Exercise 2

Weight: 20%
Deadline: 17 November 2025 by 23:59.

Exercise Overview

This exercise is designed to assess your understanding and application of core Java concepts. You will work with a partially completed UCMS project. Some code lines have been removed. Your tasks involve restoring functionality by implementing object-oriented principles and handling exceptions to improve program robustness.

The exercise is divided into three sections:

1. Section A: Abstract Classes (6 %)
2. Section B: Interfaces (7 %)
3. Section C: Exception Handling (7 %)

Instructions for opening and setting up the project files can be found on page 6.

Section A: Abstract Classes (6 %)

The User.java file contains a class named **`User`**. It defines generic attributes and behaviours for UCMS users. Through Inheritance, we have derived classes that represent specific users like Student, Admin, and Lecturer. As such we do not intend to instantiate objects directly from User. Therefore, your task is to convert it into an abstract class with one abstract method. This method will generate reports for various system users. But this exercise focuses on the report for Admin whose sample output is depicted below.

```
============================================================
                    UCMS SYSTEM REPORT
============================================================
Total Number of Courses          : 10
Total Number of Students Enrolled : 500
Total Number of Lecturers        : 50
------------------------------------------------------------
Report generated by: Admin
Date: 06-Nov-2025
============================================================
Press any key to continue...
```

Figure 1: Admin Report Format

Instructions 1 – 7 describe the task in detail:

1. Convert the **`User`** class into an abstract class.
2. Navigate to line 76 in the User.java file and declare the following abstract method:

   `generateReport(String[] reportParameters)`

3. Only **`Admin`** class should implement the method declared in Step 2. All other subclasses of **`User`** must override it with an empty body. This means that they acknowledge the method but do not provide any functionality for it.

4. Override the `generateReport(String[] reportParameters)` method and implement code for generating the report illustrated in Figure 1. This should be done on lines 135 – 150 inside Admin.java, replacing the existing placeholder code where necessary. Your implementation should display data received through the `reportParameters` argument.

   Note: Lines 135–150 of this file currently contain code for formatting the output to match the layout depicted in **Figure 1**.

5. Open UCMS.java and navigate to lines 190 - 206. This block contains code for gathering the data that is passed to `generateReport()`, which you have overridden and implemented in Admin. We do not expect detailed report. It should display basic statistics such as number of students, courses, and lecturers. The report should also capture report generation date and time as well as the name of the admin who generated it. Some skeleton code is already provided, for example, the `numStudents` variable is correctly set. However, `numCourses` and `numLecturers` are currently set to static values. You must replace the static values with dynamic data retrieved from the data structures that store respective objects.
6. In the same UCMS.java class, navigate to line 158. Below this line, implement logic that uses the report generation method declared in the `Admin` class.

Section B: Interfaces (7 %)

This section requires you to implement interfaces. As discussed in class, Interfaces are special constructs used to define **shared behaviour** that can be implemented by related and unrelated classes. An interface named `Assignable<T>` is provided in the custom Interfaces package.

Interface: `Assignable<T> {}`

You are provided with an interface named `Assignable<T>{}` located in Interfaces package (folder). This interface utilizes the concept of Generics in Java. The parameter `<T>` serves as a placeholder for data type that will be specified later (when the interface, class, or method is used). This interface provides behaviour declaration that can be shared between unrelated classes or subclasses. The `Assignable` interface provides the following behaviour declarations:

```java
package Interfaces;

public interface Assignable <T> {
    void assignTo(T object);
    T getAssignee();
    boolean isAssigned();
    void unassign();
}
```

Figure 2: Interfaces Code Snippet

Task Description

Update the following classes to implement the **`Assignable <T>`** interface:

1. Module.java
2. Lecturer.java

Each class must correctly specify the type parameter **`<T>`** and provide concrete implementations for all behaviour declarations defined in the **`Assignable`** interface. Table 1 describes the expected behaviour of each method.

#	Method	Description/Expected Behaviour
1	**`assignTo(T object)`**	Assign the object parameter by updating the corresponding field in the implementing class. - In Module.java, this method should assign a lecturer to the module by updating the **`moduleLecturer`** field. You should also update the **`modulesTaught`** field on the lecturer object. - In Lectuer.java, it should assign the course to the lecturer by updating the **`assignedCourse`** field.
2	**`getAssignee( )`**	Return the assigned object. - In Module class, this should the assigned lecturer. - In Lecturer class, it should return the course or courses to which the lecturer is assigned.
3	**`isAssigned( )`**	Check whether the current object (e.g., a module or lecturer) has been assigned. Return true if assigned, otherwise false.
4	**`unassign(T obj)`**	Remove the current assignment by resetting the reference field. For module, this method should first check whether the module is assigned.

Section C: Basic Exception Handling (7%)

Task 1: Evaluate and Refactor Exception Handling

1. Review all methods in **User.java** that declare or throw exceptions. You should assess whether the current use of exception handling is appropriate for the types of errors that may realistically occur.
2. Refactor the code where necessary by either:
    a. Removing unnecessary or redundant exception-handling statements, or
    b. Replacing them with more suitable error-handling mechanisms, such as conditional checks, input validation, or meaningful error messages.
3. Add notes in the form of comments to justify the changes or replacements made.

Expected Outcome:

You should demonstrate the ability to identify when exceptions are truly necessary, distinguishing between exceptional conditions and normal control flow.

Task 2: Handle Invalid User Input Gracefully

The Utility.java file in the Utilities package contains code that prompts the user and reads input. Users are expected to select their role by entering a non-negative integer between 1 and 4. Currently, if a user enters invalid input, such as "one" instead of 1, the program terminates abruptly and displays the error message shown in Figure 3. You are required to update the program to handle invalid input gracefully when selecting a role.

```
============================================================
                        PICK YOUR ROLE
============================================================
    1. Admin
    2. Lecturer
    3. Student
    4. Exit
************************************************************
Enter your choice: onw
Exception in thread "main" java.util.InputMismatchException Create breakpoint
    at java.base/java.util.Scanner.throwFor(Scanner.java:964)
    at java.base/java.util.Scanner.next(Scanner.java:1619)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2284)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2238)
    at Utilities.Utility.printMenu(Utility.java:50)
    at UCMS.determineRole(UCMS.java:37)
    at Main.main(Main.java:3)
```

Figure 3: Invalid Input Consequences

Your implementation should:

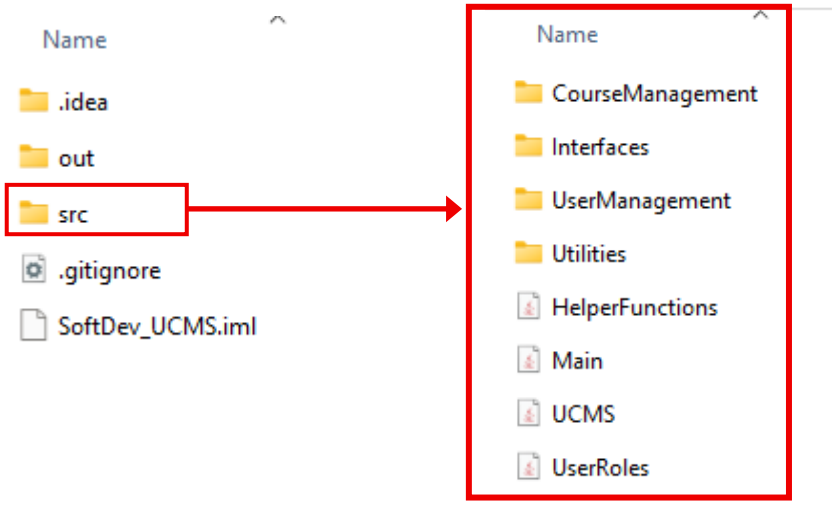
1. Handle invalid input errors without terminating the program.
2. Provide clear feedback to the user, for example: “Invalid selection. Please enter a valid number.”
3. Allow the user to retry input until a valid value is entered.

Submission Instructions

1. Ensure your project runs without errors.
2. Compress the project folder and rename it using the following format: [YourName]_UCMS_CA1-2.zip. Ensure the directory contains all project files (.java) in their respective packages (CourseManagement, UserManagement, Interfaces, Utilities).
3. Do not create or introduce any new java class files.
4. Upload the zipped folder on Blackboard before the deadline.

Additional Instructions

1. After downloading the file, extract it to any directory on your computer. It is recommended to place it in the same folder as your other Java project files. For example, if you are using IntelliJ IDEA, this could be under: C://Users/[Username]/IdeaProjects/
2. If you open the directory in File Explorer, you should see the following folder structure:



3. Open the project in IntelliJ IDEA by navigating to File → Open and selecting the folder where you extracted the project.
4. After completing Step 3, IntelliJ IDEA should display the project as shown below: