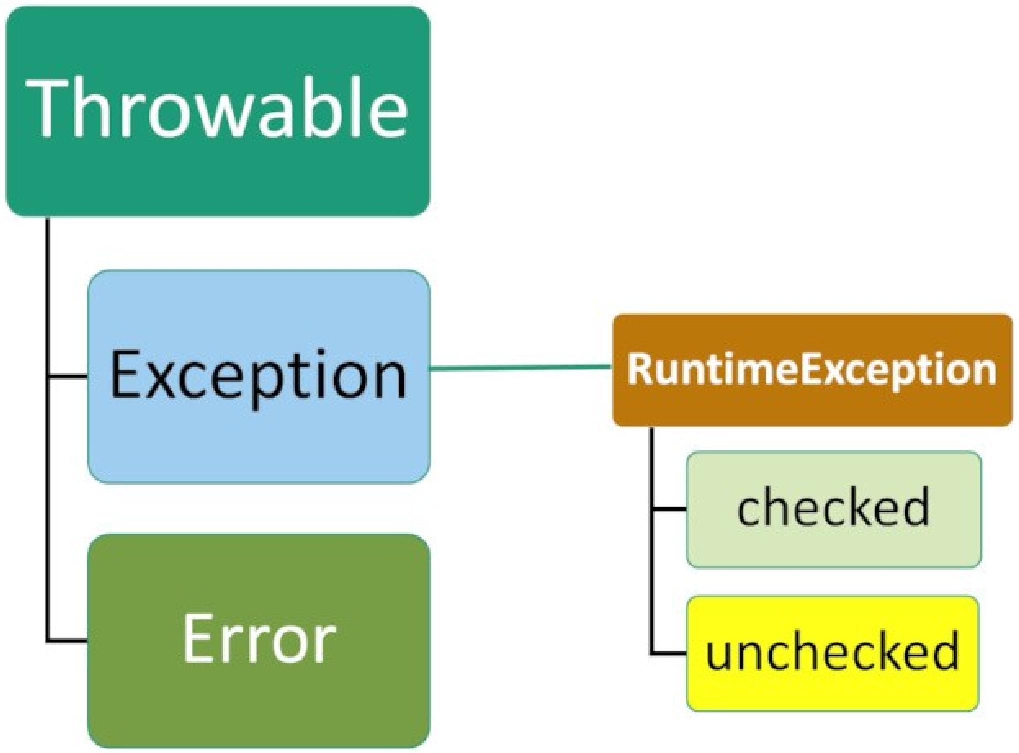
Exception handling

Exception handling



try
catch
finally
throw
throws

Exception handling

- When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.
- When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an exception (throw an error).

Exception handling

- Exception handling in Java is a mechanism that allows you to deal with runtime errors and abnormal situations in a program gracefully, preventing the program from crashing.
- In Java, exceptions are represented by objects, and the process of handling exceptions involves the use of three keywords:
  - try,
  - catch,
  - throw,
  - Throws and
  - finally.

Exception handling

- In Java, exceptions are events that occur during the execution of a program that disrupts the normal flow of instructions.
- Exceptions are used to handle errors and abnormal situations in a controlled manner.
- There are two main types of exceptions in Java:
  - checked exceptions and
  - unchecked exceptions.

Exception handling

- **Checked exceptions:**
- They are exceptions that the compiler forces you to handle.
- They are subclasses of the Exception class (excluding subclasses of RuntimeException).
- Here are some common checked exceptions:
  - IOException
  - SQLException
  - ClassNotFoundException

Exception handling

- **Checked exceptions:**
- IOException
  - This exception is thrown when there is a problem with input or output operations, such as reading from or writing to a file.

```
try {
// Code that may cause an IOException
} catch (IOException e) {
    // Handle the IOException
}
```

Exception handling

- **Checked exceptions:**
- SQLException:
  - This exception is thrown when there is a problem with database access or SQL operations.

```
try {
// Code that may cause an SQLException
} catch (SQLException e) {
    // Handle the SQLException
}
```

Exception handling

- **Checked exceptions:**
- ClassNotFoundException:
  - This exception is thrown when a class is not found at runtime, usually when trying to load a class dynamically.

```
try {
// Code that may cause a ClassNotFoundException
} catch (ClassNotFoundException e) {
// Handle the ClassNotFoundException
}
```

Exception handling

- **Unchecked exceptions:**
- They are also known as runtime exceptions, are exceptions that the compiler does not force you to catch.
- They are subclasses of the RuntimeException class.
- Here are some common unchecked exceptions:
  - ArithmeticException
  - NullPointerException
  - ArrayIndexOutOfBoundsException
  - IllegalArgumentException

Exception handling

- **Unchecked exceptions:**
- ArithmeticException
  - This exception is thrown when an arithmetic operation is attempted with an illegal argument.

```
try {
// Code that may cause an ArithmeticException
} catch (ArithmeticException e) {
// Handle the ArithmeticException
}
```

Exception handling

- **Unchecked exceptions:**
- NullPointerException
  - This exception is thrown when a program attempts to access an object or invoke a method on a null reference.

```
try {
    // Code that may cause a NullPointerException
} catch (NullPointerException e) {
    // Handle the NullPointerException
}
```

Exception handling

- ***Exception handling in Java involves several fundamental keywords and constructs:***
  - ***try,***
  - ***catch,***
  - ***throw,***
  - ***throws, and***
  - ***finally.***

Exception handling

- ***Try and catch**:*
- *The try statement allows you to define a block of code to be tested for errors while it is being executed.*
- *The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.*
- *The try and catch keywords come in pairs:*

```
try {
    // Code that may throw an exception
} catch (ExceptionType e) {
    // Handle the exception
}
```

Exception handling

- ***try and catch***:
- *In the above example we can use try...catch to catch the error and execute some code to handle it.*

```
public class Main {
  public static void main(String[] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    }
  }
}
```

Exception handling

- ***throw**:*
- *The throw statement allows you to create a custom error.*
- *The throw statement is used together with an exception type.*
- *There are many exception types available in Java: ArithmeticException, FileNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc:*

```
public class Main {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18 years old.");
    }
    else {
      System.out.println("Access granted - You are old enough!")
    }
  }
  public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
  }
}
```

Exception handling

- ***throws***:
- *The throws keyword is used in the method declaration to indicate that a method might throw certain types of exceptions.*
- *It is part of the method signature.*
- *When a method includes a throws clause, it informs the caller that the method may throw exceptions of the specified types, and the caller is responsible for handling those exceptions.*

```
public class Main {
 static void checkAge(int age) {
  if (age < 18) {
   throw new ArithmeticException("Access denied - You must be at least 18 years old.");
  }
  else {
   System.out.printlnpublic class ThrowsExample {
  public static void main(String[] args) {
    try {
      performIOOperation("file.txt");
    } catch (IOException e) {
      System.out.println("Caught exception: " + e.getMessage());
    }
  }
  public static void performIOOperation(String filename) throws IOException {
    // Code that may throw IOException
    // ...
    throw new IOException("Error reading file");
  }
}
("Access granted - You are old enough!");
  }
 }
 public static void main(String[] args) {
  checkAge(15); // Set age to 15 (which is below 18...)
 }
}
```

Exception handling

- ***finally***:
- *The finally statement lets you execute code, after try...catch, regardless of the result:*

```
public class Main {
  public static void main(String[] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.")
    } finally {
      System.out.println("The 'try catch' is
finished.");
    }
  }
}
```

Exception handling

- *Here's a complete example demonstrating the use of try, catch, throw, throws, and finally.*
- *In this example, the divide method throws an ArithmeticException if the divisor is 0.*
- *The main method catches this exception in the catch block and executes the finally block regardless of whether an exception occurred or not.*

```
public class ExceptionHandlingExample {
    public static void main(String[] args) {
        try {
            // Code that may throw an exception
            int result = divide(10, 0);
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            // Handle the exception
            System.err.println("Error: " + e.getMessage());
        } finally {
            // Code that always executes
            System.out.println("This block always executes.");
        }
    }
    public static int divide(int a, int b) {
        if (b == 0) {
            // Throw an exception if the divisor is 0
            throw new ArithmeticException("Division by zero\n");
        }
        return a / b;
    }
}
```

Assignment

- Explain the concept of exceptions in Java. Provide examples of scenarios where exceptions might occur.
- Differentiate between checked and unchecked exceptions in Java. Give examples of each.
- What is the purpose of the finally block in exception handling? Provide a scenario where it would be useful.
- Describe the purpose and usage of the try, catch, and finally blocks in Java's exception handling mechanism with example.