Lesson –12 - Exception Handling

Problem 1: Unchecked Exception Practice. Use this ShoppingCart.java file to handle Unchecked Exceptions to satisfy the given Tasks part.

```
import java.util.Scanner;
public class ShoppingCart {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      System.out.print("Enter quantity of items to add (1-50): ");
      int quantity = sc.nextInt();

      System.out.println("Successfully added " + quantity + " items to your cart!");
   }
}
```

Tasks

1. **Handle wrong input type**
   - If the user enters a string, decimal, or symbol instead of an integer, catch `InputMismatchException` and display a user-friendly message.
2. **Handle invalid range**
   - If the quantity entered is **less than 1** or **greater than 50**, throw and handle `IllegalArgumentException` with a proper message.
3. **Graceful retry**
   - After catching an exception, prompt the user again until valid input is entered. (Use While loop)

Problem 2: Custom Exception

In your Assignment you can get the CustomerAccount Java class with the implemented methods. Your task is to satisfy the given Part A-C requirement and need to create a TestClass with the main() method.

Part A: Create Custom Exception

- Define **one class** → `AccountException extends RuntimeException`.
- Use its constructor to accept a **message** string.
- Example messages:
    - `"Insufficient funds! Withdrawal amount exceeds balance."`
    - `"Low balance warning! Balance cannot go below $100.`

Part B: Modify Methods

1. **Withdraw method (`withdraw(double amount)`)**
    - Apply a check for withdrawal greater than the balance → raise your custom exception with the message *"Insufficient funds! Withdrawal amount exceeds balance."*
    - Apply a check if the withdrawal reduces the balance below **$100** → raise your custom exception with the message *"Low balance warning! Balance cannot go below $100."*
2. **Deposit method (`deposit(double amount)`)**
    - Ensure the deposit amount is positive.
    - If a negative deposit is attempted, raise an appropriate **API exception** (`IllegalArgumentException`) with a clear message.
3. **Get Balance method (`getBalance()`)**
    - No exception handling is needed. Simply return the balance.

Part C: Driver/Test Class

- In `main()`, create an account and test:
    1. Deposit with negative input → `IllegalArgumentException`.
    2. Withdraw more than balance → `AccountException` with "Insufficient funds…" message.
    3. Withdraw valid amount that drops below $100 → `AccountException` with "Low balance warning…" message.
    4. Successful deposit and withdraw.

Problem 3: You have fully implemented Stack.java and TestStack.java given in the assignments as a startup code. Your job is to complete the given tasks for the proper Exceptions usage.

Tasks

1. **Underflow (use API exception)**
   - For `pop()` and `peek()` when the stack is empty, throw **`java.util.EmptyStackException`**.
2. **Overflow (use API exception)**
   - For `push(...)` when the stack is full, throw **`IllegalStateException`** with a clear message, e.g., `"Stack Overflow: cannot push, stack is full."`
3. **Null value (use API exception)**
   - For `push(null)`, throw **`NullPointerException`** with a clear message, e.g., `"Null values are not allowed."`
4. **Update Test Code**
   - In `TestStack`, add **`try/catch` blocks** to demonstrate:
     - Overflow on `push`
     - Null push
     - Underflow on `pop` and on `peek`
   - Print friendly messages when exceptions are caught.