In this post, we will discuss the difference between checked and unchecked exceptions in Java with examples.

**Checked Exceptions**

**Definition:**Exceptions that are checked at compile-time are called checked exceptions.

**Subclass of:** Directly or indirectly derived from *java.lang.Exception* but not from *java.lang.RuntimeException*.

**Handling Requirement:**A checked exception must be handled either by re-throwing or with a try-catch block.

**Purpose:** Represent abnormal situations that can occur in the program and which can be reasonably anticipated and recovered from. Examples: FileNotFoundException, IOException, ClassNotFoundException.

**Recovery:** Developers are expected to provide recovery mechanisms for these exceptions.

**Example:**Reading from a file that doesn't exist will throw a *FileNotFoundException*, which is a checked exception.

import java.io.\*;

public class CheckedExample {

public static void main(String[] args) {

try {

BufferedReader reader = new BufferedReader(new FileReader("nonExistentFile.txt"));

String line = reader.readLine();

while (line != null) {

System.out.println(line);

line = reader.readLine();

}

reader.close();

} catch (FileNotFoundException e) {

System.out.println("Error: " + e.getMessage());

} catch (IOException e) {

System.out.println("IO Error: " + e.getMessage());

}

}

}

Output:

Error: nonExistentFile.txt (No such file or directory)

**Unchecked Exceptions**

**Definition:**Exceptions that are not checked at compile-time but are checked at runtime are called unchecked exceptions.

**Subclass of:** Derived from *java.lang.RuntimeException* and *java.lang.Error*.

**Handling Requirement:**Unchecked exception isn’t required to be handled.

**Purpose:** Mainly arise due to programming mistakes, incorrect assumptions, or logical errors.

**Examples:**NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException.

**Recovery:** Often, the best remedy is to fix the code that led to the exception rather than attempting recovery during runtime.

**Example:**Attempting to access an index of an array that doesn't exist will result in *ArrayIndexOutOfBoundsException*, which is an unchecked exception.

public class UncheckedExample {

public static void main(String[] args) {

int[] arr = {1, 2, 3};

try {

System.out.println("Value at index 5 is: " + arr[5]);

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Error: " + e.getMessage());

}

}

}

Output:

Error: Index 5 out of bounds for length 3

**Difference Between Checked and Unchecked Exceptions in Java**

Here's a comparison table for checked vs. unchecked exceptions:

| **Criteria** | **Checked Exceptions** | **Unchecked Exceptions** |
| --- | --- | --- |
| **Definition** | Checked at compile-time. | Checked at runtime. |
| **Subclass of** | All the subclasses of *java.lang.Exception* are checked exceptions. | All the subclasses of *java.lang.RunTimeException* are unchecked exceptions. |
| **Handling Requirement** | A checked exception must be handled either by re-throwing or with a try-catch block. | An unchecked exception isn’t required to be handled. |
| **Purpose** | Anticipate and recover from abnormal situations that are external to the application. | Result from programming mistakes, incorrect assumptions, or logical errors in the code. |
| **Examples** | *FileNotFoundException*, *IOException*, *ClassNotFoundException*. | *NullPointerException*, *ArrayIndexOutOfBoundsException*, *ArithmeticException*. |
| **Recovery Strategy** | Expected to provide recovery mechanisms. | Typically fix the code rather than attempting runtime recovery. |