Week 6: Computer Science 1

Iteration Continued & Error Handling

Break and Continue

Break

The `break` statement is used to exit a loop early. We have seen this in the past with the `switch` statement. It can also be used with `for` and `while` loops.

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break;
    }
    System.out.println(i);
}
```

This will print out the numbers 0-4 and then exit the loop.

Continue

The `continue` statement is used to skip the rest of the code inside a loop for the current iteration only. The loop does not terminate but continues on with the next iteration.

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        continue;
    }
    System.out.println(i);
}
```

This will print out the numbers 0-4 and 6-9. It will skip the number 5.

Error Handling

At this point, you have encountered a few different types of errors in your programs. These can be broken down into two categories:

- Syntax Errors
- Runtime Errors

Syntax errors are caught by the compiler and are usually easy to fix. Runtime errors are errors that occur while the program is running and can be more difficult to track down.

Let's look at how to handle runtime errors.

Errors that occur during the execution of a program are called **exceptions**. Exceptions can occur for many reasons. Some examples:

- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- The program tries to perform an operation that is not allowed.

An example of the last bullet point is trying to divide by zero. This will cause a runtime error called an `ArithmeticException`. When this happens, we say that the exception has been "thrown". If not handled properly, the program will terminate.

When an error is "thrown" we want to "catch" it and handle it properly.

Let's try dividing by zero and see what happens.

```java
public class Example {
    public static void main(String[] args) {
        int x = 5;
        int y = 0;
        int z = x / y;
        System.out.println(z);
    }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Example.main(Example.java:4)
```

The error message is telling us what exception was thrown and where it occurred. We can use this information to handle the exception properly.

Note that the line number will depend on where the error occurs in your program.

There are many different types of exceptions in Java. Here are a few examples:

- `ArithmeticException`
- `InputMismatchException`
- `ArrayIndexOutOfBoundsException`
- `FileNotFoundException`

We can also create our own exceptions, but we won't cover that in this class.

Let's stick to the first two for now.

Try-Catch Blocks

To handle exceptions, we use a `try-catch` block. The `try` block contains the code that might throw an exception. The `catch` block contains the code that handles the exception.

```
try {
    // code that might throw an exception
} catch (ExceptionType e) {
    // code to handle the exception
}
```

When an exception is thrown, the program will jump to the `catch` block and execute the code there. If no exception is thrown, the `catch` block is skipped.

Another example of an exception is a mismatched data type. If we prompt the user for an integer and they enter a string, we will get a `InputMismatchException`.

Try running the following code and entering a string when prompted for an integer.

```java
import java.util.Scanner;

public class Example {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int number = input.nextInt();
        System.out.println("You entered: " + number);
    }
}
```

We need a way to catch this exception and handle it properly.

We can do this with a `try-catch` block.

```java
import java.util.Scanner;
import java.util.InputMismatchException;

public class Example {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        try {
            System.out.print("Enter a number: ");
            int number = input.nextInt();
            System.out.println("You entered: " + number);
        } catch (InputMismatchException e) {
            System.out.println("Invalid input. Please enter a number.");
        }
    }
}
```

The `catch` block will execute if an `InputMismatchException` is thrown. We've seen what would happen if we didn't have the `try-catch` block. The program would terminate.

We are passing the exception object `e` to the `catch` block. This object contains information about the exception that was thrown. We can use this object to print out the error message. This can be useful for debugging.

Also, notice that I am importing the `InputMismatchException` class. This is necessary to use the class in our program.

Let's print out the error message.

```java
import java.util.Scanner;
import java.util.InputMismatchException;

public class Example {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        try {
            System.out.print("Enter a number: ");
            int number = input.nextInt();
            System.out.println("You entered: " + number);
        } catch (InputMismatchException e) {
            System.out.println("Invalid input. Please enter a number.");
            System.out.println(e);
        }
    }
}
```

Now we can prompt the user for input and handle any exceptions that occur.

If an exception is thrown, we can prompt the user to enter the correct input and continue the program.

```java
import java.util.*;

public class Example {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int number = 0;
        boolean valid = false;
        while (!valid) {
            try {
                System.out.print("Enter an integer: ");
                number = input.nextInt();
                valid = true;
            } catch (InputMismatchException e) {
                System.out.println("Invalid input. Please enter a number.");
                input.next();
            }
        }
        System.out.println("You entered: " + number);
    }
}
```

Let's examine two parts on the previous example.

First, look at my import statement. I am importing the entire `java.util` package with `import java.util.*;`. In previous examples, I have imported specific classes from the `java.util` package. Both are valid ways to import classes from a package.

Second, notice that I am using the `input.next();` method to clear the input buffer. If I don't do this, the program will enter an infinite loop. This is because the `nextInt()` method will keep trying to read the same input and keep throwing the same exception.

Remove the `input.next();` line and see what happens.

We will return to error handling later in the class to cover more advanced topics. This will include creating our own exceptions for specific cases.

Let's use today's class to practice iteration. We'll start with a few simple problems and then move on to some more complex ones. You can work with a partner or group to solve these problems if you'd like.

Make sure to use the tools we've learned in class:

- `for` loops
- `while` loops
- Conditional statements
- Scanner
- `break` and `continue`
- Error handling

Along with any other topics we have covered.

Problem 1:

Write a program that reads an unspecified number of integers, determines how many positive and negative integers have been read, and computes the total and average of the input values (not counting zeros). Your program ends with the input 0. Display the average as a floating-point number.

```
Enter an integer, the input ends if it is 0: 1 2 -1 3 0
The number of positives is 3
The number of negatives is 1
The total is 5.0
The average is 1.25
```

```
Enter an integer, the input ends if it is 0: 0
No numbers are entered except 0
```

```java
import java.util.Scanner;

public class Problem1 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int positive = 0;
        int negative = 0;
        int total = 0;
        int count = 0;
        System.out.print("Enter an integer, the input ends if it is 0: ");
        int number = input.nextInt();
        if (number == 0) {
            System.out.println("No numbers are entered except 0");

        } else{
            while (number != 0) {
                if (number > 0)
                    positive++;
                else
                    negative++;
                total += number;
                count++;
                number = input.nextInt();
            }
            double average = total / (double) count;
            System.out.println("The number of positives is " + positive);
            System.out.println("The number of negatives is " + negative);
            System.out.println("The total is " + total);
            System.out.println("The average is " + average);
        }
    }
}
```

Problem 2:

Write a program that creates a math test for the user to take. The test will consist of 5 questions that are randomly generated. The user will be prompted to enter their answer for each question. After the test is complete, the program will display the number of correct answers and the number of incorrect answers.

You need to use the `math.random()` method to generate random numbers for the test. The following code will generate a random number between 0 and 10.

```
int number = (int) (Math.random() * 10);
```

`math.random()` returns a number between 0 and 1. We multiply this number by 10 to get a number between 0 and 10. We then cast this number to an `int` to get a whole number.

Here is an example of what the program should look like when it runs.

```
What is 3 + 4? 7
You are correct!
What is 5 + 2? 8
Your answer is wrong.
5 + 2 should be 7
What is 9 + 1? 10
You are correct!
What is 6 + 3? 9
You are correct!
What is 7 + 2? 9
You are correct!
You got 4 correct answers.
You got 1 incorrect answers.
```

```java
import java.util.Scanner;

public class Problem2 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int correct = 0;
        int incorrect = 0;
        for (int i = 0; i < 5; i++) {
            int number1 = (int) (Math.random() * 10);
            int number2 = (int) (Math.random() * 10);
            System.out.print("What is " + number1 + " + " + number2 + "? ");
            int answer = input.nextInt();
            if (number1 + number2 == answer) {
                System.out.println("You are correct!");
                correct++;
            } else {
                System.out.println("Your answer is wrong.");
                System.out.println(number1 + " + " + number2 + " should be " + (number1 + number2));
                incorrect++;
            }
        }
        System.out.println("You got " + correct + " correct answers.");
        System.out.println("You got " + incorrect + " incorrect answers.");
    }
}
```

Problem 3:

An integer greater than 1 is prime if its only positive divisor is 1 or itself. For example, 2, 3, 5, and 7 are prime numbers, but 4, 6, 8, and 9 are not. The problem is to display the first 50 prime numbers in five lines, each of which contains
ten numbers. The problem can be broken into the following tasks:

- Determine whether a given number is prime.
- For number = 2, 3, 4, 5, 6, . . ., test whether it is prime.
- Count the prime numbers.
- Display each prime number, and display ten numbers per line.

Here is the algorithm:

- Set the number of prime numbers to be printed as a NUMBER_OF_PRIMES;
- Use count to track the number of prime numbers and
- set an initial count to 0;
- Set an initial number to 2;

```
while (count < NUMBER_OF_PRIMES) {
    Test whether number is prime;
    if number is prime {
        Display the prime number and increase the count;
    }
    Increment number by 1;
}
```

Output:

```
The first 50 prime numbers are

2 3 5 7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229
```

```java
public class Problem3 {
    public static void main(String[] args) {
        int NUMBER_OF_PRIMES = 50;
        int NUMBER_OF_PRIMES_PER_LINE = 10;
        int count = 0;
        int number = 2;
        System.out.println("The first 50 prime numbers are \n");
        while (count < NUMBER_OF_PRIMES) {
            boolean isPrime = true;
            for (int divisor = 2; divisor <= number / 2; divisor++) {
                if (number % divisor == 0) {
                    isPrime = false;
                    break;
                }
            }
            if (isPrime) {
                count++;
                if (count % NUMBER_OF_PRIMES_PER_LINE == 0) {
                    System.out.println(number);
                } else {
                    System.out.print(number + " ");
                }
            }
            number++;
        }
    }
}
```