**Exceptions:**

**\*What is base class of exception?**



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:** Checked exceptions are the one which are checked at compile-time. For example, SQLException, ClassNotFoundException, etc.

**Unchecked Exception:** Unchecked exceptions are the one which are handled at runtime because they can not be checked at compile-time. For example, ArithmaticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.

**Error:** Error cause the program to exit since they are not recoverable. For Example, OutOfMemoryError, AssertionError, etc.

**Exception Handling is a mechanism that is used to handle runtime errors. It is used primarily to handle checked exceptions. Exception handling maintains the normal flow of the program.**

**\*How to create custom excetions?**

* **Create** a new class whose name should end with **Exception** like ClassNameException. ...
* Make the class extends one of the exceptions which are subtypes of the **java**. ...
* **Create** a constructor with a String parameter which is the detail message of the **exception**.

**The following is a custom exception class which is created by following the above steps:**

public class StudentNotFoundException extends Exception {

    public StudentNotFoundException(String message) {

        super(message);

    }

}

**And the following example shows the way a custom exception is used is nothing different than built-in exception:**

public class StudentManager {

    public Student find(String studentID) throws StudentNotFoundException {

        if (studentID.equals("123456")) {

            return new Student();

        } else {

            throw new StudentNotFoundException(

                "Could not find student with ID " + studentID);

        }

    }

}

**And the following test program handles that exception:**

public class StudentTest {

    public static void main(String[] args) {

        StudentManager manager = new StudentManager();

        try {

            Student student = manager.find("0000001");

        } catch (StudentNotFoundException ex) {

            System.err.print(ex);

        }

    }

}

**\*In which case finally block does not executes?**

A **finally block** will **not execute** due to other conditions like when JVM runs out of memory when our java process is killed forcefully from task manager or console when our machine shuts down due to power failure and deadlock condition in our try **block**.

The finally block always executes when the try block exits. This ensures that the finally block is executed even if an unexpected **exception** occurs.

**\*What are types of exceptions?**

1. Buit-in exceptions

2. User defined exceptions

 **ArithmeticException**   
It is thrown when an exceptional condition has occurred in an arithmetic operation.

 **ArrayIndexOutOfBoundsException**  
It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

 **ClassNotFoundException**  
This Exception is raised when we try to access a class whose definition is not found

 **FileNotFoundException**  
This Exception is raised when a file is not accessible or does not open.

 **IOException**  
It is thrown when an input-output operation failed or interrupted

 **InterruptedException**  
It is thrown when a thread is waiting , sleeping , or doing some processing , and it is interrupted.

 **NoSuchFieldException**  
It is thrown when a class does not contain the field (or variable) specified

 **NoSuchMethodException**  
It is thrown when accessing a method which is not found.

 **NullPointerException**  
This exception is raised when referring to the members of a null object. Null represents nothing

 **NumberFormatException**  
This exception is raised when a method could not convert a string into a numeric format.

 **RuntimeException**  
This represents any exception which occurs during runtime.

 **StringIndexOutOfBoundsException**  
It is thrown by String class methods to indicate that an index is either negative or greater than the size of the string

**\*Examples of checked and unchecked exceptions?**

Checked exceptions are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using [try-catch block](https://beginnersbook.com/2013/04/try-catch-in-java/) or it should declare the exception using [throws keyword](https://beginnersbook.com/2013/04/java-throws/), otherwise the program will give a compilation error.

**E.g** FileInputStream which is used for specifying the file path and name, throws FileNotFoundException. The read() method which reads the file content throws IOException and the close() method which closes the file input stream also throws IOException.

**Unchecked exceptions** are not checked at compile time. It means if your program is throwing an unchecked exception and even if you didn’t handle/declare that exception, the program won’t give a compilation error. All Unchecked exceptions are direct sub classes of **RuntimeException** class

e.g.it would throw ArithmeticException.

**\*Can we handle the runtime exceptions?**

**No.**

**Runtime exceptions** represent problems that are a direct result of a programming problem, and as such shouldn't be **caught** since it **can**'t be reasonably expected to recover from them or handle them

A user should not attempt to **handle** this kind of an exception because it will only patch the problem and not completely fix it.

**\*Difference between class cast exceptions and arithmetic exceptions?**

**ArithmeticException** is an unchecked **exception** in Java. Usually, one would come across java. lang. **ArithmeticException**: / by zero which occurs when an attempt is made to divide two numbers and the number in the denominator is zero.

**ClassCastException** is a runtime **exception** raised in **Java** when we try to improperly **cast** a **class** from one type to another

difference throw and throws in java?

The main difference between Throw and Throws keywords in Java programming is that **you can use the Throw keyword to throw an exception explicitly in the code.**

**We have use this keywork in catch block to throw exception explicitly.**

**you can use the Throws keyword to declare that a method might throw an exception in the code**.

We have to define throws keyword in the method definition.