Research

Group Members

Dayana Vanessa Lema Yagcha

Daniela Elizabeth Yumbo Pazto

Central University of Ecuador

Systems 001

Teacher´s Name

Ing. Juanpa Guevara

Exceptions in Java

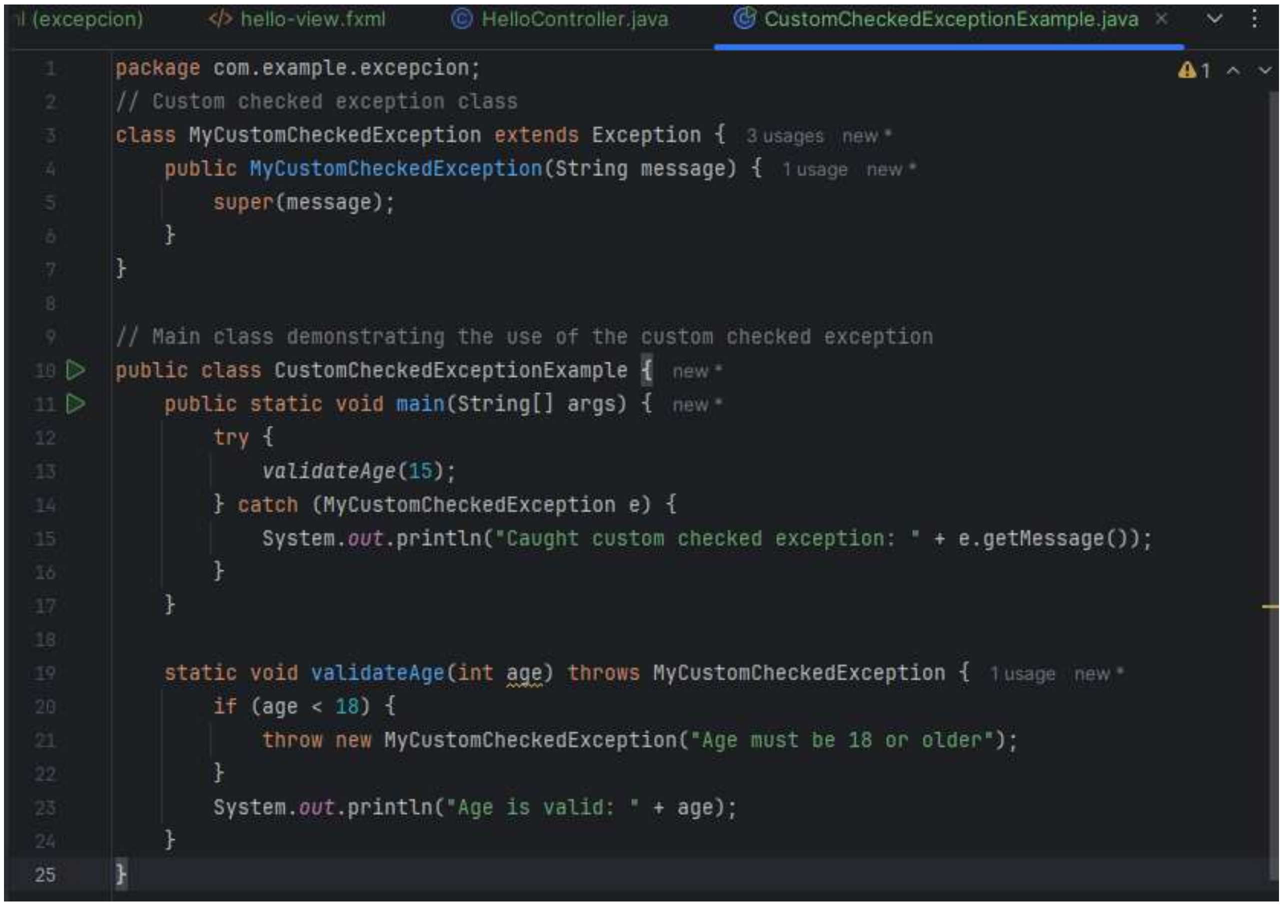
In Java, an exception is an event that disrupts the normal flow of the program's instructions during runtime. It is an object which is thrown at runtime and can be caught and handled to ensure the program doesn't crash unexpectedly.

Types of Exceptions

- Checked Exceptions: These exceptions are checked at compile-time. If a method is throwing a checked exception, it must either handle the exception using a try-catch block or declare it using the throws keyword.
- Unchecked Exceptions: These exceptions are not checked at compile-time but are checked at runtime. They include RuntimeException and its subclasses.

- Errors: Errors are serious problems that an application should not try to catch. Most of these are abnormal conditions

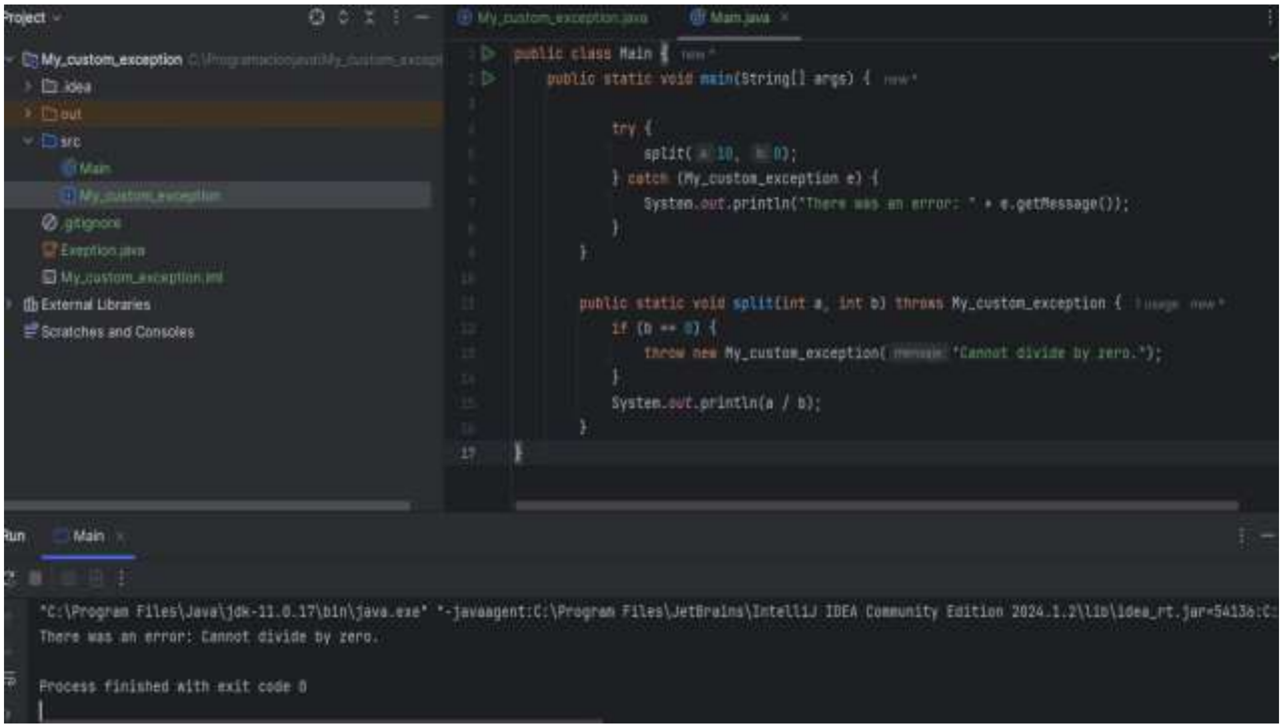
EXAMPLE



```java
package com.example.excepcion;
// Custom checked exception class
class MyCustomCheckedException extends Exception {  3 usages  new *
    public MyCustomCheckedException(String message) {  1 usage  new *
        super(message);
    }
}

// Main class demonstrating the use of the custom checked exception
public class CustomCheckedExceptionExample {  new *
    public static void main(String[] args) {  new *
        try {
            validateAge(15);
        } catch (MyCustomCheckedException e) {
            System.out.println("Caught custom checked exception: " + e.getMessage());
        }
    }

    static void validateAge(int age) throws MyCustomCheckedException {  1 usage  new *
        if (age < 18) {
            throw new MyCustomCheckedException("Age must be 18 or older");
        }
        System.out.println("Age is valid: " + age);
    }
}
```

Example 2

Execution



```java
public class Main {
    public static void main(String[] args) {

        try {
            split( 10,  0);
        } catch (My_custom_exception e) {
            System.out.println("There was an error: " + e.getMessage());
        }
    }

    public static void split(int a, int b) throws My_custom_exception {
        if (b == 0) {
            throw new My_custom_exception( "Cannot divide by zero.");
        }
        System.out.println(a / b);
    }
}
```

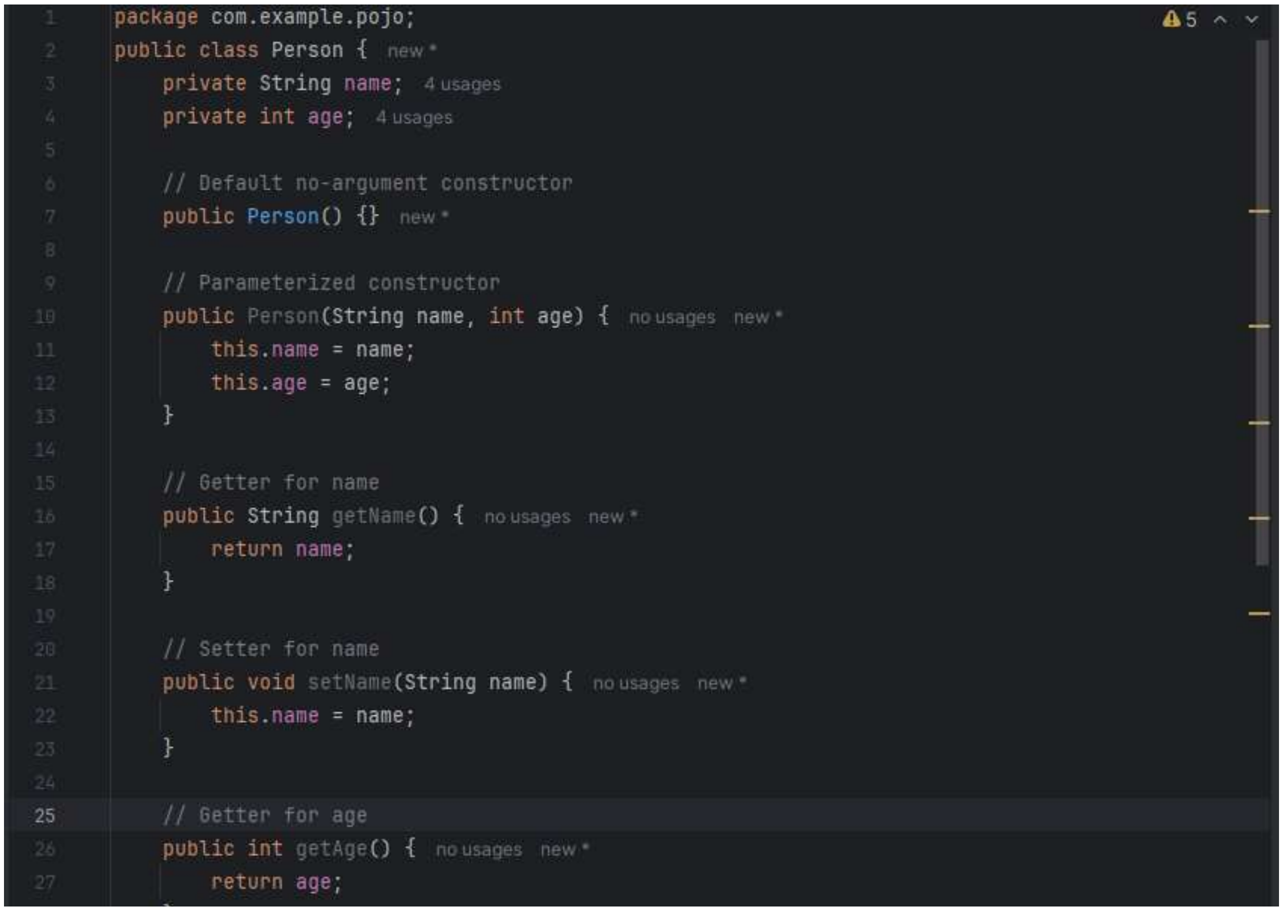
```
"C:\Program Files\Java\jdk-11.0.17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.1.2\lib\idea_rt.jar=54136:C:
There was an error: Cannot divide by zero.

Process finished with exit code 0
```

What is a POJO?

A POJO (Plain Old Java Object) is a simple Java object that is not bound by any special restriction other than those forced by the Java Language Specification. It doesn't need to follow any particular conventions or extend/implement any specific classes/interfaces. POJOs are often used for encapsulating data and are characterized by:

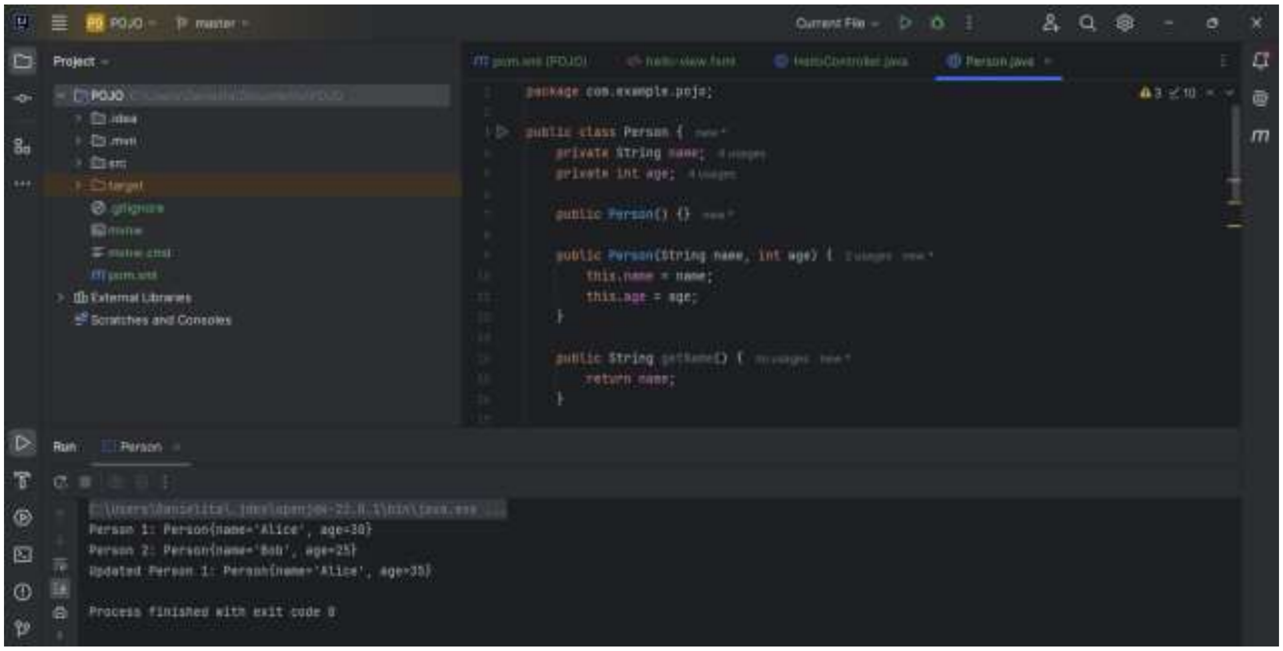
- Private Fields: POJOs typically have private fields to store data.
- Public Getters and Setters: These provide access to the fields.
- No-Arg Constructor: A no-argument constructor is usually provided.
- No Special Annotations or Inheritance: POJOs do not require any specific annotations or to extend/implement any particular classes/interfaces.

EXAMPLE



```java
package com.example.pojo;
public class Person {  new *
    private String name;  4 usages
    private int age;  4 usages

    // Default no-argument constructor
    public Person() {}  new *

    // Parameterized constructor
    public Person(String name, int age) {  no usages  new *
        this.name = name;
        this.age = age;
    }

    // Getter for name
    public String getName() {  no usages  new *
        return name;
    }

    // Setter for name
    public void setName(String name) {  no usages  new *
        this.name = name;
    }

    // Getter for age
    public int getAge() {  no usages  new *
        return age;
```

Execution



```java
package com.example.pojo;

public class Person {  new *
    private String name;  4 usages
    private int age;  4 usages

    public Person() {}  new *

    public Person(String name, int age) {  2 usages  new *
        this.name = name;
        this.age = age;
    }

    public String getName() {  no usages  new *
        return name;
    }
```

```
C:\Users\Danielitta\.jdks\openjdk-22.0.1\bin\java.exe ...
Person 1: Person{name='Alice', age=30}
Person 2: Person{name='Bob', age=25}
Updated Person 1: Person{name='Alice', age=35}

Process finished with exit code 0
```

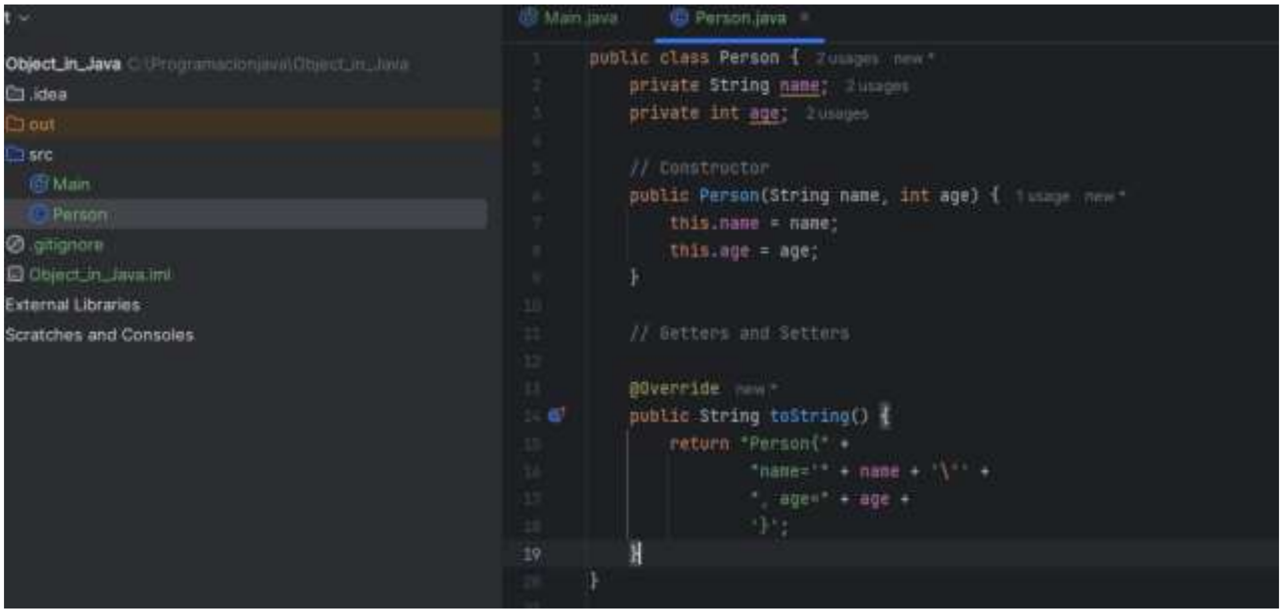
How to Print an Object in Java

Printing an object in Java involves obtaining a text representation of that object. The most common and recommended way to do this is by overriding the class's toString() method. Here is a detailed explanation of how this can be done, along with an example:

Definition

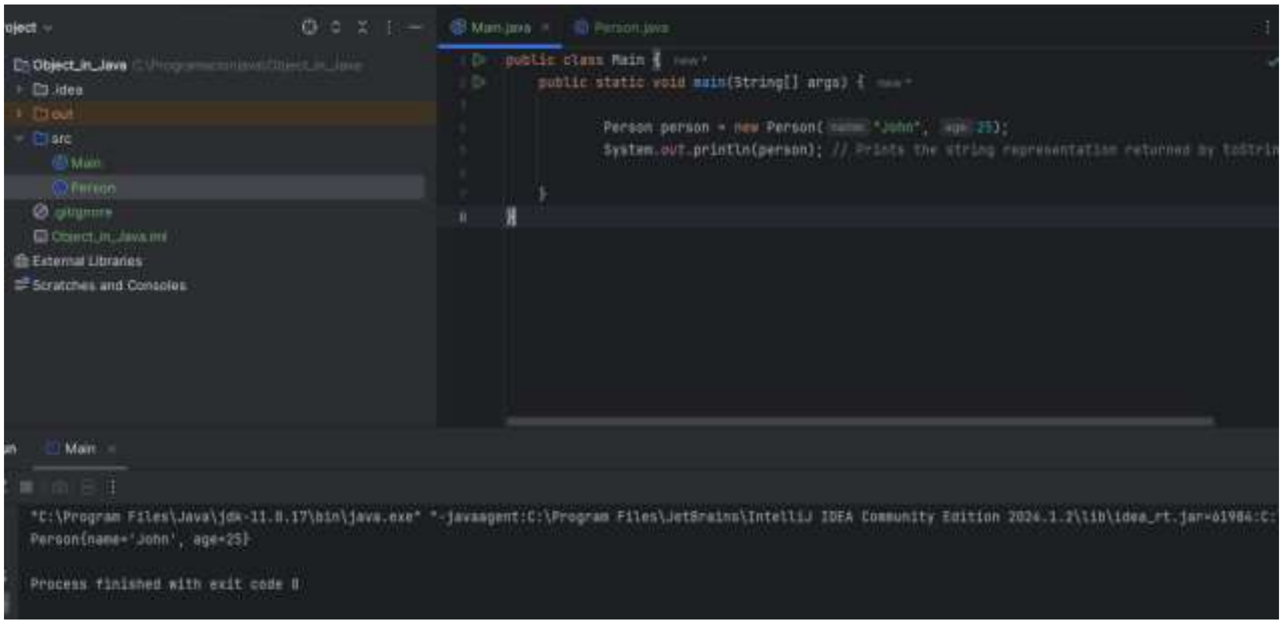
To print an object in Java, you typically override the toString() method of the object class. The toString() method is defined in the Object class (the superclass of all classes in Java), and returns a text string representation of the object. Overriding this method allows you to customize how the object is printed.

EXAMPLE



```java
public class Person {
    private String name;
    private int age;

    // Constructor
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Getters and Setters

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

Execution



```java
public class Main {
    public static void main(String[] args) {

        Person person = new Person("John", 25);
        System.out.println(person); // Prints the string representation returned by toStrin

    }
}
```

```
"C:\Program Files\Java\jdk-11.0.17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.1.2\lib\idea_rt.jar=61984:C:
Person{name='John', age=25}

Process finished with exit code 0
```

References

Excepción personalizada de Java. (s/f). Www.javatpoint.com. Recuperado el 5 de julio de 2024, de https://www.javatpoint.com/custom-exception

Irfan, M. (2021, October 16). *Imprimir objetos en Java*. Delft Stack. https://www.delftstack.com/es/howto/java/print-object-in-java/