Practices for Lesson 11: Exceptions and Assertions

Chapter 11

Practices for Lesson 11: Overview

Practices Overview

In these practices, you will use `try-catch` statements, extend the `Exception` class, and use the `throw` and `throws` keywords.

Practice 11-1: Summary Level: Catching Exceptions

Overview

In this practice, you will create a new project and catch checked and unchecked exceptions.

Assumptions

You have reviewed the exception handling section of this lesson.

Summary

You will create a project that reads from a file. The file-reading code will be provided to you. Your task is to add the appropriate exception-handling code.

Tasks

1. Perform the following tasks to create a new `CatchingExceptions11-01` project.
   a. Select File > New Project.
   b. Select Java under Categories and Java Application under Projects.
   c. Click Next.
   d. Enter the following information in the "Name and Location" dialog box:
      1) Project Name: `CatchingExceptions11-01`
      2) Project Location: `/home/oracle/labs/11-Exceptions /practices/practice1/CatchingExceptions11-01.`
      3) Check Create Main Class: `com.example.ExceptionMain`
   e. Click Finish.
2. Add the following line to the `main` method.

```
System.out.println("Reading from file:" + args[0]);
```

   Note: A command-line argument will be used to specify the file that will be read. Currently no arguments will be supplied, do not correct this oversight yet.
3. Run the project. You should see an error message similar to:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 0
    at com.example.ExceptionMain.main(ExceptionMain.java:7)
Java Result: 1
```

4. Surround the `println` line of code you added with a `try-catch` statement.
   - The catch clause should:
     - Accept a parameter of type `ArrayIndexOutOfBoundsException`
     - Print the message: `"No file specified, quitting!"`
     - Exit the application with an exit status of 1 by using the appropriate static method within the `System` class

     Note: Because the compiler did not force you to handle or declare the `ArrayIndexOutOfBoundsException`, it is an unchecked exception. Typically, you should not need to use a `try-catch` block to deal with an unchecked exception. Checking the length of the `args` array is an alternate way to ensure that a command-line argument was supplied.

5. Run the project. You should see an error message similar to:

```
No file specified, quitting!
Java Result: 1
```

6. Add a command-line argument to the project.
   a. Right-click the `CatchingExceptions11-01` project and select Properties.
   b. In the Project Properties dialog box, select the Run category.
   c. In the Arguments field, enter a value of:
      `/home/oracle/labs/resources/DeclarationOfIndependence.txt`
   d. Click OK.
7. Run the project. You should see a message similar to:

```
Reading from file:
/home/oracle/labs/resources/DeclarationOfIndependence.txt
```

   Warning: Running the project is not the same as running the file. The command-line argument will only be passed to the `main` method if you run the project.
8. Add the following lines of code to the `main` method below your previously added lines:

```
BufferedReader b =
        new BufferedReader(new FileReader(args[0]));
String s = null;
while((s = b.readLine()) != null) {
    System.out.println(s);
}
```

9. Run the Fix Imports wizard by right-clicking in the source-code window.
10. You should now see compiler errors in some of the lines that you just added. These lines potentially generate checked exceptions. By manually building the project or holding your cursor above the line with errors, you should see a message similar to:

```
unreported exception FileNotFoundException; must be caught or
declared to be thrown
```

11. Modify the project properties to support the `try`-with-resources statement.
    a. Right-click the `CatchingExceptions11-01` project and select Properties.
    b. In the Project Properties dialog box, select the Sources category.
    c. In the Source/Binary Format drop-down list, select JDK 8.
    d. Click OK.

12. Surround the file IO code provided in step 8 with a `try`-with-resources statement.
    - The line that creates and initializes the `BufferedReader` should be an automatically closed resource.
    - Add a catch clause for a `FileNotFoundException`. Within the catch clause:
      - Print `"File not found:" + args[0]`
      - Exit the application.
    - Add a catch clause for an `IOException`. Within the catch clause:
      - Print `" Error reading file:"` along with the message available in the `IOException` object
      - Exit the application.

```
try (BufferedReader b = new BufferedReader(new
FileReader(args[0]));) {
            String s = null;
            while((s = b.readLine()) != null) {
                System.out.println(s);
            }
        } catch(FileNotFoundException e) {
            System.out.println("File not found:" + args[0]);
            System.exit(1);
        } catch(IOException e) {
            System.out.println("Error reading file:" +
e.getMessage());
            System.exit(1);
        }
```

13. Run the project. You should see the content of the `/home/oracle/labs/resources/DeclarationOfIndependence.txt` file displayed in the output window.

Practice 11-1: Detailed Level: Catching Exceptions

Overview

In this practice, you will create a new project and catch checked and unchecked exceptions.

Assumptions

You have reviewed the exception handling section of this lesson.

Summary

You will create a project that reads from a file. The file-reading code will be provided to you. Your task is to add the appropriate exception-handling code.

Tasks

1. Perform the following steps to create a new `CatchingExceptions11-01` project as the main project.
   a. Click File > New Project.
   b. Select Java from Categories, and Java Application from Projects.
   c. Click Next.
   d. Enter the following information in the "Name and Location" dialog box:
      1) Project Name: `CatchingExceptions11-01`
      2) Project Location: `/home/oracle/labs/11-Exceptions /practices/practice1/CatchingExceptions11-01.`
      3) Check Create Main Class: `com.example.ExceptionMain`.
   e. Click Finish.
2. Add the following line to the `main` method.

```
System.out.println("Reading from file:" + args[0]);
```

   Note: A command-line argument will be used to specify the file that will be read. Currently no arguments will be supplied; do not correct this oversight yet.
3. Run the project. You should see an error message similar to:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 0
    at com.example.ExceptionMain.main(ExceptionMain.java:7)
Java Result: 1
```

4. Surround the `println` line of code you added with a `try-catch` statement.
   - The catch clause should:
     - Accept a parameter of type `ArrayIndexOutOfBoundsException`
     - Print the message: `"No file specified, quitting!"`
     - Exit the application with an exit status of 1 by using the `System.exit(1)` method

```
try {
    System.out.println("Reading from file:" + args[0]);
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("No file specified, quitting!");
    System.exit(1);
}
```

   Note: Since the compiler did not force you to handle or declare the `ArrayIndexOutOfBoundsException` it is an unchecked exception. Typically you should not need to use a `try-catch` block to deal with an unchecked exception. Checking the length of the `args` array is an alternate way to ensure that a command line argument was supplied.

5. Run the project. You should see an error message similar to:

```
No file specified, quitting!
Java Result: 1
```

6. Add a command-line argument to the project.
   a. Right-click the `CatchingExceptions11-01` project and click Properties.
   b. In the Project Properties dialog box, select the Run category.
   c. In the Arguments field, enter a value of: `/home/oracle/labs/resources/DeclarationOfIndependence.txt`
   d. Click OK.
7. Run the project. You should see a message similar to:

```
Reading from
/home/oracle/labs/resources/DeclarationOfIndependence.txt
```

   Warning: Running the project is not the same as running the file. The command-line argument will only be passed to the `main` method if you run the project.
8. Add the following lines of code to the `main` method below your previously added lines:

```
BufferedReader b =
        new BufferedReader(new FileReader(args[0]));
String s = null;
while((s = b.readLine()) != null) {
    System.out.println(s);
}
```

9. Run the Fix Imports wizard by right-clicking in the source-code window.
10. You should now see compiler errors in some of the lines that you just added. These lines potentially generate checked exceptions. By manually building the project or holding your cursor above the line with errors, you should see a message similar to:

```
unreported exception FileNotFoundException; must be caught or
declared to be thrown
```

11. Modify the project properties to support the `try`-with-resources statement.
    a. Right-click the `CatchingExceptions11-01` project and select Properties.
    b. In the Project Properties dialog box, select the Sources category.
    c. In the Source/Binary Format drop-down list, select JDK 8.
    d. Click OK.
12. Surround the file IO code provided in step 8 with a `try`-with-resources statement.
    - The line that creates and initializes the `BufferedReader` should be an automatically closed resource.
    - Add a catch clause for a `FileNotFoundException`. Within the catch clause:
      - Print `"File not found:" + args[0]`
      - Exit the application.
    - Add a catch clause for an `IOException`. Within the catch clause:
      - Print `" Error reading file:"` along with the message available in the `IOException` object
      - Exit the application.

```
try (BufferedReader b =
        new BufferedReader(new FileReader(args[0]));) {
    String s = null;
    while((s = b.readLine()) != null) {
        System.out.println(s);
    }
} catch(FileNotFoundException e) {
    System.out.println("File not found:" + args[0]);
    System.exit(1);
} catch(IOException e) {
    System.out.println("Error reading file:" + e.getMessage());
    System.exit(1);
}
```

13. Run the project. You should see the content of the `/home/oracle/labs/resources/DeclarationOfIndependence.txt` file displayed in the output window.

Practice 11-2: Summary Level: Extending `Exception` and Throwing `Exception`

Overview

In this practice, you will take an existing application and refactor the code to make use of a custom exception class and throwing exception using `throw` and `throws`.

Assumptions

You have reviewed the exception handling section of this lesson.

Summary

You have been given a project that implements the logic for a human resources application. The application allows for creating, retrieving, deleting, and listing of `Employee` objects.

Tasks

1. Open the `CustomExceptions11-02Prac` project as the main project.
   a. Select File > Open Project.
   b. Browse to `/home/oracle/labs/11-Exceptions/practices/practice2`
   c. Select `CustomExceptions11-02Prac`
   d. and click Open Project.
2. Expand the project directories.
3. Create a `InvalidOperationException` class in the `com.example` package.
5. Complete the `InvalidOperationException` class. The `InvalidOperationException` class should:
   - Extend the `Exception` class
   - Contain four public constructors with parameters matching those of the four public constructors present in the `Exception` class. For each constructor, use `super()` to invoke the parent class constructor with matching parameters.
6. Modify `EmployeeImpl` class.
   a. Modify the methods: `add`, `delete` and `findById`
   b. Declare that a `InvalidOperationException` may be produced during execution of these method.
   c. Within the catch block that you just created, generate a `InvalidOperationException` and deliver it to the caller of the method. The `InvalidOperationException` should contain a message String indicating what went wrong and why.
12. Modify the `EmployeeTest` class to handle the `InvalidOperationException` objects that are thrown by the `EmployeeImpl`.
    a. Modify the `main` method:

    Add the `throws` statement from the `main` method.

```
public static void main(String[] args) throws
InvalidOperationException
```

13. Run the project. Test all the operations by invoking the methods: `add`,`delete` and `findById`.

    For example: Attempt to delete an employee that does not exist. You should see a message similar to:

```
Exception in thread "main"
com.example.InvalidOperationException: Error deleting employee,
no such employee 7
```

Practice 11-2: Detailed Level: Extending `Exception` and Throwing `Exception`

Overview

In this practice, you will take an existing application and refactor the code to make use of a custom exception class and throwing exception using `throw` and `throws`.

Assumptions

You have reviewed the exception handling section of this lesson.

Summary

You have been given a project that implements the logic for a human resources application. The application allows for creating, retrieving, deleting, and listing of `Employee` objects.

Tasks

1. Open the `CustomExceptions11-02Prac` project as the main project.
   a. Select File > Open Project.
   b. Browse to `/home/oracle/labs/11-Exceptions/practices/practice2`
   c. Select `CustomExceptions11-02Prac` and Click Open Project.
2. Expand the project directories.
3. Create a `InvalidOperationException` class in the `com.example` package.
5. Complete the `InvalidOperationException` class. The `InvalidOperationException` class should:
   - Extend the `Exception` class.
   - Create four public constructors with parameters matching those of the four public constructors present in the `Exception` class. For each constructor, use `super()` to invoke the parent class constructor with matching parameters.

```
package com.example;

public class InvalidOperationException extends Exception {

    public InvalidOperationException() {
        super();
    }

    public InvalidOperationException(String message) {
        super(message);
    }

    public InvalidOperationException(Throwable cause) {
        super(cause);
    }
```

```
    public InvalidOperationException(String message, Throwable
cause) {
        super(message, cause);
    }
}
```

7. Modify the `add` method within the `EmployeeImpl` class to:
   - Declare that a `InvalidOperationException` may be produced during execution of this method.
   - Use an `if` statement to validate that an existing employee will not be overwritten by the add. If one would, generate a `InvalidOperationException` and deliver it to the caller of the method. The `InvalidOperationException` should contain a message String indicating what went wrong and why.
   - Use a `try-catch` block to catch the `ArrayIndexOutOfBoundsException` unchecked exception that could possibly be generated.
   - Within the catch block that you just created, generate a `InvalidOperationException` and deliver it to the caller of the method. The `InvalidOperationException` should contain a message String indicating what went wrong and why.

```
  public void add(Employee emp) throws InvalidOperationException
{
        if(employeeArray[emp.getId()] != null) {
            throw new InvalidOperationException("Error adding
employee , employee id already exists " + emp.getId());
        }
        try {
            employeeArray[emp.getId()] = emp;
        } catch (ArrayIndexOutOfBoundsException e) {
            throw new InvalidOperationException("Error adding
employee , id must be less than " + employeeArray.length);
        }
    }
```

8. Modify the `delete` method within the `EmployeeImpl` class to:
   - Declare that a `InvalidOperationException` may be produced during execution of this method.
   - Use an if statement to validate that an existing employee is being deleted. If one would not be, generate a `InvalidOperationException` and deliver it to the caller of the method. The `InvalidOperationException` should contain a message String indicating what went wrong and why.
   - Use a `try-catch` block to catch the `ArrayIndexOutOfBoundsException` unchecked exception that could possibly be generated.

- Within the catch block that you just created, generate a `InvalidOperationException` and deliver it to the caller of the method. The `InvalidOperationException` should contain a message String indicating what went wrong and why.

```
public void delete(int id) throws InvalidOperationException {
        if(employeeArray[id] == null) {
            throw new InvalidOperationException("Error deleting
employee, no such employee " + id);
        }
        try {
            employeeArray[id] = null;
        } catch (ArrayIndexOutOfBoundsException e) {
            throw new InvalidOperationException("Error deleting
employee, id must be less than " + employeeArray.length);
        }
    }
```

9. Modify the `findById` method within the `EmployeeImpl` class to:
   - Declare that a `InvalidOperationException` may be produced during execution of this method.
   - Use a `try-catch` block to catch the `ArrayIndexOutOfBoundsException` unchecked exception that could possibly be generated.
   - Within the catch block that you just created, generate a `InvalidOperationException` and deliver it to the caller of the method. The `InvalidOperationException` should contain a message String indicating what went wrong and why.

```
public Employee findById(int id) throws
InvalidOperationException {
        try {
            return employeeArray[id];
        } catch (ArrayIndexOutOfBoundsException e) {
            throw new InvalidOperationException("Error finding
employee ", e);
        }
    }
```

10. Modify the `EmployeeTest` class to handle the `InvalidOperationException` objects that are thrown by the `EmployeeImpl`

.

b. Modify the `main` method:

Add the `throws` statement from the `main` method.

```
public static void main(String[] args) throws
InvalidOperationException
```

11. Run the project. Test all the operations by invoking the methods: `add`,`delete` and `findById`.

    For example: Attempt to delete an employee that does not exist. You should see a message similar to:

```
Exception in thread "main"
com.example.InvalidOperationException: Error deleting employee,
no such employee 7
```