**ACS-2947-002/050**

**Assignment 1**

**Due by Monday, February 5, 11:59 pm**

**Instructions**

- Submit your `.java` files (together in a `Assign1.zip` file) via Nexus.
- Include your name and student number as a comment in every file.
- Document the classes using Javadoc notation.
- Include comments as needed and use exception handling where necessary.

**PART A - LinkedLists (55 marks)**

When a student wishes to register for a course, they are placed on a waitlist if the class is already full. Students can register from the waitlist when a spot become available and if they have permission. In our simplified simulation of a waitlist, the system randomly performs one of four operations and updates the waitlist accordingly. For simplicity, we assume that:

- only one operation can be performed in a day;
- a waitlisted student has a maximum of two days to complete their registration once they are permitted to register. If not registered after two days, the student becomes the last person on the list, in order to give other students a chance;
- only one student (first person on the list) can be granted permission to register at any point.
- A waitlisted student is a student with a waitlist status (***Waitlisted*** or ***PermittedToRegister***) and may have some number of days left to complete registration if they have been permitted to register.

The details of each operation and the rules applying are provided below:

**Operation 1: No addition to the waitlist**

This operation does not change the number of students on the waitlist but an existing status or days left to register may be changed for a student with permission to register, or the first student may be granted permission to register if no one previously had permission.

**Operation 2: A new student joins the waitlist**

Details of the student must be provided and the student added to the waitlist. A new student is always added to the end of the list with a *Waitlisted* status. An existing status or days left to register may be changed after adding the new student or the first student may be granted permission to register if no one previously had permission.

**Operation 3: The first waitlisted student registers**

First student registers if they already had permission, otherwise, they are granted permission to register.

**Operation 4: Check if a student is on the waitlist.**

Details of the student to check in the list must be provided and the result of the search displayed. An existing status or days left to register may be changed after the check or the first student may be granted permission to register if no one previously had permission. Two students are considered the same if they have the same student number and name.

The simulation randomly chooses one of these operations until the waitlist is empty or 20 operations have been performed randomly. You are expected to do the following to complete the simulator.

1. Program the generic `CircularDoublyLinkedList` class. See appendix for CDLL representation.
   - Use the circularly and doubly linked list classes from our notes as a basis.
   - Start with the DLL and think of how it can work circularly.
     - Sentinels are not required, only reference to the last node.
     - `Node` will have references to the next and previous nodes.
     - `addBetween` and `remove` private utilities will be used, all adding and removing will work like doubly.
   - Your CircularlyDoublyLinkedList class will support the following public methods:

     `size()` Returns the number of elements in the list.

     `isEmpty()` Returns true if the list is empty, and false otherwise.

     `first()` Returns (but does not remove) the first element in the list.

     `last()` Returns (but does not remove) the last element in the list.

     `addFirst(e)` Adds a new element to the front of the list.

     `addLast(e)` Adds a new element to the end of the list.

     `removeFirst()` Removes and returns the first element of the list.

     `removeLast()` Removes and returns the last element of the list.

     `rotate()` Advances to the next element in the list.

     `contains(e)` Returns true if `e` is in the list, and false otherwise

2. Create the following classes:
   a) `Student` stores the student number and name of a student. Include an `equals` method to check if an object is the same as an instance of the student. (Two students are considered the same if they have the same student number and name). Include any methods required to retrieve student details.

   b) `WaitlistedStudent` is a type of `Student` that has a `Status` and some number of days left to register.
      - Add a constructor that works the same way that a Student is created but initializes the registration status to *Waitlisted* and the number of days left to register to 0.
      - Add a constructor that allows the number of days left to register to be specified
      - Include any methods required to retrieve or update the status or number of days left to register.
      - `Status` is an enum type: *Waitlisted*, *PermittedToRegister*

   c) `CourseWaitlist` class that simulates the random operation on a course waitlist with some waitlisted students.

      **Note:** most of the work will be done here. `Assign1PartA_Driver` should have only a minimal amount of code: e.g., declare/create a `CourseWaitlist` instance, create initial waitlisted students, and initiate the course waitlist simulation.
      - Fields
        - a `CircularDoublyLinkedList` of waitlisted students
        - any other field required to manage the student waitlist operations

- Upon instantiation, set up the initial list of waitlisted students (use 4 initial waitlisted students for your demo/sample data) and grant the first student permission to register (when a student is granted permission, they have two days to register). Begin the simulation.
- The simulation randomly chooses one of the operations and continues until the list is empty or 20 operations have been performed.
- After each operation, the list of students on the waitlist is displayed together with the waitlist status and the number of days left to register (if they have permission to register).
- When the simulation stops, the termination condition of the simulation is displayed and the final waitlist is also displayed.

3. Create a driver class called `Assign1PartA_Driver` that invokes a simulation of the course waitlist using a list of four waitlisted students of your choice.

**Sample Output**

```
ACS-2947 Waitlist
1111111 : Jeremy, Status: PermittedToRegister Days left: 2
2222222 : Anne, Status: Waitlisted
3333333 : Jacob, Status: Waitlisted
4444444 : Melissa, Status: Waitlisted

System chose option 3: Jeremy registers
2222222 : Anne, Status: PermittedToRegister Days left: 2
3333333 : Jacob, Status: Waitlisted
4444444 : Melissa, Status: Waitlisted

System chose option 2: A new student joins the waitlist
Enter student name and number: 5555555 Caleb

2222222 : Anne, Status: PermittedToRegister Days left: 1
3333333 : Jacob, Status: Waitlisted
4444444 : Melissa, Status: Waitlisted
5555555 : Caleb, Status: Waitlisted

System chose option 4: Checks if a student is waitlisted
Enter student name and number: 7777777 John

Student not found in waitlist

3333333 : Jacob, Status: PermittedToRegister Days left: 2
4444444 : Melissa, Status: Waitlisted
5555555 : Caleb, Status: Waitlisted
2222222 : Anne, Status: Waitlisted

System chose option 4: Checks if a student is waitlisted
Enter student name and number: 5555555 Caleb

Student found in waitlist
3333333 : Jacob, Status: PermittedToRegister Days left: 1
4444444 : Melissa, Status: Waitlisted
5555555 : Caleb, Status: Waitlisted
2222222 : Anne, Status: Waitlisted
```

```
System chose option 1: No addition to waitlist (status/days left may change)
4444444 : Melissa, Status: PermittedToRegister Days left: 2
5555555 : Caleb, Status: Waitlisted
2222222 : Anne, Status: Waitlisted
3333333 : Jacob, Status: Waitlisted

System chose option 3: Melissa registers
5555555 : Caleb, Status: PermittedToRegister Days left: 2
2222222 : Anne, Status: Waitlisted
3333333 : Jacob, Status: Waitlisted

System chose option 1: No addition to waitlist (status/days left may change)
5555555 : Caleb, Status: PermittedToRegister Days left: 1
2222222 : Anne, Status: Waitlisted
3333333 : Jacob, Status: Waitlisted

System chose option 1: No addition to waitlist (status/days left may change)
2222222 : Anne, Status: PermittedToRegister Days left: 2
3333333 : Jacob, Status: Waitlisted
5555555 : Caleb, Status: Waitlisted

System chose option 2: A new student joins the waitlist
Enter student name and number: 6666666 Jasleen

2222222 : Anne, Status: PermittedToRegister Days left: 1
3333333 : Jacob, Status: Waitlisted
5555555 : Caleb, Status: Waitlisted
6666666 : Jasleen, Status: Waitlisted

System chose option 4: Checks if a student is waitlisted
Enter student name and number: 6666666 Jasleen

Student found in waitlist

3333333 : Jacob, Status: PermittedToRegister Days left: 2
5555555 : Caleb, Status: Waitlisted
6666666 : Jasleen, Status: Waitlisted
2222222 : Anne, Status: Waitlisted

System chose option 3: Jacob registers
5555555 : Caleb, Status: PermittedToRegister Days left: 2
6666666 : Jasleen, Status: Waitlisted
2222222 : Anne, Status: Waitlisted

System chose option 3: Caleb registers
6666666 : Jasleen, Status: PermittedToRegister Days left: 2
2222222 : Anne, Status: Waitlisted

System chose option 4: Checks if a student is waitlisted
Enter student name and number: 6666666 Jasleen
```

```
Student found in waitlist

6666666 : Jasleen, Status: PermittedToRegister Days left: 1
2222222 : Anne, Status: Waitlisted

System chose option 4: Checks if a student is waitlisted
Enter student name and number: 7777777 John

Student not found in waitlist

2222222 : Anne, Status: PermittedToRegister Days left: 2
6666666 : Jasleen, Status: Waitlisted

System chose option 4: Checks if a student is waitlisted
Enter student name and number: 2222222 Anne

Student found in waitlist

2222222 : Anne, Status: PermittedToRegister Days left: 1
6666666 : Jasleen, Status: Waitlisted

System chose option 1: No addition to waitlist (status/days left may change)
6666666 : Jasleen, Status: PermittedToRegister Days left: 2
2222222 : Anne, Status: Waitlisted

System chose option 1: No addition to waitlist (status/days left may change)
6666666 : Jasleen, Status: PermittedToRegister Days left: 1
2222222 : Anne, Status: Waitlisted

System chose option 2: A new student joins the waitlist
Enter student name and number: 7777777 John

2222222 : Anne, Status: PermittedToRegister Days left: 2
7777777 : John, Status: Waitlisted
6666666 : Jasleen, Status: Waitlisted

System chose option 2: A new student joins the waitlist
Enter student name and number: 8888888 Joey

2222222 : Anne, Status: PermittedToRegister Days left: 1
7777777 : John, Status: Waitlisted
6666666 : Jasleen, Status: Waitlisted
8888888 : Joey, Status: Waitlisted
Reached 20 days/operations!

Final waitlist:
2222222 : Anne, Status: PermittedToRegister Days left: 1
7777777 : John, Status: Waitlisted
6666666 : Jasleen, Status: Waitlisted
8888888 : Joey, Status: Waitlisted
```

**PART B Stacks (25 marks)**

Create a program that simulates the undo/redo features of an application. Implement a simple calculator that asks the user for the basic arithmetic operation that they would like to perform on the *last result*. Your program will:

- Prompt the user to enter an initial number (first operand)
- Then
    - prompt the user for the next operator and second operand
    - evaluate the expression
    - present the user with the result, which will be the new first operand
- This will continue until the user chooses to quit or undo (or redo) an operation
- Handle addition (+), subtraction (-), multiplication (* or x) and division (/) operations.

The undo operation restores the last state. The redo operation restores the next state if an undo was previously performed.

1. Create the generic `SinglyLinkedList` class that we discussed in class.
2. Create the generic `LinkedStack` that implements the provided `Stack` interface using a `SinglyLinkedList`.
3. Create a driver class called `Assign1PartB_Driver` and any other classes/methods that you may require.

You **must** use two stack objects to hold items that will restore the states based on the function (undo `"z"` or redo `"y"`) selected. Allow the user to quit (`"q"`) anytime.

**Sample Output**

```
Simple Calculator: type z to undo, y to redo, q to quit

Enter the first number:
10
Enter the next operation on 10:
+ 2
= 12
Enter the next operation on 12:
- 3
= 9
Enter the next operation on 9:
* 4
= 36
Enter the next operation on 36:
z
UNDO: 9
Enter the next operation on 9:
z
UNDO: 12
Enter the next operation on 12:
y
```

```
REDO: 9
Enter the next operation on 9:
y
REDO: 36
Enter the next operation on 36:
y
Nothing to redo.
Enter the next operation on 36:
* 2
= 72
Enter the next operation on 72:
z
UNDO: 36
Enter the next operation on 36:
z
UNDO: 9
Enter the next operation on 9:
z
UNDO: 12
Enter the next operation on 12:
z
UNDO: 10
Enter the next operation on 10:
z
Nothing to undo.
Enter a number:
3
Enter the next operation on 3:
q
Goodbye!
```

**Suggestions:**

- If you need to understand the use of undo and redo a bit more, type a few characters in a text editor and execute a series of undo/redo operations to see its effect.
- Check that your implementation work for the above sample output.
- You may assume that the calculator only works with integer values.

**Submission**

Submit your **`Assign1.zip`** file that includes all the assignment files (`CircularDoublyLinkedList.java`, `CourseWaitlist.java`, `Student.java`, `WaitlistedStudent.java`, `Status.java`, `Assign1PartA_Driver.java`, `Stack.java`, `LinkedStack.java`, `SinglyLinkedList.java`, `Assign1PartB_Driver.java`, and any other classes used in your implementation) via **Nexus**.

## Appendix A (CDLL Representation)

The structure of a CDLL storing Integer types is shown below.