UNIT-III - JAVA

Java is a class-based, object-oriented programming language. Java was first released in 1995 and is widely used for developing applications for desktop, web, and mobile devices.

JAVA was developed by James Gosling at **Sun Microsystems** Inc in the year **1995**, later acquired by Oracle Corporation. It is a simple programming language. Java makes writing, compiling, and debugging programming easy. It helps to create reusable code and modular programs.
A general-purpose programming language made for developers to *write once run anywhere* that is compiled Java code can run on all platforms that support Java. Java applications are compiled to byte code that can run on any Java Virtual Machine. The syntax of Java is similar to c/c++.

Java Virtual Machine(JVM): This is generally referred to as JVM. There are three execution phases of a program. They are written, compile and run the program.

- Writing a program is done by a java programmer like you and me.
- The compilation is done by the **JAVAC** compiler which is a primary **Java compiler** included in the **Java development kit** (JDK). It takes the Java program as input and generates bytecode as output.
- In the Running phase of a program, **JVM** executes the bytecode generated by the compiler.

Advantages of Java:

1. Platform independent: Java code can run on any platform that has a Java Virtual Machine (JVM) installed, which means that applications can be written once and run on any device.
2. Object-Oriented: Java is an object-oriented programming language, which means that it follows the principles of encapsulation, inheritance, and polymorphism.
3. Security: Java has built-in security features that make it a secure platform for developing applications, such as automatic memory management and type checking.
4. Large community: Java has a large and active community of developers, which means that there is a lot of support available for learning and using the language.
5. Enterprise-level applications: Java is widely used for developing enterprise-level applications, such as web applications, e-commerce systems, and database systems.

Disadvantages of Java:

1. Performance: Java can be slower compared to other programming languages, such as C++, due to its use of a virtual machine and automatic memory management.
2. Memory management: Java’s automatic memory management can lead to slower performance and increased memory usage, which can be a drawback for some applications.

SAMPLE PROGRAM:

```
/ Importing classes from packages

import java.io.*;

 // Main class

public class GFG {

     // Main driver method

    public static void main(String[] args)

    {

         // Print statement

        System.out.println("Welcome to GeeksforGeeks");

    }

}
```

Output

```
Welcome to GeeksforGeeks
```

Primary/Main Features of Java

1. Platform Independent: Windows, Linux, or macOS
2. Object-Oriented Programming Language:
3. Simple:
4. Robust:
5. Secure:
6. Distributed:
7. Multithreading:
8. Portable:
9. High Performance:
10. Dynamic flexibility
11. Sandbox Execution
12. Write Once Run Anywhere.
13. Power of compilation and interpretation:

Objects and Classes in Java

Classes and Objects are basic concepts of Object Oriented Programming that revolve around real life entities.

Class

1. Class is a set of object which shares common characteristics/ behavior and common properties/ attributes.
2. Class is not a real world entity. It is just a template or blueprint or prototype from which objects are created.
3. Class does not occupy memory.
4. Class is a group of variables of different data types and group of methods.

A class in java can contain:
- data member
- method
- constructor
- nested class and
- interface

Syntax to declare a class:

```
access_modifier class<class_name>
{
   data member;
   method;
   constructor;
   nested class;
   interface;
}
```

Eg:

```
public calss area
{
Int length;
Int bredth;
}
```

In general, class declarations can include these components, in order:

1. **Modifiers**: A class can be public or has default access (Refer this for details).
2. **Class keyword:** class keyword is used to create a class.
3. **Class name:** The name should begin with an initial letter (capitalized by convention).
4. **Superclass(if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
5. **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
6. **Body:** The class body is surrounded by braces, { }.

Object

It is a basic unit of Object-Oriented Programming and represents real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

1. **State**: It is represented by attributes of an object. It also reflects the properties of an object.
2. **Behavior**: It is represented by methods of an object. It also reflects the response of an object with other objects.
3. **Identity**: It gives a unique name to an object and enables one object to interact with other objects.

Example of an object: dog

Objects correspond to things found in the real world. For example, a graphics program may have objects such as “circle”, “square”, and “menu”. An online shopping system might have objects such as “shopping cart”, “customer”, and “product”.

Declaring Objects (Also called instantiating a class)

When an object of a class is created, the class is said to be **instantiated**. A single class may have any number of instances.
Syntax:

```
class_name objectname = new class_name(parameters);
```

Example:

Area a=new area();

a.length=10;

a.breadth=20;

INHERITANCE

Inheritance is an important pillar of OOP(Object-Oriented Programming). It is the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class. In Java, inheritance means creating new classes based on existing ones. A class that inherits from another class can reuse the methods and fields of that class. In addition, you can add new fields and methods to your current class as well.

Inheritance in Java: Why do we need it?

- **Code Reusability:** The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class code.
- **Method Overriding:** Method Overriding is achievable only through Inheritance. It is one of the ways by which java achieves Run Time Polymorphism.
- **Abstraction:** The concept of abstract where we do not have to provide all details is achieved through inheritance. Abstraction only shows the functionality to the user.

Important terminologies used in Inheritance:

- **Class:** Class is a set of objects which shares common characteristics/ behavior and common properties/ attributes. Class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
- **Super Class/Parent Class:** The class whose features are inherited is known as a superclass(or a base class or a parent class).
- **Sub Class/Child Class:** The class that inherits the other class is known as a subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Reusability:** Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

How to use inheritance in Java?

The **extends keyword** is used for inheritance in java. Using the extends keyword indicates you are derived from an existing class. In other words, “extends” refers to increased functionality.

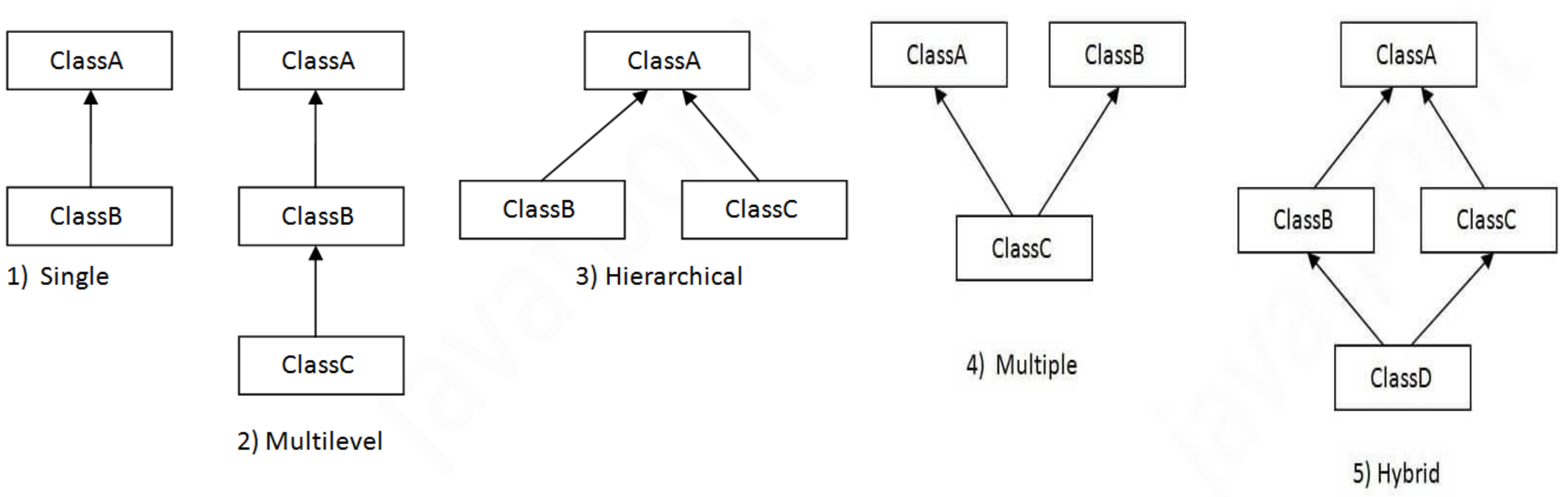
Syntax :

class derived-class extends base-class

```
{
   //methods and fields
}
```

Example: In the below example of inheritance, class Bicycle is a base class, class MountainBike is a derived class that extends the Bicycle class and class Test is a driver class to run the program.

Types of Inheritance in Java



1. Single Inheritance

In single inheritance, subclasses inherit the features of one superclass. In the image below, class A serves as a base class for the derived class B.

EX:

```
class one
{

}

class two extends one
{

}
```

OBJECT CREATION: two g = new two();

2. Multilevel Inheritance

In Multilevel Inheritance, a derived class will be inheriting a base class, and as well as the derived class also acts as the base class for other classes. In the below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C.

```
class one
{


}

class two extends one
{

}

class three extends two
 {

    }
```

OBJECT CREATION: three g = new three();

3. Hierarchical Inheritance

In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass. In the below image, class A serves as a base class for the derived classes B, C, and D.

```
class A
{
}

class B extends A
{
}

class C extends A
 {
}

class D extends A
{
}
```

OBJECT CREATION:

```
B obj_B = new B();
C obj_C = new C();
D obj_D = new D();
```

4. Multiple Inheritance (Through Interfaces)

In Multiple inheritances, one class can have more than one superclass and inherit features from all parent classes. Please note that Java does **not** support multiple inheritances with classes. In java, we can achieve multiple inheritances only through Interfaces. In the image below, Class C is derived from interfaces A and B.

EX:

```
interface one
{
}

interface two
 {
}

interface three extends one, two
{
}
class child implements three

{

}
```

OBJECT CREATION: child c = new child();

5. Hybrid Inheritance(Through Interfaces)

It is a mix of two or more of the above types of inheritance. Since java doesn’t support multiple inheritances with classes, hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through Interfaces.

EX:

```
public class SolarSystem {
}
public class Earth extends SolarSystem {
}
public class Mars extends SolarSystem {
}
public class Moon extends Earth {
}
```

OBJECT CREATION: SolarSystem s = new SolarSystem();

```
    Earth e = new Earth();
    Mars m = new Mars();
```

Java String

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

```
1. char[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=new String(ch);
```

is same as:

```
1. String s="javatpoint";
```

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

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

What is String in Java?

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

1) String Literal

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

```
1. String s="welcome";
```

2) By new keyword

```
1. String s=new String("Welcome");//creates two objects and one reference variable
```

Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

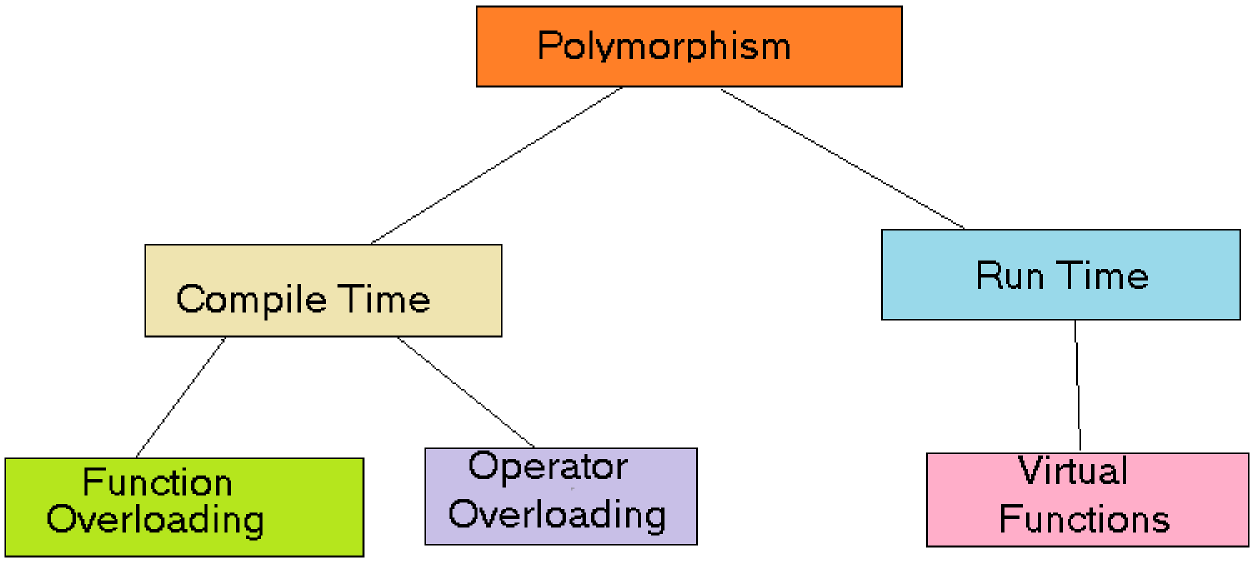
Method	Description
String substring(int beginIndex)	It returns substring for given begin index.
String substring(int beginIndex, int endIndex)	It returns substring for given begin index and end index.
boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value.
static String join(CharSequence delimiter, CharSequence... elements)	It returns a joined string.
static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	It returns a joined string.
String concat(String str)	It concatenates the specified string.
String replace(char old, char new)	It replaces all occurrences of the specified char value.
String replace(CharSequence old, CharSequence new)	It replaces all occurrences of the specified CharSequence.
static String equalsIgnoreCase(String another)	It compares another string. It doesn't check case.
String[] split(String regex)	It returns a split string matching regex.
String[] split(String regex, int limit)	It returns a split string matching regex and limit.
String intern()	It returns an interned string.

int indexOf(int ch)	It returns the specified char value index.
int indexOf(int ch, int fromIndex)	It returns the specified char value index starting with given index.
int indexOf(String substring)	It returns the specified substring index.
int indexOf(String substring, int fromIndex)	It returns the specified substring index starting with given index.
String toLowerCase()	It returns a string in lowercase.
String toUpperCase()	It returns a string in uppercase.
String trim()	It removes beginning and ending spaces of this string.

POLYMORPHISM IN JAVA

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

In Java, polymorphism refers to the ability of a class to provide different implementations of a method, depending on the type of object that is passed to the method.



In Java polymorphism is mainly divided into two types:

- Compile-time Polymorphism
- Runtime Polymorphism

Subtypes of Compile-time Polymorphism:

1. **Function Overloading**: It is a feature in C++ where multiple functions can have the same name but with different parameter lists. The compiler will decide which function to call based on the number and types of arguments passed to the function.
2. **Operator Overloading:** It is a feature in C++ where the operators such as +, -, * etc. can be given additional meanings when applied to user-defined data types.

Type 2: <u>Runtime polymorphism</u>

Subtype of Run-time Polymorphism:

1. **Virtual functions:** It allows an object of a derived class to behave as if it were an object of the base class. The derived class can override the virtual function of the base class to provide its own implementation. The function call is resolved at runtime, depending on the actual type of the object.

Advantages of Polymorphism in Java:

1. Increases code reusability by allowing objects of different classes to be treated as objects of a common class.
2. Improves readability and maintainability of code by reducing the amount of code that needs to be written and maintained.
3. Supports dynamic binding, enabling the correct method to be called at runtime, based on the actual class of the object.
4. Enables objects to be treated as a single type, making it easier to write generic code that can handle objects of different types.

Disadvantages of Polymorphism in Java:

1. Can make it more difficult to understand the behavior of an object, especially if the code is complex.
2. May lead to performance issues, as polymorphic behavior may require additional computations at runtime.

Interface in Java

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

How to declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Syntax:

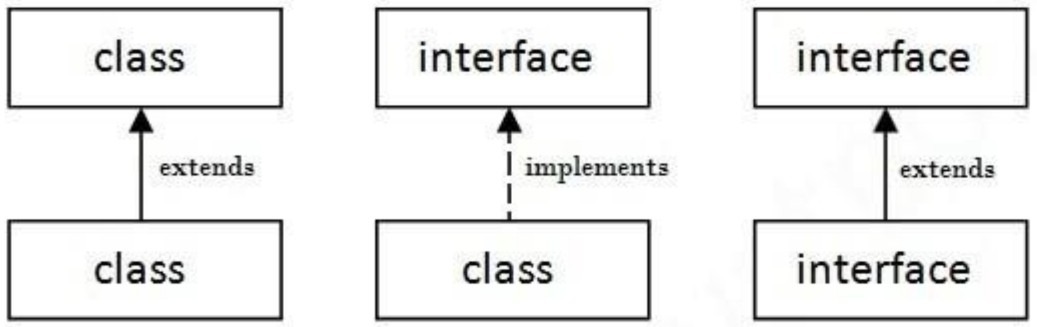
```
interface <interface_name>{



}
```

The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



Java Interface Example: Bank

Let's see another example of java interface which provides the implementation of Bank interface.

File: TestInterface2.java

```
interface Bank{
float rateOfInterest();
}
class SBI implements Bank{
public float rateOfInterest(){return 9.15f;}
}
class PNB implements Bank{
public float rateOfInterest(){return 9.7f;}
}
class TestInterface2{
public static void main(String[] args){
Bank b=new SBI();
System.out.println("ROI: "+b.rateOfInterest());
}}
```

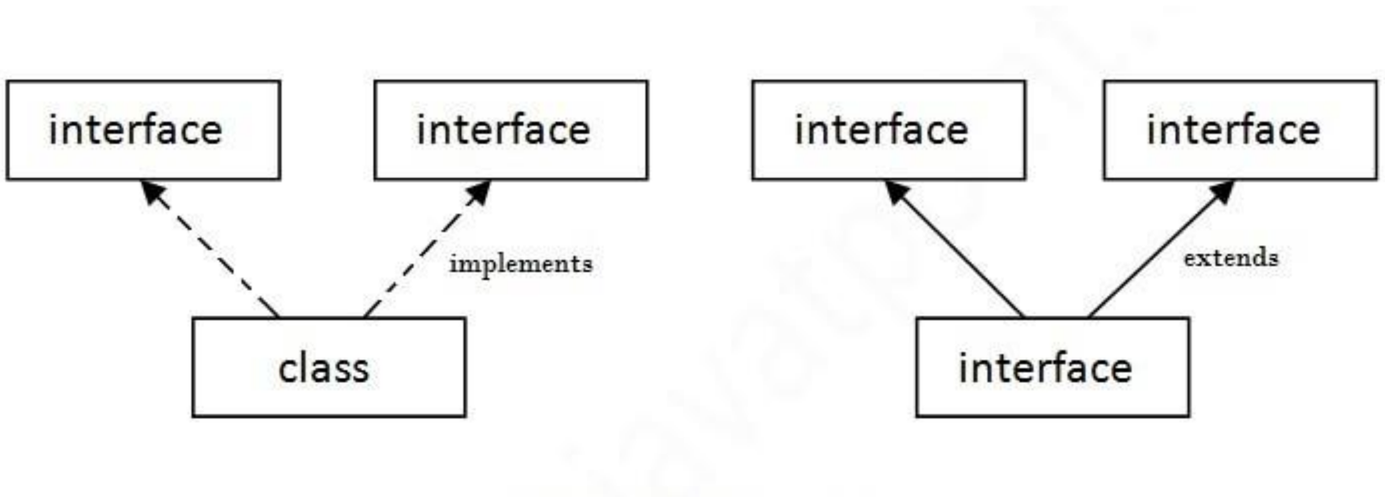
Test it Now

Output:

```
ROI: 9.15
```

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

```
interface Printable{
void print();
}
interface Showable{
void show();
}
class A7 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

Test it Now

```
Output:Hello
       Welcome
```

Regular Expressions

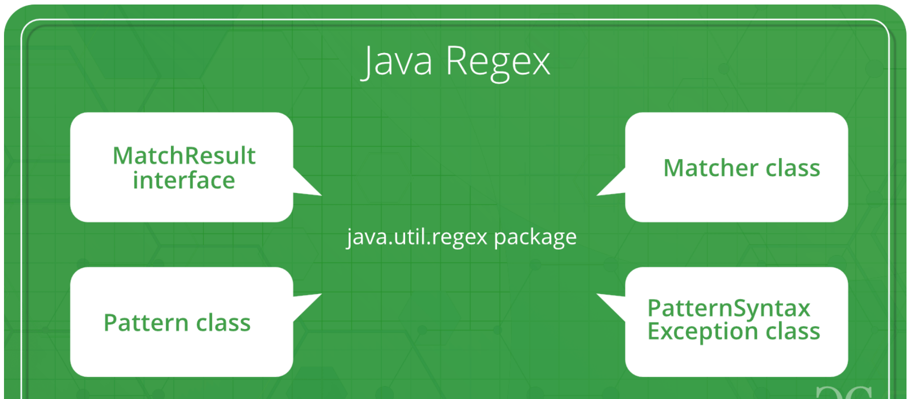
Regular Expressions or Regex (in short) in Java is an API for defining String patterns that can be used for searching, manipulating, and editing a string in Java. Email validation and passwords are a few areas of strings where Regex is widely used to define the constraints. Regular Expressions are provided under **java.util.regex** package. This consists of **3 classes and 1 interface**. The **java.util.regex** package primarily consists of the following three classes as depicted below in tabular format as follows:

S. No.	Class/Interface	Description
1.	Pattern Class	Used for defining patterns
2.	Matcher Class	Used for performing match operations on text using patterns
3.	PatternSyntaxException Class	Used for indicating syntax error in a regular expression pattern
4.	MatchResult Interface	Used for representing the result of a match operation

Regex in java provides us with 3 classes and 1 interface listed below as follows:

1. Pattern Class
2. Matcher Class
3. PatternSyntaxException Class
4. MatchResult Interface

More understanding can be interpreted from the image provided below as follows:



Exception Handling in Java

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.

In this tutorial, we will learn about Java exceptions, it's types, and the difference between checked and unchecked exceptions.

What is Exception in Java?

Dictionary Meaning: Exception is an abnormal condition.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is Exception Handling?

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

The core advantage of exception handling is **to maintain the normal flow of the application**.

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

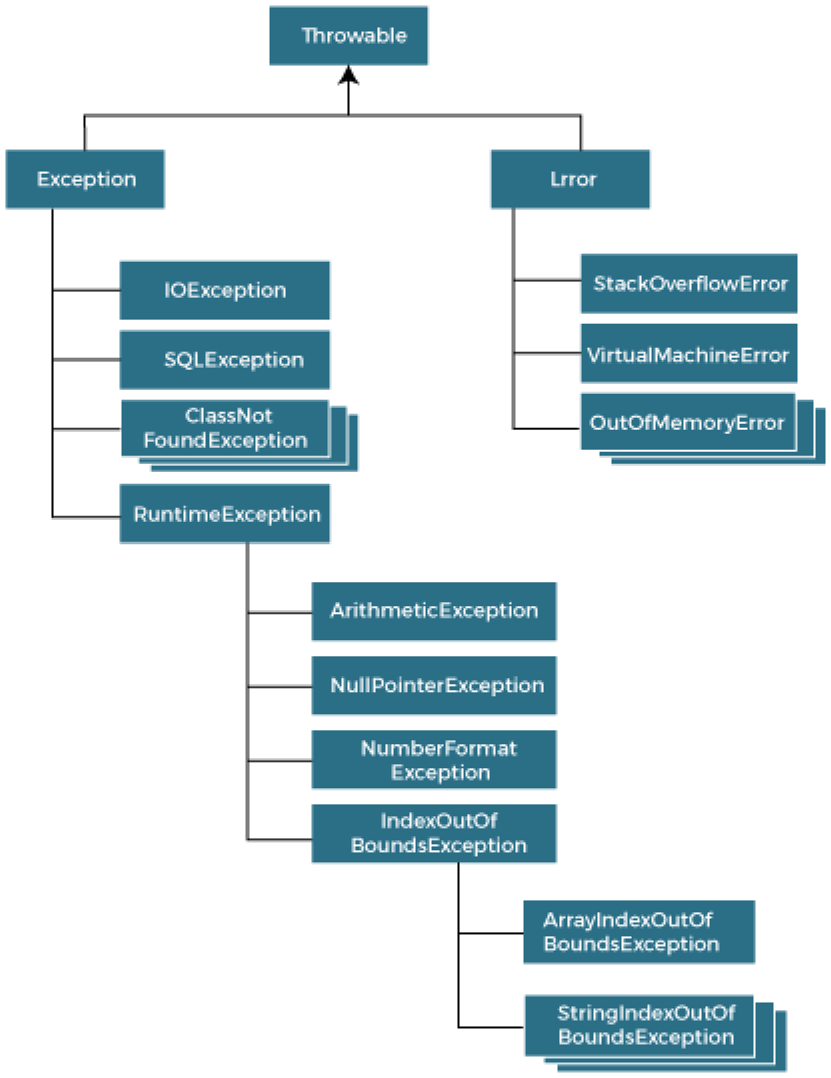
Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed. That is why we use exception handling in Java.

Do You Know?

- What is the difference between checked and unchecked exceptions?
- What happens behind the code int data=50/0;?
- Why use multiple catch block?
- Is there any possibility when the finally block is not executed?
- What is exception propagation?
- What is the difference between the throw and throws keyword?
- What are the 4 rules for using exception handling with method overriding?

Hierarchy of Java Exception classes

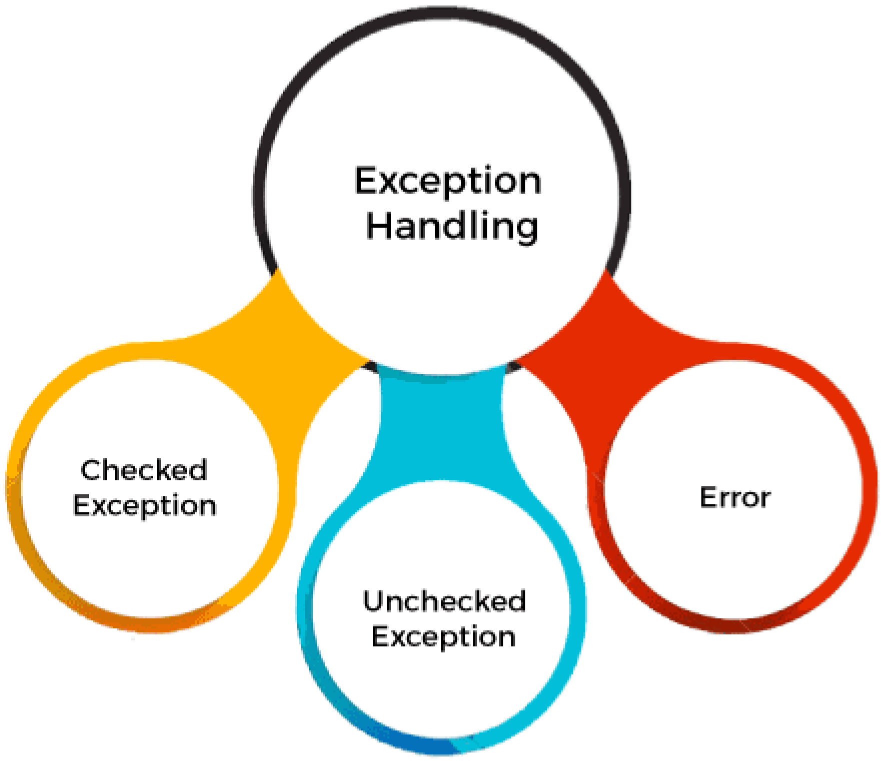
The java.lang.Throwable class is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error. The hierarchy of Java Exception classes is given below:



Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1. Checked Exception
2. Unchecked Exception
3. Error



Difference between Checked and Unchecked Exceptions

1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Keywords

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

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