Exception Handling Practice

Estimated time needed: 40 minutes

In this lab, you will learn how to handle runtime and user-define exceptions.

You are currently viewing this lab in a Cloud-based Integrated Development Environment (Cloud IDE). It is a fully online, integrated development environment that is pre-installed with JDK 21, allowing you to code, develop, and learn in one location.

Learning Objectives

After completing this lab, you will be able to:

- Understand what exceptions are
- Identify the characteristics of different types of exceptions
- Learn what happens when an excpetion is not caught
- Know how to create your own Exception

Exception in Java

When something is correct in most circumstances but is different or off-beat once in a while, it is called an exception. In Java, it is the same. Exceptions are events that disrupt the normal flow of a program. Say for example when you are dividing numbers based on numbers input by a user. The expectation is that the user will never enter 0 as the divisor as zero division is invalid. But if the user enters 0, it is an exceptional case which you have to handle.

There are two types of exceptions.

- `Checked Exceptions` - These are exceptions that are checked at compile-time. The programmer must handle or declare them.
- `Unchecked Exceptions` - These are exceptions that are checked at runtime. They are usually caused by logical errors in the code. You may have encountered this kind of exceptions in some of the labs you have done so far in the course.

1. Create a project directory by running the following command.

   ```
   mkdir my_exception_proj
   ```

2. Run the following code to create the directory structure.

   ```
   mkdir -p my_exception_proj/src
   mkdir -p my_exception_proj/classes
   mkdir -p my_exception_proj/test
   cd my_exception_proj
   ```

3. Now create a file named `ExceptionTrial.java` inside the src directory.

   ```
   touch /home/project/my_exception_proj/src/ExceptionTrial.java
   ```

4. Click the following button to open the file for editing.

Open **ExceptionTrial.java** in IDE

5. Read each statement in the following code carefully. What kind of exception do you think you may encounter in the code? Copy and paste it in `ExceptionTrial.java`.

```
import java.util.Scanner;
public class ExceptionTrial {
    public static void main(String s[]) {
        // Create a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);
        // Initialize an array to store up to 5 strings
        String strArr[] = new String[5];
        // Variable to track the current index for adding strings
        int strIdx = 0;
        // Infinite loop to keep the program running until the user chooses to exit
        while (true) {
            // Display the menu options to the user
            System.out.println(
                "Press 1 to add String, " +
                "\n2 to get String from a particular index, " +
                "\n3 to get the length string in any index, " +
                "\n4 to get all the strings in the array " +
                "\nany other key to exit");
            // Read the user's choice
            String userAction = scanner.nextLine();
            // Option 1: Add a string to the array
            if (userAction.equals("1")) {
                // Check if the array is already full
                if (strIdx == 5) {
                    System.out.println("There are already 5 strings in the array!");
                } else {
                    // Prompt the user to enter a string
                    System.out.println("Enter the String ");
                    String inputStr = scanner.nextLine();
                    // Add the string to the array and increment the index
                    strArr[strIdx++] = inputStr;
                }
            }
            // Option 2: Retrieve a string from a specific index
            else if (userAction.equals("2")) {
                // Prompt the user to enter the index
                System.out.println("Enter the index you want to retrieve ");
                int retIdx = Integer.parseInt(scanner.nextLine());
                // Retrieve and print the string at the specified index
                System.out.println(strArr[retIdx]);
            }
            // Option 3: Get the length of a string at a specific index
            else if (userAction.equals("3")) {
                // Prompt the user to enter the index
                System.out.println("Enter the index you check the length of ");
                int retIdx = Integer.parseInt(scanner.nextLine());
                // Retrieve the string at the specified index and print its length
                System.out.println(strArr[retIdx].length());
            }
            // Option 4: Get all the strings in the array
            else if (userAction.equals("4")) {
                for (int i=0; i<5; i++) {
                    System.out.println(strArr[i]);
                }
            }
            // Exit the program if the user enters any other key
            else {
                break;
            }
        }
    }
}
```

The program is interactive and allows the user to perform operations on an array of strings. It demonstrates basic input/output, array manipulation, and control flow.

This program allows the user to:

- Add strings to an array.
- Retrieve a string from a specific index in the array.
- Get the length of a string at a specific index.
- Get all the strings in the array.
- Exit the program.

As you look through the code, you will see possibilities of exceptions.

`Array Index Out of Bounds` - If the user enters an index greater than or equal to 5 (or less than 0), it will throw an ArrayIndexOutOfBoundsException.

`Null Pointer Exception` - If the user tries to retrieve the length of a string at an index that hasn't been populated yet, it will throw a NullPointerException.

`Number Format Exception` - If the user enters a non-integer value when prompted for an index, Integer.parseInt will throw a NumberFormatException.

6. Compile the Java program, specifying the destination directory as the `classes` directory that you created.

```
javac -d classes src/ExceptionTrial.java
```

7. Set the `CLASSPATH` variable.

```
export CLASSPATH=$CLASSPATH:/home/project/my_exception_proj/classes
```

8. Run the program and test with variable combinations.

```
java ExceptionTrial
```

See the following outputs with different exceptions

ArrayIndexOutOfBoundsException

When the index being retrieved is invalid. A valid int but less than 0 or greater than 4, given the length of the array is 5.

```
Press 1 to add String,
2 to get String from a particular index,
3 to get the length string in any index,
4 to get all the strings in the array
any other key to exit
1
Enter the String
Sears Tower
Press 1 to add String,
2 to get String from a particular index,
3 to get the length string in any index,
4 to get all the strings in the array
any other key to exit
2
Enter the index you want to retrieve
7
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 7 out of bounds for length 5
        at ExceptionTrial.main(ExceptionTrial.java:49)
```

NumberFormatException

When the index position is meant to be entered as a number but it is not.

```
Press 1 to add String,
2 to get String from a particular index,
3 to get the length string in any index,
4 to get all the strings in the array
any other key to exit
3
Enter the index you check the length of
two
Exception in thread "main" java.lang.NumberFormatException: For input string: "two"
```

```
        at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
        at java.base/java.lang.Integer.parseInt(Integer.java:588)
        at java.base/java.lang.Integer.parseInt(Integer.java:685)
        at ExceptionTrial.main(ExceptionTrial.java:56)
```

NullPointerException

When you the user tries to retrieve the length of a string at an index that hasn't been populated yet.

```
Press 1 to add String,
2 to get String from a particular index,
3 to get the length string in any index,
4 to get all the strings in the array
any other key to exit
3
Enter the index you check the length of
2
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.length()" because "<local2>[<local5>]" is null
        at ExceptionTrial.main(ExceptionTrial.java:59)
```

As you see these exceptions have interuppted the flow of the program. The code abruptly exits, when the exception is thrown. You can handle these exceptions, to ensure that they don't interrupt the flow of the program.

Exception Handling

You will now work with the same program, but this time you will handle the exceptions so the program can continue to work until the user opts out.

1. Click the following button to open the file for editing, if it is not already open.

Open **ExceptionTrial.java** in IDE

2. Replace the existing code with the following code.

```
import java.util.Scanner;
public class ExceptionTrial {
    public static void main(String s[]) {
        // Create a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);
        // Initialize an array to store up to 5 strings
        String strArr[] = new String[5];
        // Variable to track the current index for adding strings
        int strIdx = 0;
        // Infinite loop to keep the program running until the user chooses to exit
        while (true) {
            // Display the menu options to the user
            System.out.println(
                "Press 1 to add String, " +
                "\n2 to get String from a particular index, " +
                "\n3 to get the length string in any index, " +
                "\n4 to get all the strings in the array " +
                "\nany other key to exit");
            // Read the user's choice
            String userAction = scanner.nextLine();
            // Option 1: Add a string to the array
            if (userAction.equals("1")) {
                // Check if the array is already full
                if (strIdx == 5) {
                    System.out.println("There are already 5 strings in the array!");
                } else {
                    // Prompt the user to enter a string
                    System.out.println("Enter the String ");
                    String inputStr = scanner.nextLine();
                    // Add the string to the array and increment the index
                    strArr[strIdx++] = inputStr;
                }
            }
            // Option 2: Retrieve a string from a specific index
            else if (userAction.equals("2")) {
                try {
                    // Prompt the user to enter the index
                    System.out.println("Enter the index you want to retrieve ");
                    int retIdx = Integer.parseInt(scanner.nextLine());
                    System.out.println("String at index " + retIdx + ": " + strArr[retIdx]);
                } catch (NumberFormatException e) {
```

```
                    System.out.println("Invalid input! Please enter a valid integer.");
                } catch (ArrayIndexOutOfBoundsException e) {
                    System.out.println("Invalid index! Please enter an index between 0 and " + (strArr.length - 1));
                }
            }
            // Option 3: Get the length of a string at a specific index
            else if (userAction.equals("3")) {
                try {
                    // Prompt the user to enter the index
                    System.out.println("Enter the index you check the length of ");
                    int retIdx = Integer.parseInt(scanner.nextLine());
                    System.out.println("Length of string at index " + retIdx + ": " + strArr[retIdx].length());
                } catch (NumberFormatException e) {
                    System.out.println("Invalid input! Please enter a valid integer.");
                } catch (ArrayIndexOutOfBoundsException e) {
                    System.out.println("Invalid index! Please enter an index between 0 and " + (strArr.length -1));
                } catch (NullPointerException e) {
                    System.out.println("No string exists at the specified index!");
                }
            }
            // Option 4: Get all the strings in the array
            else if (userAction.equals("4")) {
                System.out.println("Strings in the array:");
                for (int i = 0; i < strIdx; i++) {
                    if (strArr[i] != null) {
                        System.out.println("Index " + i + ": " + strArr[i]);
                    } else {
                        System.out.println("Index " + i + ": (null)");
                    }
                }
            }
            // Exit the program if the user enters any other key
            else {
                break;
            }
        }
        // Close the scanner to avoid resource leak
        scanner.close();
    }
}
```

3. Compile the Java program, specifying the destination directory as the `classes` directory that you created.

```
javac -d classes src/ExceptionTrial.java
```

4. Run the program and test with variable combinations.

```
java ExceptionTrial
```

See the following outputs with different exceptions

ArrayIndexOutOfBoundsException handled

When the index being retrieved is invalid.

```
Press 1 to add String,
2 to get String from a particular index,
3 to get the length string in any index,
4 to get all the strings in the array
any other key to exit
2
Enter the index you want to retrieve
```

```
7
Invalid index! Please enter an index between 0 and 4
Press 1 to add String,
2 to get String from a particular index,
3 to get the length string in any index,
4 to get all the strings in the array
any other key to exit
```

NumberFormatException handled

When the index position is meant to be entered as a number but it is not.

```
Press 1 to add String,
2 to get String from a particular index,
3 to get the length string in any index,
4 to get all the strings in the array
any other key to exit
2
Enter the index you want to retrieve
two
Invalid input! Please enter a valid integer.
Press 1 to add String,
2 to get String from a particular index,
3 to get the length string in any index,
4 to get all the strings in the array
any other key to exit
```

NullPointerException handled

When you the user tries to retrieve the length of a string at an index that hasn't been populated yet.

```
Press 1 to add String,
2 to get String from a particular index,
3 to get the length string in any index,
4 to get all the strings in the array
any other key to exit
3
Enter the index you check the length of
4
No string exists at the specified index!
Press 1 to add String,
2 to get String from a particular index,
3 to get the length string in any index,
4 to get all the strings in the array
any other key to exit
```

Checked Exception

A checked exception is a type of exception that is checked at compile-time. The compiler ensures that these exceptions are either handled using a try-catch block, or declared in the method signature using the throws keyword. If a checked exception is not handled or declared, the program will fail to compile.

You may recollect working with one such use case while implementing Cloneable interface in this lab.

1. Create a new file named `StudentManagement.java`.

```
touch src/StudentManagement.java
```

2. Click the following button to open the file for editing, if it is not already open.

Open **StudentManagement.java** in IDE

3. Read each statement in the following program and focusing on how the `clone` method throws `CloneNotSupportedException` and paste it in `StudentManagement.java`.

```
// Student class implementing Cloneable interface
class Student implements Cloneable {
    // Fields
    private String name;
    private int age;
    private String major;
    // Constructor
    public Student(String name, int age, String major) {
        this.name = name;
        this.age = age;
        this.major = major;
    }
    // Getter and Setter methods
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getMajor() {
        return major;
    }
    public void setMajor(String major) {
        this.major = major;
    }
    // Override toString() method to provide a string representation of the object
    @Override
    public String toString() {
        return "Student [name=" + name + ", age=" + age + ", major=" + major + "]";
    }
    // Override clone() method to support cloning
    @Override
    public Student clone() throws CloneNotSupportedException {
        // Call the clone() method of the Object class
        return (Student) super.clone();
    }
}
public class StudentManagement {
    public static void main(String[] args) {
        // Create a Student object
        Student student1 = new Student("John Doe", 20, "Computer Science");
        // Clone the Student object
        Student student2 = student1.clone();
        // Print the original and cloned objects
        System.out.println("Original Student: " + student1);
        System.out.println("Cloned Student: " + student2);
        // Modify the cloned object
        student2.setName("Jane Doe");
        student2.setAge(21);
        student2.setMajor("Mathematics");
        // Print the original and cloned objects after modification
        System.out.println("\nAfter modifying the cloned object:");
        System.out.println("Original Student: " + student1);
        System.out.println("Cloned Student: " + student2);
    }
}
```

4. Compile the Java program, specifying the destination directory as the `classes` directory that you created.

```
javac -d classes src/StudentManagement.java
```

This compilation will fail as you have not handled the checked exception. You will see the compilation error as below

```
StudentManagement.java:60: error: unreported exception CloneNotSupportedException; must be caught or declared to be thrown
        Student student2 = student1.clone();
                                    ^
1 error
```

5. Replace the code with the code below.

```
// Student class implementing Cloneable interface
class Student implements Cloneable {
    // Fields
    private String name;
    private int age;
    private String major;
    // Constructor
    public Student(String name, int age, String major) {
        this.name = name;
        this.age = age;
        this.major = major;
    }
    // Getter and Setter methods
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getMajor() {
        return major;
    }
    public void setMajor(String major) {
        this.major = major;
    }
    // Override toString() method to provide a string representation of the object
    @Override
    public String toString() {
        return "Student [name=" + name + ", age=" + age + ", major=" + major + "]";
    }
    // Override clone() method to support cloning
    @Override
    public Student clone() throws CloneNotSupportedException {
        // Call the clone() method of the Object class
        return (Student) super.clone();
    }
}
public class StudentManagement {
    public static void main(String[] args) {
        try {
            // Create a Student object
            Student student1 = new Student("John Doe", 20, "Computer Science");
            // Clone the Student object
            Student student2 = student1.clone();
            // Print the original and cloned objects
            System.out.println("Original Student: " + student1);
            System.out.println("Cloned Student: " + student2);
            // Modify the cloned object
            student2.setName("Jane Doe");
            student2.setAge(21);
            student2.setMajor("Mathematics");
            // Print the original and cloned objects after modification
            System.out.println("\nAfter modifying the cloned object:");
            System.out.println("Original Student: " + student1);
            System.out.println("Cloned Student: " + student2);
        } catch(CloneNotSupportedException cnse) {
            System.out.println("You can't clone this object!");
        }
    }
}
```

In this case you are handling the exception.

6. Compile the Java program, specifying the destination directory as the `classes` directory that you created.

```
javac -d classes src/StudentManagement.java
```

This compilation will be successful.

7. Run the class and see the program run smoothly.

User-defined Exception

You can create your own Checked Exceptions to handle situations like validation. Let's assume you want to check that the student age is never less than 18.

1. Create a new file named `StudentUnderAgeException.java`.

```
touch src/StudentUnderAgeException.java
```

2. Click the following button to open the file for editing, if it is not already open.

Open **StudentUnderAgeException.java** in IDE

3. Read each statement in the following code and paste it in `StudentUnderAgeException.java`.

```
// Custom exception class for underage students
public class StudentUnderAgeException extends Exception {
    public StudentUnderAgeException(String message) {
        super(message);
    }
}
```

4. Compile the code.

```
javac -d classes src/StudentUnderAgeException.java
```

5. Edit `StudentManagement.java` to handle the age validation. If age is not a number it should handle `NumberFormatException`. If age is a number but is less than 18, it should handle the `StudentUnderAgeException`.

```
import java.util.Scanner;
// Student class implementing Cloneable interface
```

```
class Student implements Cloneable {
    // Fields
    private String name;
    private int age;
    private String major;
    // Constructor
    public Student(String name, int age, String major) throws StudentUnderAgeException {
        this.name = name;
        // Check if the provided age is less than 18
        if (age < 18) {
            // If the age is less than 18, throw a StudentUnderAgeException
            // with a custom error message
            throw new StudentUnderAgeException("Student age has to be 18 or more");
        }
        // If the age is valid (18 or older), set the age field to the provided value
        this.age = age;
        this.major = major;
    }
    // Getter and Setter methods
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
    public String getMajor() {
        return major;
    }

    // Override toString() method to provide a string representation of the object
    @Override
    public String toString() {
        return "Student [name=" + name + ", age=" + age + ", major=" + major + "]";
    }
    // Override clone() method to support cloning
    @Override
    public Student clone() throws CloneNotSupportedException {
        // Call the clone() method of the Object class
        return (Student) super.clone();
    }
}
public class StudentManagement {
    public static void main(String[] args) {
        // Create a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);
        try {
            System.out.println("Enter student name");
            String name = scanner.nextLine();
            System.out.println("Enter student age");
            int age = Integer.parseInt(scanner.nextLine());
            System.out.println("Enter student major");
            String major = scanner.nextLine();
            // Create a Student object
            Student student1 = new Student(name, age, major);
            Student student2 = student1.clone();
            System.out.println("Student: " + student1+ " successfully created!");
            System.out.println("Student: " + student2+ " successfully cloned!");
        } catch(CloneNotSupportedException cnse) {
            System.out.println("You can't clone this object!");
        } catch (NumberFormatException nfe) {
            System.out.println("Age has to be a number");
        } catch(StudentUnderAgeException suae) {
            System.out.println(suae.getMessage());
        }
    }
}
```

6. Compile the code.

```
javac -d classes src/StudentManagement.java
```

7. Run the code.

```
java StudentManagement
```

You can refer below for a sample output.

NumberFormatException handled

```
Enter student name
Lav
Enter student age
forty
Age has to be a number
```

StudentUnderAgeException handled

```
Enter student name
Lav
Enter student age
17
Enter student major
Data Science
Student age has to be 18 or more
```

Practice Exercise

1. Create a user defined exception that ensure that the name is valid.

   Hint: Use the `matches` method in String class like this. `name.matches("^[a-zA-Z' -]+$")`

▶ Click here for sample code

Conclusion

In this lab, you learned how about Exceptions and how to handle them. You also learned how to create user defined exception.

Author(s)

Lavanya