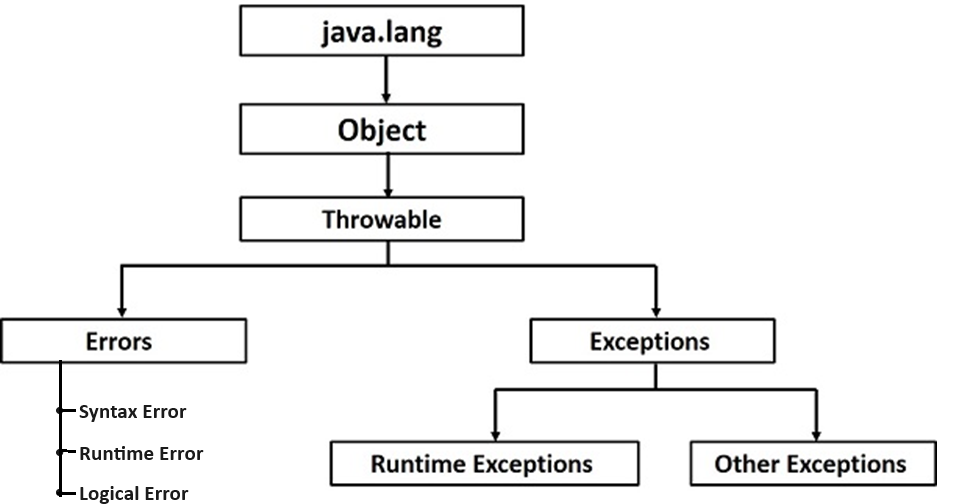
|  |  |
| --- | --- |
| Week6 | Exception Handling |

When executing Java code, different errors can occur in which Java program will normally stop and generate an error message (i.e, **exception** ).

|  |  |
| --- | --- |
| **Error** | **Exception** |
| **Error** is an fatal & irrecoverable event that cannot be handled at runtime.  **Ex:**OutOfMemoryError, StackOverflowError. | **Exception** is an non-fatal & recoverable event that can be handled at runtime.  **Ex:**IOException, NullPointerException |

****

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.

|  |  |
| --- | --- |
| **Without Exception Hndling** | **With Exception Handling** |
| **public** **class** ExceptionDemo {  **private** **static**  String inputString = "123.33";  **public** **static** **void** main(String[] args) {  **int** a = Integer.parseInt(inputString);  }  }  **Output**:  Exception in thread “main”: java.lang.NumberFormatException:null  **Note**:-program terminates abnormally. | **public** **class** ExceptionDemo {  **private** **static**  String inputString = "123.33";  **public** **static** **void** main(String[] args) {  **try** {  **int** a = Integer.parseInt(inputString);  }**catch**(NumberFormatException ex){  System.err.println("Invalid string in argumment");   }  }  }  **Output**: Invalid string in argumment |

The **try statement** allows you to define a block of code to be tested

The **catch statement** allows you to define a block of code to be executed, if an error

# INTERVIEW QUESTION

Rules for writing try and catch blocks:

|  |  |  |
| --- | --- | --- |
| **Rule1:**try block must be followed by catch block | **Rule2:**try block can have multiple catch blocks | **Rule3:**finally block must be last block after try-catch blocks |
| **try {**  // Block of code to try  **} catch(Exception e) {**  // catch block1  **}** | **try {**  // Block of code to try  **}catch(Exception e) {**  // catch block1  **} catch(Exception e) {**  // catch block2  **}** | **try {**  // Block of code to try  **}catch(Exception e1) {**  // catch block1  **} catch(Exception e2) {**  // catch block2  **}finally {**  // Optional: Code that will be executed regardless of whether an exception occurred or not  **}** |

|  |  |
| --- | --- |
| Rule4: For each try block there can be 0 or more catch blocks, but only one finally block. | |
| **try {**  // Block of code to try  **}catch(Exception e1) {**  // catch block1  **} catch(Exception e2) {**  // catch block2  **}finally {**  // Code that will be executed regardless of whether an exception occurred or not } | **try {**  // Block of code to try  **}finally {**  // Code that will be executed regardless of whether an exception occurred or not } |

# INTERVIEW QUESTION

# Q. final vs finally vs finalize in Java

|  |  |  |
| --- | --- | --- |
| **final** | **finally** | **finalize** |
| final (lowercase) is a reserved keyword in java  **final with Variables:**The value of the variable cannot be changed once initialized.  **final with Class:** The class cannot be inherited  **final with Method:** The method cannot be overridden by a subclass. | Finally is a reserved keyword in java  A finally block of code is always executed whether an exception has occurred or not. | Finalize is a method in java  The finalize method in java is called by the garbage collector before an object is garbage collected. This method can be used to perform any necessary cleanup before the object is destroyed, |

# INTERVIEW QUESTION

Q. Differentiate throw & throws

|  |  |
| --- | --- |
| **throw** | **throws** |
| Keyword used to throw the Exception explicitly. | If Exception is not handled then “throws” Keyword used for propagating the Excepttion to the next level |

**USER DEFINED EXCEPTION**

In Java, a user defined exception is a powerful tool for dealing with specific requirements and scenarios. They enable programmers to provide more descriptive error messages and manage exceptions more effectively and efficiently.

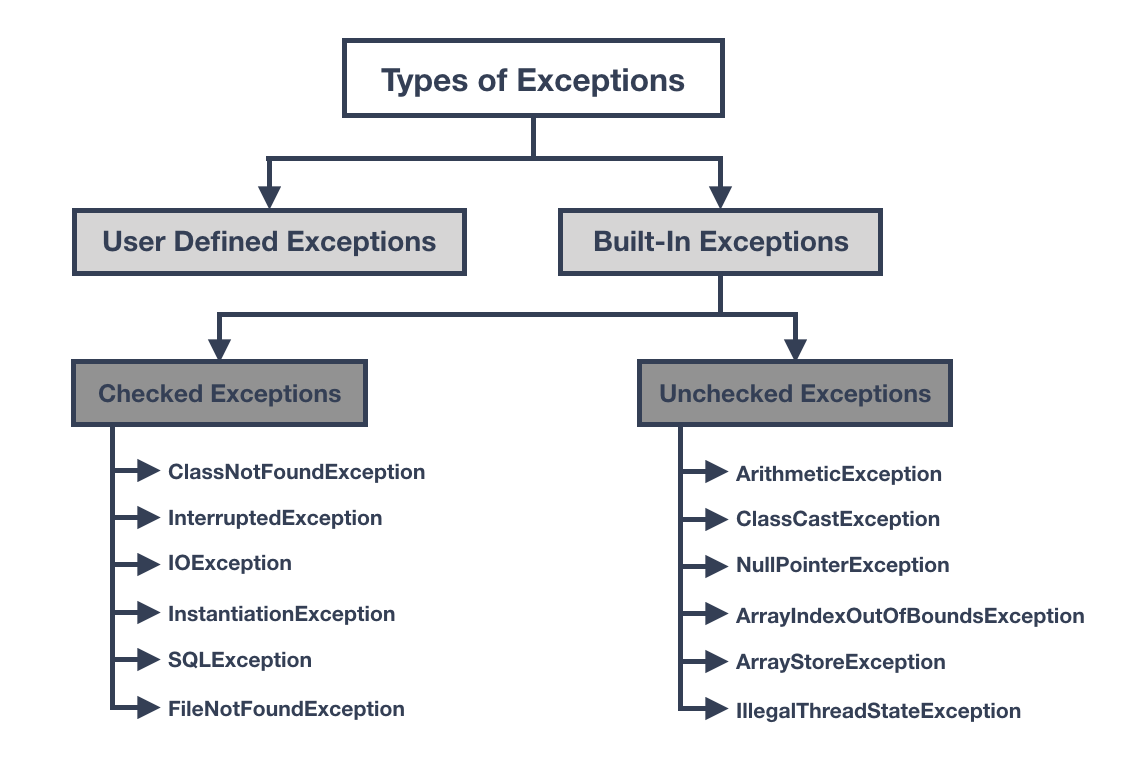
In order to create custom exception, we need to extend Exception class that belongs to java.lang package.

To implement a user-defined exception in Java, follow these steps:

1. Create a custom exception class that extends the base exception class (java.lang.Exception).
2. Define the constructor for the custom exception class. The constructor can accept parameters to provide more information about the error.
3. Override the toString() method to provide a custom error message.
4. Use the "throw" keyword to throw an instance of the custom exception when the error condition occurs.
5. Use a "try-catch" block to catch the exception and handle it appropriately.

**CHECKED vs UNCHECKED EXCEPTIONS**

|  |  |
| --- | --- |
| **CHECKED EXCEPTIONS** | **UNCHECKED EXCEPTIONS** |
| In Java, checked exceptions are exceptions that the Java compiler checks for and requires you to handle or declare in your code. Checked exceptions are typically used to indicate a recoverable error, such as when a file cannot be found or a network connection cannot be established.  Java compiler raises a compile-time error if a checked exception is left unhandled or undeclared.  **Example:-**  **IO Exception**  **SQL Exception**  **Interrupted Exception** | In Java, unchecked exceptions are exceptions that the Java compiler does not enforce checking for. These exceptions are usually associated with runtime errors that occur when a program violates an assumption about the system’s state, such as going beyond the bounds of an array or encountering a null pointer  **Example:-**  **NullPointer Exception**  **Arithmetic Exception**  **Array Index Out Of Bounds Exception** |



| **Feature** | **Checked Exceptions** | **Unchecked Exceptions** |
| --- | --- | --- |
| Definition | Exceptions that must be either caught or declared in the method signature. | Exceptions that do not need to be handled. |
| Use case | For conditions that can be reasonably recovered from. | For conditions that cannot be recovered from, such as programming errors. |
| Propagation | Must be either caught or declared in the method signature. | Can be propagated up the call stack without being caught. |
| Coding overhead | More coding overhead due to the requirement to handle or declare exceptions. | Less coding overhead, since exceptions do not need to be handled. |
| Example | FileNotFoundException, IOException, SQLException. | NullPointerException, ArrayIndexOutOfBoundsException, IllegalArgumentException. |
| Syntax | Must be surrounded by a try-catch block or declared in the method signature. | Do not need to be surrounded by a try-catch block |

**/\***

**1) ArrayIndexOutOfBoundsException**

**2) IOException**

**3) RuntimeException**

**4) FileNotFoundException**

**5) Exception**

**1 4 2 3 5**

**1 3 4 2 5**

**4 2 1 3 5**

**4 1 2 3 5**

**\*/**

# INTERVIEW QUESTION

**Q. What is the difference between checked and unchecked exceptions in Java?**Checked exceptions in Java are exceptions that must be explicitly caught or declared in the method that throws them, whereas unchecked exceptions do not. Unless they are derived from the base unchecked exception class (java.lang.RuntimeException), user-defined exceptions are typically treated as checked exceptions.