
Java Exception
Press Space for next page →

Agenda

1. What is Exception?
2. Java try block
3. Java catch block
4. Java Multiple catch block
5. Java finally block
6. Java Exception Hierarchy
7. Types of Exception
8. Java Exception Declaration
9. Java Throwing Exception

What is Exception?

- Error events that can occur during the execution of a program and disrupt the normal flow of instructions are called exceptions.
- When an exceptional situation occurs, an object of the Exception class or one of its subclasses is created and thrown.
- If this exception is not caught and handled properly, it will terminate the normal flow of the program and print an error message on the console.

> The core advantage of exception handling is to maintain the normal flow of the application.

Java try block

- Java try block is used to enclose the code that might throw an exception. It must be used within the method.
- If an exception occurs at the particular statement in the try block, the rest of the block code will not execute. **So, it is recommended not to keep the code in try block that will not throw an exception.**
- Java try block must be followed by either catch or finally block.

```
try {
    // code that might throw an exception
}
catch and finally blocks...
```

Java catch block

- Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception (i.e., Exception) or the generated exception type. However, the good approach is to declare the generated type of exception.
- The catch block must be used after the try block only. **You can use multiple catch block with a single try block.**

```
1   public class Main {
2       public static void main(String[] args) {
3           try {
4               int[] numbers = {1, 2, 3};
5               System.out.println(numbers[10]);
6           } catch (ArrayIndexOutOfBoundsException e) {
7               System.out.println("Array index out of b
8           }
9       }
10  }
```

Java Multiple catch block

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler.

So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

Points to remember:

- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from **most specific to most general**, i.e. catch for ArithmeticException must come before catch for Exception.

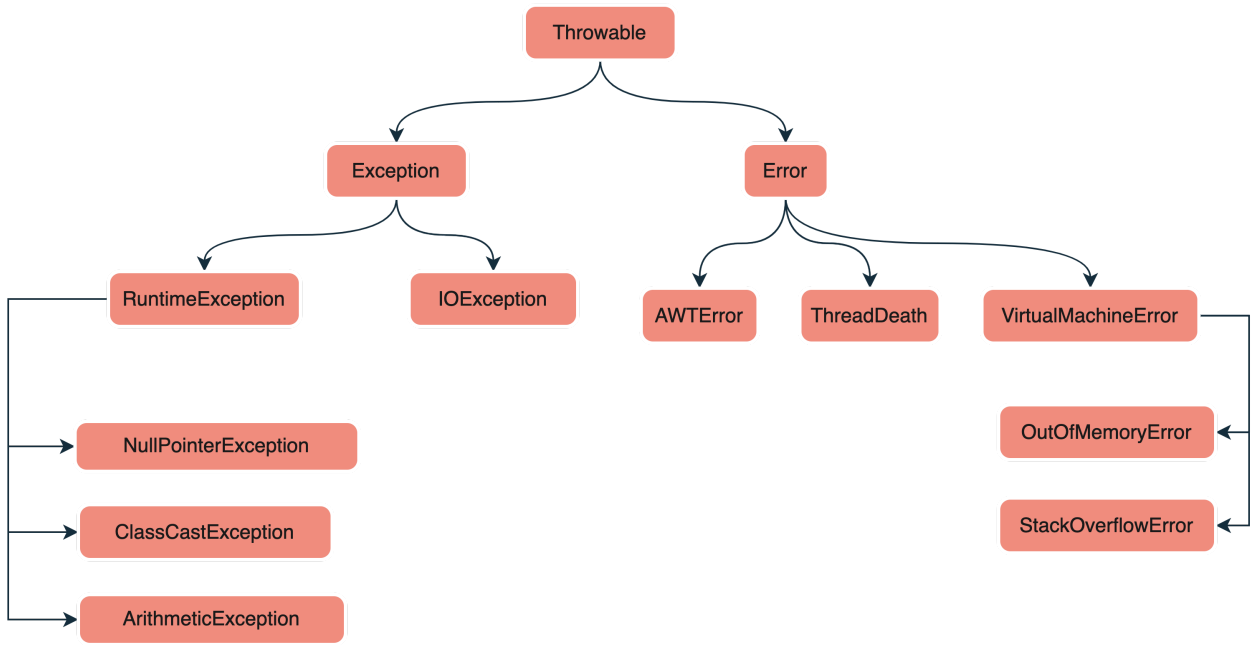
```
1   public class Main {
2       public static void main(String[] args) {
3           try {
4               int[] numbers = {1, 2, 3};
5               System.out.println(numbers[10]);
6           } catch (ArrayIndexOutOfBoundsException e) {
7               System.out.println("Array index out of b
8           } catch (Exception e) {
9               System.out.println("Parent exception occ
10          }
11      }
12  }
```

Java finally block

- Java finally block is a block used to execute important code such as closing the connection, etc.
- Java **finally block is always executed whether an exception is handled or not**. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.
- The finally block **follows the try-catch block**.

```
1  public class Main {
2      public static void main(String[] args) {
3          try {
4              int[] numbers = {1, 2, 3};
5              System.out.println(numbers[10]);
6          } catch (ArrayIndexOutOfBoundsException e) {
7              System.out.println("Array index out of b
8          } finally {
9              System.out.println("Finally block is alw
10         }
11     }
12 }
```

Java Exception Hierarchy



Types of Exception

- **Checked Exception**
  - The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions.
  - Example: `IOException`, `SQLException`, etc.
- **Unchecked Exception**
  - The classes that inherit the RuntimeException are known as unchecked exceptions.
  - Unchecked exceptions are not checked at compile-time, but they are checked at runtime.
  - Example: `ArithmeticException`, `NullPointerException`, `ArrayIndexOutOfBoundsException`, etc.
- **Error**
  - Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Declaration

> Tells the Java compiler that a method might throw an exception.

- The throws keyword is used in a method signature to declare the types of checked exceptions that the method may throw. This is required for **checked exceptions because the compiler needs to know which exceptions might be thrown** so that it can enforce proper exception handling.

```
public void method() throws IOException {
    // code
}
```

```
import java.io.FileReader;
import java.io.IOException;

public class FileOperations {
    // Declares that this method might throw an IOException
    public void readFile(String fileName) throws IOException {
        FileReader file = new FileReader(fileName);  // IOException may occur
        System.out.println("File opened successfully.");
    }
}
```

Java Throwing Exception

The throw keyword is **used to explicitly throw an exception**. This is useful when you want to create your own custom exceptions or when you want to throw an exception based on certain conditions.

```
throw new ArithmeticException("Arithmetic Exception");
```

```
public class AgeValidator {
  // Method to validate age, throws IllegalArgumentException if age is invalid
  public void validate(int age) throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Age is not valid");
        }
    }
}
```

Thank you ❤️

qa-june-2024-automation-with-java-slides