

HISTORY OF JAVA

Java was developed by Sun

Microsystems (which is now the subsidiary
of Oracle) in the year 1995

James Gosling is known as the father of
Java. Before Java, its name was Oak

HISTORY OF JAVA

- ▶ Motivation behind the development was the need for a platform-independent (that is, architecture-neutral)
- ▶ language that could be used to create software to be embedded in various consumer electronic devices, such as microwave ovens and remote controls.

What is Java?

- ▶ Java is a general-purpose programming language that is class-based, object-oriented, and designed to have as few implementation dependencies as possible.
- ▶ Java is platform independent and portable
- ▶ The moto behind the development of java is code once run anywhere anytime

- ▶ As of 2019, Java was one of the most popular programming languages in use according to GitHub,^{[17][18]} particularly for client-server web applications, with a reported 9 million developers.^[1]

Types of Java Applications

- ▶ There are mainly 4 types of applications that can be created using Java programming:
 - ▶ 1) Standalone Application
 - ▶ 2) Web Application
 - ▶ 3) Enterprise Application
 - ▶ 4) Mobile Application

What is a standalone Application?

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

Web Application

- ▶ An application that runs on the server side and creates a dynamic page is called a web application.

Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

Enterprise Application

- ▶ An application that is distributed in nature, such as banking applications, etc. is called enterprise application.
- ▶ It has advantages of the high-level security, load balancing, and clustering.
- ▶ In Java, EJB is used for creating enterprise applications.

Mobile Application

An application which is created for mobile devices is called a mobile application.

Currently, Android and Java ME are used for creating mobile applications.

SOFTWARE REQUIREMENTS

- ▶ You have to install jdk version 8 as iam using this version in the video
- ▶ JDK is the short form for java development kit
- ▶ <https://www.oracle.com/technetwork/java/javase/downloads/index.html>
- ▶ Download can be done from the above link

IDE FOR JAVA

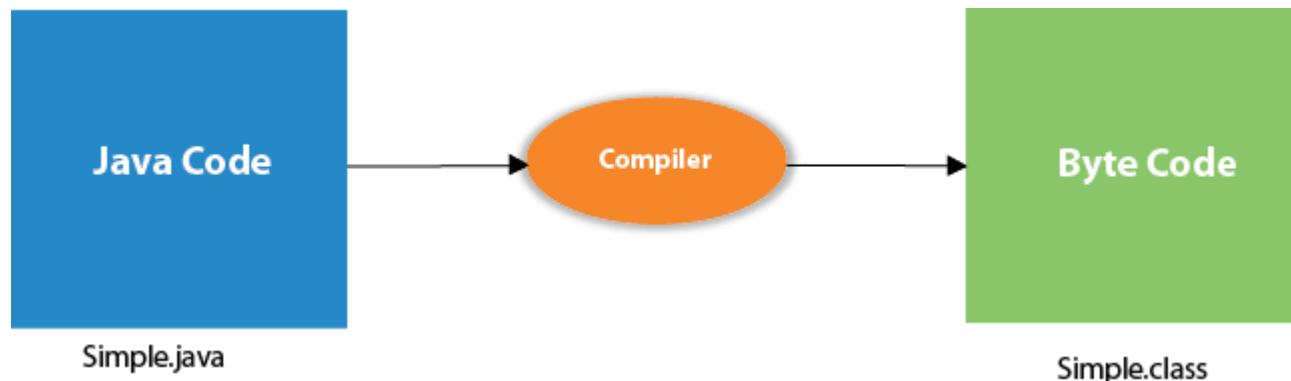
- ▶ NETBEANS is the IDE which is going to be used in the examples
- ▶ Download link
- ▶ <https://netbeans.org/downloads/8.2/>
- ▶ The other popular IDE 's are Eclipse and IntelliJ

Usages

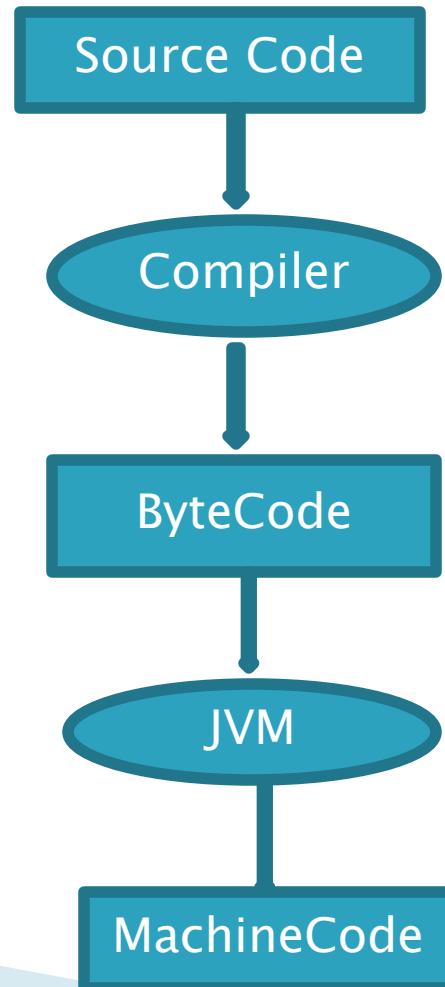
- ▶ Java has been used in different domains. Some of them are listed below:
- ▶ **Banking:** To deal with transaction management.
- ▶ **Retail:** Billing applications that you see in a store/restaurant are completely written in Java.
- ▶ **Information Technology:** Java is designed to solve implementation 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.

Java Execution Flow

- ▶ Java is a programming language which is
- ▶ compiled and then interpreted into executable code



FLOW OF EXECUTION



Javas Magic :BYTECODE

The key that allows Java to solve both the security and the portability problems

is that the output of a Java compiler is not executable code. Rather, it is bytecode.

Bytecode is

a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the *Java Virtual Machine (JVM)*.

Translating a Java program into bytecode makes it much easier to run a program in a wide variety of environments because only the JVM needs to be implemented for each platform. Once the run-time package exists for a given system, any Java program can run on it. Remember, although the details of the JVM will differ from platform to platform,

Java Language

Identifiers

Identifiers are used to name things, such as classes, variables, and methods. An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters. (The dollar-sign character is not intended for general use.) They must not begin with a number, lest they be confused with a numeric literal. Again, Java is case-sensitive, so **VALUE** is a different identifier than **Value**.



EXAMPLES OF IDENTIFIERS

Valid Identifiers:

AvgTemp	count	a4	&test
---------	-------	----	-------

Invalid identifiers:

2count	High-temp	Not/ok		
--------	-----------	--------	--	--

Data Types

- ▶ **The Primitive Types**

- ▶ Java defines eight *primitive types of data*: *byte*, *short*, *int*, *long*, *char*, *float*, *double*, and *boolean*.
- ▶ • **Integers**
- ▶ This group includes byte, short, int, and long, which are for whole-valued signed numbers.
- ▶ • **Floating-point numbers**
- ▶ This group includes float and double, which represent numbers with fractional precision.

- ▶ • **Characters**
- ▶ This group includes `char`, which represents symbols in a character set,
 - ▶ like letters and numbers.
- ▶ • **Boolean**
- ▶ This group includes `boolean`, which is a special type for representing
 - ▶ true/false values.

Integers

Name	Width	Range
long	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	-2,147,483,648 to 2,147,483,647
short	16	-32,768 to 32,767
byte	8	-128 to 127

Floating–Point Types

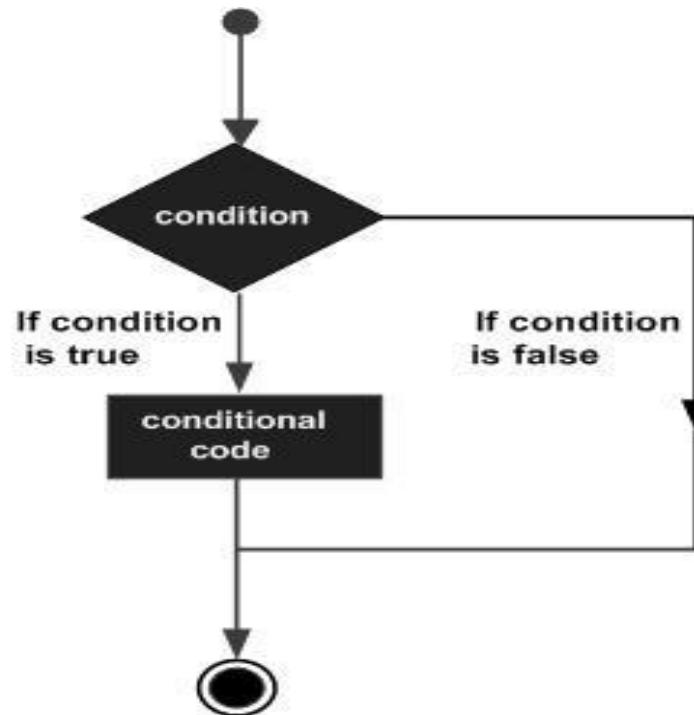
Name	Width in bits	Range
float	32	4.9e-324 to 1.8e+308
double	64	1.4e-045 to 3.4e+038

- ▶ Floating–point numbers, also known as *real numbers, are used when evaluating expressions*
- ▶ that require fractional precision

Java Keywords

- ▶ Java **keywords** are also known as **reserved words**. Keywords are particular words whose meaning is known to the compiler.
- ▶ These predefined words cannot be used as a variable or object name.

Control Statements



Syntax

- ▶ **if(condition){**
- ▶ **//code if condition is true**
- ▶ **}else{**
- ▶ **//code if condition is false**
- ▶ **}**

If... else if... else

- ▶ **if(condition1){**
- ▶ //code to be executed if condition1 is true
- ▶ **}else if(condition2){**
- ▶ //code to be executed if condition2 is true
- ▶ **}**
- ▶ **else if(condition3){**
- ▶ //code to be executed if condition3 is true
- ▶ **}**
- ▶ **...**
- ▶ **else{**
- ▶ //code to be executed if all the conditions are false
- ▶ **}**

Loops in Java

- ▶ In programming languages, loops are used to execute a set of instructions/functions repeatedly when some conditions become true.
- ▶ There are three types of loops in java.
- ▶ for loop
- ▶ while loop
- ▶ do-while loop
- ▶

Syntax

- ▶ `for(init;condition;incr/decr)`
- ▶ `{`
- ▶ `// code to be executed`
- ▶ `}`

- ▶ `while(condition)`
- ▶ `{`
- ▶ `//code to be executed`
- ▶ `}`

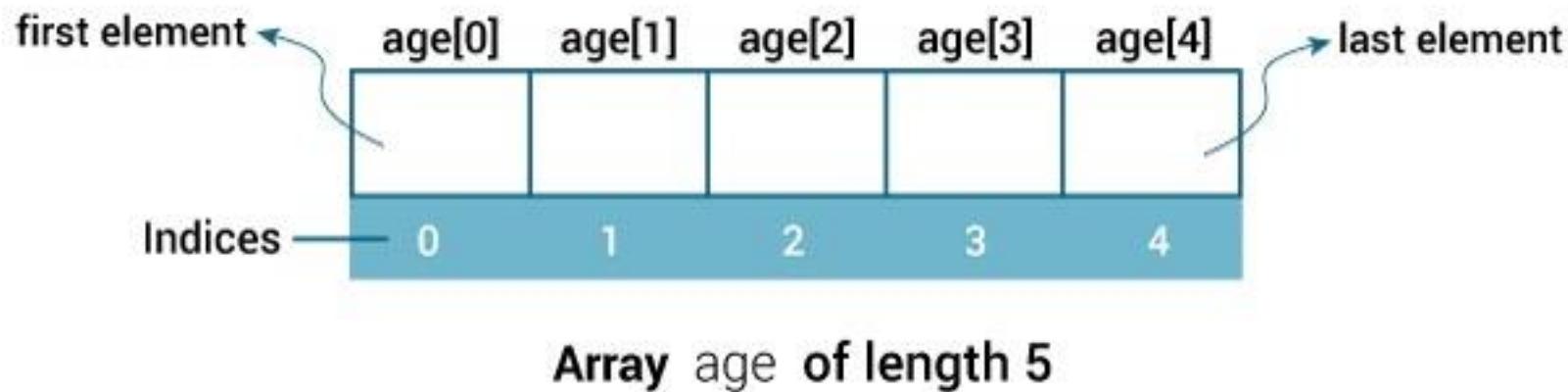
- ▶ `Do`
- ▶ `{ //code to be executed }`
- ▶ `while(condition);`

JAVA ARRAYS

- ▶ Java provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type.
- ▶ Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. To declare an array, define the variable type with square brackets: `String[] cars;` We have now declared a variable that holds an array of strings.

ARRAY DECLARATION

- ▶ `int[] age = new int[5];`



- ▶ The first element of array is age[0], second is age[1] and so on.
- ▶ If the length of an array is n, the last element will be arrayName[n-1].
- ▶ Since the length of age array is 5, the last element of the array is age[4] in the above example.

▶ Java Constructors

- ▶ A constructor in Java is a special method that is used to initialize objects.
- ▶ The constructor is called when an object of a class is created.
- ▶ It can be used to set initial values for object attributes:
- ▶ Note that the constructor name must match the class name, and it cannot have a return type (like void).

- ▶ Also note that the constructor is called when the object is created.
- ▶ All classes have constructors by default:
- ▶ if you do not create a class constructor yourself, Java creates one for you.
- ▶ However, then you are not able to set initial values for object attributes.

What is Class and Object in Java OOPS?

- ▶ What is Class?
- ▶ A class is an entity that determines how an object will behave and what the object will contain.
- ▶ In other words, it is a blueprint or a set of instruction to build a specific type of object.
- ▶ **Syntax**

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

► What is an Object?

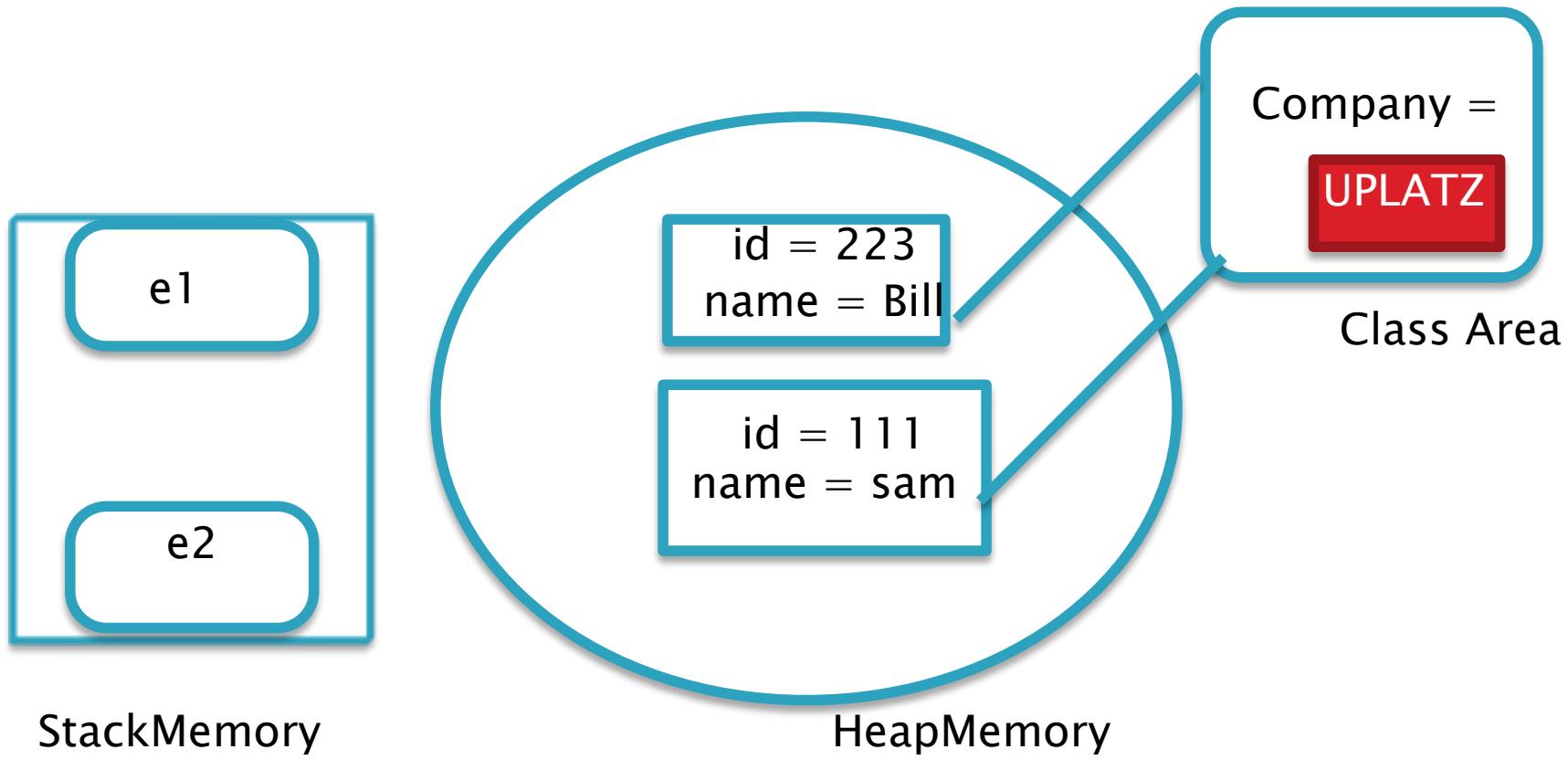
- ▶ An object is nothing but a self-contained component which consists of methods and properties to make a particular type of data useful.
- ▶ Object determines the behavior of the class. When you send a message to an object, you are asking the object to invoke or execute one of its methods.

What is the Difference Between Object & Class?

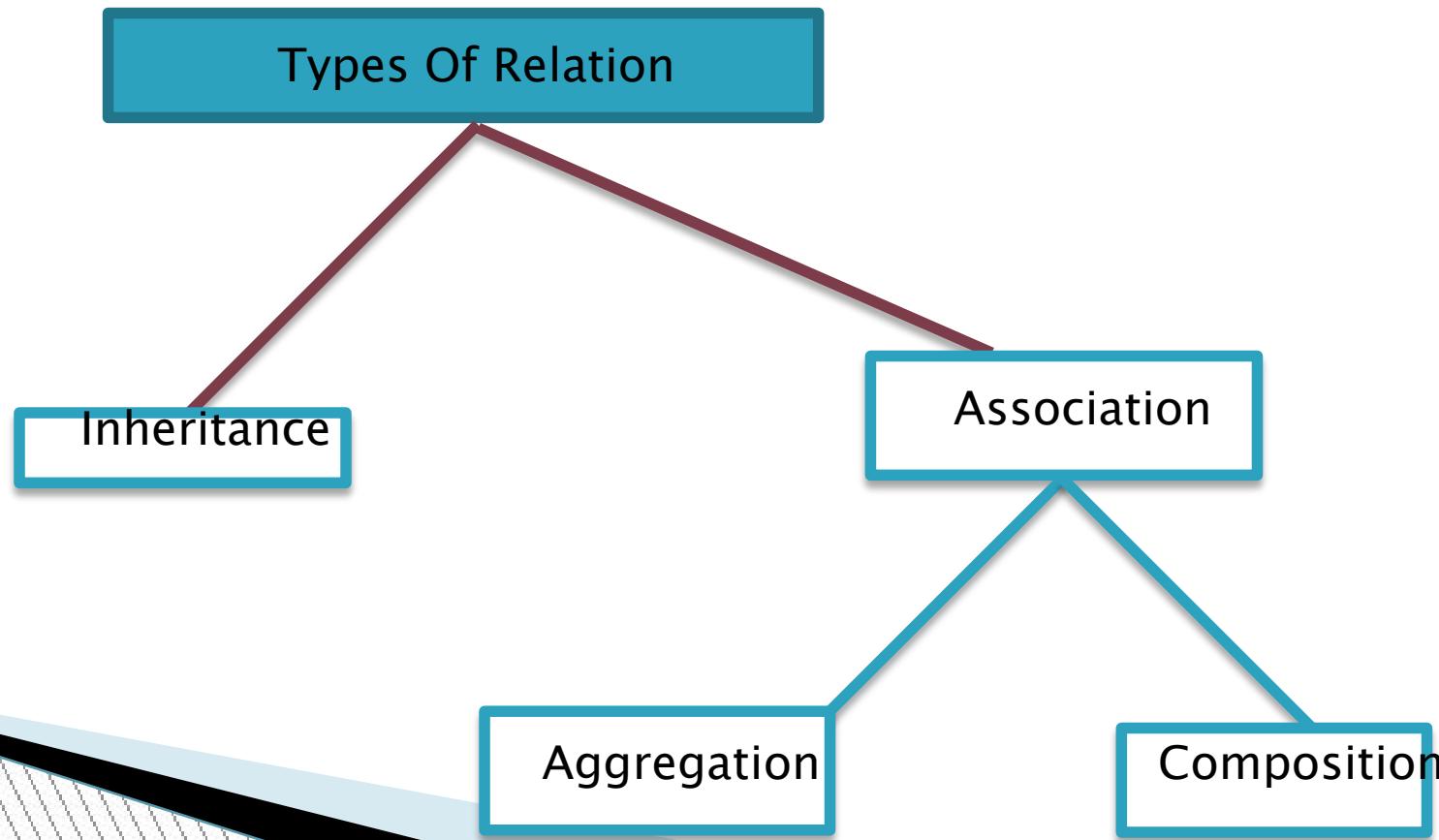
- ▶ A class is a blueprint or prototype that defines the variables and the methods (functions) common to all objects of a certain kind.
- ▶ An object is a specimen of a class. Software objects are often used to model real-world objects you find in everyday life.

UNDERSTANDING STATIC

- ▶ The static keyword in Java is used for memory management mainly.
- ▶ We can apply static keyword with variables, methods, blocks and nested classes.
- ▶ The static keyword belongs to the class than an instance of the class.



RELATIONSHIP AMONG CLASSES



INHERITANCE (IS A) RELATIONSHIP

INHERITANCE IS
CALLED (IS A)
RELATIONSHIP

ASSOCIATION IS CALLED
(HAS A) RELATIONSHIP

INHERITANCE

- ▶ The process by which one class acquires the properties(data members) and functionalities(methods) of another class is called inheritance.
- ▶ The aim of inheritance is to provide the reusability of code so that a class has to write only the unique features and rest of the common properties and functionalities can be extended from the other class.
- ▶ Child Class:
- ▶ The class that extends the features of another class is known as child class, sub class or derived class.

- ▶ Child Class: The class that extends the features of another class is known as child class, sub class or derived class.
- ▶ Parent Class: The class whose properties and functionalities are used(inherited) by another class is known as parent class, super class or Base class.

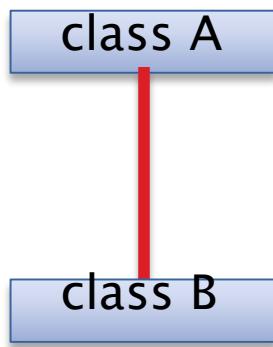
▶ SYNTAX

- ▶ To inherit a class we use extends keyword.
Here class XYZ is child class and class ABC is parent class.
- ▶ The class XYZ is inheriting the properties and methods of ABC class.
- ▶ class XYZ extends ABC
- ▶ {
- ▶ }

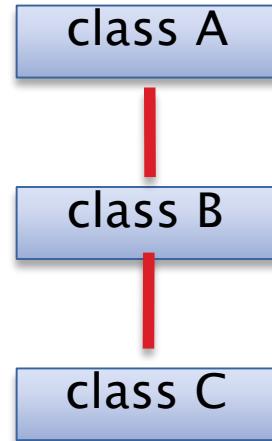
ADVANTAGES

- ▶ 1) Code Reusability
- ▶ 2) Cost Cutting
- ▶ 3) Reduced Redundancy

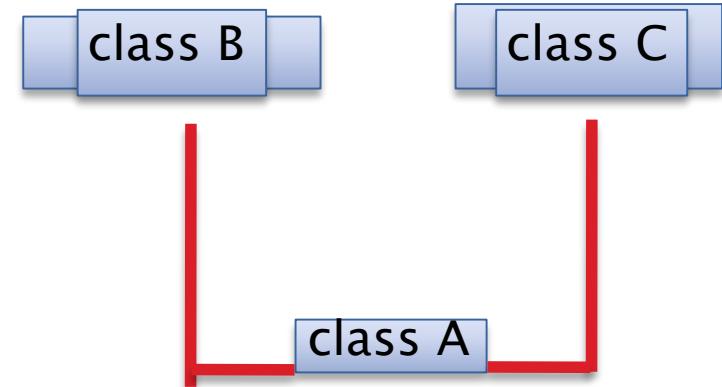
TYPES OF INHERITANCE



SINGLE
INHERITANCE



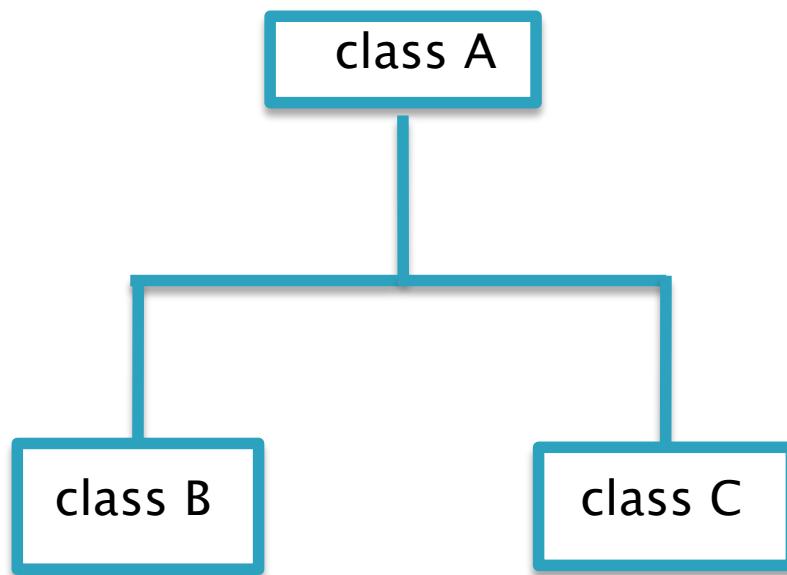
MULTILEVEL
INHERITANCE



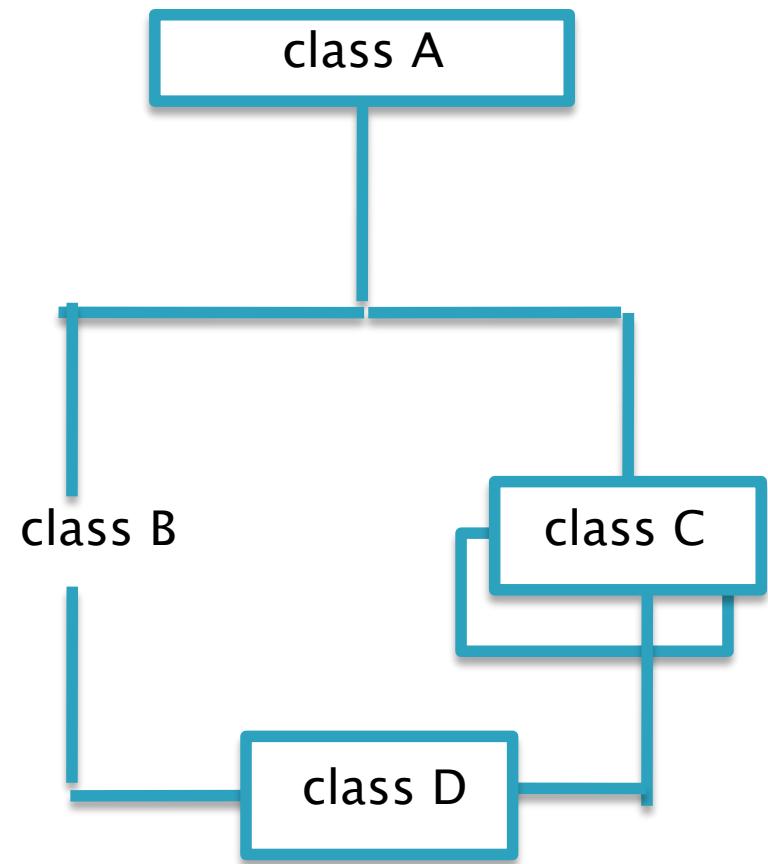
MULTIPLE
INHERITANCE

TYPES OF INHERITANCE

HIERARCHICAL INHERITANCE



HYBRID INHERITANCE



METHOD OVERLOADING

- ▶ Method Overloading is a feature that allows a class to have more than one method with the same name but their argument lists are different.

Different ways to overload the method

- ▶ There are two ways to overload the method in java
- ▶ By changing number of arguments
- ▶ By changing the data type

EXAMPLE

- ▶ For example: This is a valid case of overloading
- ▶ In order to overload a method, the argument lists of the methods must differ in either of these:
 - ▶ 1. Number of parameters.
 - ▶ add(int, int)
 - ▶ add(int, int, int)

EXAMPLE OF METHOD OVERLOADING

- ▶ Data type of parameters.
 - ▶ add(int, int)
 - ▶ add(int, float)

Invalid case of method overloading:

- ▶ If two methods have same name, same parameters and have different return type, then this is not a valid method overloading example. This will throw compilation error.
- ▶ `int sum(int, int)`
- ▶ `float sum(int, int)`

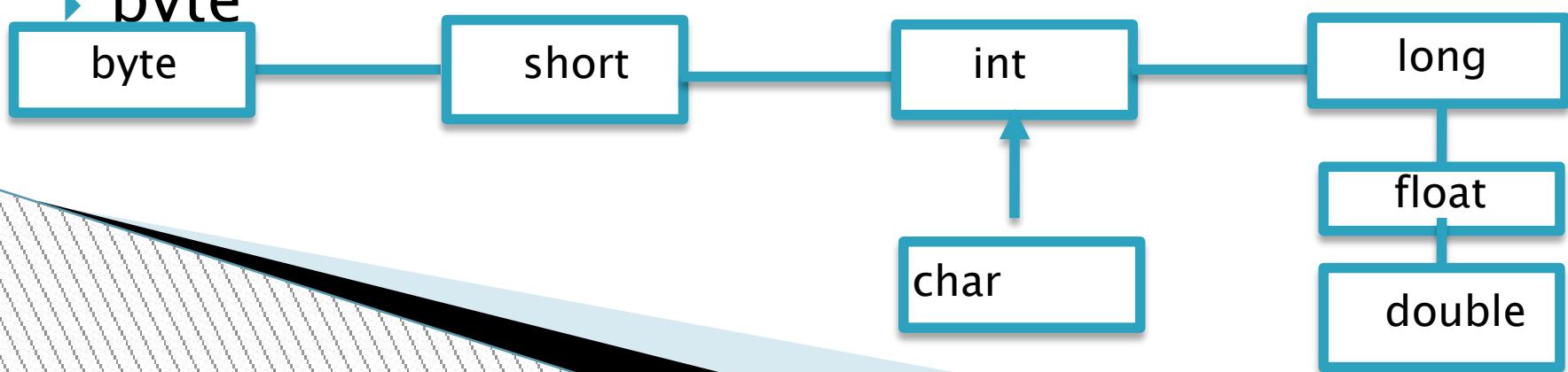
STATIC POLYMORPHISM

- ▶ Method overloading is an example of Static Polymorphism
- ▶ Static Polymorphism is also known as compile time binding or early binding.
- ▶ Static binding happens at compile time.
Method overloading is an example of static binding where binding of method call to its definition happens at Compile time.

Method Overloading and Type Promotion

- ▶ When a data type of smaller size is promoted to the data type of bigger size than this is called type promotion.
- ▶ Example: byte data type can be promoted to short, a short data type can be promoted to int, long, double etc.

▶ **bvte**



Member Access and Inheritance

Although a subclass includes all of the members of its superclass, it cannot access those members of the superclass that have been declared as **private**

class A

{

public int x;

private int y;//not available for inheritance

}

Using **super**

A subclass can call a constructor defined by its superclass by use of the following form of **super**: `super(arg-list);`

METHOD OVERRIDING

In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to *override* the method in the superclass.

When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass.

The version of the method defined by the superclass
is ~~hidden~~

Usage of Java Method Overriding

- ▶ Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- ▶ Method overriding is used for runtime polymorphism

RULES FOR METHOD OVERRIDING

- ▶ The method must have the same name as in the parent class
- ▶ The method must have the same parameter as in the parent class.
- ▶ There must be an IS-A relationship (inheritance).

Covariant Return Type

- ▶ The covariant return type specifies that the return type may vary in the same direction as the subclass.
- ▶ Before Java5, it was not possible to override any method by changing the return type. But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type.

UNDERSTANDING SUPER

- ▶ The super keyword in Java is a reference variable which is used to refer immediate parent class object.
- ▶ Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage

- ▶ super can be used to refer immediate parent class instance variable.
- ▶ super can be used to invoke immediate parent class method
- ▶ super() can be used to invoke immediate parent class constructor.

FINAL

- ▶ The final keyword in java is used to restrict the user.
- ▶ The java final keyword can be used in the following context.
- ▶ **Final can be:**
 - ▶ variable
 - ▶ method
 - ▶ class

FINAL VARIABLE

- ▶ If you declare a variable as final you cannot change the value of the variable.
- ▶ It will be CONSTANT throughout the program
- ▶ Any effort to modify the value will give a compile time error
- ▶ Example final int x = 300;
- ▶ x=324; //compile time error

final method

- ▶ If you make any method as final, you cannot override it.

final class

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

blank final variable

- ▶ A final variable that is not initialized at the time of declaration is known as blank final variable.
- ▶ If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed,
- ▶ it is useful. For example PAN CARD number of an employee.
- ▶ It can be initialized only in constructor.

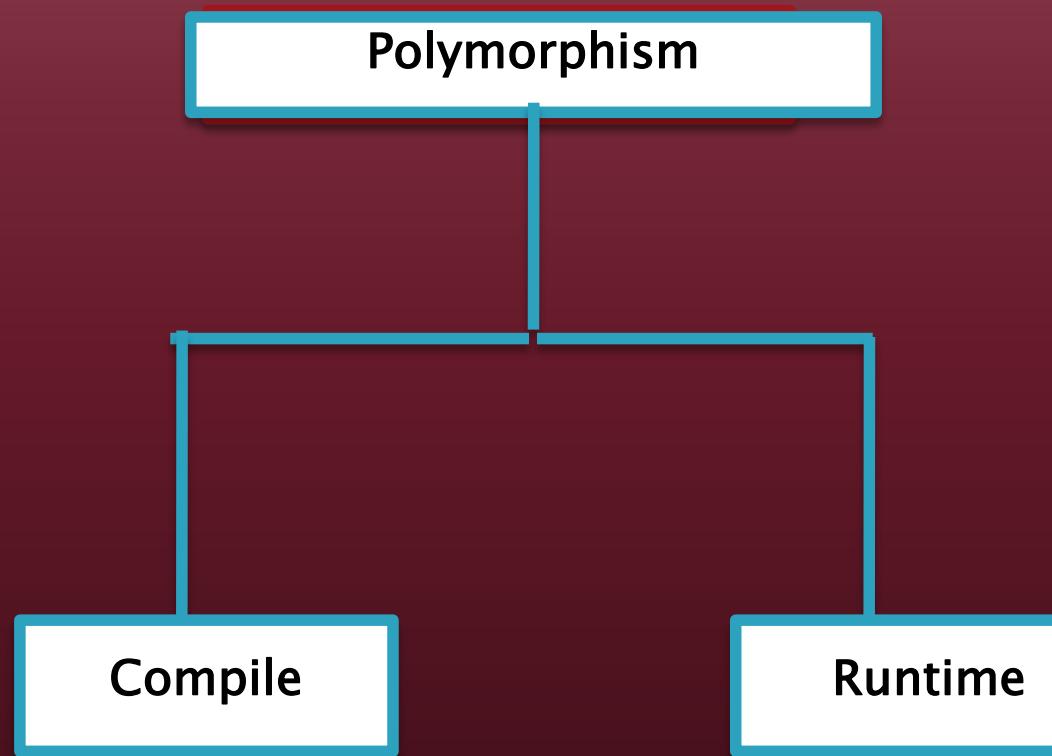
static blank final variable

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

POLYMORPHISM

- ▶ Polymorphism is a greek word meaning many forms
- ▶ poly -> many
- ▶ morphism forms
- ▶ Can be broadly classified into two categories

CLASSIFICATION



Runtime Polymorphism

- ▶ Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time.
- ▶ In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.
- ▶ Let's first understand the upcasting before Runtime Polymorphism.

▶ Upcasting

- ▶ If the reference variable of Parent class refers to the object of Child class, it is known as upcasting
- ▶ class A{}
- ▶ class B extends A{}
- ▶ A a=new B(); //upcasting

- ▶ **Static Binding and Dynamic Binding**
- ▶ Connecting a method call to the method body is known as binding.
- ▶ There are two types of binding
 - ▶ Static Binding (also known as Early Binding).
 - ▶ Dynamic Binding (also known as Late Binding).

COMPILE TIME POLYMORPHISM

Overloading Methods

In Java it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different.

When this is the case, the methods are said to be *overloaded*, and the process is referred to as *method overloading*.

Method overloading is one of the ways that Java implements compile time polymorphism.

When an overloaded method is invoked, Java uses the type and/or number of arguments as its guide to determine which version of the overloaded method to actually call.

Thus, overloaded methods must differ in the type and/or number of their parameters.

- . When Java encounters a call to an overloaded method, it simply executes the version of the method whose parameters match the arguments used in the call.

Overloading Constructors

In addition to overloading normal methods, you can also overload constructors

ABSTRACT CLASS

- ▶ A class which is declared with the abstract keyword is known as an abstract class in Java.
- ▶ It can have abstract and non-abstract methods (method with the body).
- ▶ It needs to be extended and its method implemented. It cannot be instantiated.

ABSTRACT METHOD

- ▶ A method which is declared as abstract and does not have implementation is known as an abstract method.
- ▶ Example of abstract method
- ▶ `abstract void printStatus(); //no method body and abstract`

- ▶ 1) Abstract method has no body.
- ▶ 2) Always end the declaration with a semicolon(;) .
- ▶ 3) It must be overridden. An abstract class must be extended and in a same way abstract method must be overridden.
- ▶ 4) A class has to be declared abstract to have abstract methods.

Points to Remember

- ▶ An abstract class must be declared with an abstract keyword.
- ▶ It can have abstract and non-abstract methods.
- ▶ It cannot be instantiated.
- ▶ It can have constructors and static methods also.
- ▶ It can have final methods which will force the subclass not to change the body of the method.

Why can't we create the object of an abstract class?

- ▶ Because these classes are incomplete, they have abstract methods that have no body so if java allows you to create object of this class then if someone calls the abstract method using that object then What would happen?
- ▶ Also because an object is concrete. An abstract class is like a template, so you have to extend it and build on it before you can use it.

NOTE

- ▶ The class that extends the abstract class, have to implement all the abstract methods of it, else you have to declare that class abstract as well.

INTERFACE

- ▶ In the last video we discussed abstract class which is used for achieving partial abstraction.
- ▶ Unlike abstract class an interface is used for full abstraction.
- ▶ Abstraction is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user

WHAT IS AN INTERFACE

- ▶ An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method signature, no body, see: Java abstract method).
- ▶ Also, the variables declared in an interface are public, static & final by default.

WHAT IS THE USE OF AN INTERFACE?

- ▶ They are used to achieve full abstraction in java
- ▶ Since methods in interfaces do not have body, they have to be implemented by the class before you can access them.
- ▶ The class that implements interface must implement all the methods of that interface.

INTERFACE AND INHERITANCE

- ▶ Interfaces can be extended
- ▶ This is how multiple inheritance can be implemented in java

TAG OR MARKER INTERFACE

- ▶ An empty interface is known as tag or marker interface. For example Serializable, EventListener, Remote(java.rmi.Remote) are tag interfaces.
- ▶ These interfaces do not have any field and methods in it.

VARIABLES IN AN INTERFACE

Variables declared in interface are public, static and final by default.

```
interface Demo
{
    int x=40;
    public int x=40;
    public static final int x=40;
    final int x=40;
    static int x=40;
}
```

- ▶ Interface variables must be initialized at the time of declaration otherwise compiler will throw an error.
- ▶ interface Try
- ▶ {
- ▶ int x; //Compile-time error
- ▶ }

- ▶ Inside any implementation class, you cannot change the variables declared in interface because by default, they are public, static and final.
- ▶ class Sample implements Try
- ▶ {
- ▶ public static void main(String args[])
- ▶ {
- ▶ x=20; //compile time error
- ▶ }

- ▶ A class can implement any number of interfaces.
- ▶ If there are two or more same methods in two interfaces and a class implements both interfaces, implementation of the method once is enough.

Advantages of interface in java:

- ▶ Without bothering about the implementation part, we can achieve the security of implementation
- ▶ In java, multiple inheritance is not allowed, however you can use interface to make use of it as you can implement more than one interface.

Java 8 Default Method in Interface

- ▶ Since Java 8, we can have method body in interface.
- ▶ But we need to make it default method. Let's see an example:

Java 8 Static Method in Interface

- ▶ Since Java 8, we can have static method in interface.
- ▶ Let's see an example:

ADVANTAGES OF DEFAULT METHODS

- ▶ Java interface default methods will help us in extending interfaces without having the fear of breaking implementation classes.
- ▶ Java interface default methods has bridge down the differences between interfaces and abstract classes.
- ▶ Java interface default methods are also referred to as Defender Methods or Virtual extension method

STATIC METHODS

- ▶ Java interface static method is similar to default method except that we can't override them in the implementation classes.
- ▶ This feature helps us in avoiding undesired results incase of poor implementation in implementation classes

ADVANTAGES OF STATIC METHODS

- ▶ Java interface static method is part of interface, we can't use it for implementation class objects.
- ▶ Java interface static methods are good for providing utility methods, for example null check, collection sorting etc.
- ▶ Java interface static method helps us in providing security by not allowing implementation classes to override them.

NOTE

- ▶ The scope of the static method definition is within the interface only.
- ▶ If same name method is implemented in the implementation class then that method becomes a static member of that respective class.

DIFFERENCES BETWEEN ABSTRACT CLASS AND INTERFACE

- ▶ 1) Abstract class can have abstract and non-abstract methods. Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
- ▶ 2) Abstract class doesn't support multiple inheritance. Interface supports multiple inheritance.
- ▶ 3) Abstract class can have final, non-final, static and non-static variables. Interface has only static and final variables.

DIFFERENCES

- ▶ 4) Abstract class can provide the implementation of interface. Interface can't provide the implementation of abstract class.
- ▶ 5) The abstract keyword is used to declare abstract class. The interface keyword is used to declare interface.
- ▶ 6) An abstract class can extend another Java class and implement multiple Java interfaces.
An interface can extend another Java interface only.

DIFFERENCES

- ▶ 7) An abstract class can be extended using keyword "extends". An interface can be implemented using keyword "implements".
- ▶ 8) A Java abstract class can have class members like private, protected, etc.
Members of a Java interface are public by default.

PACKAGES

- ▶ A package as the name suggests is a pack(group) of classes, interfaces and other packages.
- ▶ In java we use packages to organize our classes and interfaces.
- ▶ We have two types of packages in Java: **built-in packages** and the packages we can create (also known as **user defined package**).

BUILT IN PACKAGES

- ▶ `import java.util.Date;`
- ▶ Here:
 - ▶ → java is a top level package
 - ▶ → util is a sub package
 - ▶ → and Date is a class which is present in the sub package util.

ADVANTAGES OF USING A PACKAGE

- ▶ **Reusability:**
- ▶ While developing a project in java, we often feel that there are few things that we are writing again and again in our code. Using packages, you can create such things in form of classes inside a package and whenever you need to perform that same task, just import that package and use the class.

- ▶ **Better Organization:**
- ▶ Again, in large java projects where we have several hundreds of classes, it is always required to group the similar types of classes in a meaningful package name so that you can organize your project better and when you need something you can quickly locate it and use it, which improves the efficiency.

- ▶ Name Conflicts:
- ▶ We can define two classes with the same name in different packages so to avoid name collision, we can use packages

TYPES OF PACKAGES

- ▶ As mentioned in the beginning of this guide that we have two types of packages in java.
- ▶ 1) User defined package: The package we create is called user-defined package.
- ▶ 2) Built-in package: The already defined package like `java.io.*`, `java.lang.*` etc are known as built-in packages.

► SUB PACKAGES IN JAVA

- ▶ A package inside another package is known as sub package.
- ▶ For example If I create a package inside letmecalculate package then that will be called sub package.

MULTIPLE IMPORT STATEMENTS

- ▶ A class can have only one package declaration but it can have more than one package import statements. For example:
- ▶ package abcpackage; //This should be one
- ▶ import xyzpackage;
- ▶ import anotherpackage;
- ▶ import anything;

ACCESS MODIFIERS IN JAVA

- ▶ The access modifiers in Java specifies the accessibility of field, method, constructor, or class.
- ▶ We can change the access level of fields, constructors, and class by applying the access modifier on it.
- ▶ There are four types of Java access modifiers:
- ▶ **public, private, protected, default**

- ▶ **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
- ▶ **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

ENCAPSULATION

- ▶ Encapsulation in Java is a process of wrapping up of code and data together into a single unit, called **class**
- ▶ We can create a fully encapsulated class in Java by making all the data members of the class **private**. Now we can use setter and getter methods to set and get the data in it.

String Class

- ▶ In Java, string is basically an object that represents sequence of char values.
- ▶ In java, string is an immutable object which means it is constant and cannot be changed once it has been created.
- ▶ In this video we will learn about String class and String methods in detail

Ways To Create String?

- ▶ Creating a String
- ▶ There are two ways to create a String in Java
 - ▶ String literal
 - ▶ Using new keyword

► String literal

- ▶ In java, Strings can be created like this: Assigning a String literal to a String instance:
- ▶ `String str1 = "Welcome";`
- ▶ `String str2 = "Welcome";`
- ▶ The problem with this approach: As I stated in the beginning that String is an object in Java. However we have not created any string object using new keyword above.
- ▶ The compiler does that task for us it creates a string object having the string literal (that we have provided , in this case it is “Welcome”) and assigns it to the provided string instances.

- ▶ But if the object already exist in the memory it does not create a new Object rather it assigns the same old object to the new instance, that means even though we have two string instances above(str1 and str2) compiler only created one string object (having the value “Welcome”) and assigned the same to both the instances.

▶ Using New Keyword

- ▶ As we saw above that when we tried to assign the same string object to two different literals, compiler only created one object and made both of the literals to point the same object. To overcome that approach we can create strings like this:

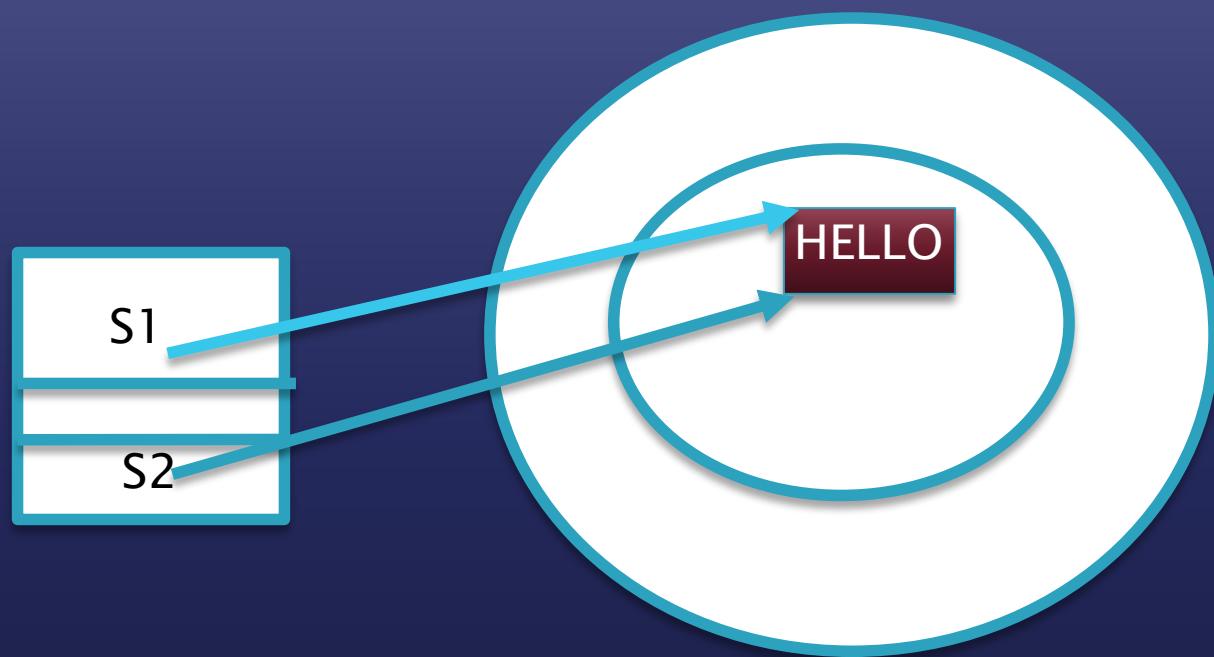
- ▶ String str1 = new String("Welcome");
- ▶ String str2 = new String("Welcome");
- ▶ In this case compiler would create two different object in memory having the same string.

Immutable

- ▶ The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created.
- ▶ For mutable strings, you can use StringBuffer and StringBuilder classes.

▶ String Literal

- ▶ Java String literal is created by using double quotes. For Example:
- ▶ String s="welcome";
- ▶ Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:
- ▶ String s1="HELLO";
- ▶ String s2="HELLO";//It doesn't create a new instance



- ▶ In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "HELLO" in string constant pool, that is why it will create a new object. After that it will find the string with the value "HELLO" in the pool, it will not create a new object but will return the reference to the same instance.

How to create mutable Strings In java?

- ▶ By using **StringBuilder** And **StringBuffer** you can create mutable Strings in java
- ▶ Why string objects are immutable in java?
- ▶ Because java uses the concept of string literal. Suppose there are 5 reference variables, all refers to one object "John".
- ▶ If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

▶ Java String compare

- ▶ java string comparison
- ▶ We can compare string in java on the basis of content and reference
- ▶ There are three ways to compare string in java:
 - ▶ 1)By equals() method
 - ▶ 2)By == operator
 - ▶ 3)By compareTo() method

String Concatenation in Java

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

► Substring in Java

- ▶ A part of string is called substring. In other words, substring is a subset of another string. In case of substring startIndex is inclusive and endIndex is exclusive.
- ▶ Note: Index starts from 0.
- ▶ You can get substring from the given string object by one of the two methods:
- ▶ `public String substring(int startIndex)`: This method returns new String object containing the substring of the given string from specified startIndex (inclusive).
- ▶ `public String substring(int startIndex, int endIndex)`: This method returns new String object containing the substring of the given string from specified startIndex to endIndex.

Methods of String Class

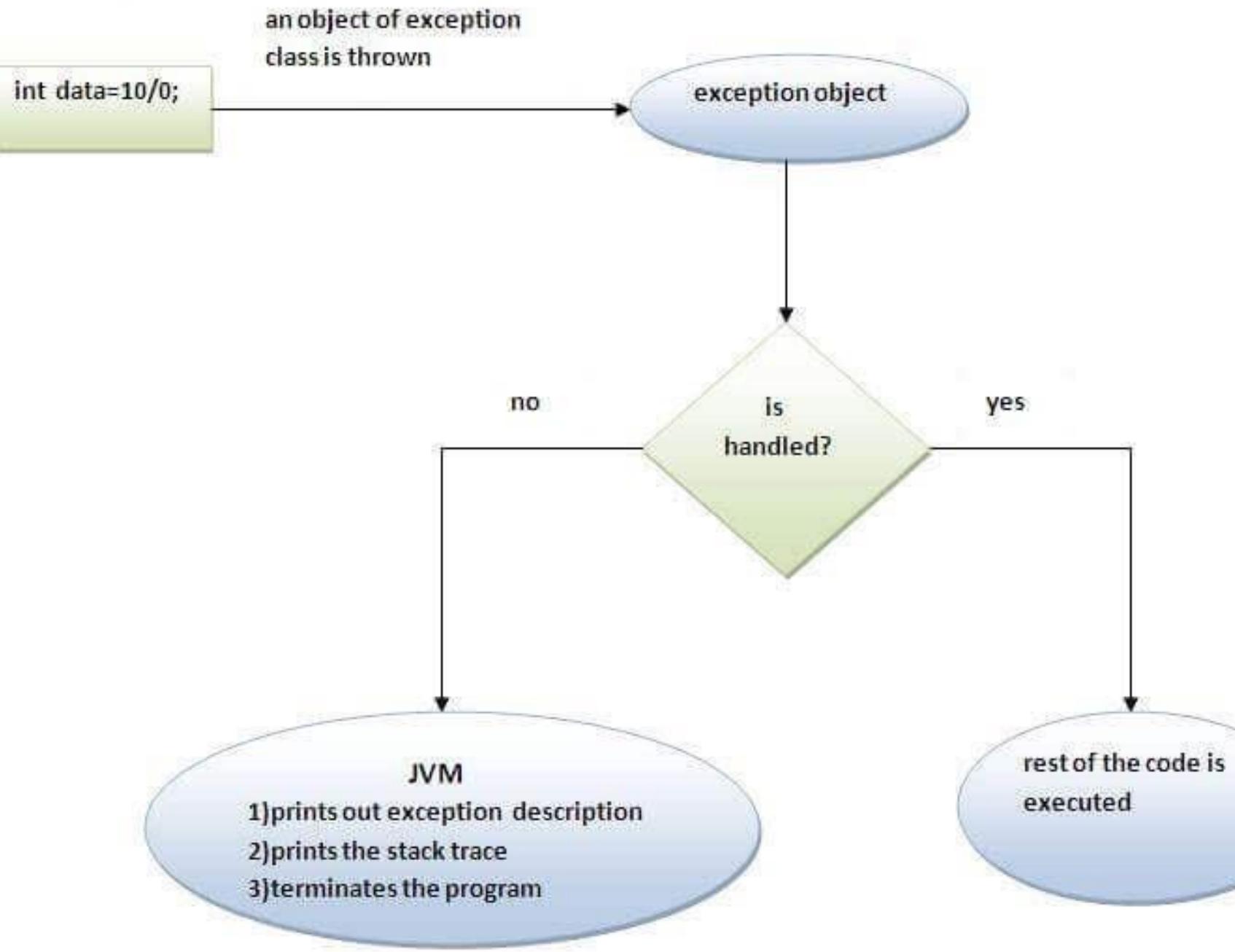
- ▶ Here are the list of the methods available in the Java String class. These methods are explained in the separate tutorials with the help of examples. Links to the tutorials are provided below:
- ▶ `char charAt(int index)`: It returns the character at the specified index. Specified index value should be between 0 to `length() - 1` both inclusive. It throws `IndexOutOfBoundsException` if `index < 0 || index >= length of String`.
- ▶ `boolean equals(Object obj)`: Compares the string with the specified string and returns true if both matches else false.
- ▶ `boolean equalsIgnoreCase(String string)`: It works same as `equals` method but it doesn't consider the case while comparing strings. It does a case insensitive comparison.
- ▶ `int compareTo(String string)`: This method compares the two strings based on the Unicode value of each character in the strings.
- ▶ `int compareIgnoreCase(String string)`: Same as `CompareTo` method however it ignores the case during comparison.
- ▶ `boolean startsWith(String prefix, int offset)`: It checks whether the substring (starting from the specified offset index) is having the specified prefix or not.
- ▶ `boolean startsWith(String prefix)`: It tests whether the string is having specified prefix, if yes then it returns true else false.
- ▶ `boolean endsWith(String suffix)`: Checks whether the string ends with the specified suffix.
- ▶ `int hashCode()`: It returns the hash code of the string.
- ▶ `int indexOf(int ch)`: Returns the index of first occurrence of the specified character ch in the string.

EXCEPTION HANDLING IN JAVA

- ▶ **WHAT IS AN EXCEPTION?**
- ▶ An Exception is an event that occurs during the execution of a program and it interrupts the normal flow of program execution.
- ▶ It is an object which is thrown at runtime.

ADVANTAGE OF EXCEPTION HANDLING

- ▶ The core advantage of exception handling is to maintain the normal flow of the application. An exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:
 - ▶ statement 1;
 - ▶ statement 2;
 - ▶ statement 3;
 - ▶ statement 4;
 - ▶ statement 5; //exception occurs
 - ▶ statement 6;
 - ▶ statement 7;
 - ▶ statement 8;
- ▶ Suppose there are 8 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 8 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.



DEFAULT EXCEPTION HANDLER

- ▶ Default Exception Handling : Whenever inside a method, if an exception has occurred, the method creates an Object known as Exception Object and hands it off to the run-time system(JVM).
- ▶ The exception object contains name and description of the exception, and current state of the program where exception has occurred.
- ▶ Creating the Exception Object and handing it to the run-time system is called throwing an Exception. There might be the list of the methods that had been called to get to the method where exception was occurred.
- ▶ This ordered list of the methods is called Call Stack. Now the following procedure will happen.

- ▶ The run-time system searches the call stack to find the method that contains block of code that can handle the occurred exception. The block of the code is called Exception handler.
- ▶ The run-time system starts searching from the method in which exception occurred, proceeds through call stack in the reverse order in which methods were called.
- ▶ If it finds appropriate handler then it passes the occurred exception to it. Appropriate handler means the type of the exception object thrown matches the type of the exception object it can handle.

- ▶ If run-time system searches all the methods on call stack and couldn't have found the appropriate handler then run-time system handover the Exception Object to default exception handler , which is part of run-time system. This handler prints the exception information in the following format and terminates program abnormally.

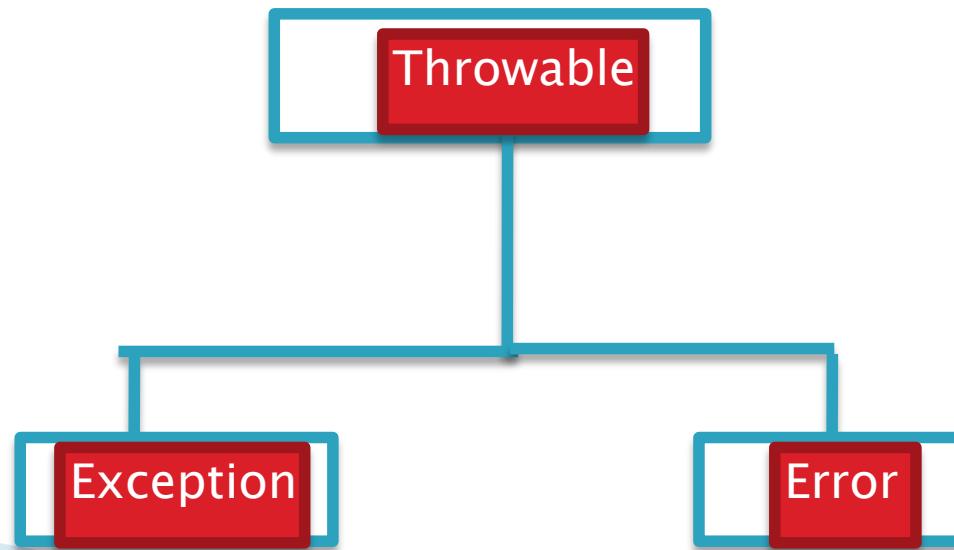
- ▶ Exception in thread "xxx" Name of Exception : Description
- ▶ // Call Stack

FINALLY BLOCK

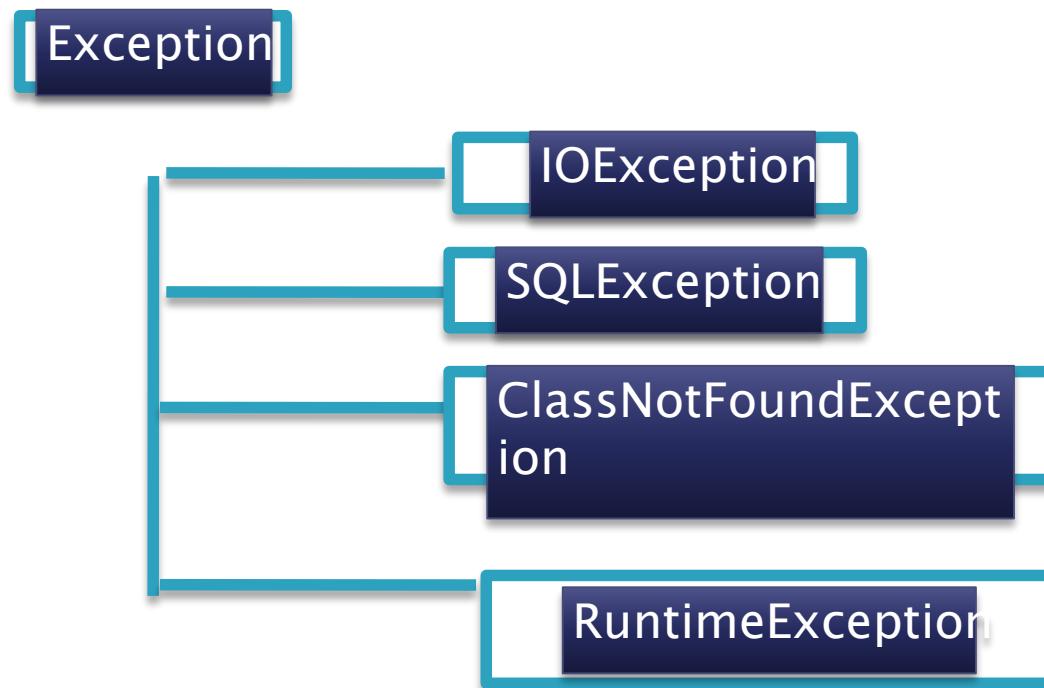
- ▶ Java finally block is a block that is used to execute important code such as closing connection, stream etc.
- ▶ Java finally block is always executed whether exception is handled or not.
- ▶ Java finally block follows try or catch block.

HIERARCHY OF JAVA EXCEPTION CLASSES

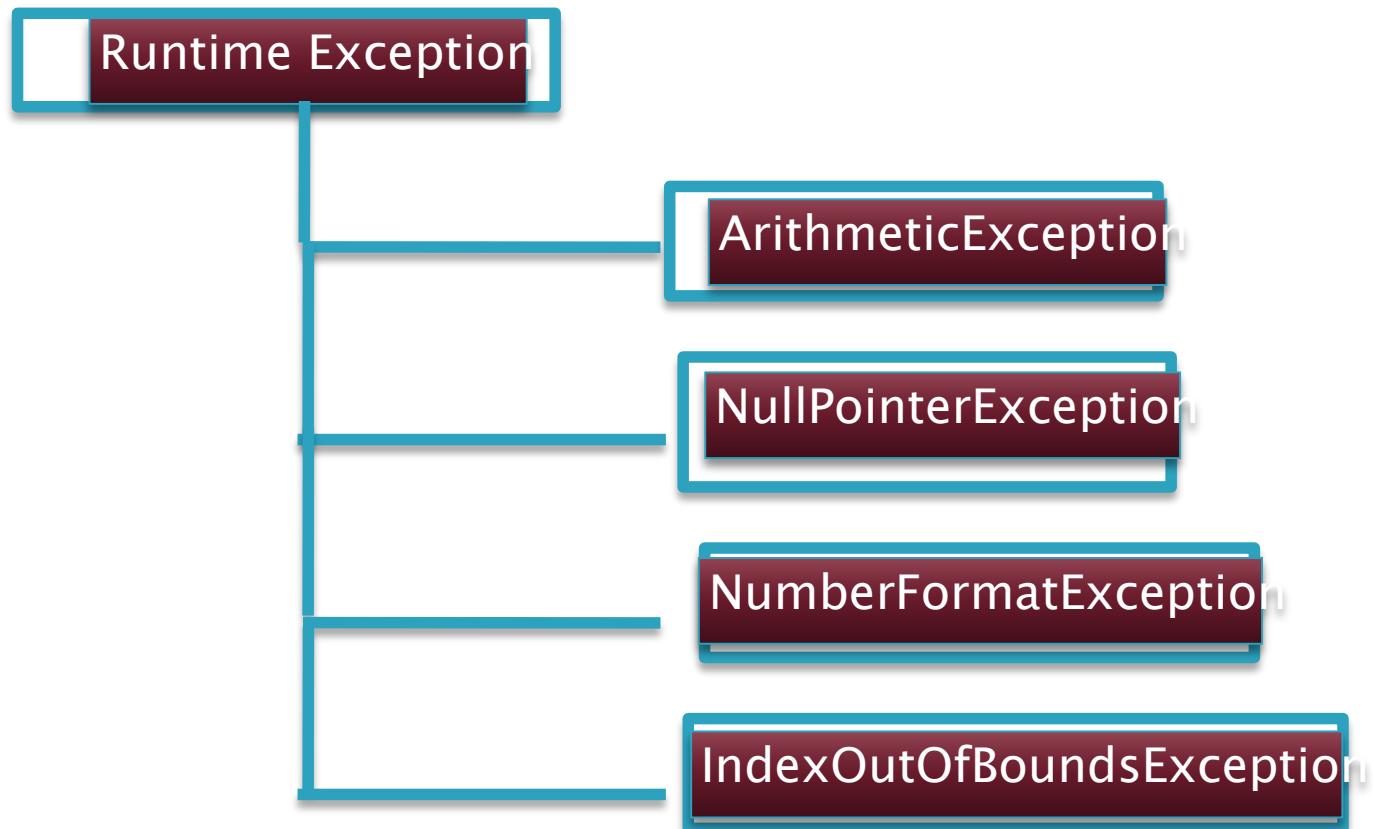
- ▶ The `java.lang.Throwable` class is the root class of Java Exception hierarchy which is inherited by two subclasses:
 - ▶ **Exception and Error.**



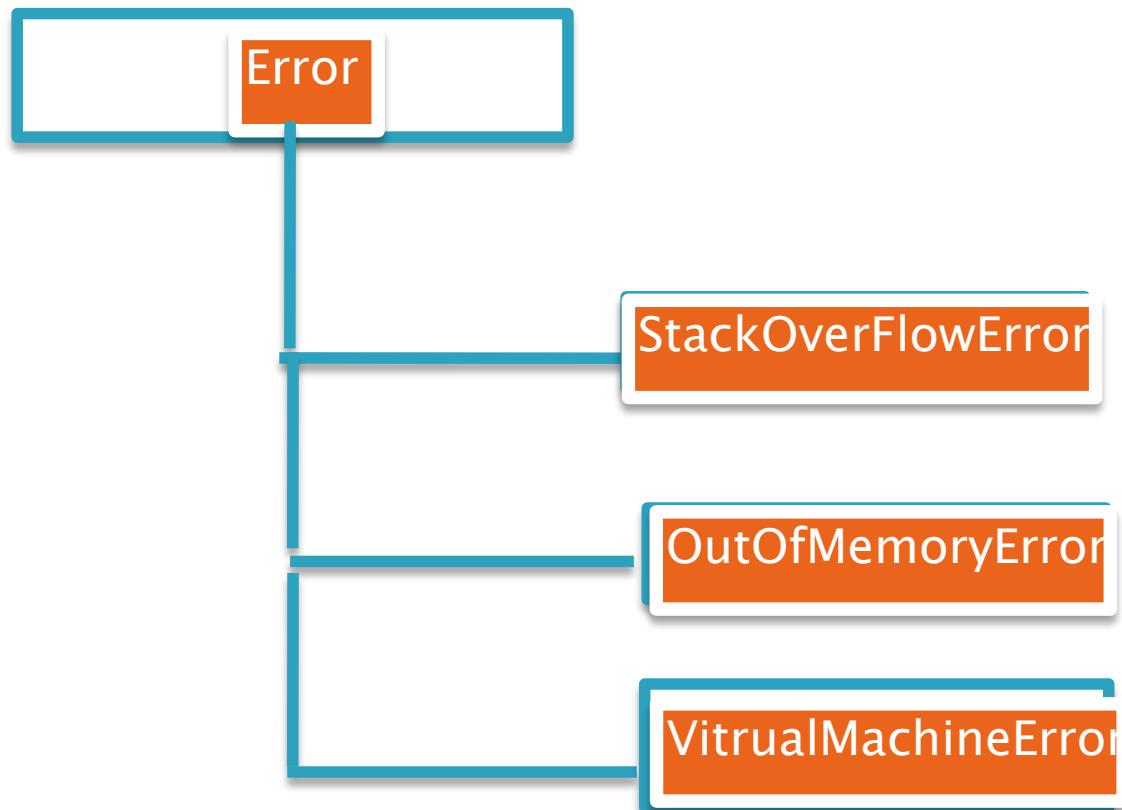
EXCEPTION



RUNTIMEEXCEPTION



ERROR



TYPES OF EXCEPTIONS

- ▶ There are mainly two types of exceptions: **checked and unchecked**. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:
 - ▶ Checked Exception
 - ▶ Unchecked Exception
 - ▶ Error

- ▶ Difference between Checked and Unchecked Exceptions
- ▶ 1) Checked Exception
- ▶ The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. **Checked exceptions are checked at compile-time.**
- ▶ 2) Unchecked Exception
- ▶ The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. **Unchecked exceptions are not checked at compile-time, but they are checked at runtime.**

KEYWORDS

- ▶ **try:** The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
- ▶ **catch:** The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.

FIVE KEYWORDS

- ▶ **finally** The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
- ▶ **throw** The "throw" keyword is used to throw an exception.
- ▶ **throws** The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Java try–catch block

- ▶ Syntax of Java try–catch
- ▶ try{
- ▶ //code that may throw an exception
- ▶ }catch(Exception_class_Name ref){}
- ▶ Syntax of try–finally block
- ▶ try{
- ▶ //code that may throw an exception
- ▶ }finally{}

THROWS KEYWORD

- ▶ The Java throws keyword is used to declare an exception.
- ▶ It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.
- ▶ .

THROWS KEYWORD USES

- ▶ **throws** is a keyword used to propagate the exceptions to the caller method by specifying at method level.
- ▶ You can define any checked or unchecked exceptions at method level.
- ▶ You can define any built-in or user defined exceptions at method level.
- ▶ When the exception is unchecked, then throws keyword is optional. But for checked exception throws keyword is mandatory.

SYNTAX OF THROWS KEYWORD

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

JAVA CATCH BLOCK

- ▶ Java catch block is used to handle the Exception by declaring the type of exception within the parameter.
- ▶ The declared exception must be the parent class exception (i.e., Exception) or the generated exception type. However, the good approach is to declare the generated type of exception.
- ▶ The catch block must be used after the try block only. You can use multiple catch block with a single try block.

MULTICATCH BLOCK

- ▶ A try block can be followed by one or more catch blocks.
- ▶ Each catch block must contain a different exception handler.
- ▶ So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

POINTS TO NOTE

- ▶ At a time only one exception occurs and at a time only one catch block is executed.
- ▶ All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

FINALLY BLOCK

- ▶ A finally block contains all the crucial statements that must be executed whether exception occurs or not.
- ▶ The statements present in this block will always execute regardless of whether exception occurs in try block or not such as closing a connection, stream etc.

ONE EXAMPLE

- ▶ Suppose you opened a database connection
- ▶ The code inside the try block will be
 - ▶ try
 - ▶ {
 - ▶ open dbconnectoin;
 - ▶ read data;
 - ▶ close dbconnection
 - ▶ }
- ▶ If an exception happens while reading data
 - ▶ close dbeconnection would be skipped

WHY TO USE FINALLY?

- ▶ try
- ▶ {
- ▶ open deconnection;
- ▶ read data;
- ▶ }catch(Exception e)
- ▶ {
- ▶ handle exception ;
- ▶ close deconnection;
- ▶ }//the disadvantage is that if no exception happens then catch block is skipped

CLEAN UP CODE IN FINALLY

```
▶ try
▶ {
▶   open dbconnection;
▶   read data;
▶ }catch(Exception e)
▶ {
▶   handle exception
▶ }

▶ finally
▶ {close dbconnection;
}
```

SYNTAX OF FINALLY

```
▶ try {  
▶   //Statements that may cause an exception  
▶ }  
▶ catch {  
▶   //Handling exception  
▶ }  
▶ finally {  
▶   //Statements to be executed  
▶ }
```

FINALLY BLOCK AND SYSTEM.EXIT(0)

- ▶ whenever System.exit() gets called in try block then Finally block doesn't execute. Here is a code snippet that demonstrate the same:

```
▶ ....  
▶ try {  
▶   //try block  
▶   System.out.println("Inside try block");  
▶   System.exit(0)  
▶ }  
▶ catch (Exception exp) {  
▶   System.out.println(exp);  
▶ }  
▶ finally {  
▶   System.out.println("Java finally block");  
▶ }
```

TRY CATCH FINALLY BLOCK

- ▶ Either a try statement should be associated with a catch block or with finally.
- ▶ Since catch performs exception handling and finally performs the cleanup, the best approach is to use both of them.

USAGES OF FINALLY

- ▶ **Case 1**
- ▶ where exception doesn't occur.
- ▶ **Case 2**
- ▶ where exception occurs and not handled
- ▶ **Case 3**
- ▶ where exception occurs and handled.

FEW IMPORTANT POINTS

- ▶ A finally block must be associated with a try block, you cannot use finally without a try block. You should place those statements in this block that must be executed always.
- ▶ 2. Finally block is optional, as we have seen in previous tutorials that a try–catch block is sufficient for exception handling, however if you place a finally block then it will always run after the execution of try block.
- ▶ 3. In normal case when there is no exception in try block then the finally block is executed after try block. However if an exception occurs then the catch block is executed before finally block.
- ▶ 4. An exception in the finally block, behaves exactly like any other exception.
- ▶ 5. The statements present in the finally block execute even if the try block contains control transfer statements like return, break or continue.

ADVANTAGES

- ▶ By providing only a setter or getter method, you can make the class read-only or write-only. In other words, you can skip the getter or setter methods.
- ▶ It is a way to achieve data hiding in Java because other class will not be able to access the data through the private data members.
- ▶ The encapsulate class is easy to test. So, it is better for unit testing.

JAVA THROW KEYWORD

- ▶ The Java throw keyword is used to explicitly throw an exception.
- ▶ We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.
- ▶ The syntax of java throw keyword is given below.
- ▶ `throw new IOException("sorry device error");`

EXCEPTION PROPAGATION

- ▶ An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method, If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack.
- ▶ This is called exception propagation.

JAVA THROWS KEYWORD

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

SYNTAX

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

► ADVANTAGE OF THROWS

- ▶ Now Checked Exception can be propagated (forwarded in call stack).
- ▶ It provides information to the caller of the method about the exception.

Difference between throw and throws in Java

- ▶ 1) Java **throw** keyword is used to explicitly throw an exception.
- ▶ Java **throws** keyword is used to declare an exception.
- ▶ 2) Checked exception cannot be propagated using **throw** only.
- ▶ Checked exception can be propagated with **throws**.

► Java Custom Exception

- ▶ If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.
 - ▶ By the help of custom exception, you can have your own exception and message.
- ▶ Let's see a simple example of java custom exception.

Difference between throw and throws

- ▶ 3) **Throw** is used within the method.
Throws is used with the method signature.
- ▶ 4) You cannot throw multiple exceptions.
You can declare multiple exceptions e.g.
- ▶ `public void method()throws
IOException,SQLException.`

- ▶ **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
- ▶ **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

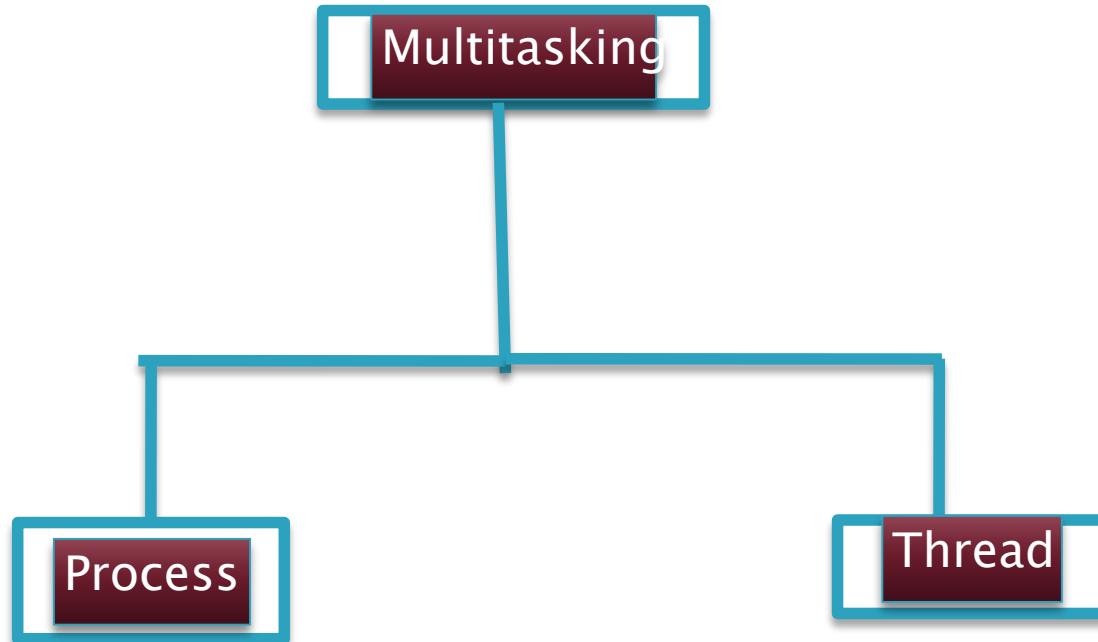
class	package	subclass same package	subclass different package	outside class
public	yes	yes	yes	yes
private	yes	no	no	no
protected	yes	yes	yes	no
default	yes	yes	yes	no

▶ Java Access Modifiers with Method Overriding

- ▶ If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

MULTITASKING

- ▶ Multitasking is a process of executing multiple tasks simultaneously.
- ▶ We use multitasking to eliminate the idle time of the CPU.
- ▶ Multitasking can be achieved in two ways:
 - ▶ Process-based Multitasking (Multiprocessing)
 - ▶ Thread-based Multitasking (Multithreading)



WHAT IS MULTIPROCESSING?

- ▶ When two or more programs run concurrently
- ▶ that is called as Multiprocessing
- ▶ when two or more independents portions of a program run concurrently that is called as MultiThreading

What is Process?

- ▶ A process is a program in execution.
- ▶ A process is an execution environment that consists of instructions, user-data, and system-data segments, as well as lots of other resources such as CPU, memory, address-space, disk and network I/O acquired at runtime

Thread

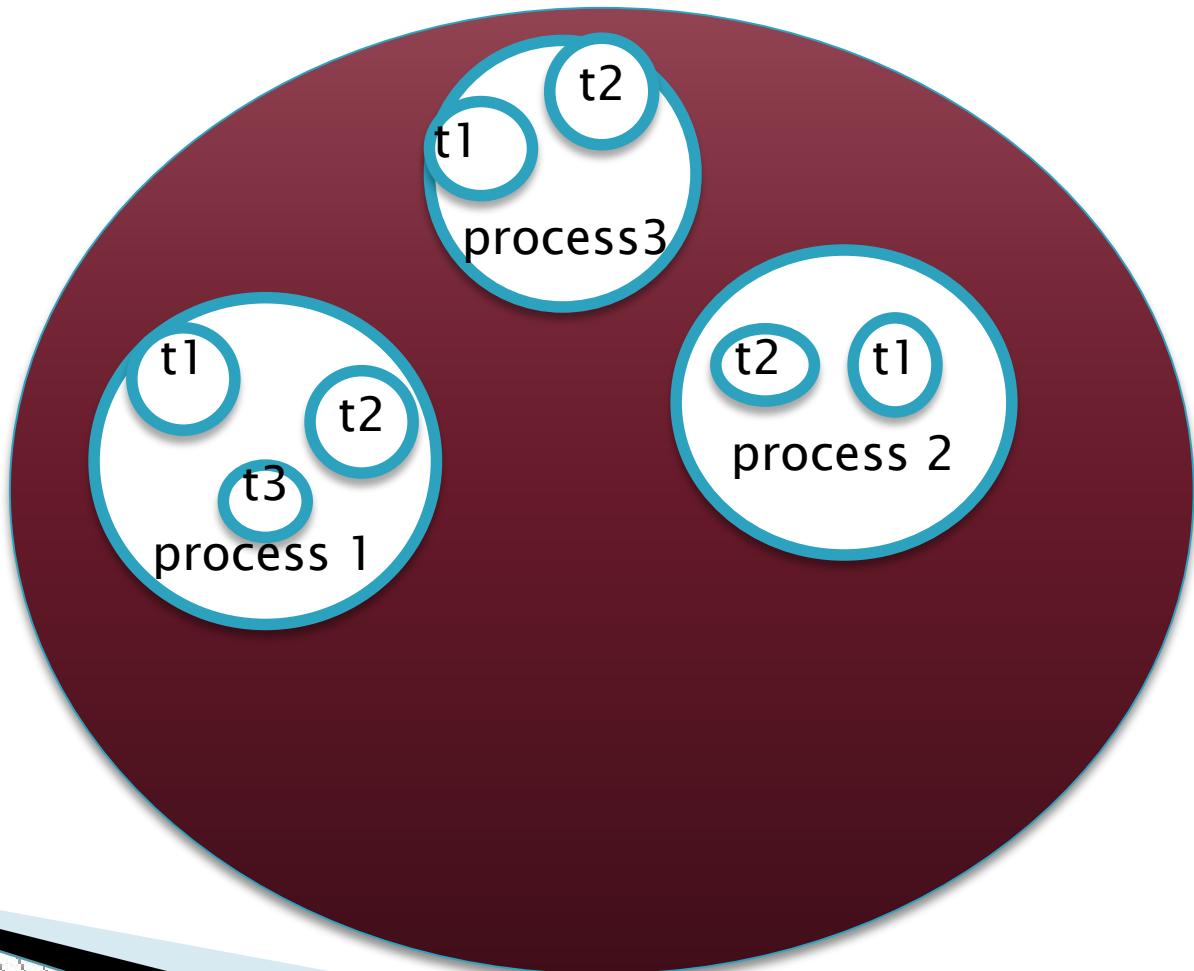
- ▶ Thread is the smallest unit of execution in a process.
- ▶ A thread simply executes instructions serially. A process can have multiple threads running as part of it.
- ▶ Usually, there would be some state associated with the process that is shared among all the threads and in turn each thread would have some state private to itself.

Characteristics of a process

- ▶ Each process has an address in memory. In other words, each process allocates a separate memory area.
- ▶ A process is heavyweight.
- ▶ Cost of communication between the process is high.
- ▶ Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

Thread Characteristics

- ▶ Threads share the same address space.
- ▶ A thread is lightweight.
- ▶ Cost of communication between the thread is low.
- ▶ Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.



- ▶ As shown in the above figure, a thread is executed inside the process.
- ▶ There IS context-switching between the threads.
- ▶ There can be multiple processes inside the OS, and one process can have multiple threads.

USAGE OF MULTITHREADING

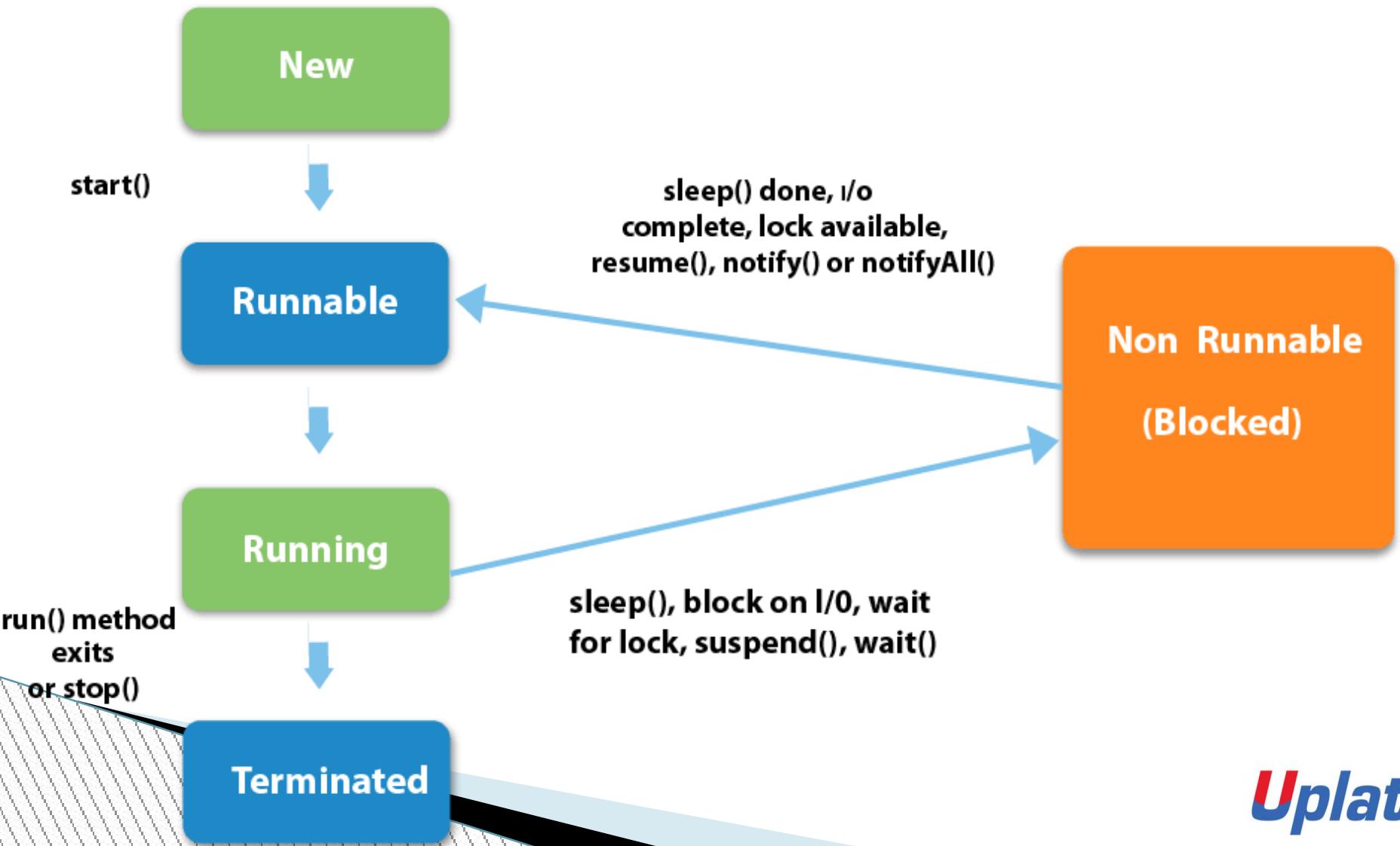
- ▶ Multimedia Graphics
- ▶ Animation pictures
- ▶ Video games
- ▶ Webserver's

Java Thread class

- ▶ Java provides Thread class to achieve thread programming.
- ▶ Thread class provides constructors and methods to create and perform operations on a thread.
- ▶ Thread class extends Object class and implements Runnable interface.

Life cycle of a Thread

- ▶ A thread can be in one of the five states. According to sun, there is only 4 states in thread life cycle in java new, runnable, non-runnable and terminated. There is no running state.
- ▶ But for better understanding the threads, we are explaining it in the 5 states.
- ▶ The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:
 - ▶ New
 - ▶ Runnable
 - ▶ Running
 - ▶ Non-Runnable (Blocked)
 - ▶ Terminated



Thread States

- ▶ 1) New
 - ▶ The thread is in new state if you create an instance of Thread class but before the invocation of start() method.
- ▶ 2) Runnable
 - ▶ The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.
- ▶ 3) Running
 - ▶ The thread is in running state if the thread scheduler has selected it.

Thread States

- ▶ 4) Non-Runnable (Blocked)
 - ▶ This is the state when the thread is still alive, but is currently not eligible to run.
- ▶ 5) Terminated
 - ▶ A thread is in terminated or dead state when its run() method exits.

Characteristics of Process

- ▶ Each process has an address in memory. In other words, each process allocates a separate memory area.
- ▶ A process is heavyweight.
- ▶ Cost of communication between the process is high.
- ▶ Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

How to create thread

- ▶ There are two ways to create a thread:
- ▶ By extending Thread class
- ▶ By implementing Runnable interface.

▶ Thread class:

- ▶ Thread class provide constructors and methods to create and perform operations on a thread.
- ▶ Thread class extends Object class and implements Runnable interface.
- ▶ Thread()
- ▶ Thread(String name)
- ▶ Thread(Runnable r)
- ▶ Thread(Runnable r, String name)

▶ Commonly used Constructors of Thread class:

- ▶ Thread()
- ▶ Thread(String name)
- ▶ Thread(Runnable r)
- ▶ Thread(Runnable r, String name)

▶ Thread Scheduler in Java

- ▶ Thread scheduler in java is the part of the JVM that decides which thread should run.
- ▶ There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.
- ▶ Only one thread at a time can run in a single process.

► Sleep method in java

- ▶ The sleep() method of Thread class is used to sleep a thread for the specified amount of time.
- ▶ Syntax of sleep() method in java
- ▶ The Thread class provides two methods for sleeping a thread:
- ▶ `public static void sleep(long miliseconds) throws InterruptedException`
- ▶ `public static void sleep(long miliseconds, int nanos) throws InterruptedException`

- ▶ What if we call run() method directly instead start() method?
- ▶ Each thread starts in a separate call stack.
- ▶ Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.

▶ Can we start a thread twice

- ▶ No. After starting a thread, it can never be started again. If you does so, an `IllegalThreadStateException` is thrown. In such case, thread will run once but for second time, it will throw exception.
- ▶ Let's understand it by the example given below:

Naming Thread and Current Thread

- ▶ The Thread class provides methods to change and get the name of a thread.
- ▶ By default, each thread has a name i.e. thread-0, thread-1 and so on.
- ▶ By we can change the name of the thread by using setName() method.
- ▶ The syntax of setName() and getName() methods are given below:
 - ▶ public String getName(): is used to return the name of a thread.
 - ▶ public void setName(String name): is used to change the name of a thread.

Current Thread

- ▶ The `currentThread()` method returns a reference of currently executing thread.
- ▶ `public static Thread currentThread()`

- ▶ Priority of a Thread (Thread Priority):
- ▶ Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread scheduler schedules the threads according to their priority (known as preemptive scheduling).
- ▶ But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.
- ▶ .

- ▶ 3 constants defined in Thread class

- ▶ :
- ▶ public static int MIN_PRIORITY
- ▶ public static int NORM_PRIORITY
- ▶ public static int MAX_PRIORITY
- ▶ Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10

The join() method

- ▶ The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.
- ▶ Syntax:
- ▶ `public void join()throws InterruptedException`
- ▶ `public void join(long milliseconds)throws InterruptedException`

Daemon Thread in Java

- ▶ Daemon thread in java is a service provider thread that provides services to the user thread.
- ▶ Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.
- ▶ There are many java daemon threads running automatically e.g. gc, finalizer etc.

- ▶ Points to remember for Daemon Thread in Java
- ▶ It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
- ▶ Its life depends on user threads.
- ▶ It is a low priority thread.

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

► Synchronization in java

- ▶ Since java is a multi-threaded language so, when two or more thread uses a shared resources that may lead to **two kind of errors: thread interference and memory consistency error**, to avoid this error you need to synchronize the object, that the resource will be used by one thread at a time and the process by which this is achieved is called **synchronization**.

- ▶ To put it in simple words when multiple threads have access to a shared resource
- ▶ we should ensure that only one thread uses the resource at a time.
- ▶ Otherwise this will create inconsistency issues

▶ Thread Synchronization

- ▶ There are two types of thread synchronization mutual exclusive and inter-thread communication.
- ▶ 1) **Mutual Exclusive**
- ▶ Synchronized method.
- ▶ Synchronized block.
- ▶ static synchronization.
- ▶ 2)**Cooperation** (Inter-thread communication in java)

▶ Mutual Exclusive

- ▶ Mutual Exclusive helps keep threads from interfering with one another while sharing data.
- ▶ This can be done by three ways in java:

- ▶ **by synchronized method**
- ▶ **by synchronized block**
- ▶ **by static synchronization**

▶ Concept of Lock in Java

- ▶ Synchronization is built around an internal entity known as the lock or monitor.
- ▶ Every object has an lock associated with it.
- ▶ By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.
- ▶ From Java 5 the package `java.util.concurrent.locks` contains several lock implementations.

- ▶ when a thread calls a synchronized method it is said to enter a locked state.
- ▶ until this thread exits the method no other thread is allowed to call not only this method
- ▶ but any other synchronised method on the same object

- ▶ **Java synchronized method**
- ▶ If you declare any method as synchronized, it is known as synchronized method.
- ▶ Synchronized method is used to lock an object for any shared resource.
- ▶ When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

▶ Synchronized Block in Java

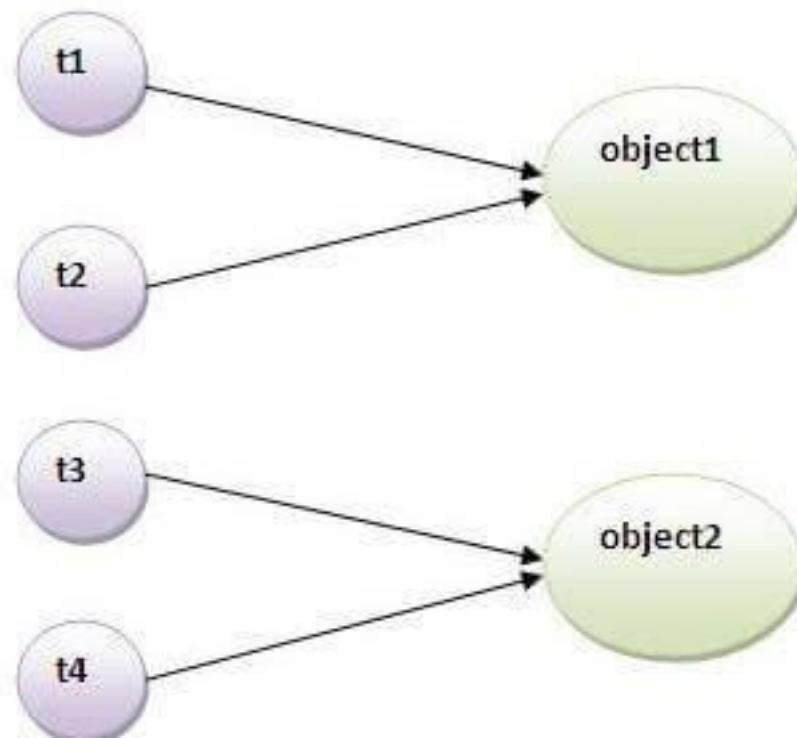
- ▶ Synchronized block can be used to perform synchronization on any specific resource of the method.
 - ▶ Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.
 - ▶ If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.
-
- ▶ Points to remember for Synchronized block
 - ▶ Synchronized block is used to lock an object for any shared resource.
 - ▶ Scope of synchronized block is smaller than the method.

► Syntax to use synchronized block

- ▶ `synchronized (object reference expression) {`
- ▶ `//code block`
- ▶ `}`

Static Synchronization

- If you make any static method as synchronized, the lock will be on the class not on object.



► Deadlock in java

- ▶ Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread.
- ▶ Since, both threads are waiting for each other to release the lock, the condition is called deadlock.

Inter-thread communication in Java

- ▶ .It is implemented by following methods of Object class:
 - ▶ wait()
 - ▶ notify()
 - ▶ notifyAll()
- ▶ These methods are kept in the object class
- ▶ so that threads are able to call them on any java object

▶ **wait() method**

- ▶ 1) Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
- ▶ The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

NOTE

- ▶ These methods must be called from the synchronised area only otherwise they will throw an IllegalMonitorStateException
- ▶ whenever the wait() is called the thread releases its lock and goes to a waiting state
- ▶ Then after getting notified it joins the runnable pool

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

Syntax

- ▶ `public final void wait()throws InterruptedException` //waits until object is notified.
- ▶ `public final void wait(long timeout)throws InterruptedException` //waits for the specified amount of time.

- ▶ 2) notify() method
- ▶ Wakes up a single thread that is waiting on this object's monitor.
- ▶ If any threads are waiting on this object, one of them is chosen to be awakened.
- ▶ The choice is arbitrary and occurs at the discretion of the implementation.
- ▶ Syntax: public final void notify()

- ▶ **notifyAll() method**
- ▶ Wakes up all threads that are waiting on this object's monitor. Syntax:
- ▶ `public final void notifyAll()`

Collections in Java

- ▶ When group of objects has to be treated as single entity we should go for Collections Framework
- ▶ Java provides Collection Framework which defines several classes and interfaces to represent a group of objects as a single unit.

ROOT INTERFACES

- ▶ The Collection interface (**java.util.Collection**) and Map interface (**java.util.Map**) are the two main “root” interfaces of Java collection classes.

- ▶ Need for Collection Framework :
- ▶ Before Collection Framework (or before JDK 1.2) was introduced, the standard methods for grouping Java objects (or collections) were Arrays or Vectors or Hashtables.
- ▶ All of these collections had no common interface.
- ▶ Accessing elements of these Data Structures was a hassle as each had a different method (and syntax) for accessing its members:

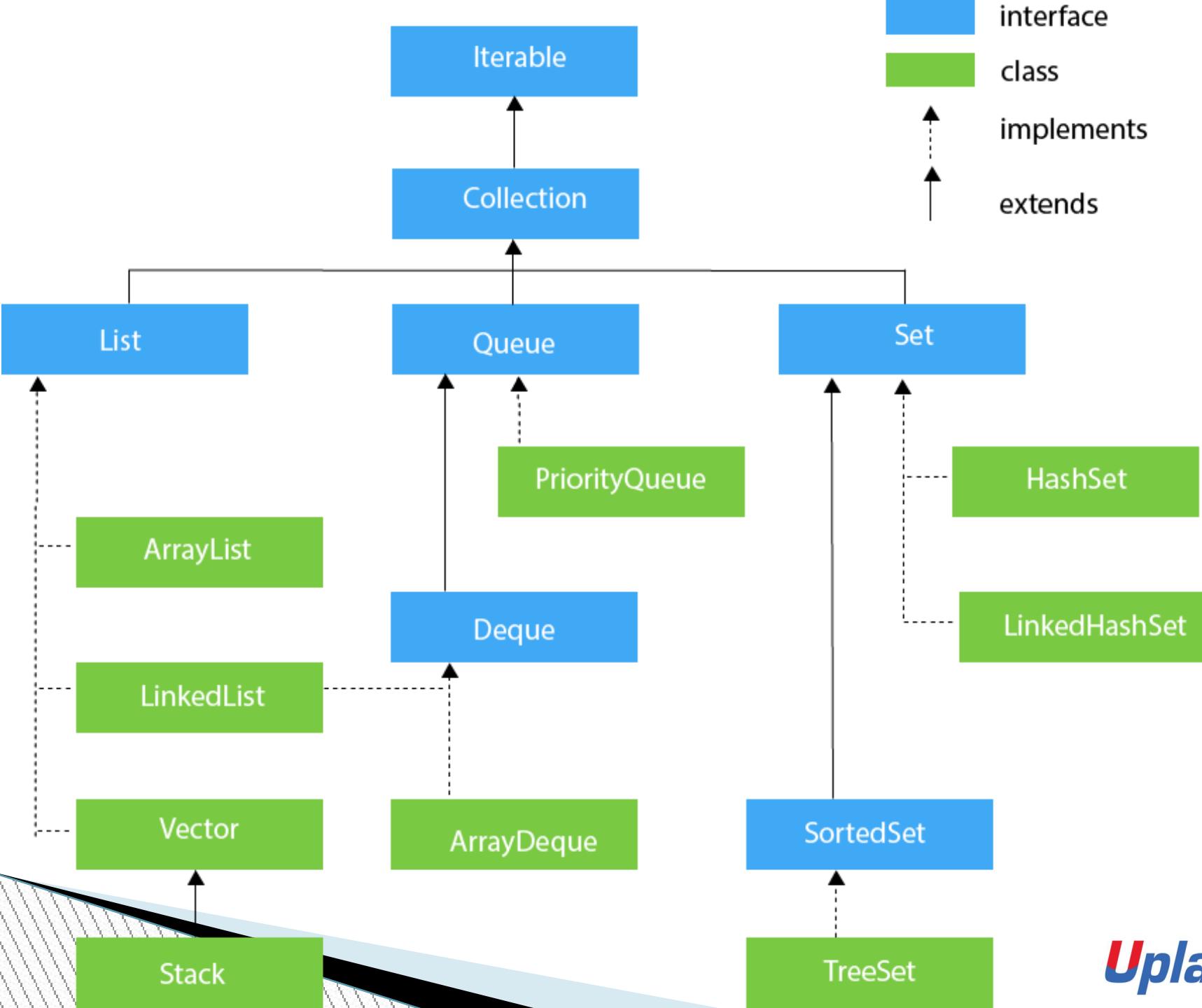
NO COMMON METHODS

```
▶ class Test
▶ {
▶   public static void main (String[] args)
▶   {
▶     // Creating instances of array, vector and hashtable
▶     int arr[] = new int[] {1, 2, 3, 4};
▶     Vector<Integer> v = new Vector();
▶     Hashtable<Integer, String> h = new Hashtable();
▶     v.addElement(1);
▶     v.addElement(2);
▶     h.put(1,"geeks");
▶     h.put(2,"4geeks");
▶
▶     // Array instance creation requires [], while Vector
▶     // and hashtable require ()
▶     // Vector element insertion requires addElement(), but
▶     // hashtable element insertion requires put()
▶
▶     // Accessing first element of array, vector and hashtable
▶     System.out.println(arr[0]);
▶     System.out.println(v.elementAt(0));
▶     System.out.println(h.get(1));
▶
▶     // Array elements are accessed using [], vector elements
▶     // using elementAt() and hashtable elements using get()
▶   }
}
```

- ▶ As we can see, none of these collections (Array, Vector or Hashtable) implement a standard member access interface.
- ▶ It was very difficult for programmers to write algorithms that can work for all kinds of Collections. Another drawback being that most of the ‘Vector’ methods are final, meaning we cannot extend the ‘Vector’ class to implement a similar kind of Collection.
- ▶ Java developers decided to come up with a common interface to deal with the above mentioned problems and introduced the Collection Framework in JDK 1.2.

What is Framework?

- ▶ It provides readymade architecture.
- ▶ It represents a set of classes and interfaces.
- ▶ It is optional.
- ▶ **What is Collection framework?**
- ▶ The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:
 - ▶ Interfaces and its implementations, i.e., classes
 - ▶ Algorithm



Nine key Interfaces

- ▶ Collection
- ▶ List
- ▶ Set
- ▶ SortedSet
- ▶ NavigableSet
- ▶ Queue
- ▶ Map
- ▶ SortedMap
- ▶ NavigableMap

Collection

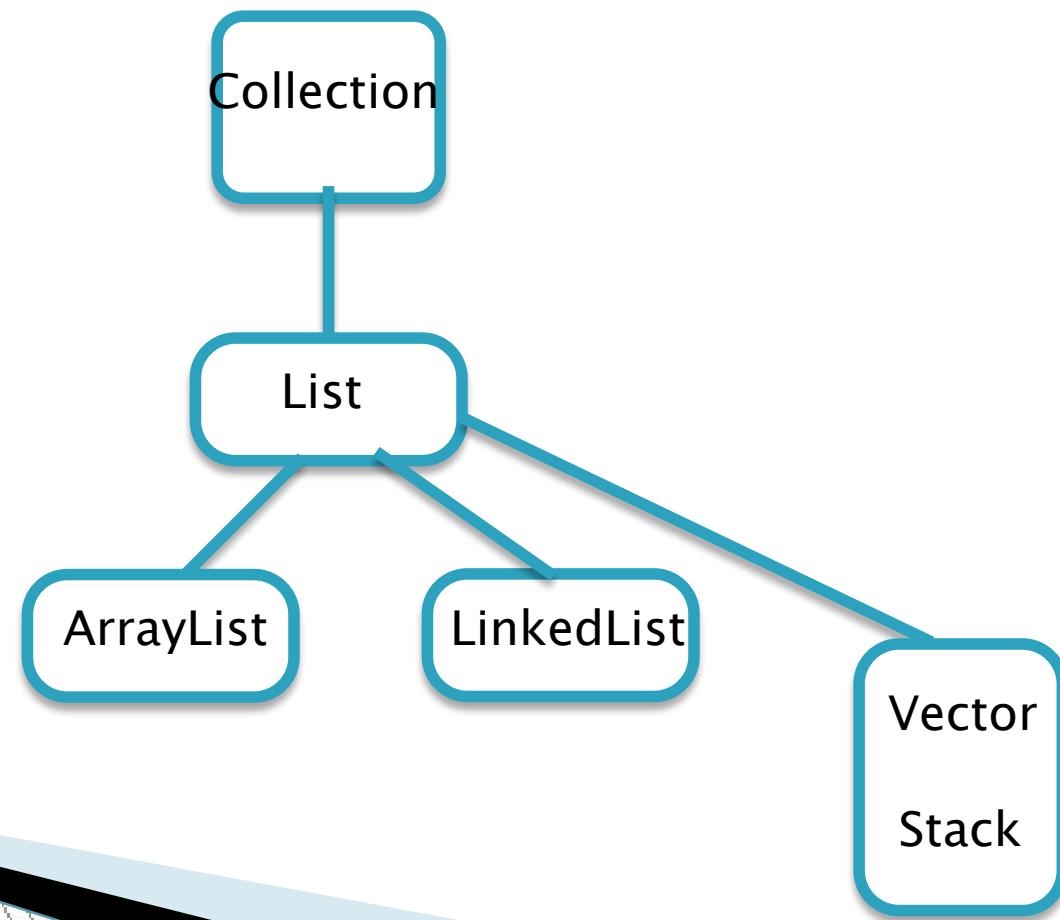
- ▶ Collection(I): If we want to represent a group of individual objects as a single entity then we should go for collection.
- ▶ **Collection interface** define the most common methods which are applicable for any collection object.
- ▶ In general collection interface is consider as root interface of collection framework.
- ▶ There is no concrete class which implements collection interface directly

▶ Difference between collection and collections

- ▶ Collection is an interface.
- ▶ If we want to represent a group of individual object as a single entity then we should go for collection
- ▶ Collections is an utility class present in `java.util` package to define several utility method for collection object(like sorting, searching etc)

LIST INTERFACE

- ▶ It is the child interface of collection.
- ▶ If we want to represent a group of individual objects as a single entity where duplicates are allowed and insertion order must be preserved then we should go for List.



- ▶ List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.
- ▶ To instantiate the List interface, we must use :
 - ▶ List <data-type> list1 = new ArrayList();
 - ▶ List <data-type> list2 = new LinkedList();
 - ▶ List <data-type> list3 = new Vector();
 - ▶ List <data-type> list4 = new Stack();

DISADVANTAGES OF ARRAYS

- ▶ The size of the array is fixed at the time of creating the array
- ▶ We can create arrays storing primitive types as well as arrays storing objects
- ▶ But while storing data inside arrays allows only homogeneous data inside them
- ▶ You cannot add extra elements to the array after declaring its size

ARRAYLIST DYNAMIC ARRAY

- ▶ Using **ArrayList** Class we can create **dynamic** arrays in java
- ▶ It **inherits AbstractList class and implements List interface.**

- ▶ The important points about Java ArrayList class are:
- ▶ Java ArrayList class can contain duplicate elements.
- ▶ Java ArrayList class maintains insertion order.
- ▶ Java ArrayList class is non synchronized.
- ▶ Java ArrayList allows random access because array works at the index basis.
- ▶ In Java ArrayList class, manipulation is slow because a lot of shifting needs to occur if any element is removed from the array list.

ArrayList Declaration

- ▶ `public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable`

- ▶ Ways to iterate the elements of the collection in java
- ▶ There are various ways to traverse the collection elements:
 - ▶ By Iterator interface.
 - ▶ By for-each loop.
 - ▶ By ListIterator interface.
 - ▶ By for loop.

▶ Java ArrayList class

- ▶ Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.
- ▶ The important points about Java ArrayList class are:
 - ▶ Java ArrayList class can contain duplicate elements.
 - ▶ Java ArrayList class maintains insertion order.
 - ▶ Java ArrayList class is non synchronized.
 - ▶ Java ArrayList allows random access because array works at the index basis.
 - ▶ In Java ArrayList class, manipulation is slow because a lot of shifting needs to occur if any element is removed from the array list.

LINKEDLIST CLASS

- ▶ Java LinkedList class uses a doubly linked list as the underlying data structure
- ▶ It inherits the AbstractList class and implements List and Deque interfaces.

Properties

- ▶ Java LinkedList class can contain duplicate elements.
- ▶ Java LinkedList class maintains insertion order.
- ▶ Java LinkedList class is non synchronized.
- ▶ In Java LinkedList class, manipulation is fast because no shifting needs to occur.
- ▶ Java LinkedList class can be used as a list, stack or queue.

LINKEDLIST DECLARATION

- ▶ `public class LinkedList<E> extends AbstractSequentialList<E> implements List<E>, Deque<E>, Cloneable, Serializable`

▶ Constructors of Java LinkedList

- ▶ Constructor Description
- ▶ `LinkedList()` It is used to construct an empty list.
- ▶ `LinkedList(Collection<? extends E> c)` It is used to construct a list containing the elements of the specified collection, in the order, they are returned by the collection's iterator.

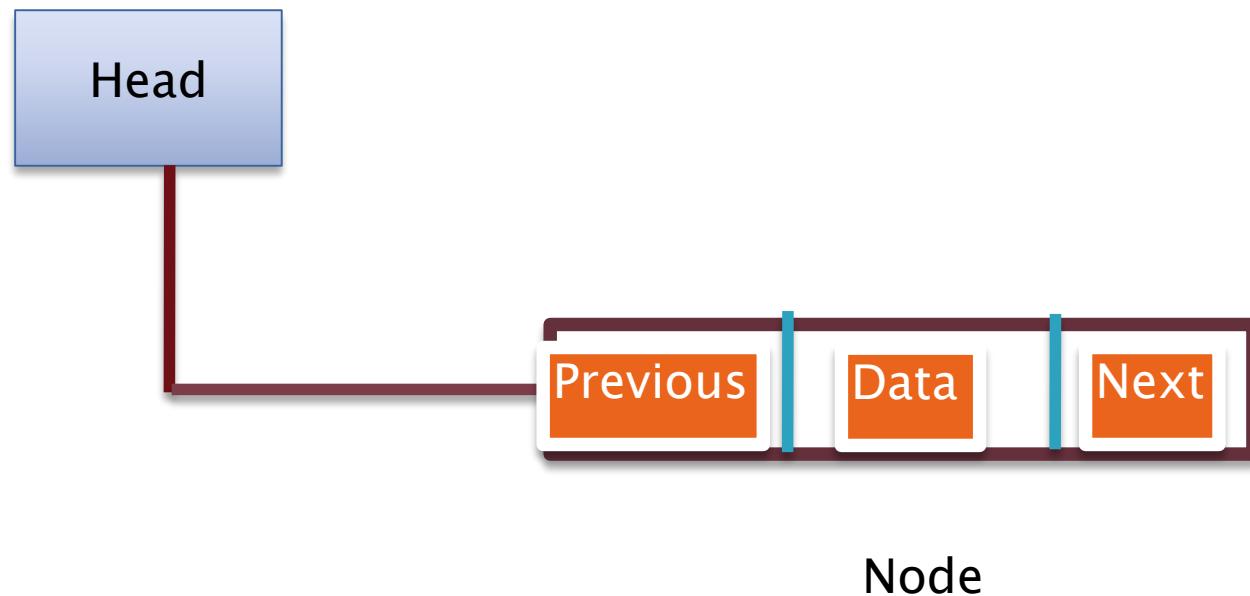
► Why do we need a Linked List?

- ▶ You must be aware of the arrays which is also a linear data structure but arrays have certain limitations such as:
- ▶ 1) Size of the array is fixed which is decided when we create an array so it is hard to predict the number of elements in advance, if the declared size fall short then we cannot increase the size of an array and if we declare a large size array and do not need to store that many elements then it is a waste of memory.
- ▶ 2) Array elements need contiguous memory locations to store their values.

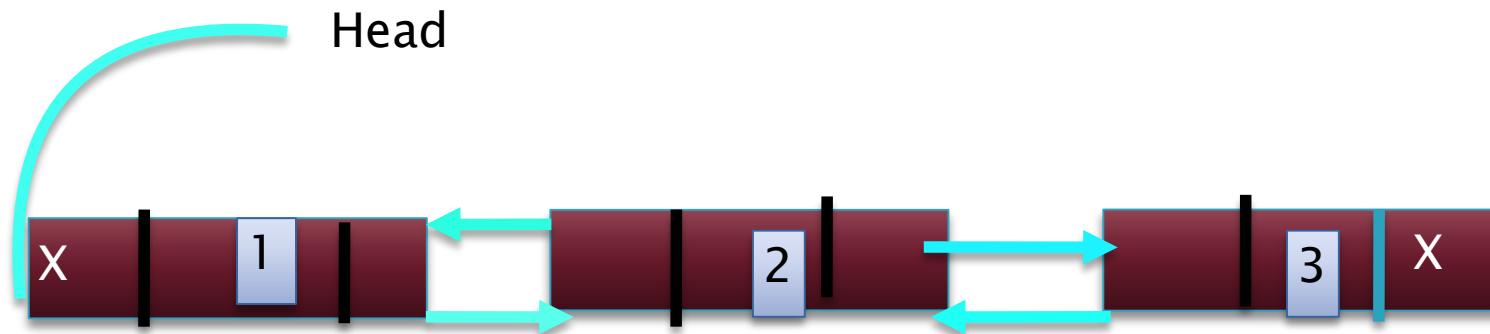
- ▶ 3) Inserting an element in an array is performance wise expensive as we have to shift several elements to make a space for the new element. For example:
- ▶ Let's say we have an array that has following elements: 10, 12, 15, 20, 4, 5, 100, now if we want to insert a new element 99 after the element that has value 12 then we have to shift all the elements after 12 to their right to make space for new element.
- ▶ Similarly deleting an element from the array is also a performance wise expensive operation because all the elements after the deleted element have to be shifted left.

- ▶ These limitations are handled in the Linked List by providing following features:
- ▶ 1. Linked list allows dynamic memory allocation, which means memory allocation is done at the run time by the compiler and we do not need to mention the size of the list during linked list declaration.
- ▶ 2. Linked list elements don't need contiguous memory locations because elements are linked with each other using the reference part of the node that contains the address of the next node of the list.
- ▶ 3. Insert and delete operations in the Linked list are not performance wise expensive because adding and deleting an element from the linked list doesn't require element shifting, only the pointer of the previous and the next node requires change.

Doubly Linked List



Doubly linked list having three nodes



- ▶ A doubly linked list containing three nodes having numbers from 1 to 3 in their data part, is shown in the above image.

- ▶ 1. Head of the LinkedList only contains the Address of the First element of the List.
- ▶ 2. The Last element of the LinkedList contains null in the pointer part of the node because it is the end of the List so it doesn't point to anything as shown in the above diagram.

- ▶ In a singly linked list, we could traverse only in one direction, because each node contains address of the next node and it doesn't have any record of its previous nodes.
- ▶ However, doubly linked list overcome this limitation of singly linked list. Due to the fact that, each node of the list contains the address of its previous node,

▶ HashSet

- ▶ Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.
- ▶ The important points about Java HashSet class are:
 - ▶ HashSet stores the elements by using a mechanism called hashing.
 - ▶ HashSet contains unique elements only.
 - ▶ HashSet allows null value.
 - ▶ HashSet class is non synchronized.
 - ▶ HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashCode.
 - ▶ HashSet is the best approach for search operations.
 - ▶ The initial default capacity of HashSet is 16, and the load factor is 0.75.

▶ Difference between List and Set

- ▶ A list can contain duplicate elements whereas Set contains **unique** elements only.
- ▶ **Hierarchy of HashSet class**
- ▶ The HashSet class extends AbstractSet class which implements Set interface.
- ▶ The Set interface inherits Collection and Iterable interfaces in hierarchical order.

- ▶ 1) HashSet()
- ▶ It is used to construct a default HashSet.
- ▶ 2) HashSet(int capacity)
- ▶ It is used to initialize the capacity of the hash set to the given integer value capacity. The capacity grows automatically as elements are added to the HashSet.

- ▶ 3) `HashSet(int capacity, float loadFactor)` It is used to initialize the capacity of the hash set to the given integer value capacity and the specified load factor.
- ▶ 4) `HashSet(Collection<? extends E> c)` It is used to initialize the hash set by using the elements of the collection c.

HashSet Methods

- ▶ boolean add(Element e): It adds the element e to the list.
- ▶ void clear(): It removes all the elements from the list.
- ▶ Object clone(): This method returns a shallow copy of the HashSet.
- ▶ boolean contains(Object o): It checks whether the specified Object o is present in the list or not. If the object has been found it returns true else false.
- ▶ boolean isEmpty(): Returns true if there is no element present in the Set.
- ▶ int size(): It gives the number of elements of a Set.
- ▶ boolean(Object o): It removes the specified Object o from the Set.

Java Comparable interface

- ▶ Java Comparable interface is used to order the objects of the user-defined class. This interface is found in `java.lang` package and contains a method named `compareTo(Object)`.
- ▶ It provides a single sorting sequence only, i.e., you can sort the elements on the basis of single data member only. For example, it may be `rollno`, `name`, `age` or anything else.
- ▶ `compareTo(Object obj)` method

- ▶ **public int compareTo(Object obj):**
- ▶ **public int compareTo(Object obj):** It is used to compare the current object with the specified object.
- ▶ positive value, if the current object is greater than the specified object.
- ▶ negative integer, if the current object is less than the specified object.
- ▶ zero, if the current object is equal to the specified object.

- ▶ We can sort the elements of:
- ▶ String objects
- ▶ Wrapper class objects
- ▶ User-defined class objects

► Java Comparator interface

- Java Comparator interface is used to order the objects of a user-defined class.
- This interface is found in `java.util` package and contains 2 methods `compare(Object obj1, Object obj2)` and `equals(Object element)`.
- It provides multiple sorting sequences, i.e., you can sort the elements on the basis of any data member, for example, `rollno`, `name`, `age` or anything else.

▶ Methods of Java Comparator Interface

▶ Method Description

▶ `public int compare(Object obj1, Object obj2)`

It compares the first object with the second object.

▶ `public boolean equals(Object obj)` It is used to compare the current object with the specified object.

▶ `public boolean equals(Object obj)` It is used to compare the current object with the specified object.

▶ Java Queue Interface

- ▶ Java Queue interface orders the element in FIFO(First In First Out) manner. In FIFO, first element is removed first and last element is removed at last.
- ▶ Queue Interface declaration
- ▶ public interface Queue<E> extends Collection<E>

▶ Methods of Java Queue Interface

- ▶ `boolean add(object)` It is used to insert the specified element into this queue and return true upon success.
- ▶ `boolean offer(object)` It is used to insert the specified element into this queue.
- ▶ `Object remove()` It is used to retrieves and removes the head of this queue.
- ▶ `Object poll()` It is used to retrieves and removes the head of this queue, or returns null if this queue is empty.
- ▶ `Object element()` It is used to retrieves, but does not remove, the head of this queue.
- ▶ `Object peek()` It is used to retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

PriorityQueue class

```
public class PriorityQueue<E> extends AbstractQueue<E> implements  
Serializable
```

- ▶ PriorityQueue class
- ▶ The PriorityQueue class provides the facility of using queue. But it does not orders the elements in FIFO manner. It inherits AbstractQueue class.

- ▶ PriorityQueue class declaration
- ▶ Let's see the declaration for java.util.PriorityQueue class.
- ▶

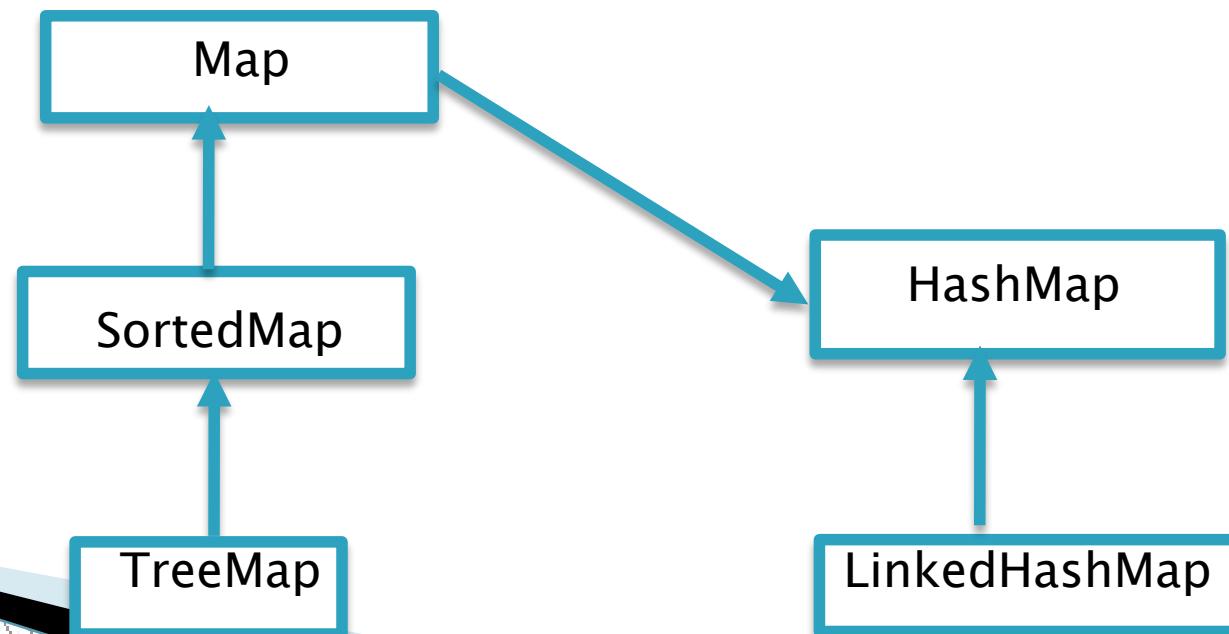
```
public class PriorityQueue<E> extends  
AbstractQueue<E> implements Serializable
```

MAP INTERFACE

- ▶ Map contains data as key value pairs
- ▶ Each entry in the map is a key value pair
- ▶ A Map is useful if you have to search, update or delete elements on the basis of a key.
- ▶ All the keys should be unique whereas
- ▶ values can have duplicate values

Hierarchy

- ▶ There are two interfaces for implementing Map in java: Map and SortedMap, and three classes: HashMap, LinkedHashMap, and TreeMap.



features

- ▶ A Map doesn't allow duplicate keys, but you can have duplicate values. HashMap and LinkedHashMap allow null keys and values, but TreeMap doesn't allow any null key or value.
- ▶ A Map can't be traversed, so you need to convert it into Set using keySet() or entrySet() method.
- ▶ Class Description
- ▶ **HashMap** HashMap is the implementation of Map, but it doesn't maintain any order.
- ▶ **LinkedHashMap** LinkedHashMap is the implementation of Map. It inherits HashMap class. It maintains insertion order.
- ▶ **TreeMap** TreeMap is the implementation of Map and SortedMap. It maintains ascending order.

HashSet Versus TreeSet

- ▶ 1) HashSet gives better performance (faster) than TreeSet for the operations like add, remove, contains, size etc.
- ▶ HashSet offers constant time cost while TreeSet offers $\log(n)$ time cost for such operations.

- ▶ 2) HashSet does not maintain any order of elements while TreeSet elements are sorted in ascending order by default.

Map.Entry Interface

- ▶ Entry is the subinterface of Map.
- ▶ we will be access it by Map.Entry name. It returns a collection-view of the map, whose elements are of this class. It provides methods to get key and value.

METHODS OF MAP.ENTRY

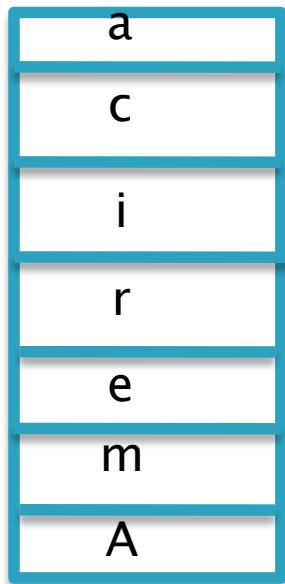
- ▶ `K getKey()` It is used to obtain a key.
- ▶ `V getValue()` It is used to obtain value.
- ▶ `int hashCode():` It is used to obtain hashCode.
- ▶ `V setValue(V value)` It is used to replace the value corresponding to this entry with the specified value.
- ▶ `boolean equals(Object o)` It is used to compare the specified object with the other existing objects.

Example

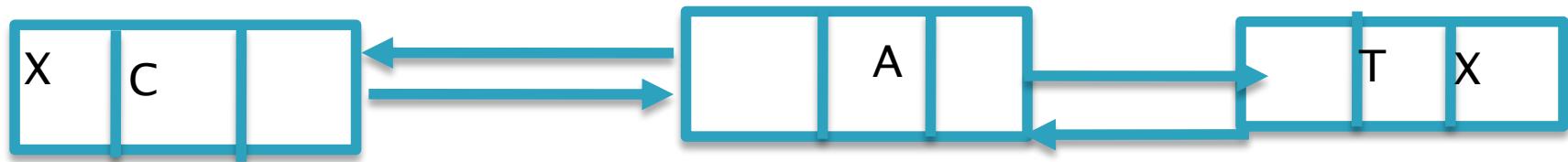
- ▶ Suppose I am storing a string “America” in a map datastructure then this is how it looks

A	2
m	1
e	1
r	1
i	1
c	1

▶ Data stored in a stack



DOUBLY LINKED LIST



- ▶ 1) Both HashSet and TreeSet does not hold duplicate elements, which means both of these are duplicate free.
- ▶ 2) If you want a sorted Set then it is better to add elements to HashSet and then convert it into TreeSet rather than creating a TreeSet and adding elements to it.
- ▶ 3) Both of these classes are non-synchronized that means they are not thread-safe and should be synchronized explicitly when there is a need of thread-safe operations.

TreeSet Class

- ▶ Java TreeSet class implements the Set interface that uses a tree for storage.
- ▶ It inherits AbstractSet class and implements the NavigableSet interface.
- ▶ The objects of the TreeSet class are stored in ascending order.

features of TreeSet

- ▶ Java TreeSet class contains unique elements only like HashSet.
- ▶ Java TreeSet class access and retrieval times are quiet fast.
- ▶ Java TreeSet class doesn't allow null element.
- ▶ Java TreeSet class is non synchronized.
- ▶ Java TreeSet class maintains ascending order.

- ▶ `TreeSet()` It is used to construct an empty tree set that will be sorted in ascending order according to the natural order of the tree set.
- ▶ `TreeSet(Collection<? extends E> c)` It is used to build a new tree set that contains the elements of the collection c.

- ▶ `TreeSet(Comparator<? super E> comparator)`
It is used to construct an empty tree set that will be sorted according to given comparator.
- ▶ `TreeSet(SortedSet<E> s)` It is used to build a TreeSet that contains the elements of the given SortedSet.

▶ Introduction to JDBC

- ▶ Java Database Connectivity(JDBC) is an Application Programming Interface(API) used to connect Java application with Database. JDBC is used to interact with various type of Databases such as Oracle, MS Access, My SQL and SQL Server. JDBC can also be defined as the platform-independent interface between a relational database and Java programming. It allows java program to execute SQL statement and retrieve result from database.

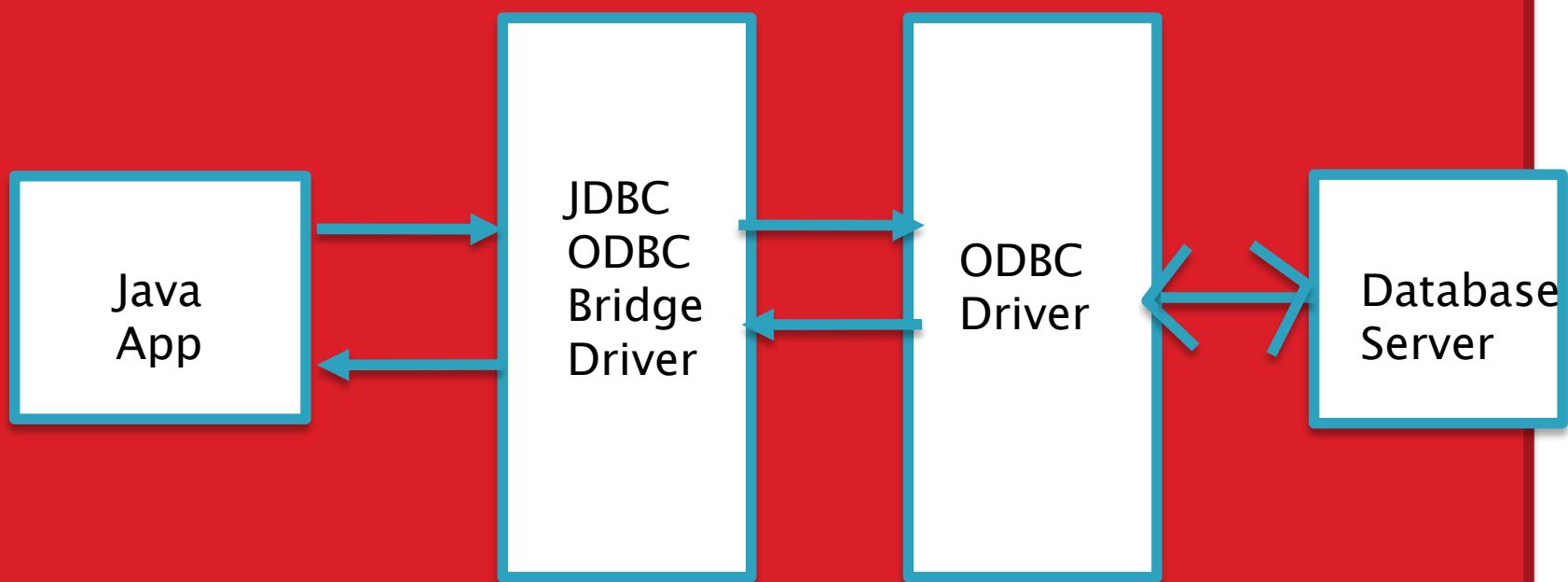


- ▶ What's new in JDBC 4.0
- ▶ JDBC 4.0 is new and advance specification of JDBC. It provides the following advance features
 - ▶ Connection Management
 - ▶ Auto loading of Driver Interface.
 - ▶ Better exception handling
 - ▶ Support for large object
 - ▶ Annotation in SQL query.

Different Types Of Drivers

- ▶ JDBC-ODBC bridge driver
- ▶ Native-API driver
- ▶ Network Protocol driver
- ▶ Thin driver
- ▶ JDBC Driver is a software component that enables java application to interact with the database.
There are 4 types of JDBC drivers:
 - ▶ JDBC-ODBC bridge driver
 - ▶ Native-API driver (partially java driver)
 - ▶ Network Protocol driver (fully java driver)
 - ▶ Thin driver (fully java driver)

Architecture of Type-1 Driver



TYPE – 1 Driver

- ▶ Also known as JDBC–ODBC Bridge Driver
- ▶ It is a part of JDK
- ▶ Internally it uses ODBC Driver

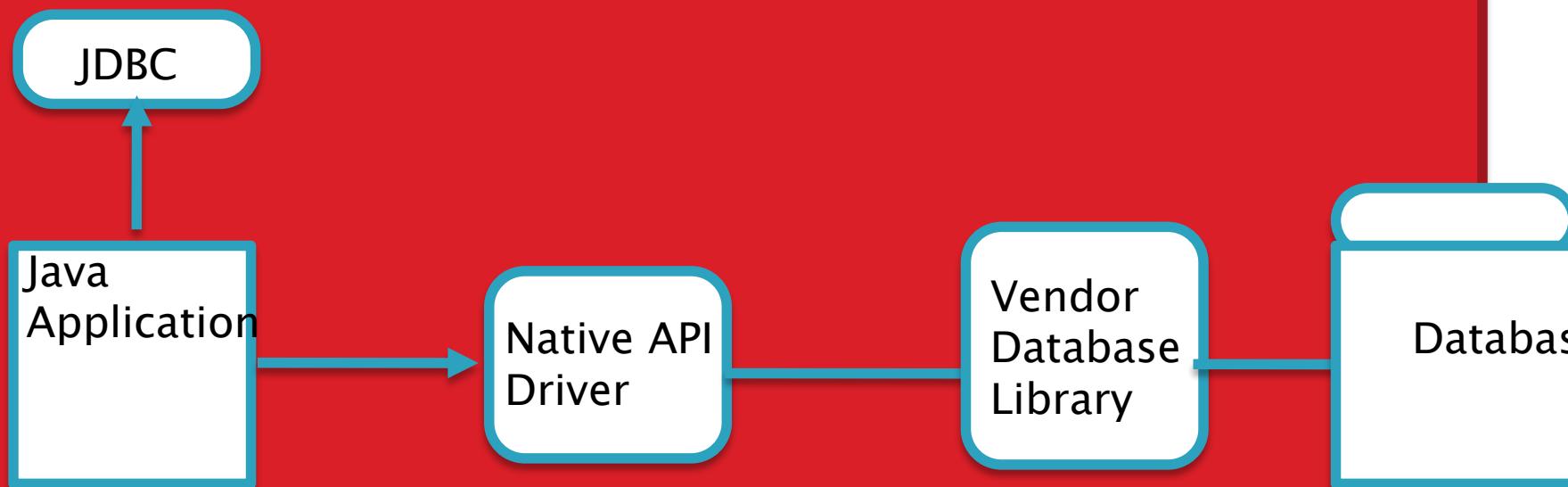
ADVANTAGES

- ▶ No separate installation required
- ▶ Very Easy to Use
- ▶ DataBase Independent Driver, Since it is not communicating directly with the database

DISADVANTAGES

- ▶ It is the slowest Driver
- ▶ Platform independence not there for Type-1
- ▶ Type-1 Driver support available only upto JDK 1.7 version

Type-2 Architecture



FEATURES OF Type-2 Driver

The Native API driver uses the client-side libraries of the database.

The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

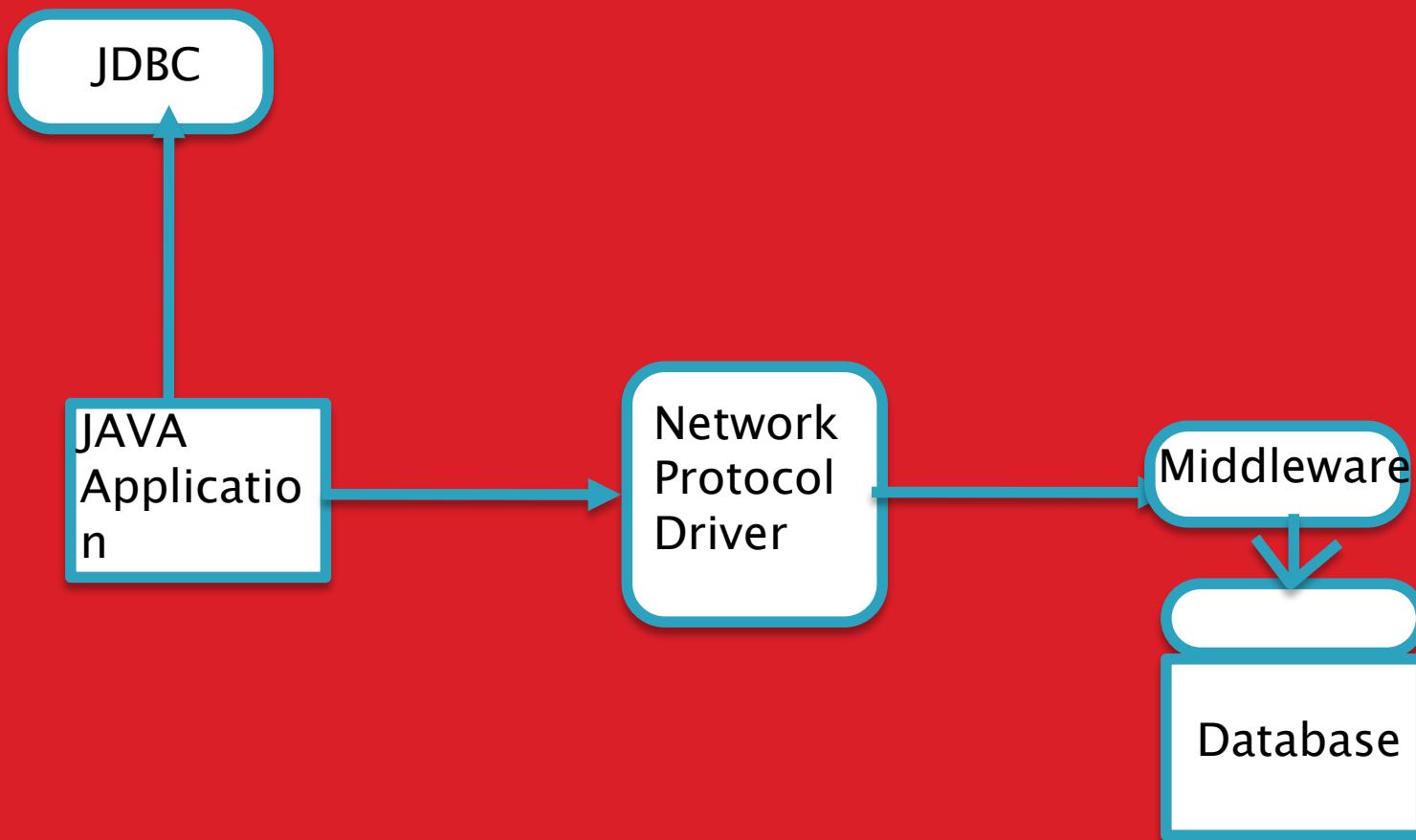
Disadvantages Of Type-2 Driver

- ▶ The Native driver needs to be installed on the each client machine.
- ▶ The Vendor client library needs to be installed on client machine.

Type-3 Driver(Network Protocol Driver)

- ▶ The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol.
- ▶ It is fully written in java.

Architecture(Type-3)



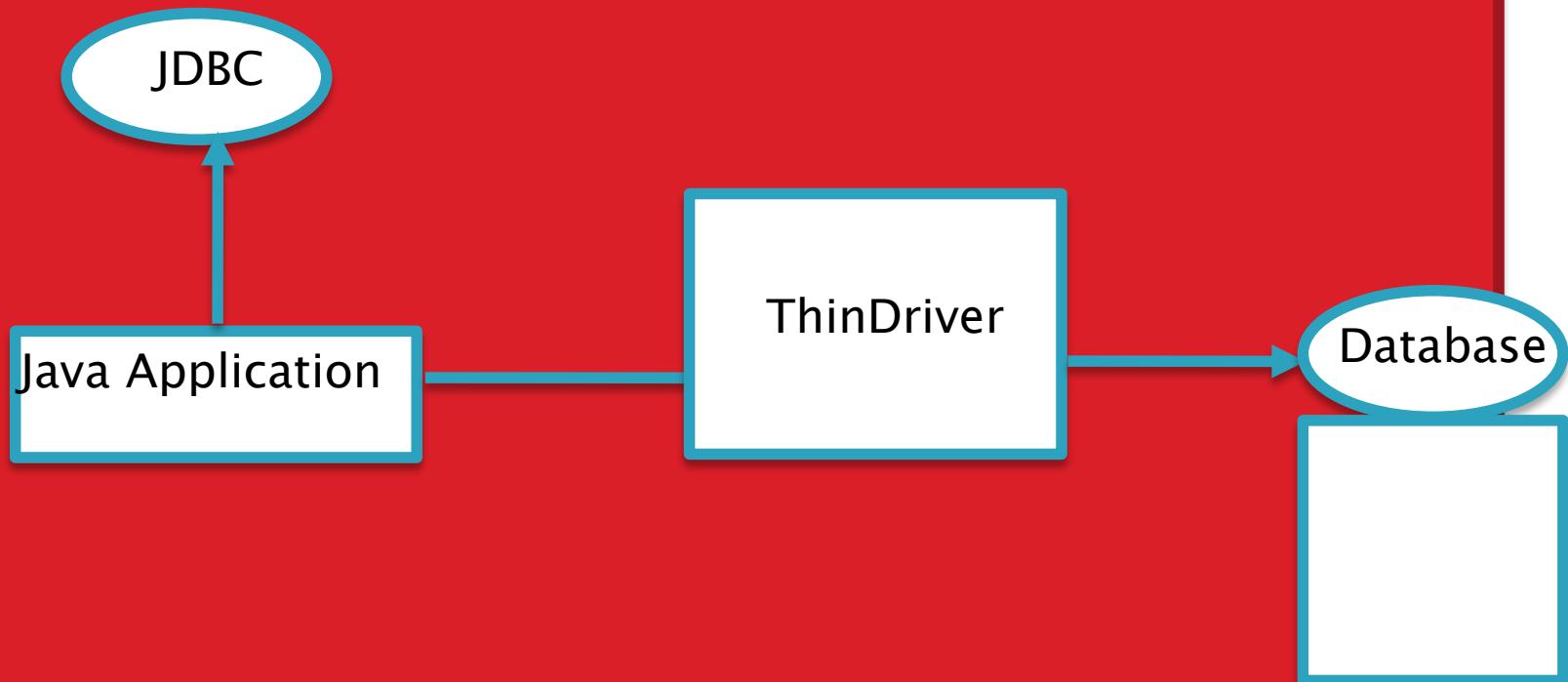
Features

- ▶ **Advantages:**
- ▶ No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- ▶ **Disadvantages:**
- ▶ Network support is required on client machine.
- ▶ Requires database-specific coding to be done in the middle tier.
- ▶ Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

Type-4 Driver (Thin Driver)

- ▶ The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver.
- ▶ It is fully written in Java language.
- ▶ Also Known as Pure Java Driver.

Architecture



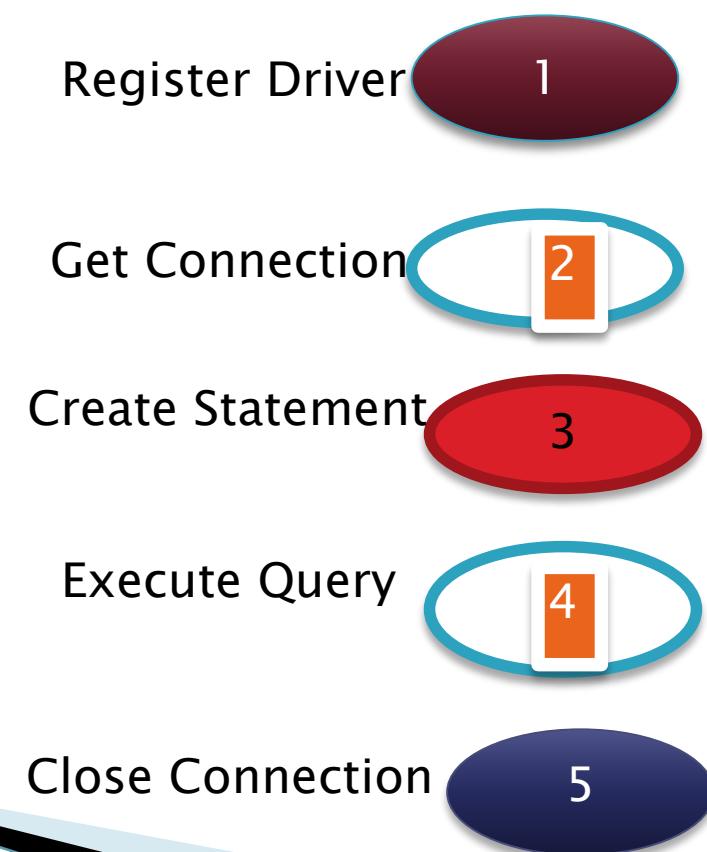
Features

- ▶ **Advantage:**
- ▶ Better performance than all other drivers.
- ▶ No software is required at client side or server side.
- ▶ **Disadvantage:**
- ▶ Drivers depend on the Database.

Java Database Connection

- ▶ In order to establish a connection with the database we have to follow these steps
- ▶ **Register the Driver class**
- ▶ **Create connection**
- ▶ **Create statement**
- ▶ **Execute queries**
- ▶ **Close connection**

Steps



How To Register The Driver?

- ▶ The `forName()` method of `Class` class is used to register the driver class.
- ▶ This method will dynamically load the driver class.
- ▶ Syntax of `forName()` method
- ▶ `public static void forName(String
className) throws ClassNotFoundException`

- ▶ `Class.forName("oracle.jdbc.driver.OracleDriver");`
- ▶ This statement loads the Oracle driver

Create the connection object

- ▶ The getConnection() method of DriverManager class is used to establish connection with the database.
- ▶ Syntax of getConnection() method
- ▶ 1) **public static Connection getConnection(String url)throws SQLException**
- ▶ 2) **public static Connection getConnection(String url,String name,String password)**
- ▶ **throws SQLException**

- ▶ How to create the Statement object?
- ▶ 3) The `createStatement()` method of `Connection` interface is used to create statement.
- ▶ The object of statement is responsible to execute queries with the database.
- ▶ **Syntax** of `createStatement()` method
- ▶ **public Statement createStatement()throws SQLException**

▶ How To Execute the query

- ▶ The executeQuery() method of Statement interface is used to execute queries to the database.
 - ▶ This method returns the object of ResultSet that can be used to get all the records of a table.
- ▶ **Syntax of executeQuery() method**
- ▶ **public ResultSet executeQuery(String sql) throws SQLException**

CODE SNIPPET

- ▶ `ResultSet rs=stmt.executeQuery("select * from employee");`
- ▶
- ▶ `while(rs.next()){`
- ▶ `System.out.println(rs.getInt(1)+"`
- ▶ `"+rs.getString(2));`
- ▶ `}`

How To Close The Connection

- ▶ By closing connection object statement and ResultSet will be closed automatically.
- ▶ The close() method of Connection interface is used to close the connection.
- ▶ **Syntax of close() method**
- ▶ **public void close()throws SQLException**
- ▶ **Example to close connection**
- ▶ **con.close();**

NOTE

- ▶ Note: Since Java 7, JDBC uses try-with-resources statement to automatically close resources of type Connection, ResultSet, and Statement.
- ▶ It avoids explicit connection closing step.

▶ Database Connectivity with Oracle

- ▶ To connect java application with the oracle database, we need to follow 5 following steps.
- ▶ Driver class: The driver class for the oracle database is `oracle.jdbc.driver.OracleDriver`.
- ▶ Connection URL: The connection URL for the oracle10G database is `jdbc:oracle:thin:@localhost:1521:xe` where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name.
- ▶ You may get all these information from the `tnsnames.ora` file.
- ▶ Username: The default username for the oracle database is `system`.
- ▶ Password: It is the password given by the user at the time of installing the oracle database.

▶ How To Create a Table?

- ▶ Before establishing connection, let's first create a table in the oracle database.
- ▶

```
create table employee(emp_id  
number(10),emp_name varchar2(40),emp_age  
number(3));
```

EXAMPLE FOR ORACLE

```
► import java.sql.*;
► class OracleCon{
► public static void main(String args[]){
► try{
► //step1 load the driver class
► Class.forName("oracle.jdbc.driver.OracleDriver");
►
► //step2 create the connection object
► Connection con=DriverManager.getConnection(
► "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
►
► //step3 create the statement object
► Statement stmt=con.createStatement();
►
► //step4 execute query
► ResultSet rs=stmt.executeQuery("select * from emp");
► while(rs.next())
► System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
►
► //step5 close the connection object
► con.close();
►
► }catch(Exception e){ System.out.println(e);}
►
► }
```

- ▶ Ways to load the jar file:
 - ▶ paste the ojdbc14.jar file in jre/lib/ext folder
 - ▶ set classpath
 - ▶ 1) paste the ojdbc14.jar file in JRE/lib/ext folder:
 - ▶ Firstly, search the ojdbc14.jar file then go to JRE/lib/ext folder and paste the jar file here.
 - ▶ 2) set classpath:

- ▶ There are two ways to set the classpath:
- ▶ temporary
- ▶ permanent
- ▶ **How to set the temporary classpath:**
- ▶ Firstly, search the ojdbc14.jar file then open command prompt and write:
- ▶ **C:>set classpath=c:\folder\ojdbc14.jar;.;**
- ▶ **How to set the permanent classpath?:**
- ▶ Go to environment variable then click on new tab. In variable name write classpath and in variable value paste the path to ojdbc14.jar by appending ojdbc14.jar;.; as C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar;.;

▶ Database Connectivity with MySQL

- ▶ To connect Java application with the MySQL database, we need to follow 5 following steps.
- ▶ In this example we are using MySql as the database. So we need to know following informations for the mysql database:
 - ▶ Driver class: The driver class for the mysql database is `com.mysql.jdbc.Driver`.
 - ▶ Connection URL: The connection URL for the mysql database is `jdbc:mysql://localhost:3306/sam` where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sam is the database name.
 - ▶ We may use any database, in such case, we need to replace the sam with our database name.
 - ▶ Username: The default username for the mysql database is `root`.
 - ▶ Password: It is the password given by the user at the time of installing the mysql database. In this example, we are going to use `root` as the password.

- ▶ create database sam;
- ▶ use sam;
- ▶ create table Student(id int(10),name varchar(40),age int(3));

- ▶ To connect java application with the mysql database, mysqlconnector.jar file is required to be loaded.
- ▶ download the jar file mysql-connector.jar
- ▶ Two ways to load the jar file:
- ▶ Paste the mysqlconnector.jar file in jre/lib/ext folder
- ▶ Set classpath

▶ DriverManager class

- ▶ The DriverManager class acts as an interface between user and drivers.
- ▶ It keeps track of the drivers and it establishes a connection between a database and the appropriate driver.
- ▶ The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method `DriverManager.registerDriver()`.

▶ Methods of DriverManager class

- ▶ Method Description
- ▶ 1) **public static void registerDriver(Driver driver):**
Registers the given driver with DriverManager.
- ▶ 2) **public static void deregisterDriver(Driver driver):**
Deregisters the given driver (drop the driver from the list) with DriverManager.
- ▶ 3) **public static Connection getConnection(String url):**
Establishes the connection with the specified url.
- ▶ 4) **public static Connection getConnection(String url, String userName, String password):** Establishes the connection with the specified url, username and password.

▶ PreparedStatement

- ▶ The PreparedStatement interface is a subinterface of Statement.
- ▶ It is used to execute parameterized query.
- ▶ Let's see the example of parameterized query:
- ▶ String sql="insert into emp values(?, ?, ?);"
- ▶ As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

- ▶ Why use PreparedStatement?
- ▶ Improves performance: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

▶ Benefits of Java Prepared Statement

- ▶ PreparedStatement in Java JDBC offers several benefits and it's a recommended way to execute SQL queries in any enterprise Java application or in production code.
- ▶ Here are few advantages of using PreparedStatement in Java:
- ▶ 1. PreparedStatement allows you to write dynamic and parametric query.

- ▶ 2) **PreparedStatement** is faster than **Statement** in Java
- ▶ One of the major benefits of using PreparedStatement is better performance.
- ▶ PreparedStatement gets pre compiled
- ▶ In database and there access plan is also cached in database, which allows database to execute parametric query written using prepared statement much faster than normal query because it has less work to do.
- ▶ You should always try to use PreparedStatement in production JDBC code to reduce load on database.

Methods Of PreparedStatement

- ▶ How to get the instance of PreparedStatement?
- ▶ The **prepareStatement()** method of Connection interface is used to return the object of PreparedStatement.
- ▶ Method
- ▶ **public void setInt(int paramInt, int value)** sets the integer value to the given parameter index.
- ▶ **public void setString(int paramInt, String value)** sets the String value to the given parameter index.
- ▶ **public void setFloat(int paramInt, float value)** sets the float value to the given parameter index.
- ▶ **public void setDouble(int paramInt, double value)** sets the double value to the given parameter index.

▶ CallableStatement Interface

- ▶ Java CallableStatement interface is used to call the stored procedures and functions.
- ▶ We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Differences between StoredProcedure and Function

- ▶ What is the difference between stored procedures and functions.
- ▶ The differences between stored procedures and functions are given below:
- ▶ Stored Procedure is used to perform business logic.function is used to perform calculation.
- ▶ must not have the return type.function must have the return type.
- ▶ may return 0 or more values. function may return only one value.
- ▶ We can call functions from the procedure. Procedure cannot be called from function.
- ▶ Procedure supports input and output parameters. Function supports only input parameter.
- ▶ Exception handling using try/catch block can be used in stored procedures. Exception handling using try/catch can't be used in user defined functions.

SYNTAX

- ▶ How to get the instance of CallableStatement?
- ▶ The prepareCall() method of Connection interface returns the instance of CallableStatement. Syntax is given below:
- ▶ `public CallableStatement prepareCall("{ call procedurename(?,?...?)}");`

Java 8 New Features

- ▶ **1) LAMBDA EXPRESSIONS:**
- ▶ Lambda expression is a new feature which is introduced in Java 8.
- ▶ A lambda expression is an anonymous function.
- ▶ A function that doesn't have a name and doesn't belong to any class.
- ▶ The concept of lambda expression was first introduced in LISP programming language.

▶ Java Lambda Expression Syntax

- ▶ To create a lambda expression, we specify input parameters (if there are any) on the left side of the lambda operator `->`, and place the expression or block of statements on the right side of lambda operator.
- ▶ For example, the lambda expression `(x, y) -> x + y` specifies that lambda expression takes two arguments `x` and `y` and returns the sum of these.

- ▶ Lambda expression vs method in Java
- ▶ A method (or function) in Java has these main parts:
 - ▶ 1. Name
 - ▶ 2. Parameter list
 - ▶ 3. Body
 - ▶ 4. return type.

- ▶ A lambda expression in Java has these main parts:
- ▶ Lambda expression only has body and parameter list.
- ▶ 1. No name – function is anonymous so we don't care about the name
- ▶ 2. Parameter list
- ▶ 3. Body – This is the main part of the function.
- ▶ 4. No return type – The java 8 compiler is able to infer the return type by checking the code. you need not to mention it explicitly.

Where to use lambda?

- ▶ To use lambda expression, you need to either create your own functional interface or use the pre defined functional interface provided by Java.
- ▶ An interface with only single abstract method is called functional interface(or Single Abstract method interface), for example: Runnable, callable, ActionListener etc.
- ▶ To use function interface:
- ▶ Pre Java 8: We create anonymous inner classes.
- ▶ Post Java 8: You can use lambda expression instead of anonymous inner classes.

▶ Java Predefined–Functional Interfaces

- ▶ Java provides predefined functional interfaces to deal with functional programming by using lambda and method references.
- ▶ You can also define your own custom functional interface. Following is the list of functional interface which are placed in `java.util.function` package.
- ▶ `InterfaceDescription`
- ▶ `BiConsumer<T,U>` It represents an operation that accepts two input arguments and returns no result.
- ▶ `Consumer<T>` It represents an operation that accepts a single argument and returns no result.
- ▶ `Function<T,R>` It represents a function that accepts one argument and returns a result.
- ▶ `Predicate<T>` It represents a predicate (boolean–valued function) of one argument.

Predicate

- ▶ The Predicate interface has an abstract method test which gives a Boolean value as a result for the specified argument. Its prototype is
- ▶ public interface Predicate
- ▶ {
- ▶ ▶ public boolean test(T t);
- ▶ }

▶ Function

- ▶ The Java Function interface (`java.util.function.Function`) interface is one of the most central functional interfaces in Java. The Function interface represents a function (method) that takes a single parameter and returns a single value. Here is how the Function interface definition looks:
- ▶

```
public interface Function<T,R> {  
    public <R> apply(T parameter);  
}
```

▶ Consumer

- ▶ The Java Consumer interface is a functional interface that represents a function that consumes a value without returning any value.
- ▶ A Java Consumer implementation could be printing out a value, or writing it to a file, or over the network etc. Here is an example implementation of the Java Consumer interface:
- ▶

```
Consumer<Integer> consumer = (value) ->
System.out.println(value);
```
- ▶ This Java Consumer implementation prints the value passed as parameter to it out to System.out.

▶ Java 8 Stream

- ▶ Java provides a new additional package in Java 8 called `java.util.stream`.
- ▶ This package consists of classes, interfaces and enum to allows functional-style operations on the elements.
- ▶ By using streams we can perform various aggregate operations on the data returned from collections, arrays, Input/Output operations.

Features Of Stream

- ▶ Stream provides following features:
- ▶ Stream does not store elements. It simply conveys elements from a source such as a data structure, an array, or an I/O channel, through a pipeline of computational operations.
- ▶ Stream is functional in nature. Operations performed on a stream does not modify it's source.
- ▶ For example, filtering a Stream obtained from a collection produces a new Stream without the filtered elements, rather than removing elements from the source collection.

Features

- ▶ Stream is lazy and evaluates code only when required.
- ▶ The elements of a stream are only visited once during the life of a stream. Like an Iterator, a new stream must be generated to revisit the same elements of the source.