Practices for Lesson 8

Practices Overview

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

Practice 8-1: 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. Create a new `ExceptionPractice` project as the main project.
   a. Select File > New Project.
   b. Select Java under Categories and Java Application under Projects. Click the Next button.
   c. Enter the following information in the “Name and Location” dialog box:
      - Project Name: `ExceptionPractice`
      - Project Location: `D:\labs\08\practices`. (or you other directory)
      - (checked) Create Main Class: `com.example.ExceptionMain`
      - (checked) Set as Main Project
   d. Click the *Finish* button.
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 `ExceptionPractice` project and select Properties.
   b. In the Project Properties dialog box, select the Run category.
   c. In the Arguments field, enter a value of: `D:\labs\resources\DeclarationOfIndependence.txt`
   d. Click the OK button.

7. Run the project. You should see a message similar to:

```
Reading from
file:D:\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 `ExceptionPractice` 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 7.
    d. Click the OK button.

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.
13. Run the project. You should see the content of the `D:\labs\resources\DeclarationOfIndependence.txt` file displayed in the output window.

Practice 8-1: 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. Create a new `ExceptionPractice` project as the main project.
    a. Select File > New Project.
    b. Select Java under Categories and Java Application under Projects. Click the Next button.
    c. Enter the following information in the “Name and Location” dialog box:
        - Project Name: `ExceptionPractice`
        - Project Location: `D:\labs\08\practices`. (or you other directory)
        - (checked) Create Main Class: `com.example.ExceptionMain`
        - (checked) Set as Main Project
    d. Click the Finish button.
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 `ExceptionPractice` project and click Properties.
   b. In the Project Properties dialog box, select the Run category.
   c. In the Arguments field, enter a value of: `D:\labs\resources\DeclarationOfIndependence.txt`
   d. Click the OK button.

7. Run the project. You should see a message similar to:

   ```
   Reading from
   file:D:\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 `ExceptionPractice` 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 7.

    d. Click the OK button.

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 `D:\labs\resources\DeclarationOfIndependence.txt` file displayed in the output window.

Practice 8-2: Summary Level: Extending `Exception`

Overview

In this practice, you will take an existing application and refactor the code to make use of a custom exception class and a custom auto-closeable resource.

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, updating, deleting, and listing of `Employee` objects. This is the same project that you completed in the “Applying the DAO Pattern” practice.

Currently the only exceptions generated by the DAO implementation (`EmployeeDAOMemoryImpl`) are unchecked exceptions such as `ArrayIndexOutOfBoundsException`.

Future DAO implementations should not require any rewriting of the application logic (`EmployeeTestInteractive`). However, some DAO implementations will generate checked exceptions that must be dealt with. By creating a custom checked exception class that will be used to wrap any DAO generated exceptions, all DAO implementations can appear to generate the same type of exception. This will completely eliminate the need to change any application logic when you create database enabled DAO implementations in later practices.

Tasks

1. Open the `DAOException` project as the main project.
   a. Select File > Open Project.
   b. Browse to `D:\labs\08\practices`. (or you other directory)
   c. Select `DAOException` and select the "Open as Main Project" check box.
   d. Click the Open Project button.
2. Expand the project directories.
3. Run the project. You should see a menu. Test all the menu choices.

```
[C]reate | [R]ead | [U]pdate | [D]elete | [L]ist | [Q]uit:
```

4. Create a `DAOException` class in the `com.example.dao` package.
5. Complete the `DAOException` class. The `DAOException` class should:
   - Extend the `Exception` class
   - Contain four 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 the `EmployeeDAO` interface.
   - All methods should declare that a `DAOException` may be thrown during execution.
   - Extend the `AutoCloseable` interface.

7. Modify the `add` method within the `EmployeeDAOMemoryImpl` class to:
    - Declare that a `DAOException` 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 `DAOException` and deliver it to the caller of the method. The `DAOException` 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 `DAOException` and deliver it to the caller of the method. The `DAOException` should contain a message String indicating what went wrong and why.

    Note: Checking the length of the `employeeArray` could be used to determine whether the `DAOException` should be thrown. However, the use of a `try-catch` block will be typical of the structure used when creating a database-enabled DAO.
8. Modify the `update` method within the `EmployeeDAOMemoryImpl` class to:
    - Declare that a `DAOException` may be produced during execution of this method.
    - Use an if statement to validate that an existing employee is being updated. If one would not be, generate a `DAOException` and deliver it to the caller of the method. The `DAOException` 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 `DAOException` and deliver it to the caller of the method. The `DAOException` should contain a message String indicating what went wrong and why.
9. Modify the `delete` method within the `EmployeeDAOMemoryImpl` class to:
    - Declare that a `DAOException` 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 `DAOException` and deliver it to the caller of the method. The `DAOException` 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 `DAOException` and deliver it to the caller of the method. The `DAOException` should contain a message String indicating what went wrong and why.
10. Modify the `findById` method within the `EmployeeDAOMemoryImpl` class to:
    - Declare that a `DAOException` 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 `DAOException` and deliver it to the caller of the method. The `DAOException` should contain a message String indicating what went wrong and why.

11. Add a `close` method within the `EmployeeDAOMemoryImpl` class to implement the `AutoCloseable` interface.

```
@Override
public void close() {
    System.out.println("No database connection to close just
yet");
}
```

Note: The `EmployeeDAOMemoryImpl` class implements `EmployeeDAO` which extends `AutoCloseable` and, therefore, `EmployeeDAOMemoryImpl` class must provide a `close` method.

12. Modify the `EmployeeTestInteractive` class to handle the `DAOException` objects that are thrown by the `EmployeeDAO`.
    a. Import the `com.example.dao.DAOException` class.
    b. Modify the `executeMenu` method to declare that it throws an additional exception of type `DAOException`.
    c. Remove the `throws` statement from the `main` method.

public static void main(String[] args) ~~throws Exception~~

    d. Modify the `main` method to use a `try`-with-resources statement.
        - Surround the `do-while` loop with a `try` block.
        - Convert the `EmployeeDAO` and `BufferedReader` references into auto-closed resources.
        - Add a catch clause for an `IOException` to the end of the `try` block to handle both I/O errors thrown from the `executeMenu` method and when auto-closing the `BufferedReader`.

```
catch (IOException e) {
    System.out.println("Error " + e.getClass().getName() +
" , quitting.");
    System.out.println("Message: " + e.getMessage());
}
```

        - Add a second catch clause for an `Exception` to the end of the `try` block to handle errors when auto-closing the `EmployeeDAO`.

```
catch (Exception e) {
    System.out.println("Error closing resource " +
e.getClass().getName());
    System.out.println("Message: " + e.getMessage());
}
```

Note: At this point the application will compile and run, but `DAOException` instances generated will cause the application to terminate. For example, if you create an employee with an ID of 100, the application will break out of the `do-while` loop and pass to this catch clause.

e. Add a nested `try-catch` block in the `main` method that handles exceptions of type `DAOException` that may be thrown by the `executeMenu` method.

```
try {
    timeToQuit = executeMenu(in, dao);
} catch (DAOException e) {
    System.out.println("Error " + e.getClass().getName());
    System.out.println("Message: " + e.getMessage());
}
```

13. Run the project. You should see a menu. Test all the menu choices.

```
[C]reate | [R]ead | [U]pdate | [D]elete | [L]ist | [Q]uit:
```

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

```
Error com.example.dao.DAOException
Message: Error deleting employee in DAO, no such employee 1
```

Practice 8-2: Detailed Level: Extending `Exception`

Overview

In this practice, you will take an existing application and refactor the code to make use of a custom exception class and a custom auto-closeable resource.

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, updating, deleting, and listing of `Employee` objects. This is the same project that you completed in the “Applying the DAO Pattern” practice.

Currently the only exceptions generated by the DAO implementation (`EmployeeDAOMemoryImpl`) are unchecked exceptions such as `ArrayIndexOutOfBoundsException`.

Future DAO implementations should not require any rewriting of the application logic (`EmployeeTestInteractive`). However, some DAO implementations will generate checked exceptions that must be dealt with. By creating a custom-checked exception class that will be used to wrap any DAO generated exceptions, all DAO implementations can appear to generate the same type of exception. This will completely eliminate the need to change any application logic when you create database enabled DAO implementations in later practices.

Tasks

1. Open the `DAOException` project as the main project.
   a. Select File > Open Project.
   b. Browse to `D:\labs\08 \practices`. (or you other directory)
   c. Select `DAOException` and select the "Open as Main Project" check box.
   d. Click the Open Project button.
2. Expand the project directories.
3. Run the project. You should see a menu. Test all the menu choices.

```
[C]reate | [R]ead | [U]pdate | [D]elete | [L]ist | [Q]uit:
```

4. Create a `DAOException` class in the `com.example.dao` package.
5. Complete the `DAOException` class. The `DAOException` class should:
   - Extend the `Exception` class.
   - Contain four 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.

```
public class DAOException extends Exception {

    public DAOException() {
        super();
    }

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

    public DAOException(Throwable cause) {
        super(cause);
    }

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

6. Modify all the methods in the `EmployeeDAO` interface.
   - All methods should declare that a `DAOException` may be thrown during execution.
   - Extend the `AutoCloseable` interface.

```
public interface EmployeeDAO extends AutoCloseable {

    public void add(Employee emp) throws DAOException;

    public void update(Employee emp) throws DAOException;

    public void delete(int id) throws DAOException;

    public Employee findById(int id) throws DAOException;

    public Employee[] getAllEmployees() throws DAOException;

}
```

7. Modify the `add` method within the `EmployeeDAOMemoryImpl` class to:
   - Declare that a `DAOException` 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 `DAOException` and deliver it to the caller of the method. The `DAOException` 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 `DAOException` and deliver it to the caller of the method. The `DAOException` should contain a message String indicating what went wrong and why.

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

Note: Checking the length of the `employeeArray` could be used to determine whether the `DAOException` should be thrown however the use of a `try-catch` block will be typical of the structure used when create a database enabled DAO.

8. Modify the `update` method within the `EmployeeDAOMemoryImpl` class to:
    - Declare that a `DAOException` may be produced during execution of this method.
    - Use an if statement to validate that an existing employee is being updated. If one would not be, generate a `DAOException` and deliver it to the caller of the method. The `DAOException` 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 `DAOException` and deliver it to the caller of the method. The `DAOException` should contain a message String indicating what went wrong and why.

```
public void update(Employee emp) throws DAOException {
    if(employeeArray[emp.getId()] == null) {
        throw new DAOException("Error updating employee in DAO, 
no such employee " + emp.getId());
    }
    try {
        employeeArray[emp.getId()] = emp;
    } catch (ArrayIndexOutOfBoundsException e) {
        throw new DAOException("Error updating employee in DAO, 
id must be less than " + employeeArray.length);
    }
}
```

9. Modify the `delete` method within the `EmployeeDAOMemoryImpl` class to:
    - Declare that a `DAOException` 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 `DAOException` and deliver it to the caller of the method. The `DAOException` 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 `DAOException` and deliver it to the caller of the method. The `DAOException` should contain a message String indicating what went wrong and why.

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

10. Modify the `findById` method within the `EmployeeDAOMemoryImpl` class to:
    - Declare that a `DAOException` 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 `DAOException` and deliver it to the caller of the method. The `DAOException` should contain a message String indicating what went wrong and why.

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

11. Add a `close` method within the `EmployeeDAOMemoryImpl` class to implement the `AutoCloseable` interface.

```
@Override
public void close() {
    System.out.println("No database connection to close just
yet");
}
```

Note: The `EmployeeDAOMemoryImpl` class implements `EmployeeDAO` which extends `AutoCloseable` and, therefore, `EmployeeDAOMemoryImpl` class must provide a `close` method.

12. Modify the `EmployeeTestInteractive` class to handle the `DAOException` objects that are thrown by the `EmployeeDAO`.
    a. Import the `com.example.dao.DAOException` class.

    ```
    import com.example.dao.DAOException;
    ```

    b. Modify the `executeMenu` method to declare that it throws an additional exception of type `DAOException`.

    ```
    public static boolean executeMenu(BufferedReader in, EmployeeDAO
    dao) throws IOException, DAOException {
    ```

    c. Remove the `throws` statement from the `main` method.

    public static void main(String[] args) ~~throws Exception~~

    d. Modify the `main` method to use a `try`-with-resources statement.
    - Surround the `do-while` loop with a `try` block.
    - Convert the `EmployeeDAO` and `BufferedReader` references into auto-closed resources.
    - Add a catch clause for an `IOException` to the end of the `try` block to handle both I/O errors thrown from the `executeMenu` method and when auto-closing the `BufferedReader`.
    - Add a second catch clause for an `Exception` to the end of the `try` block to handle errors when auto-closing the `EmployeeDAO`.

    ```
    try (EmployeeDAO dao = factory.createEmployeeDAO();
            BufferedReader in =
    new BufferedReader(new InputStreamReader(System.in))) {
        do {
                timeToQuit = executeMenu(in, dao);
        } while (!timeToQuit);
    } catch (IOException e) {
        System.out.println("Error " + e.getClass().getName() +
        " , quitting.");
        System.out.println("Message: " + e.getMessage());
    } catch (Exception e) {
        System.out.println("Error closing resource " +
        e.getClass().getName());
        System.out.println("Message: " + e.getMessage());
    }
    ```

    Note: At this point, the application will compile and run, but `DAOException` instances generated will cause the application to terminate. For example, if you create an employee with an ID of 100, the application will break out of the `do-while` loop and pass to this catch clause.

e. Add a nested `try-catch` block in the `main` method that handles exceptions of type `DAOException` that may be thrown by the `executeMenu` method.

```
try {
    timeToQuit = executeMenu(in, dao);
} catch (DAOException e) {
    System.out.println("Error " + e.getClass().getName());
    System.out.println("Message: " + e.getMessage());
}
```

13. Run the project. You should see a menu. Test all the menu choices.

```
[C]reate | [R]ead | [U]pdate | [D]elete | [L]ist | [Q]uit:
```

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

```
Error com.example.dao.DAOException
Message: Error deleting employee in DAO, no such employee 1
```