Java Exception Handling:

Certainly! Exception handling is a crucial aspect of Java programming that allows you to gracefully handle errors and exceptions that may occur during the execution of a program. Java provides a mechanism for catching and handling exceptions using `try`, `catch`, `throw`, and `finally` blocks. Let's go through two examples of exception handling in Java.

Example 1: Handling ArithmeticException

In this example, we'll handle an `ArithmeticException` that occurs when attempting to divide by zero.

```java
public class ExceptionHandlingExample1 {
    public static void main(String[] args) {
        int numerator = 10;
        int denominator = 0;

        try {
            int result = numerator / denominator; // This line may throw an ArithmeticException
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

In this code:

1. We define two integers, `numerator` and `denominator`, with `denominator` set to 0.
2. Inside the `try` block, we attempt to perform the division operation, which may throw an `ArithmeticException` when dividing by zero.
3. In the `catch` block, we catch the exception and print an error message using `e.getMessage()`.

Example 2: Using finally Block

In this example, we'll handle a `FileNotFoundException` and use a `finally` block to ensure some cleanup code always gets executed.

```java
import java.io.*;
```

```
public class ExceptionHandlingExample2 {
    public static void main(String[] args) {
        FileReader fileReader = null;
        try {
            fileReader = new FileReader("nonexistentfile.txt"); // This may throw a
FileNotFoundException
            // Code to read from the file
        } catch (FileNotFoundException e) {
            System.err.println("Error: File not found - " + e.getMessage());
        } finally {
            // Close the file reader in the finally block to ensure cleanup
            try {
                if (fileReader != null) {
                    fileReader.close();
                }
            } catch (IOException e) {
                System.err.println("Error while closing file - " + e.getMessage());
            }
        }
    }
}
```

In this code:

1. We attempt to open a file (`nonexistentfile.txt`) using `FileReader`, which may throw a `FileNotFoundException`.
2. In the `catch` block, we catch the exception and print an error message.
3. In the `finally` block, we ensure that the `fileReader` is closed to release resources, even if an exception occurs or not.

These examples demonstrate how to use `try`, `catch`, and `finally` blocks to handle exceptions in Java, ensuring that your program can handle errors gracefully and perform necessary cleanup tasks.