User Input and Exception Handling

Part A - Setting up

Create a new project in your IDE called **LuckyNumberGame.**

Copy the starter code into your project.

Run the starter code. Does it work? What happens if you enter an integer from the keyboard?

Part B - Testing your code

When you test your program, make sure you test it with a variety of values. Test the program with the following inputs. Put a ✓ if the code runs properly and ❌if there is an error.

Input test data	Expected Result	Actual Result (✓/❌)
1.65	✓	
0	✓	
3	✓	
abcd	❌	

Part C - Exception Handling

When a program has an **error**, it is known as an **exception**.

As programmers, we need to **predict** when exceptions might occur and write code to **handle** them.

This is known as **Exception Handling**.

When you input ***abcd*** into the program, you get an **InputMismatchException**.

This is a classic Java error and we need to be able to **handle** it, otherwise our program will **crash**!

In Java, we can create a **try/except** block of code to do this.

Replace your main function with:

```
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int number;

        try {
               System.out.print("Enter an integer: ");
               number = scanner.nextInt();
               System.out.println("You entered: " + number);
               scanner.close();
        } catch (InputMismatchException e) {
                System.out.println("Invalid input. Try again!");
                scanner.nextLine(); // clear the input buffer
        }
    }
}
```

Try the code above. Does it work? *Recommend you type the code rather than copy/paste*. Use the test table above to test your code.

Hopefully your program **catches** the **exception** instead of **crashing**.

Note, if there is no error, the **except** code does not run.

Part D - Keep asking!

Great! We have handled an exception!
But how can we force users to keep trying until they enter a valid integer?
Let's use a **while loop** strategy.

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int number;
    boolean valid = false;

    while(!valid) { //or while(valid==false)
       try {
            System.out.print("Enter an integer: ");
            number = scanner.nextInt();
            System.out.println("You entered: " + number);
            valid = true;
            scanner.close();
       }catch(InputMismatchException e) {
            System.out.println("Invalid input. Try again!");
            scanner.nextLine(); // clear the input buffer
       }
    }
}
```

Run and test the code. Does it work?

Hopefully, the program will keep asking you to enter a valid number from the keyboard.

.

Part E - Code Analysis

Get together with a classmate(s) and together discuss the code.

Are there lines of code that are puzzling you?

Discuss.

Later we will form a class discussion.

YOU DO NOT HAVE TO FULLY UNDERSTAND OR MEMORISE THE CODE.

BUT MAKE SURE YOU GET THE CONCEPT.

A GOOD STRATEGY TO CHECK YOUR UNDERSTANDING IS TO TALK TO YOUR CLASSMATES OR TEACHER.

ANOTHER GOOD STRATEGY IS TO USE THE CONCEPT IN FUTURE ASSIGNMENTS.