Types of Exceptions in Java

Java exceptions can be categorized into two major types:

1. Built-in Exceptions (checked, unchecked, and error)
2. Custom Exceptions (User-Defined Exceptions)

Let's explore each type in detail.

1. Built-in Exceptions

These are the exceptions that Java's standard library provides. They are essentially part of Java's API, and you've likely encountered many of them during programming.

1.1. Checked Exceptions

Checked exceptions are also known as **compile-time** exceptions as these exceptions are checked by the compiler during the compilation process to confirm whether the exception is handled by the programmer or not. If not, then the system displays a compilation error.

Java built-in checked exceptions:

- FileNotFoundException
- EOFException
- ClassNotFoundException
- SQLException
- NoSuchMethodException
- InterruptedException

Example: Let's see an example of a checked exception - FileNotFoundException

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;

public class CheckedExceptionDemo {
    public static void main(String[] args) {
        try {
            FileInputStream fis = new FileInputStream("nonexistentfile.txt");
        } catch (FileNotFoundException e) {
            System.out.println("Checked Exception: " + e.getMessage());
        }
    }
}
```

Output:

```
Checked Exception: nonexistentfile.txt (No such file or directory)
```

1.2 Unchecked Exceptions (Runtime Exceptions)

The unchecked exceptions are those exceptions that occur during the execution of the program. Hence they are also referred to as Runtime exceptions. These exceptions are generally ignored during the compilation process. They are not checked while compiling the program. For example, programming bugs like logical errors, and using incorrect APIs.

Java built-in unchecked exceptions:

- **NullPointerException**
- **ArrayIndexOutOfBoundsException**
- **StringIndexOutOfBoundsException**
- **ArithmeticException**
- **IllegalArgumentException**
- **NumberFormatException**
- **IllegalStateException**
- **ClassCastException**

Example: Let's see an example of an unchecked exception: **ArrayIndexOutOfBoundsException**

```
public class UncheckedExceptionDemo {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3};
        try {
            System.out.println(arr[5]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Unchecked Exception: " + e.getMessage());
        }
    }
}
```

Output:

```
Unchecked Exception: Index 5 out of bounds for length 3
```

Types of Exceptions in Java

author: Ramesh Fadatare

CORE JAVA EXCEPTION HANDLING

Java exceptions can be categorized into two major types:

1. Built-in Exceptions (checked, unchecked, and error)
2. Custom Exceptions (User-Defined Exceptions)

Let's explore each type in detail.

1. Built-in Exceptions

These are the exceptions that Java's standard library provides. They are essentially part of Java's API, and you've likely encountered many of them during programming.

1.1. Checked Exceptions

Checked exceptions are also known as **compile-time** exceptions as these exceptions are checked by the compiler during the compilation process to confirm whether the exception is handled by the programmer or not. If not, then the system displays a compilation error.

Java built-in checked exceptions:

- **FileNotFoundException**
- **EOFException**
- **ClassNotFoundException**
- **SQLException**
- **NoSuchMethodException**
- **InterruptedException**

Example: Let's see an example of a checked exception - **FileNotFoundException**

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;

public class CheckedExceptionDemo {
    public static void main(String[] args) {
        try {
            FileInputStream fis = new FileInputStream("nonexistentfile.txt");
        } catch (FileNotFoundException e) {
            System.out.println("Checked Exception: " + e.getMessage());
        }
    }
}
```

Output:

```
Checked Exception: nonexistentfile.txt (No such file or directory)
```

1.2 Unchecked Exceptions (Runtime Exceptions)

The unchecked exceptions are those exceptions that occur during the execution of the program. Hence they are also referred to as Runtime exceptions. These exceptions are generally ignored during the compilation process. They are not checked while compiling the program. For example, programming bugs like logical errors, and using incorrect APIs.

Java built-in unchecked exceptions:

- **NullPointerException**
- **ArrayIndexOutOfBoundsException**
- **StringIndexOutOfBoundsException**
- **ArithmeticException**
- **IllegalArgumentException**
- **NumberFormatException**
- **IllegalStateException**
- **ClassCastException**

Example: Let's see an example of an unchecked exception: **ArrayIndexOutOfBoundsException**

```
public class UncheckedExceptionDemo {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3};
        try {
            System.out.println(arr[5]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Unchecked Exception: " + e.getMessage());
        }
    }
}
```

Output:

```
Unchecked Exception: Index 5 out of bounds for length 3
```

1.3 Errors

These are exceptional conditions that are external to the application and it cannot anticipate or recover from them.

Java built-in errors:

- **OutOfMemoryError**
- **StackOverflowError**
- **NoClassDefFoundError**

Example: Let's see an example of **StackOverflowError**

```
public class ErrorDemo {
    public static void recursiveMethod() {
        recursiveMethod();
    }

    public static void main(String[] args) {
        try {
            recursiveMethod();
        } catch (StackOverflowError e) {
            System.out.println("Error: " + e.toString());
        }
    }
}
```

Output:

```
Error: java.lang.StackOverflowError
```

2. Custom Exceptions (User-Defined Exceptions)

Sometimes, built-in exceptions aren't enough. Developers may need to create custom exceptions tailored to specific application requirements.

They provide a means to represent domain-specific issues and can enhance code clarity and maintainability.

User-defined exceptions are created by extending the `Exception` class (for checked exceptions) or the `RuntimeException` class (for unchecked exceptions).

Example: Let's see an example of creating and using a custom exception:

```
class CustomException extends Exception {
    public CustomException(String message) {
        super(message);
    }
}

public class CustomExceptionDemo {
    public static void main(String[] args) {
        try {
            throw new CustomException("This is a custom exception!");
        } catch (CustomException e) {
            System.out.println("Caught Custom Exception: " + e.getMessage());
        }
    }
}
```

Output:

```
Caught Custom Exception: This is a custom exception!
```

Conclusion

In this article, we have learned about different types of exceptions in Java with example - build-in exceptions (checked, unchecked, and error) and custom exceptions (user-defined exceptions)