EQUIPMENT LOANING MANAGEMENT SYSTEM

1. OBJECTIVES

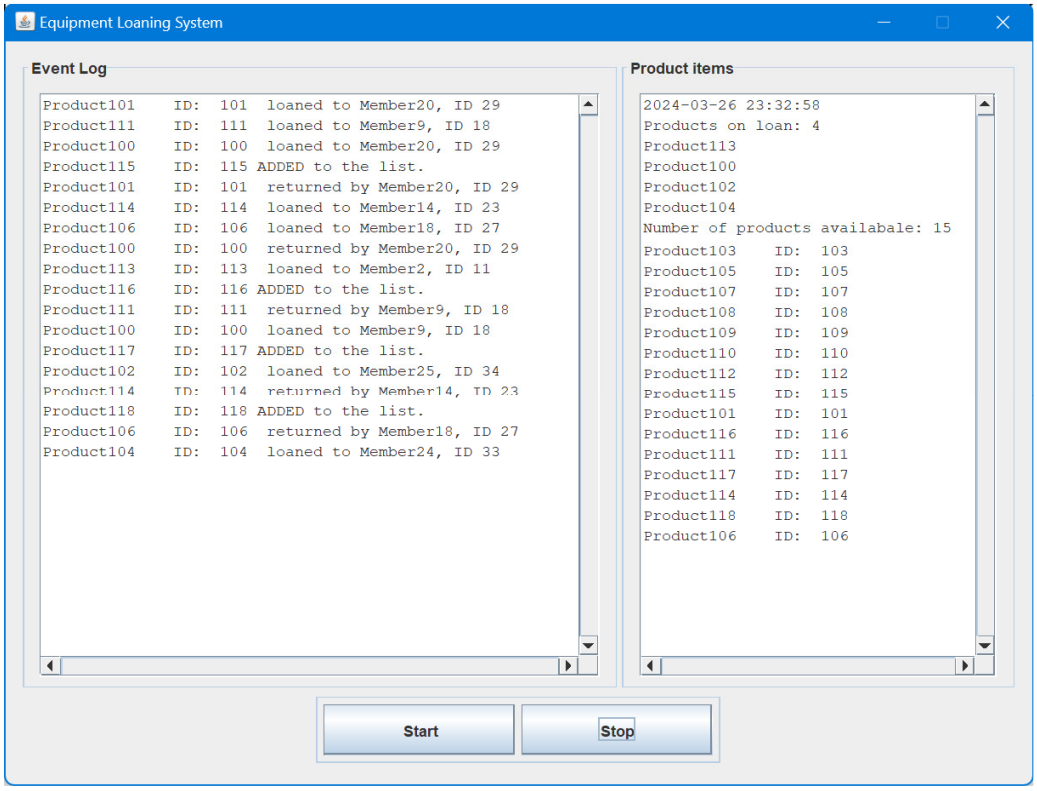
The main objectives of this assignment are:

- To learn how to create and initiate multiple threads to execute tasks concurrently.
- To utilize some of the basic features of the Thread.

2. DESCRIPTION

Your task is to develop a multithreaded application that simulates a loaning equipment system to manage the loaning of appliances and tools. The application is designed for use in a neighborhood where appliances are stored in a common warehouse. The program should simulate the process of loaning items to members, adding new items, and returning loaned items.

The system maintains three main lists: a list of products, a list of members (who can loan a product), and a list of items on loan. Each product (Product) is defined by an ID and a name, each member (Member) is identified by an ID and a name, and each loaned item (LoanItem) is associated with a member and a product. These are the minimum required pieces of information to be handled, but you can certainly add more attributes to each object type. The following scenario can serve as a guide in designing a solution for the system:



2.1 Simulation notes

Tasks such as loaning and returning items, along with other actions, should be simulated randomly, with time represented in seconds, where each second signifies one day.

When randomizing the time for a task to be performed, use the **sleep** (or C# **Sleep**)method of the **Thread** class for a randomly selected period in seconds, representing the number of days. For instance, to simulate the loan of an item for **n** days, let the thread sleep for **n** seconds:

```
Thread.sleep(sleepTimeInSeconds);  //C#: Thread.Sleep
```

Note: Use of a Timer control is not permitted.!

2.2 Manager classes:

It is recommended that each of the above -mentioned lists is managed in a separate class, such as **ProductManager**, **MemberManager**, and **LoanItemManager**.

The manager classes should contain methods for adding, removing, and fetching objects from the respective lists.

Advanced hint: You may consider creating a generic manager class that includes all common methods and then utilize this class within the manager classes (using for example inheritance).

2.3 Test values:

Upon the start of the application, generate a set of test products (e.g., 15 products) and a number of test members (e.g., 25), assigning test values to their attributes. For instance, Product100, Product101, Member1, Member2, and so forth.

However, you are encouraged to use more descriptive names such as "Makita Lawn Mower" for a product and "House 32" or "Mr. X" for member names. Alternatively, consider preparing lists of products and members in text files and loading these files into your application upon startup.

The creation of test values in the code should be managed by the ProductManager and MemberManager classes, to fill the product list and member lists, respectively.

2.4 Tasks:

The activities of the system can be represented as four separate tasks, each of which must be performed by a separate thread. These tasks are: LoanTask, ReturnTask, AdminTask, and UpdateGUITask::

1. **LoanTask**: This task simulates loaning a randomly selected Product to a randomly selected member at random intervals, e.g. ranging from 2 to 6 seconds. These two objects create a loan item, which is then added to the loan item list. The product is subsequently removed from the product list, ensuring it only contains products available for loaning. As a simplification, the product can be added to the end of the list. The member, however, is not removed from the member list, allowing them to loan another product at a later time. This

task should not proceed when the product list is empty, indicating that all elements are currently on loan.

2. **ReturnTask**: This task simulates the return of a loaned item at random times, reflecting the loan period. The return task can select to process a return after a randomly selected amount of time, such as between 3 and 15 seconds. The item to be returned is also selected randomly from the list of LoanItems.

   Once selected, the LoanItem's product is added back to the list of products before removing the loan item from the loan item list. The member is not re-added to the member list since it was not removed when the loan item was created, thus maintaining a constant member list throughout runtime.

3. **AdminTask**: This task adds a new test product to the product list at random intervals, e.g. ranging from 6 to 16 seconds.

   As an optional feature, a randomly selected product can be removed from the list of products to simulate lost or out-of-order items. The choice of adding or removing can also be randomized, with adding occurring more frequently and removal happening occasionally.

4. **UpdateGUI** (UI): This task is relatively small, focusing solely on updating the user interface by displaying the list of products currently available for loan and the list of items on loan. It simply lists the items stored in the Product list and LoanItem list, respectively. Updating can occur regularly, such as every second or every two seconds.

2.5 Threads

Each of the tasks mentioned above should be executed by a separate thread. Since there are four tasks, four threads need to be utilized, one dedicated to each task. To achieve this, create an object of each task, instantiate one thread for each task, and start the threads. Consequently, all threads will operate simultaneously, executing their tasks concurrently.

Each thread executes its assigned task within a loop, conditioned by a boolean flag:

```
public void Run()
{
  while (isRunning)
  {
     //Code
  }
 //other code
```

To start the thread execution, set isRunning to true, and set it to false to stop the thread.

Note: Do not use deprecated method of the Thread class, to terminate or suspend a thread.

3. USER-INTERFACE

While it is allowed to develop a command-line application (Console application in C#), it is recommended to create an application with a graphical user interface (GUI). The design of the user interface is flexible and can be designed according to your preferences.

4. CODING QUALITY

Make all attempts to use of a well-organized project structure and well-structured classes, avoiding lengthy methods. Ensure that methods are adequately commented to enhance code readability and understanding.

The user interface class should solely manage input and output (IO) operations and other interactions with the user. All business logic should be encapsulated within related classes.

Hint: Consider creating a class, such as LoanSystemManager, responsible for initiating the application by creating tasks, threads, and managing their execution. This class should leverage manager classes when accessing lists is necessary.

5. GRADING AND SUBMISSION

This and all future assignments are presented individually during lab hours. However, the solution must be uploaded to Canvas either before or after the presentation. Supervisors will provide feedback, and if additional work is required, you'll be given time for resubmission and a new presentation.

Before submission, compress all files, folders, and subfolders into a zip, rar, or 7z file format. Upload this compressed file via the Assignment page on Canvas. Ensure that you submit the correct version of your project and that it has been compiled and thoroughly tested beforehand.

Avoid using hard-coded file paths (such as paths to image files on your C-drive) in your source code, as they will not function properly on other computers. Projects that fail to compile and run correctly, or exhibit poor code quality, will be returned for completion and resubmission.

Good Luck!

Farid Naisan,
Course Responsible and Instructor