Doing Tasks Concurrently

# Objectives

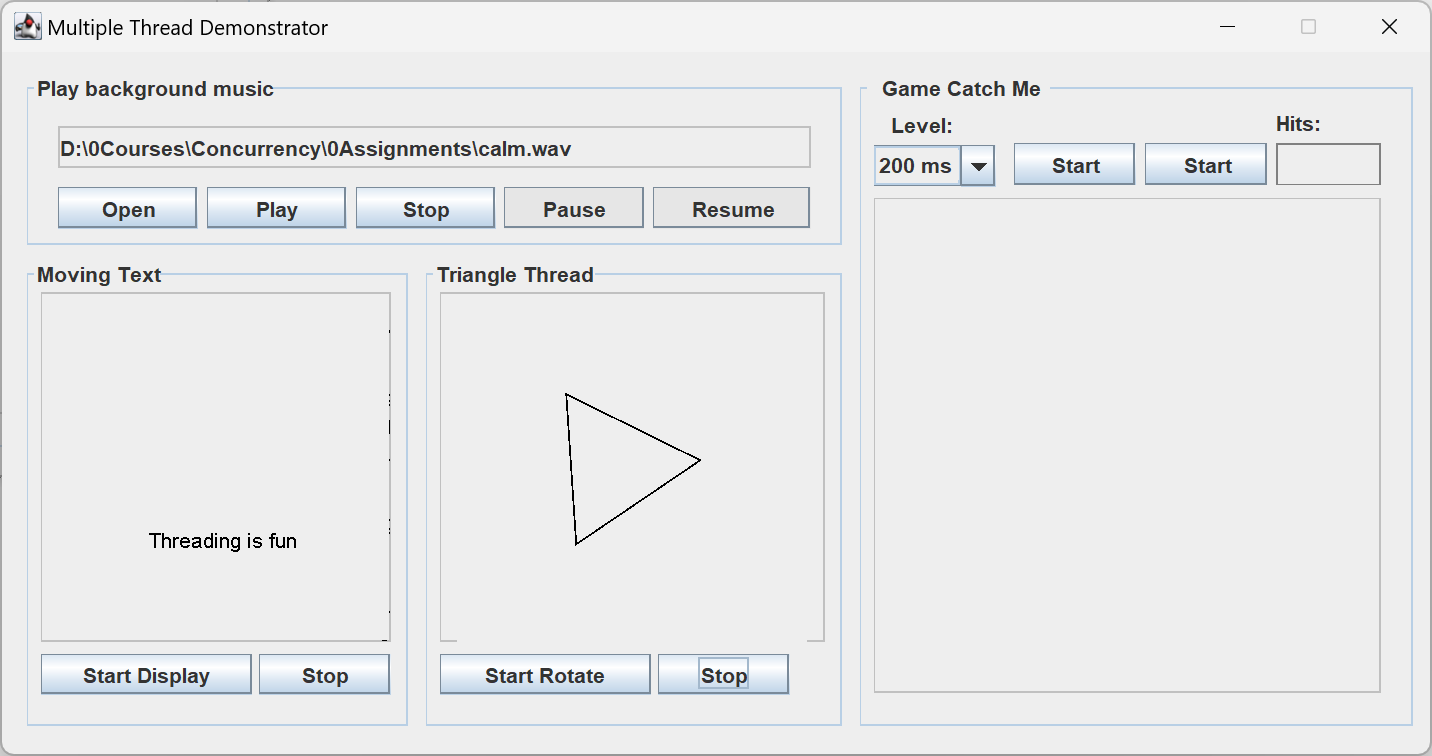
The main goal of this assignment is:

* To learn how to create and start a thread to perform a certain task, suspend and resume the task at any time.
* How to let several threads do their tasks simultaneously and operate without interference.
* To refresh your skills in programming applications with graphical user interface.

# Description

Your job in this first assignment is to write a simple GUI-based application that performs different tasks at the same time. You are given a number of tasks, of which you should pick-up at least 2 tasks, design a graphical user interface (GUI) and create an application to run the tasks concurrently. The tasks are to be executed each in its own thread. This way, you application will be running with at least three threads: the main thread (GUI) that is a part of the process (your program) and the two news threads that you will be creating for your tasks. You can, of course, select more than two tasks if you wish to practice with more threads, but to qualify for a **Pass** grade, you only need to use **two threads** among the following:

* Task 1: Displaying text randomly on a panel.
* Task 2: Turning a graphical object, rectangle, triangle or any other shape
* Task 3: Playing background music, pick one among wav, mp3 and other audio files
* Task 4: Playing a simple game: Catch Me, or a game of yours



**Programming language and development environment, IDEs**

You may use any of the languages C#, Java, C++ or Python. For C#, it is recommended to use **Visual Studio** to design your GUI and do the rest of coding.

The code for drawing the GUI will be provided for Java and C#, so you can concentrate on your multithreading programming. However, you may certainly design and program the GUI by yourself. Although, there are several development environments and tools available, we appreciate it if you use Visual Studio (C#) or **InilliJ** (Java) which make our grading job easier. The following versions are to be used for programming the assignments:

**Java** : Java 11 or higher, IntelliJ, Community version 2021 or newer.

**C#:** .NET 6 or higher (but .NET Framework 4.7 or higher will also work), VS 2022 Community version or higher.

# Task 1 – Play a background audio file

* 1. In this part, a separate is to be assigned the work of playing a music file that services as background music to your application. You can use a background or a foreground thread for playing the music. In Java, to make a thread a background thread, use the thread’s **setDaemon**(true) method. In C#, set the thread’s **IsBackground** property to true;

**Note**: The buttons “Pause” and “Resume” are optional to function.

Java tips: How to play music in java: see the link:   
<https://www.geeksforgeeks.org/play-audio-file-using-java/>

C# tips: you can use the **SoundPlayer** class or search for more tips on the internet.

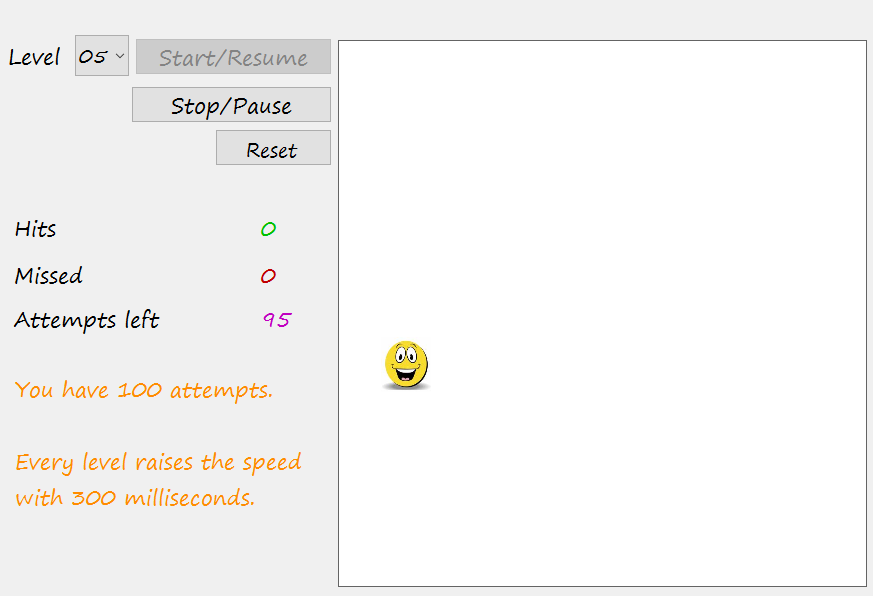
# Task 2 - Moving text randomly

* 1. As illustrated in the left-panel of the image above, use a separate thread to display a text appearing randomly on the drawing area. You may use a label or the graphical text drawing.

# Task 3 - Moving a graphical object

* 1. Create a thread to draw, move and/or rotate a shape. Draw any shape, a rectangle, a polygon, a curve or a triangle (as in the above example). The shape moves, rotates around its midpoint or bounce against the edges of the panel containing the drawing. Another example is to draw a clock (digital or analogue) showing the current time (system time or any local time). You may even load (embed) an image file and move the image around inside the panel. The main idea is that something happens in the panel dynamically.
  2. When the button **Start Rotate** is pressed, a thread is created and it starts drawing the shape on the panel. The panel is updated with an interval of a certain number of milliseconds (for example 200), and rotating a certain degree (say 10) around its midpoint, clockwise or counter-clockwise. If you choose to display a clock, it should update itself and display the time every second (Java Thread.sleep(1000), C# Thread.Sleep(1000)).
  3. There should be a **Start** and a **Stop** button for starting and stopping the thread working on each panel. The Start button should create and start the thread and the Stop button should end the thread execution. Make the buttons enabled only one at a time, depending on status. The thread can’t start something that is already running or vice versa.

# Task 4: Playing Game - Catch-Me Game

* 1. In this part, you can let one or more threads to play a simple game. If you have any game idea that is somewhat interactive, program your game instead. Otherwise, program the Catch-me game, a simple game described below.
  2. The idea behind the Catch-Me game is to let an image or an icon appear randomly on the surface of the panel, at random intervals (shorter for higher levels). The user tries to hit the image using the mouse. The program displays the number of hits on the figure, number of times missing the figure and the number of attempts left. The figure below shows a runtime example from another exercise (not using threads).

**Hint**: Write one listener for the image-click and one for the panel-click. The number of image-clicks = hits, panel-clicks = miss!

* 1. Disable the start button during a play session. At some intervals the play thread then randomly places an image on the playing panel. When the user clicks the GO button, make sure to wait one or two seconds before first image is shown so the user gets prepared.

# Specifications and Requirements for a Pass grade (G)

* 1. It is not allowed to use a **Timer** control or any similar components. All such operations should be accomplished by the threads.
  2. Do not use the deprecated methods in the Thread class for interrupting, suspending and aborting or stopping a thread. Use a variable as a flag for each thread (isRunning) and set it to true while it is running, set to false to stop running.
  3. Select two (or more) tasks and combine them as parts of your application.
  4. Your code must be well organized into classes and well object-oriented based on Encapsulation (private instance variables, short methods), inheritance and polymorphism wherever and whenever applicable. Add comments to your code to document your thoughts.
  5. Comment your code, briefly to describe what each class do and what each method of the class do as wells in places inside the code wherever needed. Setter and getter and other such methods are known methods and need not be commented.
  6. Make sure that no user action can cause the program to crash. Test your application carefully before submitting.
  7. Save each class in its own file without regard to the size of the class.
  8. Use proper variable and methods names, a, fnc, and other such short names make your code difficult to read and maintain.

**Hint**: Do not use nested classes; they make your coding more complicated risky for bugs.

# Grading and submission

Assignmenets are to be presented personally and individually during the lab-hours, but the solution must be uploaded to Canvas before or after the presentation

.

Compress all the files, folders and subfolders into a **zip**, **rar, 7z** file, and then upload it via the Assignment page on Canvas. Make sure that you submit the correct version of your project and that you have compiled and tested your project before handing in. Be careful not to use any hard-coded file paths (for example path to an image file on your C-drive) in your source code. It will not work on other computers. Projects that do not compile and run correctly, or is done with poor code quality, will be returned for completion and resubmission.

Good Luck!

Farid Naisan,

Course Responsible and Instructor