

ITSC 1213 - Exceptions

Introduction

I am pretty sure you have encountered exceptions at some point working on the various programs we've introduced in the course. For example, when entering letters as user input when the program is designed for numbers, accessing the fourth element of an array of three elements, or trying to execute a method using an object reference variable/field that has not been initialized.

This lab focuses on the use of Exceptions to catch a variety of errors that can occur, allowing your program to take appropriate corrective action.

Concepts covered in this lab:

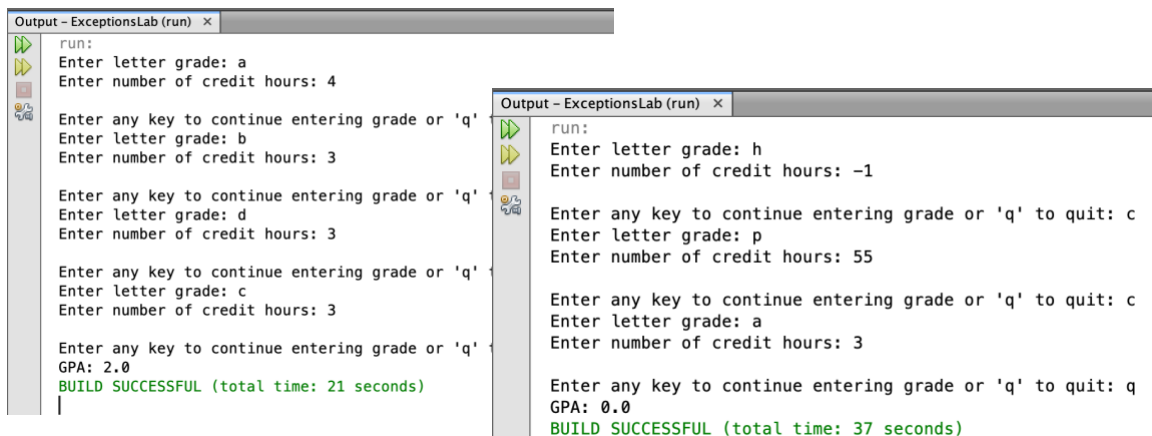
- try/catch blocks
- throws keyword

Required files or links

- Code snippets linked in document.

Part A: Setup

- 1) Create a new project in NetBeans. It doesn't matter what you call it and you don't have to add a main class as we will add one in the next step.
- 2) Add a new Java class to the newly created project and call it `GPACalculator`.
- 3) Copy and paste the code from this [code snippet](#) into this file. Make sure you include the correct package declaration at the beginning of the file to match what you have in your project.
- 4) As this file has a main method and doesn't contain any errors you should be able to run it. Go ahead and run the file to get a feel of what it does.



```
Output - ExceptionsLab (run) x
run:
Enter letter grade: a
Enter number of credit hours: 4

Enter any key to continue entering grade or 'q'
Enter letter grade: b
Enter number of credit hours: 3

Enter any key to continue entering grade or 'q'
Enter letter grade: d
Enter number of credit hours: 3

Enter any key to continue entering grade or 'q'
Enter letter grade: c
Enter number of credit hours: 3

Enter any key to continue entering grade or 'q'
GPA: 2.0
BUILD SUCCESSFUL (total time: 21 seconds)

Output - ExceptionsLab (run) x
run:
Enter letter grade: h
Enter number of credit hours: -1

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: p
Enter number of credit hours: 55

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: a
Enter number of credit hours: 3

Enter any key to continue entering grade or 'q' to quit: q
GPA: 0.0
BUILD SUCCESSFUL (total time: 37 seconds)
```

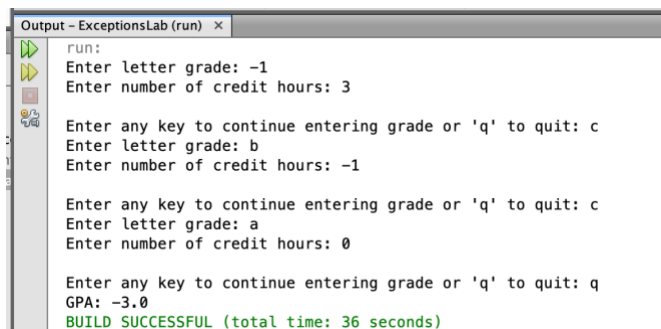
- 5) When your program works and you are satisfied with the result, show your work to the instructional team to get checked off and proceed to Part B.

Part B: Simple Error Handling

It is likely that as you were interacting with the `GPACalculator` program you encountered some runtime errors depending on the input values you tried to enter. Taking a closer look at our program we would notice that we do not have any form of error checking and handling in our code. The closest thing we have is the if statements in the `getLetterGradeValue` that return -1 when an entered letter does not match any of our letter grades to points mapping.

Let's try and see how our program behaves when we don't give it the values it expects and can handle properly.

- 1) Run the `GPACalculator` program and try the following scenario that uses negative values when prompted for a letter grade, and negative and zero values for the credit hours.



```
Output - ExceptionsLab (run) x
RUN:
Enter letter grade: -1
Enter number of credit hours: 3

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: b
Enter number of credit hours: -1

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: a
Enter number of credit hours: 0

Enter any key to continue entering grade or 'q' to quit: q
GPA: -3.0
BUILD SUCCESSFUL (total time: 36 seconds)
```

You will notice that the program runs without failing but the calculated gpa does not make much sense. What can we do to make our program more robust?

One simple way is adding some conditional statements to check for valid/invalid values before we include them into the calculation of the GPA.

- 2) Let's go ahead and add a couple of if/else statements to check for invalid user input. Update the logic inside the while loop to look like this:

```

while (quitCmd != 'q') {
    // Get user data
    System.out.print("Enter letter grade: ");
    letterGrade = scnr.nextLine().charAt(0);
    // Error checking, unknown letter grade
    if (getLetterGradeValue(letterGrade) == -1) {
        System.out.println("Invalid letter grade. Try again!");
    } else {
        System.out.print("Enter number of credit hours: ");
        creditHours = scnr.nextInt();
        // Error checking, negative credit hours
        if (creditHours < 0) {
            System.out.println("Invalid number of credit hours. "
                + "Try again!");
        } else {
            totalCreditHrs += creditHours;
        }

        qualityPoints += getLetterGradeValue(letterGrade) * creditHours;
        System.out.print("\nEnter any key to continue entering grade "
            + "or 'q' to quit: ");
        quitCmd = scnr.next().charAt(0);
        scnr.nextLine();
    }
}
}

```

Notice here we added the first if-else block to check if the user input was a letter grade that mapped to a point value. If that check fails the “Invalid letter grade. Try again!” message will appear and the user will be prompted to try again.

If the user enters a valid letter grade the next step is to check if the user entered a proper number of credit hours. In this case we treat values less than zero as invalid and the “Invalid number of credit hours. Try again!” message will appear and the user will be prompted to either continue or quit.

- 3) Run the program and try different values, both valid and invalid, when prompted for a letter grade and the credit hours to see how the program behaves and handles these situations.

Output - ExceptionsLab (run) X

```

run:
Enter letter grade: a
Enter number of credit hours: 3

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: d
Enter number of credit hours: 4

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: h
Invalid letter grade. Try again!
Enter letter grade: b
Enter number of credit hours: -3
Invalid number of credit hours. Try again!

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: b
Enter number of credit hours: 3

Enter any key to continue entering grade or 'q' to quit: c
GPA: 1.0
BUILD SUCCESSFUL (total time: 1 minute 8 seconds)

```

Output - ExceptionsLab (run) X

```

run:
Enter letter grade: -1
Invalid letter grade. Try again!
Enter letter grade: h
Invalid letter grade. Try again!
Enter letter grade: f
Enter number of credit hours: -3
Invalid number of credit hours. Try again!

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: f
Enter number of credit hours: 3

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: a
Enter number of credit hours: 4

Enter any key to continue entering grade or 'q' to quit: q
GPA: 2.0
BUILD SUCCESSFUL (total time: 47 seconds)

```

- 4) When your program works and you have tried different value and have a good understanding of the results, show your work to the instructional team to get checked off and proceed to Part C.

Part C: Error Checking using Exception Handling

In part B we were able to see that adding error-checking code using if-else statements can help prevent these types of errors from occurring but it makes our code clunky, and obscures the normal flow of the program. Luckily, Java has special constructs, try, throw, and catch, known as exception-handling constructs, to keep error-checking code separate and to reduce redundant checks.

Let's update our code to utilize these constructs and make a code a bit more efficient.

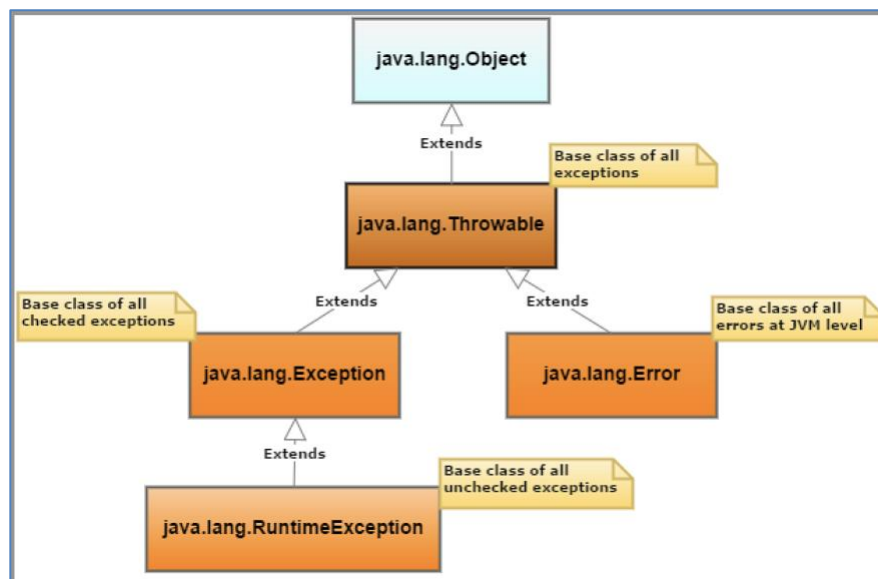
- 1) Update the logic inside the while loop to look like this:

```
while (quitCmd != 'q') {
    try {
        // Get user data
        System.out.print("Enter letter grade: ");
        letterGrade = scnr.nextLine().charAt(0);
        // Error checking, unknown letter grade
        if (getLetterGradeValue(letterGrade) == -1) {
            throw new Exception("Invalid letter grade.");
        }
        System.out.print("Enter number of credit hours: ");
        creditHours = scnr.nextInt();
        // Error checking, negative credit hours
        if (creditHours < 0) {
            throw new Exception("Invalid number of credit hours. "
                                + "Try again!");
        }
        totalCreditHrs += creditHours;

        qualityPoints += getLetterGradeValue(letterGrade) * creditHours;
    } catch (Exception excpt) {
        // Prints the error message passed by throw statement
        System.out.println(excpt.getMessage());
        System.out.println("Cannot compute GPA");
    }
    System.out.print("\nEnter any key to continue entering grade "
                    + "or 'q' to quit: ");
    quitCmd = scnr.next().charAt(0);
    scnr.nextLine();
}
```

Notice how now the normal code flow is not obscured by error-checking/handling if-else statements. The flow is clearly: Get letter grade, then get number of credit hours, when done print GPA.

Java offers several built-in Throwable types like Error, Exception, and classes derived(inherited) from these. The Exception class (and other Throwable types) has a constructor that can be passed a String, as in `throw new Exception("Invalid weight. ");`, which allocates a new Exception object and sets an internal String value that can later be retrieved using the `getMessage()` method, as in `System.out.println(excpt.getMessage());`.



Exceptions – subset hierarchy in Java

- 2) Run the program and try different values, both valid and invalid, when prompted for a letter grade and the credit hours to see how the program behaves and handles these situations.
- 3) When your program works and you have tried different values and have a good understanding of the results, show your work to the instructional team to get checked off and proceed to Part D.

```
Output - ExceptionsLab (run) x
RUN:
Enter letter grade: h
Invalid letter grade.
Cannot compute GPA

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: -1
Invalid letter grade.
Cannot compute GPA

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: a
Enter number of credit hours: -1
Invalid number of credit hours. Try again!
Cannot compute GPA

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: a
Enter number of credit hours: 3

Enter any key to continue entering grade or 'q' to quit: q
GPA: 4.0
BUILD SUCCESSFUL (total time: 25 seconds)
```

Part D: InputMismatchException

An example of where an exception is particularly useful is dealing with user input. In our program we need to read integer from the input. Users tend to be imperfect; they might mistype the input. Thus, we need to be careful to make sure that they actually enter a number to be added. The `Scanner` class helps with this as its `nextInt()` method throws an exception if the what is being read isn't an integer.

- 1) Run you the program again and now try entering something other than an integer number when prompted for the credit hours.

```
Output - ExceptionsLab (run) x
run:
Enter letter grade: -1
Enter number of credit hours: 3

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: h
Enter number of credit hours: 3

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: a
Enter number of credit hours: j
Exception in thread "main" java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
    at exceptionslab.GPACalculator.main(GPACalculator.java:33)
/Users/nanajjar/Library/Caches/NetBeans/12.6/executor-snippets/run.xml:111: The following error occurred while executing this line:
/Users/nanajjar/Library/Caches/NetBeans/12.6/executor-snippets/run.xml:68: Java returned: 1
BUILD FAILED (total time: 48 seconds)
```

```
Output - ExceptionsLab (run) x
run:
Enter letter grade: -1
Enter number of credit hours: 0

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: a
Enter number of credit hours: p
Exception in thread "main" java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
    at exceptionslab.GPACalculator.main(GPACalculator.java:33)
/Users/nanajjar/Library/Caches/NetBeans/12.6/executor-snippets/run.xml:111: The following error occurred while executing this line:
/Users/nanajjar/Library/Caches/NetBeans/12.6/executor-snippets/run.xml:68: Java returned: 1
BUILD FAILED (total time: 28 seconds)
```

We can see here that the code in the try block can throw a different type of exception. This is not uncommon for a code in a try block to throw more than one type of exception. In such situations a catch block can to be written for each type of exception that could potentially be thrown.

There are three main things to keep in mind in such cases:

- The JVM will run the first compatible catch clause found.
- The catch blocks must be listed from most specific to most general.
- If the last catch block catches `Exception`, then it will catch any type of `Exception` not caught in the proceeding blocks.

- 2) Now we need to alter the code to handle this type of exception. Add a second catch block to handle the invalid input type. Note the name of the exception class in parentheses should match the type that could be thrown. Make sure to add this in the correct order with the other catch block.

```

catch(InputMismatchException excpt){
    // Prints the error message passed by throw statement
    scnr.nextLine();//consume the new line from the last nextInt call
    System.out.println("Invalid type entered. Please try again!");
}

```

- 3) Run the program and try different values, both valid and invalid, when prompted for the number of credit hours to see how the program behaves and handles these situations.
- 4) When your program works and you have tried different values and have a good understanding of the results, show your work to the instructional team to get checked off and proceed to Part E.

Part E: ArithmeticException

As our program performs an arithmetic operation that involves integer division it is possible to run into a scenario where the program attempts to divide by a zero value.

- 1) Run the program and enter 0 when prompted for the number of credit hours.

```

Output - ExceptionsLab (run) x
run:
Enter letter grade: a
Enter number of credit hours: 0

Enter any key to continue entering grade or 'q' to quit: c
Enter letter grade: b
Enter number of credit hours: 0

Enter any key to continue entering grade or 'q' to quit: q
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at exceptionslab.GPACalculator.computeIntGPA(GPACalculator.java:84)
    at exceptionslab.GPACalculator.main(GPACalculator.java:55)
/Users/nanajjar/Library/Caches/NetBeans/12.6/executor-snippets/run.xml:111: The following error occurred while executing this line:
/Users/nanajjar/Library/Caches/NetBeans/12.6/executor-snippets/run.xml:68: Java returned: 1
BUILD FAILED (total time: 11 seconds)

```

- 2) Now we need to alter the code to handle this type of exception. You might find yourself wondering where to place the try-catch block. Should we place it inside the method computeIntGPA where the calculations performed or in the main method when we call that method?

One principle to follow when it comes to Exception handling is “throw early catch late”. The basic idea here is we should throw an exception as soon as we can, and catch it late as much as possible. This allows for having all the information to handle it properly. In our case it would be best to handle it in the main method where the method call happens. Go ahead and add the try-catch block to surround the call to computeIntGPA:

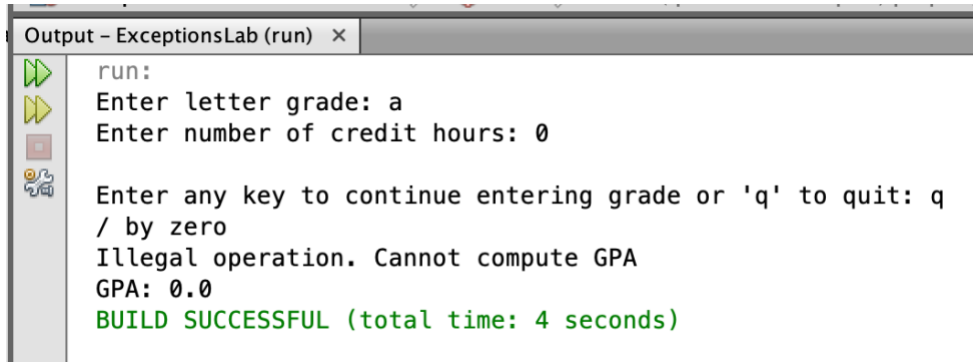
```

// calculate GPA - Divide the total quality points by the total credit hours.
try {
    double gpa = computeIntGPA(qualityPoints, totalCreditHrs);
    //Print user GPA info
    System.out.println("GPA: " + gpa);
} catch (ArithmeticException excpt) {
    System.out.println(excpt.getMessage());
    System.out.println("Illegal operation. Cannot compute GPA");
}

```

Now if you run the program, it will try the calculation, throw a division by 0 exception, catch the exception in the catch block, and calculate the monthly pension using another non-zero value. This way, the program can execute fully.

- 3) Run the program and try entering zero when prompted for the number of credit hours to see how the program behaves.



The screenshot shows an IDE output window titled "Output - ExceptionsLab (run)". The output text is as follows:

```
run:
Enter letter grade: a
Enter number of credit hours: 0

Enter any key to continue entering grade or 'q' to quit: q
/ by zero
Illegal operation. Cannot compute GPA
GPA: 0.0
BUILD SUCCESSFUL (total time: 4 seconds)
```

The IDE interface includes a vertical toolbar on the left with icons for running (green play button), stepping through (yellow play button), and debugging (red bug icon).

- 4) When your program works and you have tried different values and have a good understanding of the results, show your work to the instructional team to get checked off for this lab.