17.2 Exception Handling using throw , throws

🚨 Understanding `throw` and `throws`

Exception handling in Java is a powerful mechanism that helps you manage runtime errors, making your applications more robust and error-resilient.

In a previous post, we explored the `try-catch-finally` blocks, which are used to catch and handle exceptions. In this blog, we'll go deeper and explore two other critical keywords in exception handling: `throw` and `throws`.

🔹 What is `throw` in Java?

The `throw` keyword is used to **explicitly throw an exception** in Java. This is helpful when you want to signal that an error has occurred, either by throwing a built-in exception or a custom one.

✅ **Syntax:**

```
throw new ExceptionType("Error Message");
```

✅ **Example:**

```
public class ThrowExample {
    public static void main(String[] args) {
        int age = 15;
        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!");
        }
    }
}
```

📝 **Note**: When using `throw`, you're creating an instance of an exception and throwing it at runtime.

🔹 What is `throws` in Java?

The `throws` keyword is used in a method declaration to indicate that **the method might throw one or more exceptions**. It informs the caller of the method to handle those exceptions.

This is particularly useful for **checked exceptions**, which Java requires to be either caught or declared in the method signature.

✅ **Syntax:**

```
returnType methodName() throws ExceptionType1, ExceptionType2 {
    // method code
}
```

✅ **Example:**

```java
import java.io.IOException;

public class ThrowsExample {
    public static void main(String[] args) {
        try {
            readFile();
        } catch (IOException e) {
            System.out.println("Exception caught: " + e.getMessage());
        }
    }

    public static void readFile() throws IOException {
        throw new IOException("File not found");
    }
}
```

🔄 `throw` vs `throws` – Key Differences

Feature	`throw`	`throws`
Purpose	Used to **explicitly throw** an exception	Used to **declare** exceptions in method signature
Placement	Inside a method or block	After the method signature
Follows By	An **instance** of `Throwable` class	One or more **exception classes**
Number Allowed	One exception at a time	Multiple exceptions (comma-separated)
Example	`throw new IOException();`	`public void readFile() throws IOException`

🔧 When to Use `throw` and `throws`

- Use `**throw**` when you want to **manually trigger** an exception based on certain conditions in your logic.
- Use `**throws**` when a method might throw a **checked exception**, and you want the calling method to handle it.

🧪 Real-world Example: Custom Exception

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

public class CustomExceptionDemo {
    static void validate(int age) throws InvalidAgeException {
        if (age < 18)
            throw new InvalidAgeException("Not eligible to vote");
        else
            System.out.println("Eligible to vote");
    }

    public static void main(String[] args) {
        try {
            validate(16);
        } catch (InvalidAgeException e) {
            System.out.println("Caught the exception: " + e.getMessage());
        }
    }
}
```

📝 Final Thoughts

Using `throw` and `throws` gives you fine-grained control over how and when exceptions occur and how they are handled. By combining these with `try-catch-finally`, you can create resilient Java applications that handle errors gracefully and maintain smooth user experience.

Document by ***Suyash*** 😇