Exception Handling in Java

Exception Handling in Java

- Exception handling in Java is a powerful mechanism or technique that allows us to handle runtime errors in a program so that the normal flow of the program cannot be stopped.
- All the exceptions occur only at runtime.
- A syntax error occurs at compile time.

Exception Handling in Java

- An exception can be identified only at runtime, not at compile time. Therefore, it is also called runtime errors that are thrown as exceptions in Java. They occur while a program is running.

- When JVM faces these kinds of runtime errors in a program, it creates an exception object and throws it to inform us that an error has occurred.

- If the exception object is not caught and handled properly, JVM will display an error message and will terminate the rest of the program abnormally.

- By handling the occurrence of exception, we can provide a meaningful message to the user about the error rather than a system-generated error message is nothing but a Exception Handling in Java.

Advantage of Exception Handling

1. The main advantage of exception handling technique is to maintain the normal flow of the program.
2. It provides flexibility in handling situations of errors.
3. It allows us to define a user-friendly message to handle the exception.
4. The exception handling technique helps to separate “error-handling code” from “regular code”.

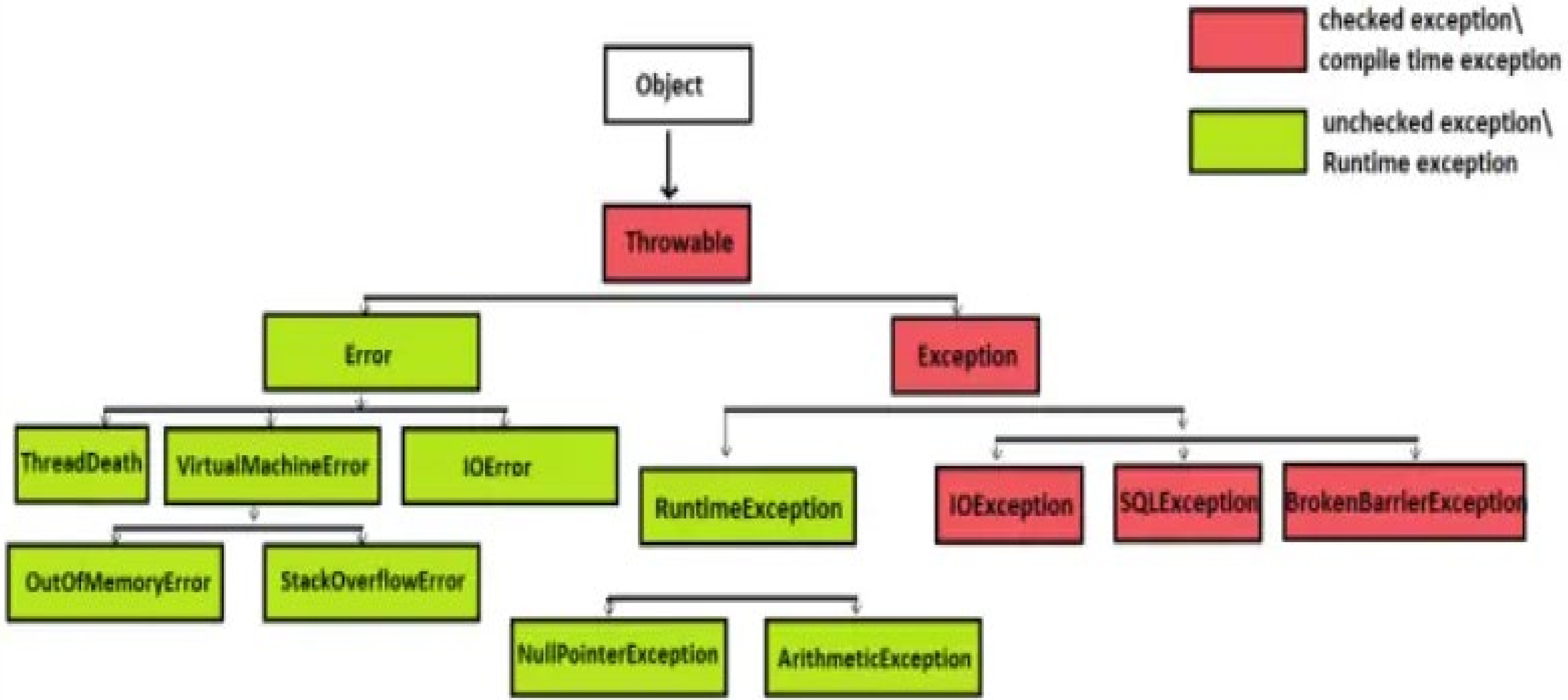
Types of Exceptions in Java

There are two types of exceptions in java API. They are:

1. Predefined Exceptions (Built-in-Exceptions)
2. Custom Exceptions (User defined Exception)

- When a predefined exception occurs, JVM (Java runtime system) creates an object of predefined exception class.
- All exceptions are derived from **java.lang.Throwable** class but not all exception classes are defined in the same package.
- All the predefined exceptions supported by java are organized as subclasses in a hierarchy under the **Throwable** class.
- The **Throwable** class is the root of exception hierarchy and is an immediate subclass of Object class.

The hierarchy of Java Exception classes



The hierarchy of Java Exception classes

Object: Object is the super most class of all classes available in java.

Throwable: For all types of exceptions **java.lang.Throwable** is the superclass. It has two main subclasses

1. Error
2. Exception

Error: Error is the super-most class of all exceptions that occur because of JVM failure. These errors we cannot handle.

Exception: Exception is the super most class of all exceptions that may occur because of programmer logic failure. These exceptions we can handle.

Differences between Error and Exception

Error class is the subclass of **Throwable** class and a superclass of all the runtime error classes.

It terminates the program if there is problem-related to a system or resources **(JVM).**

It does not occur by programmer mistakes. It generally occurs if the system is not working properly or resource is not allocated properly.

Example:

1. VirtualMachineError,
2. StackOverFlowError,
3. AssertionError,
4. LinkageError,
5. OutOfMmeoryError etc.

Differences between Error and Exception (Conti..)

Exception type exceptions are thrown due to the problem that occurred in java program logic, like If we divide an integer number with zero, then JVM terminates program execution by throwing Exception type exception "**java.lang.ArithmeticException**".

If we pass the array size as a negative number, then JVM terminates the program execution by throwing the exception type exception "**java.lang.NegativeArraySizeException**".

The exception class provides two constructors:

1. public Exception() (Default constructor)
2. public Exception(String message) (It takes a message string as argument)

Differences between Error and Exception (Conti..)

We cannot catch an Error type exception because an error type exception is not thrown in our application and once this error type exception is thrown JVM is terminated.

We can catch an Exception type exception because an exception type exception is thrown in our program and moreover JVM is not directly terminated because of an exception type exception. JVM is only terminated if the thrown exception is not caught.

predefined exceptions Exceptions in Java

We have the following two types of Exceptions:

1. Checked Exceptions
2. Unchecked Exceptions (**RuntimeException)**

Checked Exceptions in Java

Exceptions that are identified at compilation time and occurred at runtime are called checked exceptions.

These checked exceptions are also called Compile Time Exceptions.

An exception said to be checked exception whose exception handling is mandatory as per the compiler.

Example:

IOException, ClassNotFoundException, CloneNotSupportedException, FileNotFoundException, InterruptedException, NoSuchMethodException, NoSuchFieldException, IllegalAccessException, InstantiationException, InterruptedException etc.

Unchecked Exceptions in Java

Exceptions that are identified and occurred at run-time are called **Unchecked Exceptions**.

These Unchecked Exceptions are also called **Runtime Exceptions**.

An exception is said to be an unchecked exception whose exception handling is optional as per the compiler.

Example:

ArithmeticException, NumberFormatException, NoSuchMethodError, . ClassCastException, IllegalArgumentException, NumericFormatException,

IllegalThreadStateException, IndexOutOfBoundsException, NullPointerException, ArrayStoreException, NegativeArraySizeException

Difference between Checked and Unchecked Exceptions

- The classes that directly inherit **Throwable** class except **RuntimeException** and **Error** are called checked exceptions whereas, classes that directly inherit **RuntimeException** are called unchecked exceptions.
- **Checked** exceptions are checked and handled at compile-time whereas, unchecked exceptions are not checked and handled at compile time. They are checked at runtime.
- **Checked** exceptions in java extends Exception class, whereas unchecked exceptions extends RuntimeException class.
- **Checked** exception happens when there is a chance of a higher failure rate. whereas unchecked exceptions occur, mostly due to programming mistakes/errors.

The Exception Handling in Java is a 4 steps procedure

1. Preparing the exception object appropriate to the current logical mistake.
2. Throwing that exception to the appropriate exception handler.
3. Catching that exception
4. Taking necessary actions against that exception

How can we handle an exception in Java?

There are two ways to handle the exception in Java.

1. Logical implementation
2. Try catch implementation

Keywords to handle Exception in Java ?

Java provides five essential keywords to handle an exception.
They are as:

1. try
2. catch
3. finally
4. throw
5. throws

Handling Exception in Java

The three possible forms of try block are as follows:

1. **try-catch:** A try block is always followed by one or more catch blocks.
2. **try-finally:** A try block followed by a finally block.
3. **try-catch-finally:** A try block followed by one or more catch blocks followed by a finally block.

Syntax of try-catch block

```
try
{
  statement 1;
  statement 2;
  statement 3;
}
catch(exception_class  var)
{
  statement 4;
}
statement 5;
```

Rules for using Try-Catch block in Java

There are some rules for using try-catch block in java program. They are as follows:

1. Java try-catch block must be within a method.
2. A try block can not be used without a catch or finally block. It must be followed by at least one catch block otherwise, the compilation error will occur.
3. A catch block must be followed by try block. There should not be any statement between the end of try block and the beginning of catch block.
4. A finally block cannot come before catch block.

Points to Remember:

1. An exception can be handled using try, catch, and finally blocks.
2. We can handle multiple exceptions using multiple catch blocks.
3. There can be a possibility for several exceptions inside the try block but at a time only one exception will be raised.
4. A single try block in Java can be followed by several catch blocks.
5. A catch block cannot be without try block but a try block can have without catch block.
6. We cannot write any statement between try and catch blocks.
7. We can also write a try block within another try block that is called nested try blocks.

Finally Block in Java

- A finally block contains all the crucial codes such as closing connections, stream, etc that is always executed whether an exception occurs within a try block or not.
- When finally block is attached with a try-catch block, it is always executed whether the catch block has handled the exception thrown by try block or not.
- Syntax:

```
try
{
//  statements;
}
finally
{
 // statements;
}
```

Important rules of using finally block

1. A finally block is optional but at least one of the catch or finally block must exist with a try block.

2. It must be defined at the end of last catch block. If finally block is defined before a catch block, the program will not compile successfully.

3. Unlike catch, multiple finally blocks cannot be declared with a single try block. That is there can be only one finally clause with a single try block.

Use of finally block in Java

1. Finally block or clause is used for freeing up resources, cleaning up code, database closing connection, io stream, etc.

2. A java finally block is used to prevent resource leak. While closing a file or recovering resources, the code is put inside the finally block to ensure that the resource is always recovered.

Conditions where finally block does not execute

1. When **System.exit()** method is invoked before executing finally block.

2. When an exception happens in the finally block.

3. When the return statement is declared in the finally block, the control is transferred to the calling routine, and statements after return statement inside finally block will not be executed.

Throw Keyword in Java

In all the Java Programs of exception handling , Java runtime system (JVM) was responsible for identifying exception class, creating its object, and throwing that object.

JVM automatically throws system-generated exceptions. All those exceptions are called implicit exceptions.

If we want to throw an exception manually or explicitly, for this, Java provides a keyword **throw**.

Using throw keyword, we can throw either checked or unchecked exceptions in Java programming.

Points to remember:

1. In Java exception handling, we use throw keyword to throw a single exception explicitly. It is followed by an instance variable.
2. Using throw keyword, we can throw either checked or unchecked exception in Java.
3. The keyword throw raises an exception by creating a subclass object of Exception explicitly.
4. We mainly use throw keyword to throw custom exception on the basis of some specified condition.
5. We use keyword throw inside the body of method or constructor to invoke an exception.
6. With the help of throw keyword, we cannot throw more than one exception at a time.

Syntax of Throw Keyword

Syntax:

```
throw exception_name;
```

Here, **exception_name** is a reference to an object of **Throwable** class or its subclass.

For example:

1) throw new ArithmeticException();

    Or,

    ArithmeticException ae = new ArithmeticException();

    throw ae;

2) throw new NumberFormatException();

Throws Keyword

1. Throws keyword in Java is used in the method declaration.
2. It provides information to the caller method about exceptions being thrown and the caller method has to take the responsibility of handling the exception.
3. Throws keyword is used in case of checked exception only.
4. When the code generates a checked exception inside a method but the method does not handle it, Java compiler detects it and informs us about it to handle that exception.

Points to remember:

1. In Java exception handling, we use throws keyword to define a list of exceptions which may be thrown by that method.

2. We can use throws keyword in method or constructor declaration (signature) to denote exceptions that can be possibly thrown by that method.

3. Throws is followed by the exception class name.

4. With throws clause, we can declare more than one exception.

5. When throws keyword is used with a method declaration, the method calling a method with throws keyword must be enclosed within try-catch block.

Syntax to declare Throws Keyword

Syntax:

```
access_specifier return_type method_name(parameter list) throws exception
{
  // body of the method.
}
```

Or

```
access_specifier return_type method_name(parameter_list) throws
exception1, exception2, . . . .  exceptionN
{
  // body of the method.
}
```

Difference between Throw and Throws

1. The keyword throw is used to throw an exception explicitly, while throws clause is used to declare an exception.

2. Throw is followed by an instance variable, while throws is followed by the name of exception class.

3. We use throw keyword inside method body to call an exception, while throws clause is used in method signature.

4. With throw keyword, we cannot throw more than one exception at a time, while we can declare multiple exceptions with throws.

User defined Exception in Java

- User-defined exceptions in Java are those exceptions that are created by a programmer (or user) to meet the specific requirements of the application.
- That’s why it is also known as user-defined exception. It is useful when we want to properly handle the cases that are highly specific and unique to different applications.

How to Create Your Own User-defined Exception in Java?

1. User-defined exceptions can be created simply by extending the Exception class.

```
class MyClass extends Exception
```

2. If you do not want to store any exception details, define a default constructor in your own exception class.

```
MyClass()
{


}
```

How to Create Your Own User-defined Exception in Java? (Conti..)

3. If you want to store exception details, define a parameterized constructor with string as a parameter, call the superclass (Exception) constructor

```
MyClass(String str)
{
    super(str);        // Call superclass exception constructor
}
```

4. In the last step, we need to create an object of the user-defined exception class and throw it using throw clause.

```
MyClass obj = new MyClass ("Exception details");
throw obj;
or,
throw new MyClass ("Exception details");
```

Chained Exceptions in Java

1. Chained exception in Java is a technique to handle exceptions that occur one after another in a program. This technique helps us to know when one exception causes another in a program.

2. This process is called chaining of exceptions in Java, and the exceptions involved in this process are called chained exceptions. This feature helps the programmer to know when and where is the cause for the exception.

Throwable Class

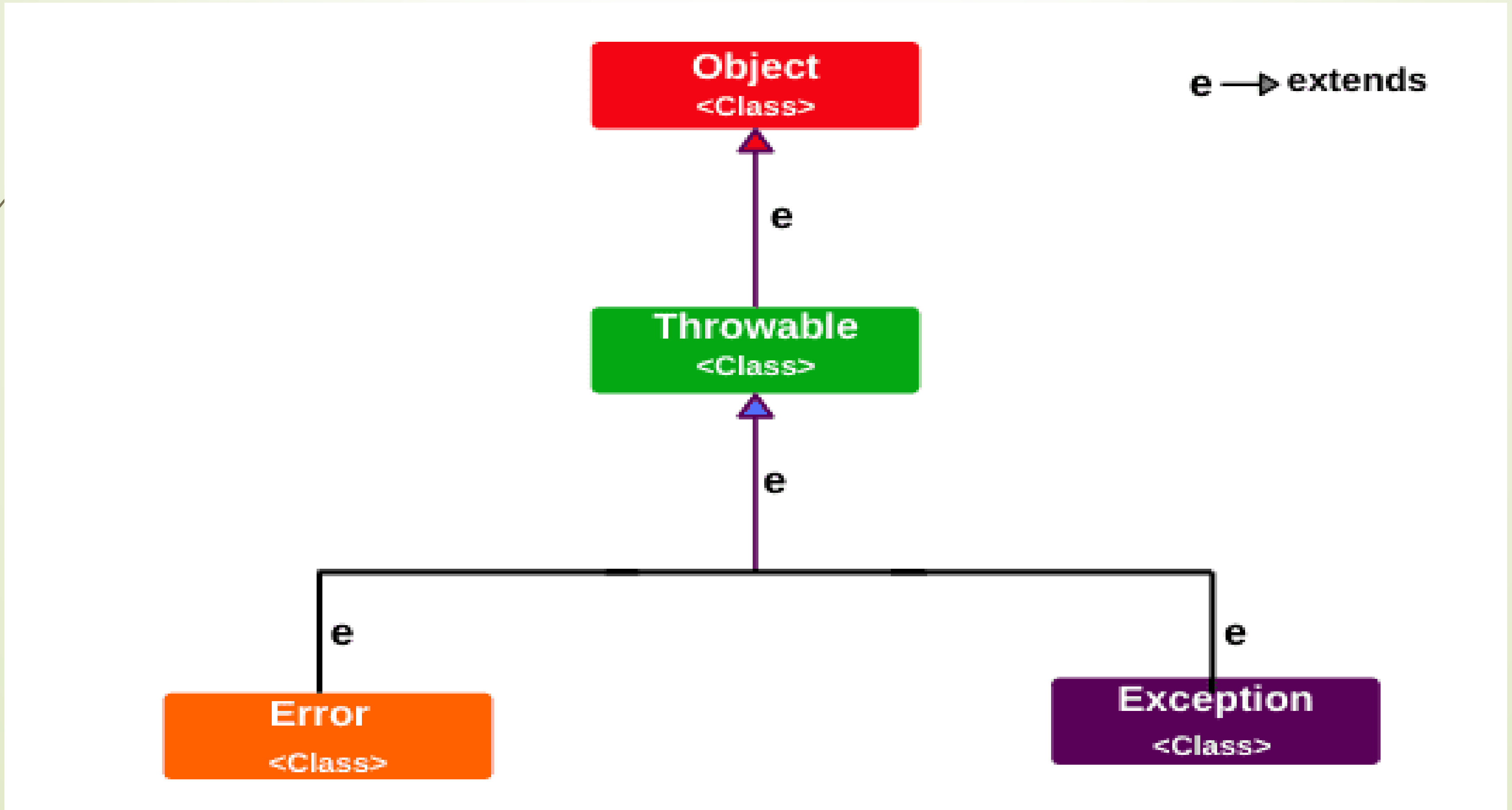
Throwable in Java is a predefined class that is the superclass of all kinds of exceptions and errors which may occur in Java program. This class is present in java.lang package.

Throwable class extends Object class, which is at the root of the class hierarchy. Therefore, **Throwable** class is itself a subclass of the superclass of all Java classes, Object class.

JVM throws only those exception objects, which are instances of **Throwable** class or one of its subclasses. In other words, when an exceptional situation occurs in a Java program, **JVM** generates an exception object to represent that situation. And this exception object is of a type derived from the **Throwable** class.

Throwable Class Hierarchy in Java

Throwable class is the top most class in the exception class hierarchy but it is also an immediate subclass of Object class that is located at the root of exception hierarchy.



Constructors of Throwable Class to Support Chained Exceptions

To implement chained exceptions feature in JDK 1.4 version, Java added two constructors and two methods to **Throwable** class.

1. Throwable(Throwable causeExc):

This form of constructor creates a new **Throwable** object with the specified cause. It takes only one parameter: **Throwable** causeExc, which represents an exception that causes the current exception.

If the **causeExc** is null, it will return the null of message. Otherwise, it will return string representation of the message. The message contains the name of class and the detailed information of the cause.

2. Throwable(String msg, causeExc):

This form of constructor creates a new **Throwable** object with the specified detail message and cause. It takes two parameters: String **msg** and **Throwable** causeExc. Here, **msg** is an exception message and **causeExc** is an exception that causes the current exception.

Methods of Throwable Class to Support Chained Exceptions

1. toString(): This method returns an exception followed by a description of the exception. The general syntax is as follows:

public String toString()

2. getMessage(): It returns the description of exception. The general syntax is as:

public String getMessage()

3. printStackTrace(): This method displays stack trace. It returns nothing. The general syntax is given below.

public void printStackTrace()

4. getCause(): The **getCause**() method returns the exception that caused the occurrence of current exception. If there is no caused exception then null is returned. The syntax is as follows:

public Throwable getCause()

Assertion (JDK 1.4)

1. JDK 1.4 introduced a new keyword called **assert**, to support the so-called assertion feature.
2. Assertion enables you to test your assumptions about your program logic (such as pre-conditions, post-conditions, and invariants).
3. Each assertion contains a **boolean expression** that you believe will be true when the program executes.
4. If it is not true, the **JVM** will throw an **AssertionError**.
5. This error signals you that you have an invalid assumption that needs to be fixed.
6. **Assertion** is much better than using if-else statements, as it serves as proper documentation on your assumptions, and it does not carry performance liability in the production environment

Assertion (JDK 1.4)

The assert statement has two forms:

1. **assert** booleanExpr;
2. **assert** booleanExpr : errorMessageExpr;