otherwise.
3	`Object get(int index)` Returns the object stored at the specified index within the invoking collection.
4	`int indexOf(Object obj)` Returns the index of the first instance of `obj` in the invoking list. If `obj` is not an element of the list, .1 is returned.
5	`int lastIndexOf(Object obj)` Returns the index of the last instance of `obj` in the invoking list. If `obj` is not an element of the list, -1 is returned.

List Interface (methods) (cont.)

Sr.	Method & Description
6	`ListIterator listIterator( )` Returns an iterator to the start of the invoking list.
7	`ListIterator listIterator(int index)` Returns an iterator to the invoking list that begins at the specified `index`.
8	`Object remove(int index)` Removes the element at position `index` from the invoking list and returns the deleted element. The resulting list is compacted. That is, the indexes of subsequent elements are decremented by one
9	`Object set(int index, Object obj)` Assigns obj to the location specified by index within the invoking list.
10	`List subList(int start, int end)` Returns a list that includes elements from `start` to end-1 in the invoking list. Elements in the returned list are also referenced by the invoking object.

Iterator

- `Iterator` interface is used to cycle through elements in a collection, eg. displaying elements.
- `ListIterator` extends `Iterator` to allow bidirectional traversal of a list, and the modification of elements.
- Each of the collection classes provides an `iterator( )` method that returns an iterator to the start of the collection. By using this iterator object, you can access each element in the collection, one element at a time.
- To use an iterator to cycle through the contents of a collection, follow these steps:
  1. Obtain an iterator to the start of the collection by calling the collection's `iterator( )` method.
  2. Set up a loop that makes a call to `hasNext( ).` Have the loop iterate as long as `hasNext( )` returns true.
  3. Within the loop, obtain each element by calling `next( ).`

Iterator - Example

```
import java.util.*;
public class IteratorDemo {
    public static void main(String args[]) {
        ArrayList<String> al = new ArrayList<String>();
        al.add("C");
        al.add("A");
        al.add("E");
        al.add("B");
        al.add("D");
        al.add("F");
        System.out.print("Contents of list: ");
        Iterator<String> itr = al.iterator();
        while(itr.hasNext()) {
            Object element = itr.next();
            System.out.print(element + " ");
        }
    }
}
```

```
G:\Darshan\Java 2019\PPTs\HAD\Pr
Contents of list: C A E B D F
```

Iterator - Methods

Sr.	Method & Description
1	`boolean` **`hasNext()`** Returns true if there are more elements. Otherwise, returns false.
2	`E` **`next()`** Returns the next element. Throws NoSuchElementException if there is not a next element.
3	`void` **`remove()`** Removes the current element. Throws IllegalStateException if an attempt is made to call `remove()` that is not preceded by a call to `next()`

Comparator

- Comparator interface is used to set the sort order of the object to store in the sets and lists.
- The Comparator interface defines two methods: compare( ) and equals( ).
- `int compare(Object obj1, Object obj2)`

  `obj1` and `obj2` are the objects to be compared. This method returns zero if the objects are equal. It returns a positive value if `obj1` is greater than `obj2`. Otherwise, a negative value is returned.
- `boolean equals(Object obj)`

  `obj` is the object to be tested for equality. The method returns true if `obj` and the invoking object are both Comparator objects and use the same ordering. Otherwise, it returns false.

```java
import java.util.*;
class Student {
    String name;
    int age;
    Student(String name,
    int age){
        this.name = name;
        this.age = age;
    }
}
```

```java
class AgeComparator implements
Comparator<Object>{
    public int compare(Object o1,Object o2){
        Student s1=(Student)o1;
        Student s2=(Student)o2;
        if(s1.age==s2.age) return 0;
        else if(s1.age>s2.age) return 1;
        else return -1;
    }
}
```

```java
public class ComparatorDemo {
    public static void main(String args[]){
        ArrayList<Student> al=new ArrayList<Student>();
        al.add(new Student("Vijay",23));
        al.add(new Student("Ajay",27));
        al.add(new Student("Jai",21));
        System.out.println("Sorting by age");
        Collections.sort(al,new AgeComparator()
        Iterator<Student> itr2=al.iterator();
        while(itr2.hasNext()){
            Student st=(Student)itr2.next();
            System.out.println(st.name+" "+st.age);
        }
    }
}
```

```
G:\Darshan\Java 2019
Sorting by age
Jai 21
Vijay 23
Ajay 27
```

Vector Class

- **`Vector`** implements a dynamic array.
- It is similar to **`ArrayList`**, but with two differences:
  - Vector is synchronized.
  - Vector contains many legacy methods that are not part of the collection framework
- Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change sizes over the lifetime of a program.
- Vector is declared as follows:

```
Vector<E> = new Vector<E>;
```

```java
import java.util.*;
public class VectorDemo {
	public static void main(String args[]) {
		//Create an empty vector with initial capacity 4
		Vector<String> vec = new Vector<String>(4);
		//Adding elements to a vector
		vec.add("Tiger");
		vec.add("Lion");
		vec.add("Dog");
		vec.add("Elephant");
		//Check size and capacity
		System.out.println("Size is: "+vec.size());
		System.out.println("Default capacity is: "+vec.capacity());
		//Display Vector elements
		System.out.println("Vector element is: "+vec);
		vec.addElement("Rat");
		vec.addElement("Cat");
		vec.addElement("Deer");
		//Again check size and capacity after two insertions
		System.out.println("Size after addition: "+vec.size());
		System.out.println("Capacity after addition is: "+vec.capacity());
		//Display Vector elements again
		System.out.println("Elements are: "+vec);
		//Checking if Tiger is present or not in this vector
		if(vec.contains("Tiger"))
		{
			System.out.println
		}
		else
		{
			System.out.println
		}
		//Get the first elemer
		System.out.println("Th
		//Get the last element
		System.out.println("Th
	}
}
```

```
G:\Darshan\Java 2019\PPTs\HAD\Programs>java VectorDemo
Size is: 4
Default capacity is: 4
Vector element is: [Tiger, Lion, Dog, Elephant]
Size after addition: 7
Capacity after addition is: 8
Elements are: [Tiger, Lion, Dog, Elephant, Rat, Cat, Deer]
Tiger is present at the index 0
The first animal of the vector is = Tiger
The last animal of the vector is = Deer
```

Vector - Constructors

Sr.	Constructor & Description
1	**`Vector( )`** This constructor creates a default vector, which has an initial size of 10
2	**`Vector(int size)`** This constructor accepts an argument that equals to the required size, and creates a vector whose initial capacity is specified by size:
3	**`Vector(int size, int incr)`** This constructor creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward
4	**`Vector(Collection c)`** creates a vector that contains the elements of collection c

Vector - Methods

Sr.	Method & Description
7	`boolean containsAll(Collection c)` Returns true if this Vector contains all of the elements in the specified Collection.
8	`Enumeration elements()` Returns an enumeration of the components of this vector.
9	`Object firstElement()` Returns the first component (the item at index 0) of this vector.
10	`Object get(int index)` Returns the element at the specified position in this Vector.
11	`int indexOf(Object elem)` Searches for the first occurence of the given argument, testing for equality using the equals method.
12	`boolean isEmpty()` Tests if this vector has no components.

Vector - Methods (cont.)

Sr.	Method & Description
13	`Object lastElement()` Returns the last component of the vector.
14	`int lastIndexOf(Object elem)` Returns the index of the last occurrence of the specified object in this vector.
15	`Object remove(int index)` Removes the element at the specified position in this Vector.
16	`boolean removeAll(Collection c)` Removes from this Vector all of its elements that are contained in the specified Collection.
17	`Object set(int index, Object element)` Replaces the element at the specified position in this Vector with the specified element.
18	`int size()` Returns the number of components in this vector.

Stack

- `Stack` is a subclass of `Vector` that implements a standard last-in, first-out stack.
- `Stack` only defines the default constructor, which creates an empty stack.
- `Stack` includes all the methods defined by `Vector` and adds several of its own.
- `Stack` is declared as follows:

```
Stack<E> st = new Stack<E>();
```

where E specifies the type of object.

```java
import java.util.*;
public class StackDemo {
    static void showpush(Stack<Integer> st, int a) {
        st.push(new Integer(a));
        System.out.println("push(" + a + ")");
        System.out.println("stack: " + st);
    }
    static void showpop(Stack<Integer> st) {
        System.out.print("pop -> ");
        Integer a = (Integer) st.pop();
        System.out.println(a);
        System.out.println("stack: " + st);
    }
    public static void main(String args[]) {
        Stack<Integer> st = new Stack<Integer>
        System.out.println("stack: " + st);
        showpush(st, 42);
        showpush(st, 66);
        showpush(st, 99);
        showpop(st);
        showpop(st);
        showpop(st);
        try {
            showpop(st);
        } catch (EmptyStackException e) {
            System.out.println("empty stack");
        }
    }
}
```

```
G:\Darshan\Java 2019\PP
stack: []
push(42)
stack: [42]
push(66)
stack: [42, 66]
push(99)
stack: [42, 66, 99]
pop -> 99
stack: [42, 66]
pop -> 66
stack: [42]
pop -> 42
stack: []
pop -> empty stack
```

Stack - Methods

- Stack includes all the methods defined by Vector and adds several methods of its own.

Sr.	Method & Description
1	`boolean empty()` Returns true if the stack is empty, and returns false if the stack contains elements.
2	`E peek()` Returns the element on the top of the stack, but does not remove it.
3	`E pop()` Returns the element on the top of the stack, removing it in the process.
4	`E push(E element)` Pushes element onto the stack. Element is also returned.
5	`int search(Object element)` Searches for element in the stack. If found, its offset from the top of the stack is returned. Otherwise, -1 is returned.

Queue

- `Queue` interface extends `Collection` and declares the behaviour of a queue, which is often a first-in, first-out list.
- `LinkedList` and `PriorityQueue` are the two classes which implements Queue interface
- `Queue` is declared as follows:

```
Queue<E> q = new LinkedList<E>();
Queue<E> q = new PriorityQueue<E>();
```

where E specifies the type of object.

Queue Example

```
G:\Darshan\Java 2019\PPTs\HAD\Programs>java QueueDemo
Elements in Queue:[Tom, Jerry, Mike, Steve, Harry]
Removed element: Tom
Head: Jerry
poll(): Jerry
peek(): Mike
Elements in Queue:[Mike, Steve, Harry]
```

```
        q.add("Harry");
        System.out.println("Elements in Queue:"+q);
        System.out.println("Removed element:
        "+q.remove());
        System.out.println("Head: "+q.element());
        System.out.println("poll(): "+q.poll());
        System.out.println("peek(): "+q.peek());
        System.out.println("Elements in Queue:"+q);
    }
}
```

Queue - Methods

Sr.	Method & Description
1	`E element()` Returns the element at the head of the queue. The element is not removed. It throws NoSuchElementException if the queue is empty.
2	`boolean offer(E obj)` Attempts to add obj to the queue. Returns true if obj was added and false otherwise.
3	`E peek()` Returns the element at the head of the queue. It returns null if the queue is empty. The element is not removed.
4	`E poll()` Returns the element at the head of the queue, removing the element in the process. It returns null if the queue is empty.
5	`E remove()` Returns the element at the head of the queue, returning the element in the process. It throws NoSuchElementException if the queue is empty.

PriorityQueue

- `PriorityQueue` extends `AbstractQueue` and implements the `Queue` interface.
- It creates a queue that is prioritized based on the queue's comparator.
- `PriorityQueue` is declared as follows:

```
PriorityQueue<E> = new PriorityQueue<E>;
```

- It builds an empty queue with starting capacity as 11.

PriorityQueue - Example

```
import java.util.*;
public class PriorityQueueExample {
    public static void main(String[] args) {
        PriorityQueue<Integer> numbers = new
PriorityQueue<>();
        numbers.add(750);
        numbers.add(500);
        numbers.add(900);
        numbers.add(100);
        while (!numbers.isEmpty()) {
            System.out.println(numbers.remove());
        }
    }
}
```

```
G:\Darshan\Java
100
500
750
900
```

PriorityQueue - Constructors

Sr.	Constructor & Description
1	`PriorityQueue()` Creates a PriorityQueue with the default initial capacity (11) that orders its elements according to their natural ordering.
2	`PriorityQueue(Collection<? extends E> c)` Creates a PriorityQueue containing the elements in the specified collection.
3	`PriorityQueue(int initialCapacity)` Creates a PriorityQueue with the specified initial capacity that orders its elements according to their natural ordering.
4	`PriorityQueue(int initialCapacity, Comparator<? super E> comparator)` Creates a PriorityQueue with the specified initial capacity that orders its elements according to the specified comparator.
5	`PriorityQueue(PriorityQueue<? extends E> c)` Creates a PriorityQueue containing the elements in the specified priority queue.
6	`PriorityQueue(SortedSet<? extends E> c)` Creates a PriorityQueue containing the elements in the specified sorted set.

PriorityQueue - Methods

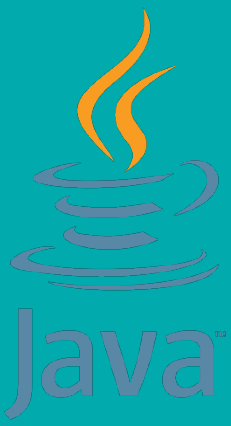
Sr.	Method & Description
1	`boolean add(E e)` Inserts the specified element into this priority queue.
2	`void clear()` Removes all of the elements from this priority queue.
3	`Comparator<E> comparator()` Returns the comparator used to order the elements in this queue, or null if this queue is sorted according to the natural ordering of its elements.
4	`boolean contains(Object o)` Returns true if this queue contains the specified element.
5	`Iterator<E> iterator()` Returns an iterator over the elements in this queue.
6	`boolean offer(E e)` Inserts the specified element into this priority queue.

PriorityQueue - Methods

Sr.	Method & Description
7	`E peek()` Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
8	`E poll()` Retrieves and removes the head of this queue, or returns null if this queue is empty.
9	`boolean remove(Object o)` Removes a single instance of the specified element from this queue, if it is present.
10	`int size()` Returns the number of elements in this collection.
11	`Object[] toArray()` Returns an array containing all of the elements in this queue.

3140705
Object Oriented Programming - I

Unit – 11
Sets & Maps



Prof. Hardik A. Doshi

9978911553

hardik.doshi@darshan.ac.in

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

List v/s Sets

List	Set
Lists allow duplicates.	Sets allow only unique
List is an ordered collection.	Sets is an unordered
Popular implementation of List interface includes ArrayList, Vector and LinkedList.	Popular implementation of Set interface includes HashSet, TreeSet and LinkedHashSet.

When to use List and Set?

Lists - If insertion order is maintained during insertion and allows duplicates.

Sets – If unique collection without any duplicates without maintaining order.

Singleton & Unmodifiable Collection

- java.util.Collections.singleton() method is a java.util.Collections class (static) method.
- It creates a immutable set over a single specified element.
- An application of this method is to remove an element from Collections like List and Set.

Example

```
myList : {"God", "code", "Practice", " Error",
    "Java", "Class", "Error", "Practice",
                    "Java" }
```

- To remove all "Error" elements from the list at once, we use singleton() method as,

```
myList.removeAll(Collections.singleton("Error"
                    ));
```

- After using `singleton()` and `removeAll()`, we get following.

```
{"God", "code", "Practice", "Java", "Class",
                "Practice", "Java"}
```

Maps

- A map is an object that stores associations between keys and values, or key/valuepairs.
- Given a key, you can find its value. Both keys and values are objects.
- The keys must be unique, but the values may be duplicated. Some maps can accept a null key and null values, others cannot.
- Maps don't implement the Iterable interface. This means that you cannot cycle through a map using a for-each style for loop. Furthermore, you can’t obtain an iterator to a map.

Map Interfaces

Interface	Description
Map	Maps unique keys to values.
Map.Entry	Describes an element (a key/value pari) in a map. This is an inner class of Map.
NavigableMap	Extends SortedMap to handle the retrieval of entries based on closest-match searches.
SortedMap	Extends Map so that the keys are maintained in ascending order.

Map Classes

Class	Description
AbstractMap	Implements most of the Map interface.
EnumMap	Extends AbstractMap for use with enum keys.
HashMap	Extends AbstractMap to use a hash table.
TreeMap	Extends AbstractMap to use a tree.
WeakHashMap	Extends AbstractMap to use a hash table with weak keys.
LinkedHashMap	Extends HashMap to allow insertion-order iterators.
IdentityHashMap	Extends AbstractMap and uses reference

HashMap Class

- The HashMap class extends AbstractMap and implements the Map interface.
- It uses a hash table to store the map. This allows the execution time of get() and put() to remain constant even for large sets.
- HashMap is a generic class that has declaration:

```
class HashMap<K,V>
```

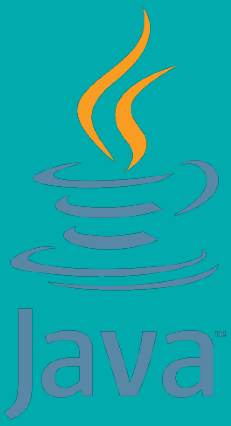
```java
import java.util.*;
class HashMapDemo {
    public static void main(String args[]) {
        // Create a hash map.
        HashMap<String, Double> hm = new HashMap<String, Double>();
        // Put elements to the map
        hm.put("John Doe", new Double(3434.34));
        hm.put("Tom Smith", new Double(123.22));
        hm.put("Jane Baker", new Double(1378.00));
        hm.put("Tod Hall", new Double(99.22));
        hm.put("Ralph Smith", new Double(-19.08));
        // Get a set of the entries.
        Set<Map.Entry<String, Double>> set = hm.entrySet();
        // Display the set.
        for(Map.Entry<String, Double> me : set) {
            System.out.print(me.getKey() + ": ");
            System.out.println(me.getValue());
        }
        System.out.println();
        //Deposit 1000 into John Doe's account.
        double balance = hm.get("John Doe");
        hm.put("John Doe", balance + 1000);
        System.out.println("John Doe's new balance: " +
        hm.get("John Doe"));
    }
}
```

HashMap - Constructors

Sr.	Constructor & Description
1	**`HashMap()`** Constructs an empty HashMap with the default initial capacity (16) and the default load factor (0.75).
2	**`HashMap(int initialCapacity)`** Constructs an empty HashMap with the specified initial capacity and the default load factor (0.75).
3	**`HashMap(int initialCapacity, float loadFactor)`** Constructs an empty HashMap with the specified initial capacity and load factor.
4	**`HashMap(Map<? extends K,? extends V> m)`** Constructs a new HashMap with the same mappings as the specified Map.

3140705
Object Oriented Programming - I

Unit – 12 Concurrency / Multithreading



Prof. Hardik A. Doshi

9978911553

hardik.doshi@darshan.ac.in

Dedicated Faculty, Committed Education

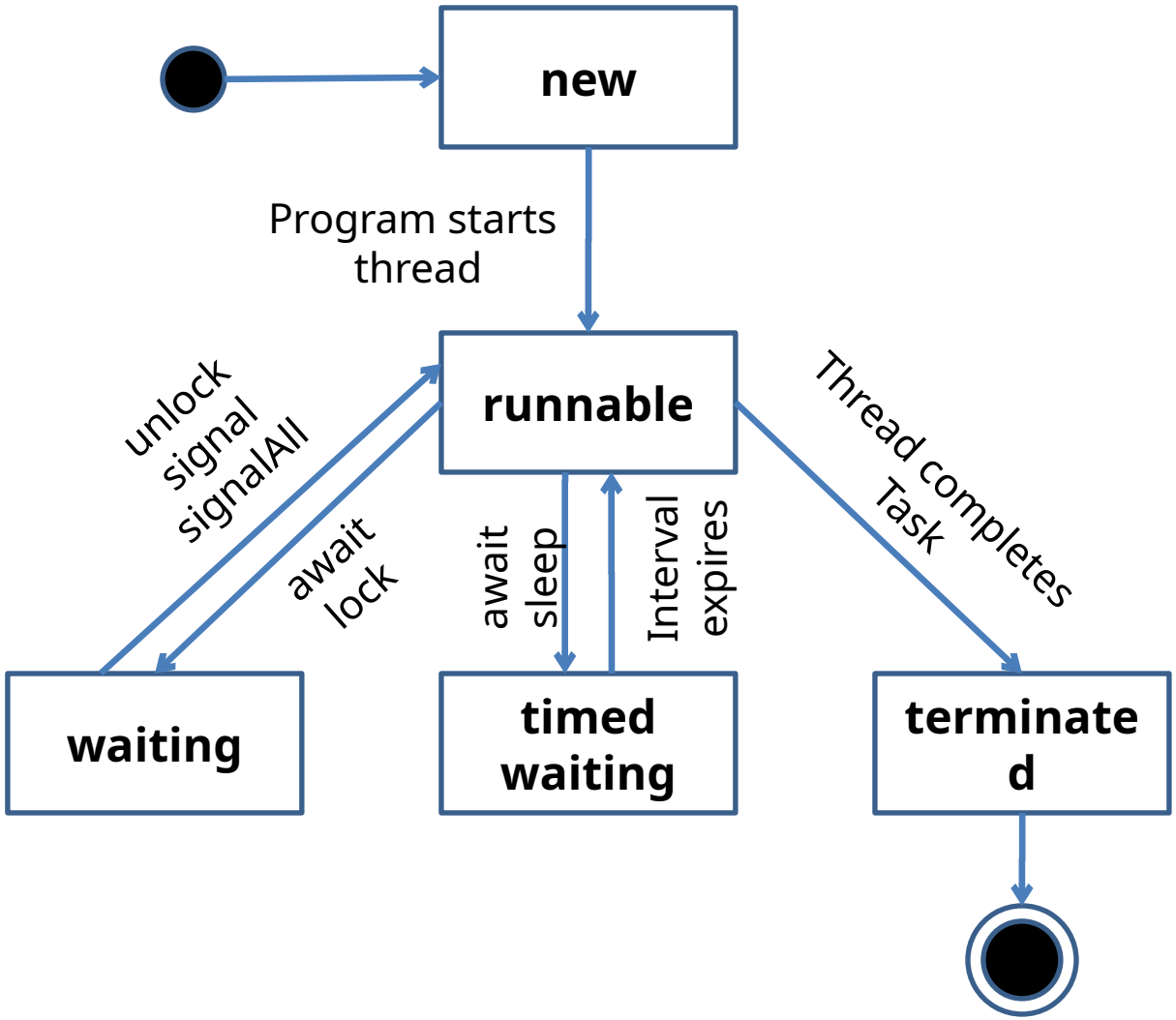
Darshan

Institute of Engineering & Technology

What is Multithreading?

- Multithreading in Java is a process of *executing multiple threads* simultaneously.
- A thread is a *lightweight sub-process*, the smallest unit of processing.
- Multiprocessing and multithreading, both are used to achieve multitasking.
- Threads use a *shared memory area*. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Life cycle of a Thread



Life cycle of a Thread (Cont.)

- A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies.
- There are **5** stages in the life cycle of the Thread
  - **New:** A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
  - **Runnable:** After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
  - **Waiting:** Sometimes a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals waiting thread to continue.
  - **Timed waiting:** A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
  - **Terminated:** A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Creating a Thread in Java

- There are two ways to create a Thread
  1. **extending** the **Thread class**
  2. **implementing** the **Runnable interface**

1) Extending Thread Class

- One way to create a thread is to create a new class that extends `Thread`, and then to create an instance of that class.
- The extending class must override the `run( )` method, which is the entry point for the new thread.
- It must also call `start( )` to begin execution of the new thread.

```java
class NewThread extends Thread {
    NewThread() {
        super("Demo Thread");
        System.out.println("Child thread: " + this);
        start(); // Start the thread
    }
    public void run() {
        try {
            for (int i = 5; i > 0; i--) {
                System.out.println("Child Thread: " + i);
                Thread.sleep(500);
            }
        } catch (InterruptedException e) {
            System.out.println("Child interrupted.");
        }
        System.out.println("Exiting child thread.");
    }
}
class ExtendThread {
    public static void main(String args[])
        new NewThread(); // create a new th
        try {
            for (int i = 5; i > 0; i--) {
                System.out.println("Main T
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
             System.out.println("Main threa
        }
        System.out.println("Main thread exi
    }
}
```

```
G:\Darshan\Java 2019\PPTs\HAD\Programs>ja
Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Exiting child thread.
Main Thread: 2
Main Thread: 1
Main thread exiting.
```

2) Implementing Runnable Interface

- To implement thread using `Runnable` interface, `Runnable` interface needs to be implemented by the class.

```
class NewThread implements Runnable
```

- Class which implements `Runnable` interface should override the `run()` method which containts the logic of the thread.

```
public void run( )
```

- Instance of `Thread` class is created using following constructor.

```
Thread(Runnable threadOb, String threadName);
```

- Here `threadOb` is an instance of a class that implements the `Runnable` interface and the name of the new thread is specified by `threadName`.
- `start()` method of Thread class will invoke the `run()` method.

```
class NewThread implements Runnable {
    NewThread() {
        super("Demo Thread");
        System.out.println("Child thread: " + this);
        start(); // Start the thread
    }
    public void run() {
        try {
            for (int i = 5; i > 0; i--) {
                System.out.println("Child Thread: " + i);
                Thread.sleep(500);
            }
        } catch (InterruptedException e) {
            System.out.println("Child interrupted.");
        }
        System.out.println("Exiting child thread.");
    }
}
class ExtendThread {
    public static void main(String args[]) {
        new NewThread(); // create a new th
        try {
            for (int i = 5; i > 0; i--) {
                System.out.println("Main T
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Main threa
        }
        System.out.println("Main thread exi
    }
}
```

```
G:\Darshan\Java 2019\PPTs\HAD\Programs>ja
Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Exiting child thread.
Main Thread: 2
Main Thread: 1
Main thread exiting.
```

Thread using Executor Framework

- Steps to execute thread using Executor Framework are as follows:

1. Create a task (Runnable Object) to execute
2. Create Executor Pool using Executors
3. Pass tasks to Executor Pool
4. Shutdown the Executor Pool

```java
import java.util.concurrent.*;
class Task implements Runnable {
     private String name;
     public Task(String s) {
          name = s;
     }
     public void run() {
          try {
               for (int i = 1; i<=5; i++) {
                    System.out.println(name+" - task number - "+i);
                    Thread.sleep(1000);
               }
               System.out.println(name+" complete");
          }
          catch(InterruptedException e) {
                e.printStackTrace();
          }
     }
}
public class ExecutorThreadDemo {
     public static void main(String[] args) {
          Runnable r1 = new Task("task 1");
          Runnable r2 = new Task("task 2");
          Runnable r3 = new Task("task 3");
          Runnable r4 = new Task("task 4");
          Runnable r5 = new Task("task 5");
          ExecutorService pool = Executors.newFixedThreadPool(3);
          pool.execute(r1);
          pool.execute(r2);
          pool.execute(r3);
          pool.execute(r4);
          pool.execute(r5);
          pool.shutdown();
     }
}
```

```
G:\Darshan\Java 2019\PPTs\HAD\Pro
task 1 - task number - 1
task 2 - task number - 1
task 3 - task number - 1
task 1 - task number - 2
task 2 - task number - 2
task 3 - task number - 2
task 2 - task number - 3
task 1 - task number - 3
task 3 - task number - 3
task 1 - task number - 4
task 2 - task number - 4
task 3 - task number - 4
task 1 - task number - 5
task 2 - task number - 5
task 3 - task number - 5
task 2 complete
task 1 complete
task 4 - task number - 1
task 3 complete
task 5 - task number - 1
task 4 - task number - 2
task 5 - task number - 2
task 4 - task number - 3
task 5 - task number - 3
task 4 - task number - 4
task 5 - task number - 4
task 4 - task number - 5
task 5 - task number - 5
task 4 complete
task 5 complete
```

Thread Synchronization

- When we start *two or more threads* within a program, there may be a situation when multiple threads try to *access the same resource* and finally they can produce unforeseen result due to concurrency issues.
- For example, if multiple threads try to write within a same file then they may corrupt the data because one of the threads can override data or while one thread is opening the same file at the same time another thread might be closing the same file.
- So there is a need to synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time.
- Java programming language provides a very handy way of creating threads and synchronizing their task by using *synchronized methods & synchronized blocks.*

Problem without synchronization (Example)

```
class Table {
  void printTable(int n) {
    for (int i = 1; i <= 5; i++) {
      System.out.print(n * i + " ");
      try {
        Thread.sleep(400);
      } catch (Exception e) {
        System.out.println(e);
      }
    }
  }
}
```

```
class MyThread1 extends
Thread {
  Table t;
  MyThread1(Table t) {
    this.t = t;
  }
  public void run() {
    t.printTable(5);
  }
}
```

```
class MyThread2 extends
Thread {
  Table t;
  MyThread2(Table t) {
    this.t = t;
  }
  public void run() {
    t.printTable(100);
  }
```

```
public class TestSynchronization {
  public static void main(String
args[]){
    Table obj = new Table();
    MyThread1 t1 = new MyThread1(obj);
    MyThread2 t2 = new MyThread2(obj);
    t1.start();
    t2.start();
  }
}
```

```
C:\WINDOWS\system32\cmd.exe
D:\DegreeDemo\PPTDemo>javac TestSynchronization.java
D:\DegreeDemo\PPTDemo>java TestSynchronization
5 100 200 10 300 15 400 20 500 25
```

Solution with synchronized method

```
class Table {
  synchronized void printTable(int n)
{
    for (int i = 1; i <= 5; i++) {
      System.out.print(n * i + " ");
      try {
        Thread.sleep(400);
      } catch (Exception e) {
        System.out.println(e);
      }
    }
  }
}
```

```
class MyThread1 extends
Thread {
  Table t;
  MyThread1(Table t) {
    this.t = t;
  }
  public void run() {
    t.printTable(5);
  }
}
```

```
class MyThread2 extends
Thread {
  Table t;
  MyThread2(Table t) {
    this.t = t;
  }
  public void run() {
    t.printTable(100);
  }
```

```
public class TestSynchronization {
  public static void main(String
args[]){
    Table obj = new Table();
    MyThread1 t1 = new MyThread1(obj);
    MyThread2 t
    t1.start();
    t2.start();
  }
}
```

```
C:\WINDOWS\system32\cmd.exe
D:\DegreeDemo\PPTDemo>javac TestSynchronization.java
D:\DegreeDemo\PPTDemo>java TestSynchronization
5 10 15 20 25 100 200 300 400 500
```

Solution with synchronized blocks

```
class Table {
  void printTable(int n) {
    for (int i = 1; i <= 5; i++) {
      System.out.print(n * i + " ");
      try {
        Thread.sleep(400);
      } catch (Exception e) {
        System.out.println(e);
      }
    }
  }
}
```

```
class MyThread1 extends
Thread {
  Table t;
  MyThread1(Table t) {
    this.t = t;
  }
  public void run() {
    synchronized (t) {
      t.printTable(5);
    }
  }
}
```

```
public class TestSynchronization {
  public static void main(String
args[]){
    Table obj = new Table();
    MyThread1 t1 = new MyThread1(obj);
    MyThread2 t2 = new MyThread2(obj);
    t1.start();
    t2.start();
  }
}
```

```
class MyThread1 extends
Thread {
  Table t;
  MyThread1(Table t) {
    this.t = t;
  }
  public void run() {
    synchronized (t) {
```

```
C:\WINDOWS\system32\cmd.exe
D:\DegreeDemo\PPTDemo>javac TestSynchronization.java
D:\DegreeDemo\PPTDemo>java TestSynchronization
5 10 15 20 25 100 200 300 400 500
```