Java Exceptions

Lab setup

- Create a new Java project called "JavaExceptions"
- Add the provided files to the project
- Students are encouraged to watch Tutorial R5: Exception Handling under Module 01 on Canvas

Exceptions as error-management

An exception is an event that disrupts the normal flow of a program. It is a form of error-management.

When an error occurs within a method it can create an `exception` object containing information about the error and `throws` the exception. The runtime will then try to find something in the call stack to handle or `catch` the exception.

If an appropriate handler is not found, the program terminates.

The example we will use for this lab is a prize redemption system for an arcade using tickets won during games. There are some possible errors that are handled using `if` statements in the current implementation. We will start with defining exceptions for these errors and then implementing how to handle them.

These errors are:

InsufficientFundsException
A prize redemption was attempted by a user without enough tickets

RequestedItemException
The prize a user is attempting to redeem is unavailable

InvalidUserException
The specified user does not exist in the database

Exception class

The `Exception` class is a form of `Throwable` representing conditions that an application can catch and handle.

Writing our own exceptions

Graded Submission Task 1

1. Create an exception class for each error that is checked in the program. An example is provided in Figure 1.

```
public class InsufficientFundsException extends Exception {
    double amount;
    //...
}
```

Figure 1 InsufficentFundsException example

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 1. You will submit this copy at the end of this lab.

Throwing exceptions

Methods which throw exceptions are declared using the keyword `throws` followed by a list of exceptions it can throw (the list is separated by commas). The actual statement which throws the exception uses the keyword `throw` followed by an exception object.

Graded Submission Task 2

1. Declare that the `PrizeMachine` method `vendItem` throws the `RequestedItemException`.
2. Declare the method `RedeemTickets` in `Main` throws all three defined exceptions.
3. Add the appropriate `throw` statements where each exception can occur.

The program will not build at this point, but you will still submit as-is.

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 2. You will submit this copy at the end of this lab.

Handling exceptions

Once thrown, the program needs to handle the exception, or else the program will terminate. We do this using a combination of try/catch blocks.

```
try {
    RedeemTickets(u, machines.get(0))
}//...
```

Figure 2 Try-block surrounding method which throws an Exception

`try` block

A try block (shown in Figure 2) surrounds a portion of protected code which can throw exceptions. Immediately following the try block should be `catch` or `finally` blocks.

try-with-resources

The try-with-resources block is like the `try` block, and it allows you to declare one or more resources for use in the block. Resources are objects that must be closed once the program is done working with them. The try-with-resources block will automatically handle closing the resources that are declared.

```
try (FileReader reader = new FileWriter(path)){
    ///...
}
```

Figure 3 Try-with-resources block

`catch` block

A catch block begins with a catch statement declaring the type of exception it is trying to catch. The code which is run in the event the exception is caught is placed inside the block. This code most likely handles the error in some way to restore the state of the program and logs some information regarding what caused the exception or gracefully terminates the program.

```
// immediately after try-block
catch (InsufficientFundsException ex){
    //...
}
```

Figure 4 Catch-block for InsufficientFundsException

Multiple catch blocks can be placed after a try block to handle different kinds of exceptions. A single catch block can also handle more than one exception as shown in Figure 5.

```
catch (IOException|FileNotFoundException ex){
    //...
}
```

Figure 5 Catch-block for IOExeption and FileNotFoundExepction

`finally` block

A finally block follows a try or catch block. This code always executes even if no exception occurred. It can be used for cleanup such as after opening a file.

```
finally {
    //...
}
```

Figure 6 Finally block

Graded Submission Task 3

Add exception handling to the code that requires it in the given example. For now, it suffices to simply print the stack trace.

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 3. You will submit this copy at the end of this lab.

Nested try-catch blocks

Try blocks can be nested within other try blocks and paired with corresponding catch blocks for better control in exception handling. Take the following code as an example.

```
try (DirectoryStream<Path> dir = Files.newDirectoryStream(Path.of("data"))) {
    for (Path p : dir) {
        BufferedReader br = Files.newBufferedReader(p);
        br.readLine();
    }
} catch (IOException e) {
    System.err.println("Terminate reading directory");
}
```

Figure 7 Reading files from a directory (one try-catch).

In this code, we are opening a directory stream to iterate over the files in a directory. This block of code can throw an IOException in a couple places. First, when the directory stream is first created, then whenever we open the BufferedReader for each of the files. In its current state, if any of the BufferedReaders throw an exception then the code will stop reading from the entire directory. Suppose that isn't what we want, then we can nest another try-catch block within the outer try-catch block which handles IOExceptions thrown by the BufferedReaders.

```
try (DirectoryStream<Path> dir = Files.newDirectoryStream(Path.of("data"))) {
    for (Path p : dir) {
        try (BufferedReader br = Files.newBufferedReader(p)) {
            br.readLine();
        } catch (IOException e) {
            System.err.println("Error reading file %s".formatted(p.toString()));
            System.err.println("Skipping...");
        }
    }
} catch (IOException e) {
    System.err.println("Terminate reading directory");
}
```

Figure 8 Reading files from a directory (nested try-catch).

In this new code, when a BufferedReader throws an IOException it is caught by the inner catch block. Now, we are handling what happens when a BufferedReader fails. In this case, we will simply skip the file and move to the next file in the directory.

Lab wrap-up

In this lab we discussed exceptions, how to define our own and how to handle them for error management. We can define our own exceptions by extending the Exception base class. Methods can throw exceptions using the keyword throw. They also need to show what exceptions they can throw as part of the method header using the keyword throws.

We handle exceptions by catching them. We surround code which can throw exceptions in a try block. Following the try block, we include catch blocks for each exception that can be thrown. If we want something to be done after exception handling, whether an exception was thrown or not, we can use the finally block as the last block in the sequence of blocks.

Submission notes for graded tasks:

- Submit a zipped folder containing the Eclipse projects for each graded task.
- For each project, ensure you have included the src folder containing the .java file, the .classpath file, and the .project file.
- Ensure that each project is named appropriately to correspond with each task.
- Zip the projects together and name the zipped folder "Lab20_cslogin", where cslogin is your login ID for the computers at the Department of Computer Science at ODU.
- Submit the zipped file to the respective Canvas link.

Rubrics		
Task 1 (points allotted) - Compilation (2) - InsufficientFundsException (2) - RequestedItemException (2) - InvalidUserException (2) - Each new exception class defined (2)	Task 2 (points allotted) - vendItem throws RequestedItemException (2) - RedeemTickets throws all 3 exceptions (2) - Throw statements added where needed (6)	Task 3 (points allotted) - Compilation (2) - Exceptions handled in Main (4) - Each catch prints stack trace (4)