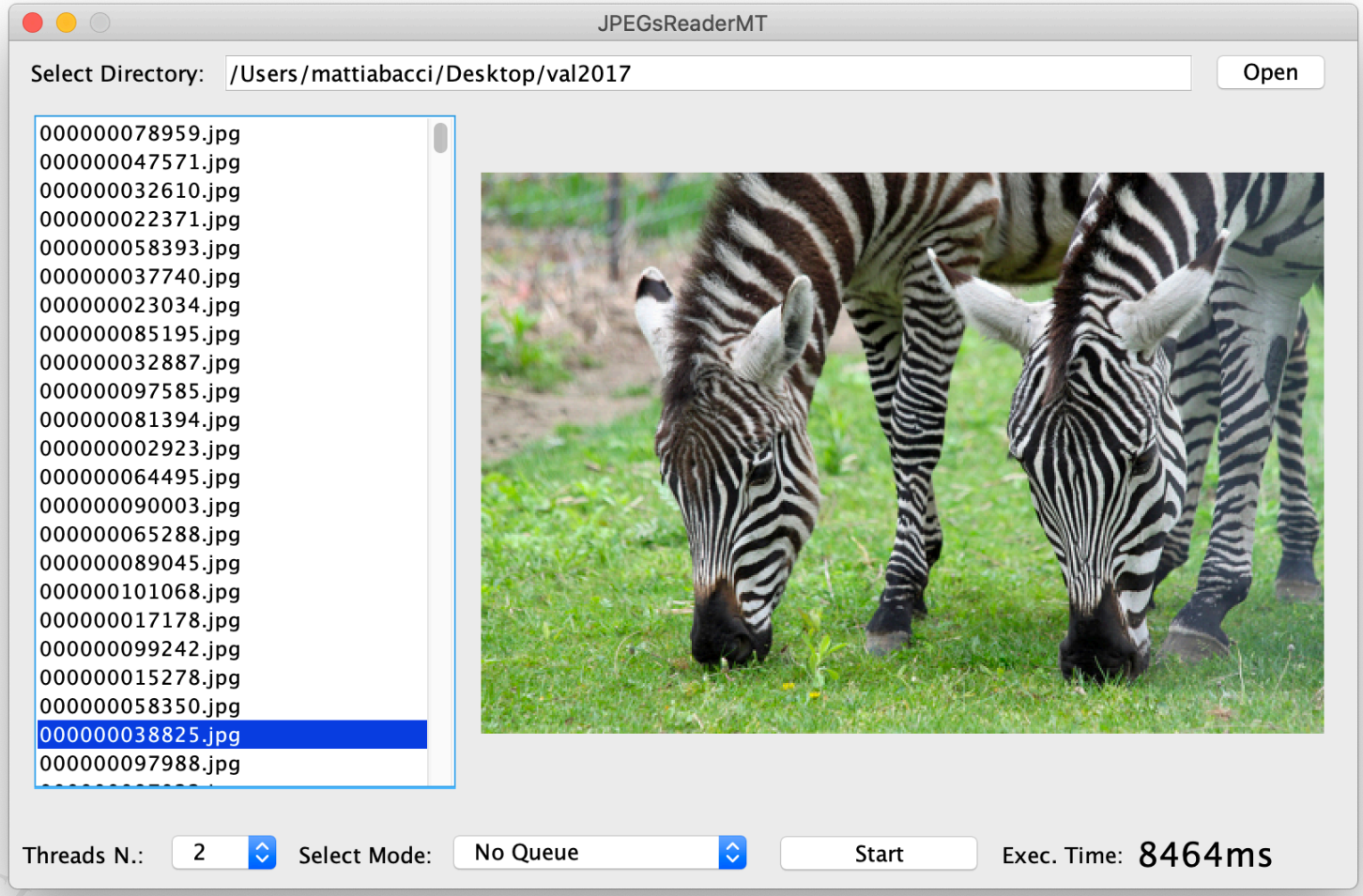
Non Blocking JPEGs Reader

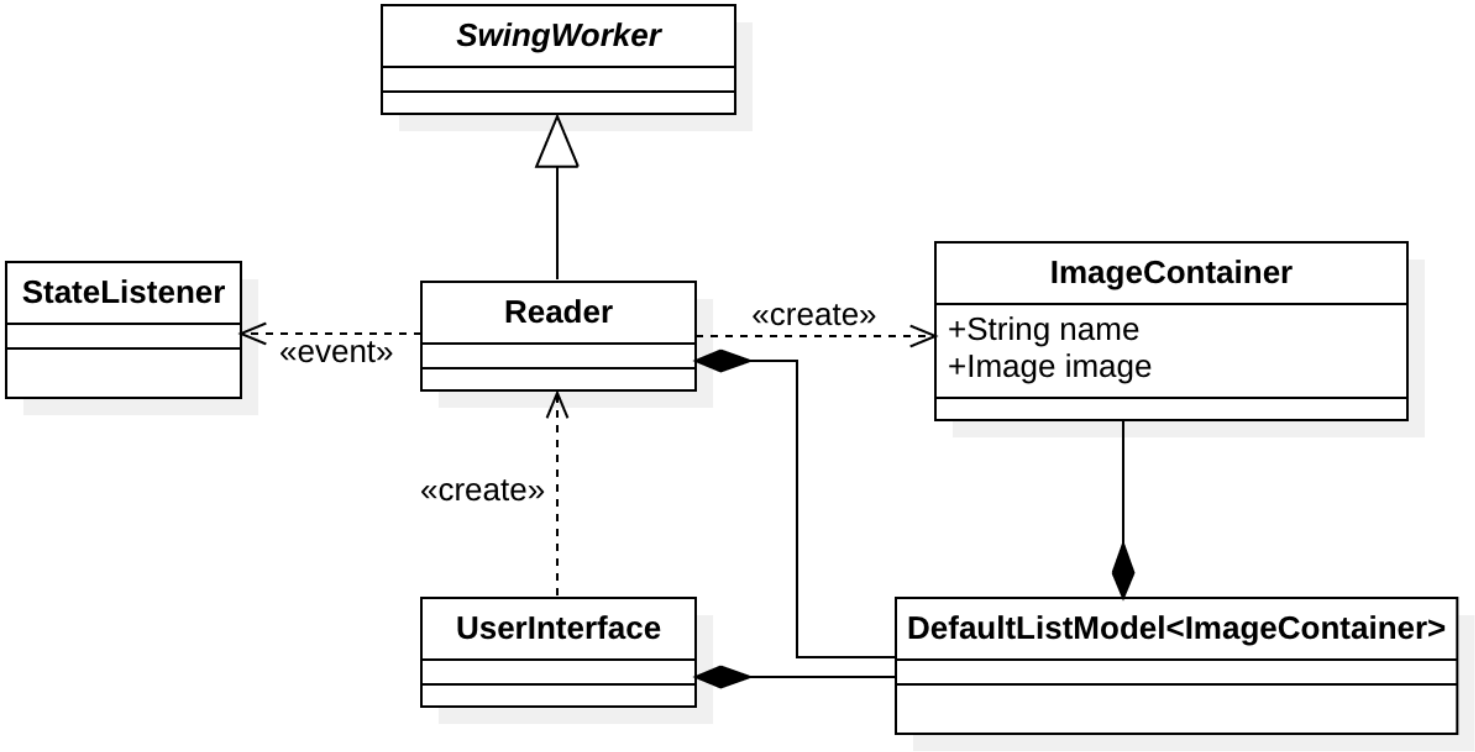
Mattia Bacci 7060637
Elaborato Mid-Term

13/01/2021

UI


JPEGsReaderMT
Select Directory:
/Users/mattiabacci/Desktop/val2017
Open
000000078959.jpg
000000047571.jpg
000000032610.jpg
000000022371.jpg
000000058393.jpg
000000037740.jpg
000000023034.jpg
000000085195.jpg
000000032887.jpg
000000097585.jpg
000000081394.jpg
000000002923.jpg
000000064495.jpg
000000090003.jpg
000000065288.jpg
000000089045.jpg
000000101068.jpg
000000017178.jpg
000000099242.jpg
000000015278.jpg
000000058350.jpg
000000038825.jpg
000000097988.jpg
Threads N.:
2
Select Mode:
No Queue
Start
Exec. Time:
8464ms

UML


SwingWorker
StateListener
«event»
Reader
«create»
ImageContainer
+String name
+Image image
«create»
UserInterface
DefaultListModel<ImageContainer>

UI and MultiThread – The Problem

```
startButton.addActionListener(e -> {
        System.out.println(«Executing on EDT»);
        Thread thread = new  Thread();
        thread.run();
});
```

Executing on EDT, if it is a long running task, UI will became unresponsive and eventually will freeze.

Creating a new Thread for the long running task will resolve the problem.

Swing Worker<T,V>

When writing a multi-threaded application using Swing, there are two constraints to keep in mind:

- *Time-consuming tasks should not be run on the Event Dispatch Thread. Otherwise the application becomes unresponsive.*
- *Swing components should be accessed on the Event Dispatch Thread only.*

All Implemented Interfaces: Runnable, Future<T>, RunnableFuture<T>

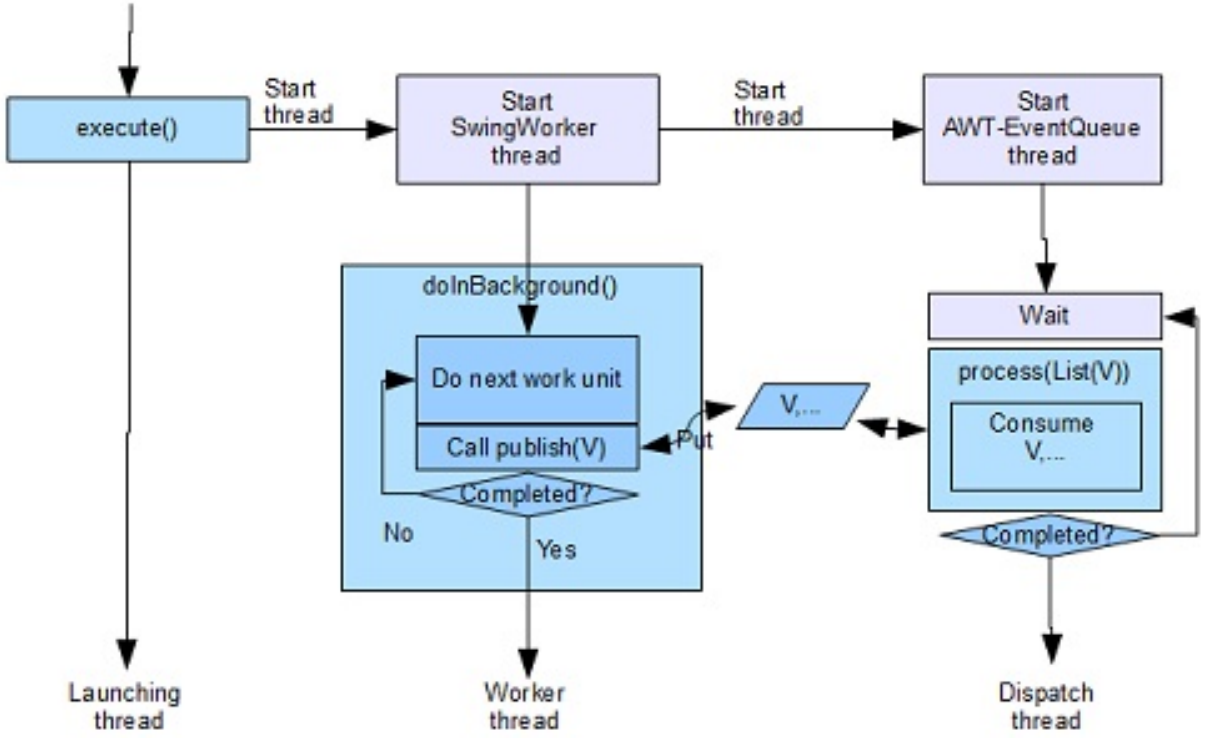
protected abstract T **doInBackground**()

protected final void **publish**(V... chunks)

protected void **process**(List<V> chunks)

Swing Workers uses Thread Pools for its execution with a ***maximumPoolSize*** set to 10.

Swing Worker<T,V>



protected abstract T **doInBackground**()

protected final void **publish**(V... chunks)

protected void **process**(List<V> chunks)

Stopping the Timer

When a Reader doInBackground() method terminate its execution the done() method will be automatically called. This method will launch and event captured by a PropertyChangeListerner object that is the ame for all the Threads.

```
@Override
public void propertyChange(PropertyChangeEvent evt) {
    String value = evt.getNewValue().toString();
    if (value.equals("DONE")) {
        if (++i == THREADS_NUM) {
            long end = System.currentTimeMillis();
            label.setText((end - start) + "ms");
            startButton.setEnabled(true);
        }
    }
}
```

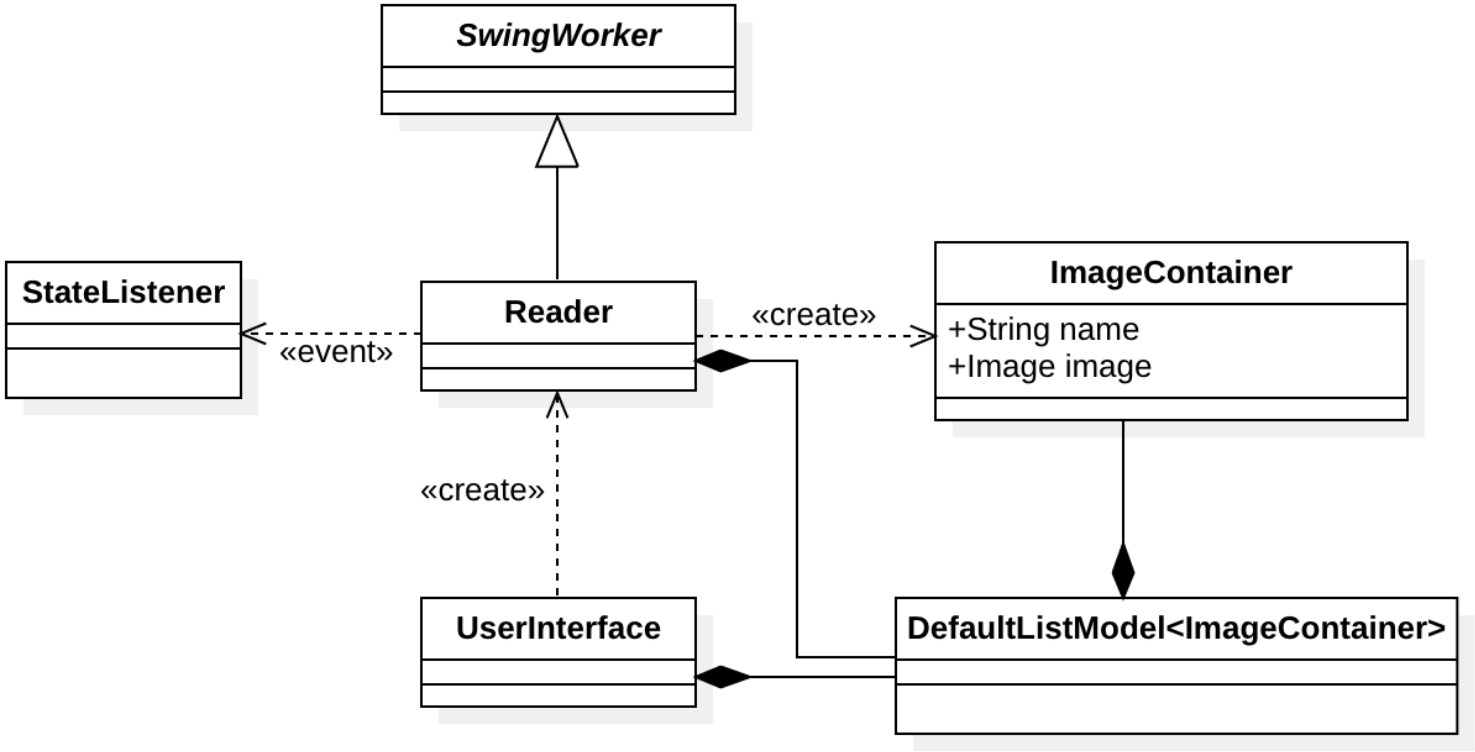
Stopping the Timer

When a Reader doInBackground() method terminate its execution the done() method will be automatically called. This method will launch and event captured by a PropertyChangeListerner object that is the ame for all the Threads.

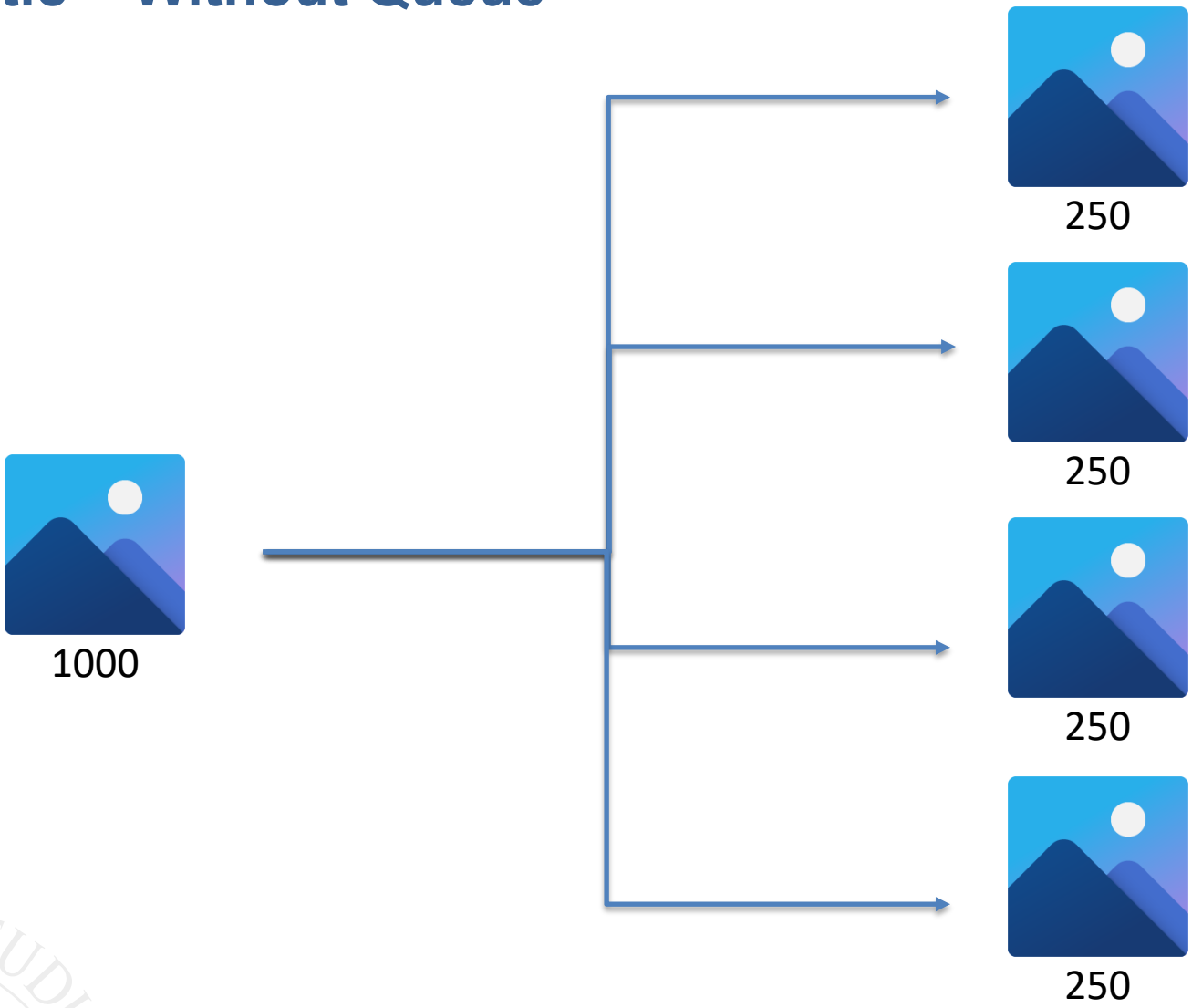
```
@Override
public void propertyChange(PropertyChangeEvent evt) {
    String value = evt.getNewValue().toString();
    if (value.equals("DONE")) {
        if (++i == THREADS_NUM) {
            long end = System.currentTimeMillis();
            label.setText((end - start) + "ms");
            startButton.setEnabled(true);
        }
    }
}
```

The increment doesn't need to be Synchronized or Atomized, The code is Executed in the EDT!

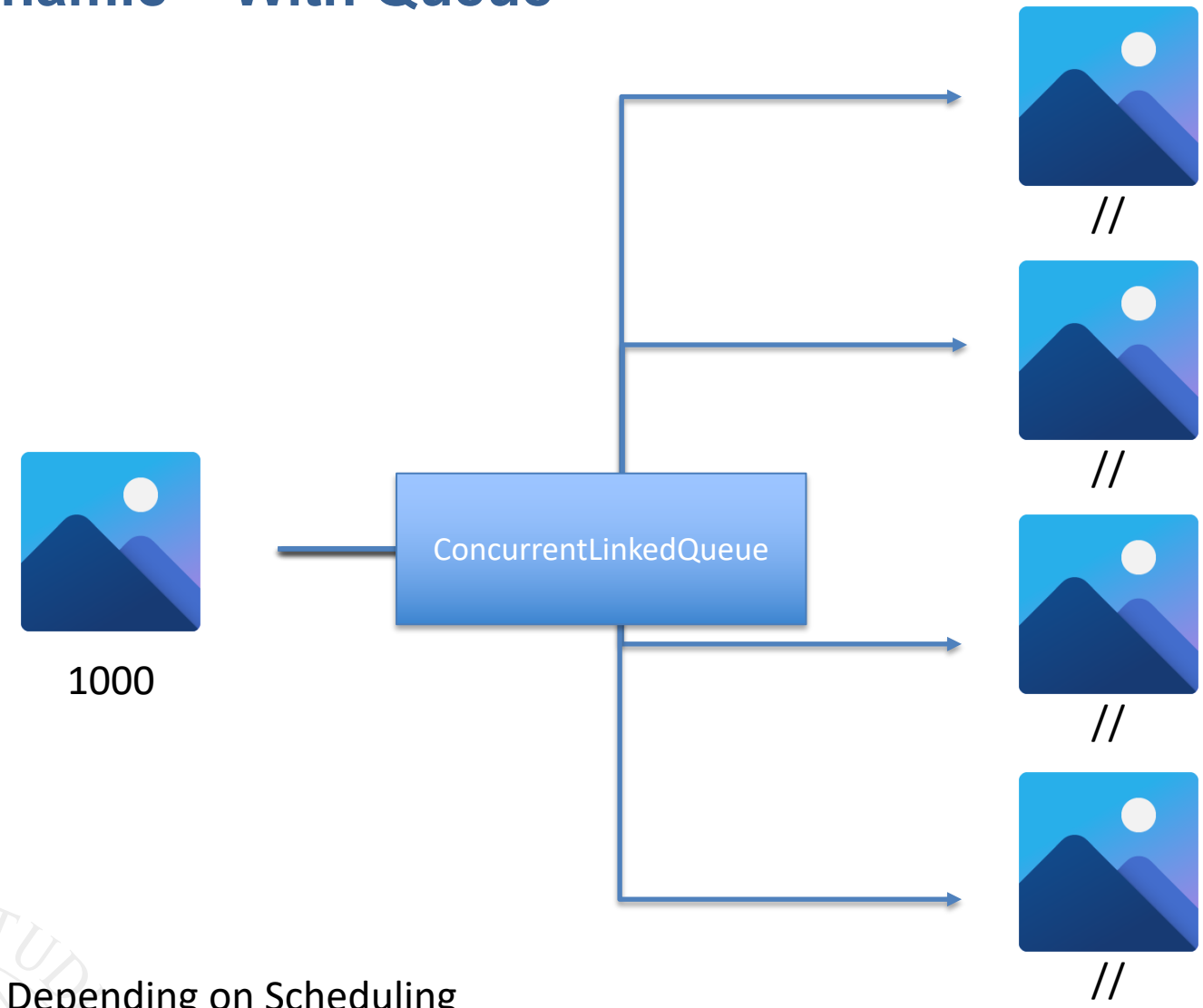
UML


SwingWorker
StateListener
«event»
Reader
«create»
ImageContainer
+String name
+Image image
«create»
UserInterface
DefaultListModel<ImageContainer>

Static – Without Queue



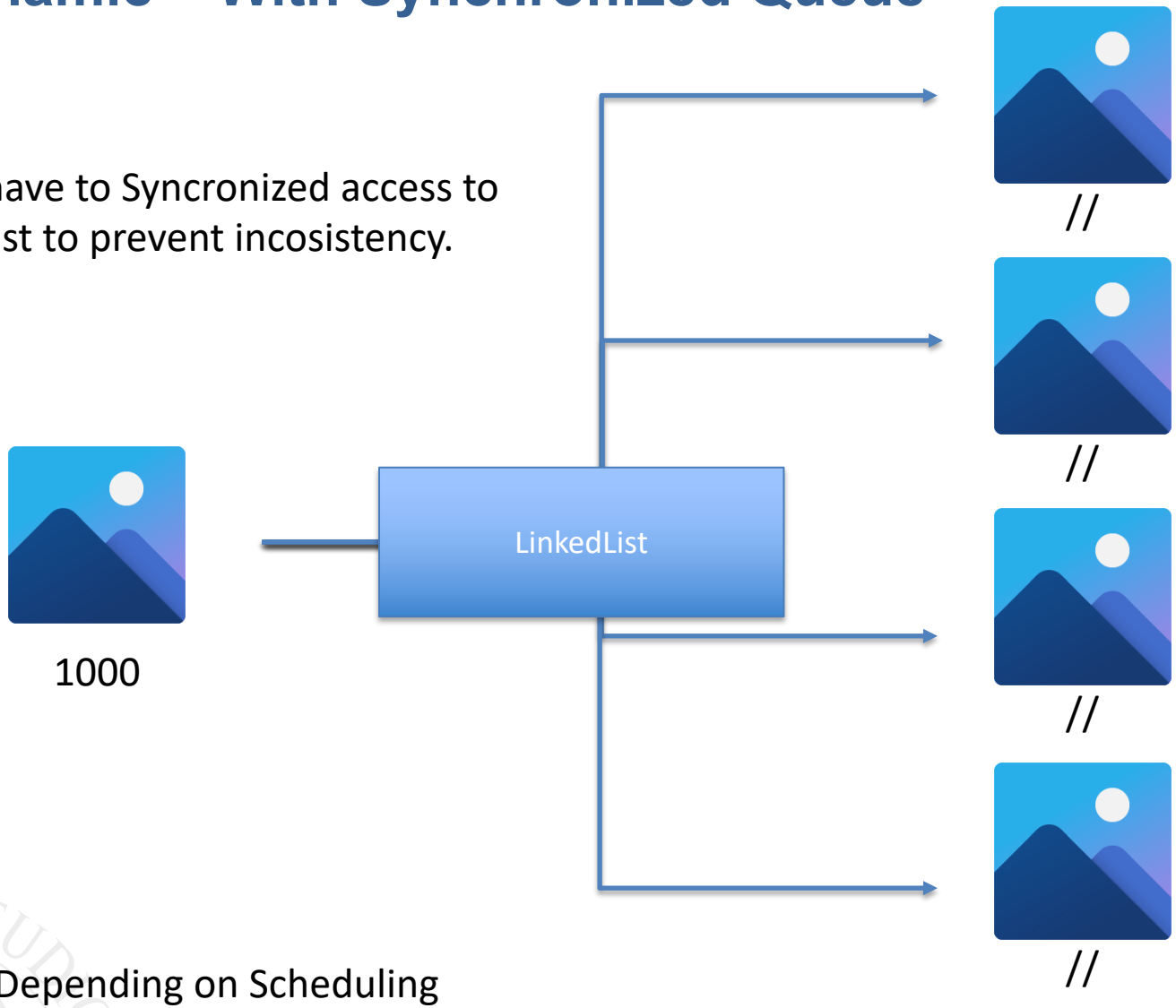
Dynamic – With Queue



// - Depending on Scheduling

Dynamic – With Synchronized Queue

We have to Syncronized access to the list to prevent incosistency.



// - Depending on Scheduling

Results

N. Immagini	Metodo Esecuzione	Tempo Sequenziale	Tempo MultiThread	Efficienza
900	Senza Coda	16100 ms	4500 ms	90%
900	Con coda	16100 ms	4700 ms	85%
900	Con coda sincronizzata	16200 ms	4900 ms	83%
1400	Senza Coda	26100 ms	7800 ms	84%
1400	Con coda	25900 ms	8200 ms	79%
1400	Con coda sincronizzata	26700 ms	8000 ms	83%

Figure 1. Intel Core i5 (I5-4570S) 4-Core

N. Immagini	Metodo Esecuzione	Tempo Sequenziale	Tempo MultiThread	Efficienza
900	Senza Coda	17900 ms	8300	100%
900	Con coda	18700 ms	8600	100%
900	Con coda sincronizzata	18200 ms	9100	100%
1400	Senza Coda	28500 ms	14600	100%
1400	Con coda	28900 ms	14900	100%
1400	Con coda sincronizzata	29100 ms	15100	100%

Figure 2. Intel Core i5 (I5-7267U) 2-Core

- 7° Gen intel core i5 2-Core has always better efficency.
- Syncronized Queue has the same execution time of the other modalities.