

# Threads and Non-blocking Reusable Controls

## General part

Explain about Thread Programming including:

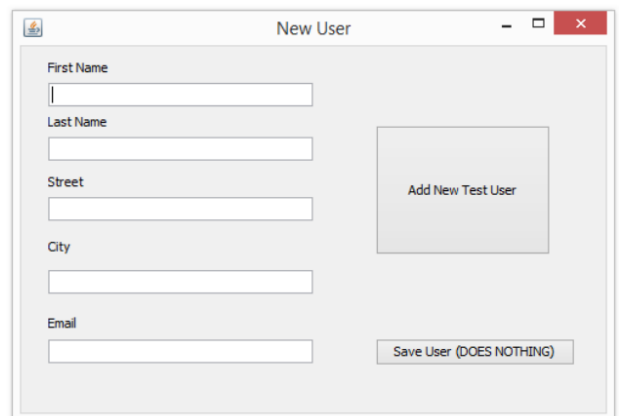
- When and why we will use Threads in our programs?
- Explain about the Race Condition Problem and ways to solve it in Java
- Explain how Threads can help us in making responsive User Interfaces
- Explain how we can write reusable non-blocking Java Controls using Threads and the observer Pattern

## Practical part

While developing new GUI-based systems, it is often cumbersome, that while testing, we have to type in test data, over and over again.

The code supplied with this example contains a Form with a big "ugly" test button, that when pressed, will fill out the form with auto generated test data. Since data is retrieved over the network, it can take time (made very clear by a prolonged random delay in the control).

Run the code and identify and explain the problem related to pressing the button. Fix the problem by Implementing the following changes:



You should solve the problem in two steps.

- Implement the Observer Pattern<sup>1</sup>, so when the big button is pressed you signal that the GUI would like to be notified when data is ready.
- Move the time Consuming task into a separate Thread

**Rewrite the code using Threads and the Observer Pattern as explained below**

1. Change the declaration of RandomUserControl to Extend `java.util.Observable`<sup>2</sup>
2. Change the return type of the method `fetchRandomUser()` into void and replace the return statement with (explain what happens):  
`setChanged();`  
`notifyObservers(user);`
3. Change the RandomUserForm to implement the `java.util.Observer` interface<sup>3</sup>. When the RandomUserControl call `notifyObservers(..)`, the `update(..)` method is called on all Observers and the arg-argument will include a RandomUser.

---

<sup>1</sup> [https://en.wikipedia.org/wiki/Observer\\_pattern](https://en.wikipedia.org/wiki/Observer_pattern)

<sup>2</sup> A Java class that gives you the functionality found in Subject, in Wikipedia's Observer Article.

<sup>3</sup> A Java interface Corresponding to Observer, in Wikipedia's Observer Article

4. Move all the `*.setText(..)` methods from `btn1Clicked(..)` into the `update(..)` method (Create the variable `random`, as a `RandomUser` and initialize it by casting `arg` into a `RandomUser`).
5. Register the Form as an `Observer` on the `randomUserController` and call `fetchRandomUser()` in the `btn1Clicked(..)` method.
6. Verify that everything works but, with the same problem as when we started (what have we gained by using the `Observer Pattern` so far?)
7. Move the contents of `fetchRandomUser(..)` into a separate `Thread`, that has visibility to the `RandomUserController`, so its `notifyObservers(..)` method when it has received the data. This method notifies that all registered listeners.
8. Finally, make sure that the code running on the GUI thread in the Forms `update(..)` method is thread-safe (see guidelines for threads and Swing)

**Note: this part is NOT a part of the exercise, it's meant as FYI and this section will not be included with the real question.**

*Since this exercise is given in week one, it provides some help you won't find in the real question.*

- *For the real question you are supposed to be able to identify the blocking/time-consuming parts by yourself, and implement the `Observer Pattern` without the hints given in this exercise.*
- *You could also be requested to write the code that actually fetches the data via the network (don't worry, this should be a piece of cake, by the end of this semester ;-)*