```
public class ExceptionHandling {

    // Handling Multiple Exceptions
    public void handleMultipleExceptions() {
        try {
            // Code that might throw multiple exceptions
            int x = 10 / 0; // Arithmetic exception
            String str = null;
            str.length(); // NullPointerException
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic exception occurred: " + e.getMessage());
        } catch (NullPointerException e) {
            System.out.println("Null pointer exception occurred: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("General exception occurred: " + e.getMessage());
        }
    }

    // Re-throwing Exceptions
    public void rethrowException(int x) throws IOException {
        if (x < 0) {
            throw new IOException("Input cannot be negative");
        }

        // Code that might throw other exceptions
        // ...
    }

    // Resource Management
    public void manageResources() {
        try (FileReader reader = new FileReader("data.txt");
             BufferedReader bufferedReader = new BufferedReader(reader)) {
            String line;
            while ((line = bufferedReader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }

    // Chaining Exceptions
    public void chainExceptions() {
        try {
```

```
        // Code that might throw exceptions
        // ...
      } catch (Exception e) {
        e.printStackTrace(); // Print the stack trace
        throw new RuntimeException("Error occurred", e); // Chain the exception
      }
    }
}
```

-Task Breakdown

Sure, here's a breakdown of the tasks for creating a class for exception handling in Java, dividing the responsibility among team members:

Team Member 1:

- Topic: Handling Multiple Exceptions
- Responsibility:
    - Implement the `handleMultipleExceptions()` method.
    - Demonstrate how to catch and handle multiple exceptions using specific catch blocks.
    - Use appropriate exception types for each scenario.

Team Member 2:

- Topic: Re-throwing Exceptions
- Responsibility:
    - Implement the `rethrowException()` method.
    - Illustrate how to rethrow an exception from a method.
    - Explain the purpose of rethrowing exceptions and its usage scenarios.

Team Member 3:

- Topic: Resource Management
- Responsibility:
    - Implement the `manageResources()` method.
    - Demonstrate the use of try-with-resources for resource management.
    - Explain the benefits of try-with-resources and its automatic resource closing mechanism.

Team Member 4:

- Topic: Chaining Exceptions
- Responsibility:
    - Implement the `chainExceptions()` method.
    - Illustrate how to chain exceptions.
    - Explain the purpose of chaining exceptions and its usage scenarios.
    - Demonstrate how to print the stack trace of the original exception and its cause.

tuneshare
more_vert