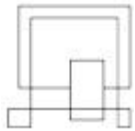
VIA University College



Software Development with UML and Java 2

Autumn 2021

Learning Objectives

- By the end of class today, you will be able to:
  - ✓explain the concept of – Java threads and multithreading
  - ✓ explain the different thread states
  - ✓ demonstrate their usage

What is a thread?

- Thread: an independent path of execution through program code
- Multithreading: when multiple threads execute byte-code instruction sequences in the same program

http://cbitss.in/java-multithreading-concept/multithreading/

User thread vs Daemon

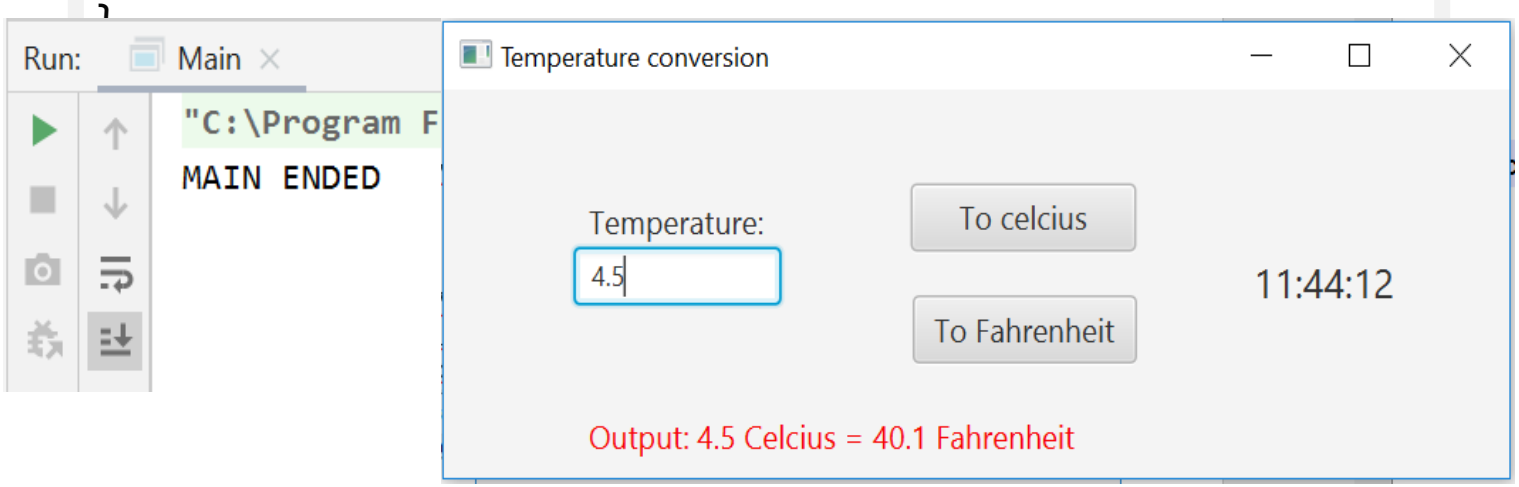
- User thread (default), initiated by an application (main thread is one such an example)
- Daemon thread that works in the background in support of user thread activity (more details coming later)

How do I create a Thread?

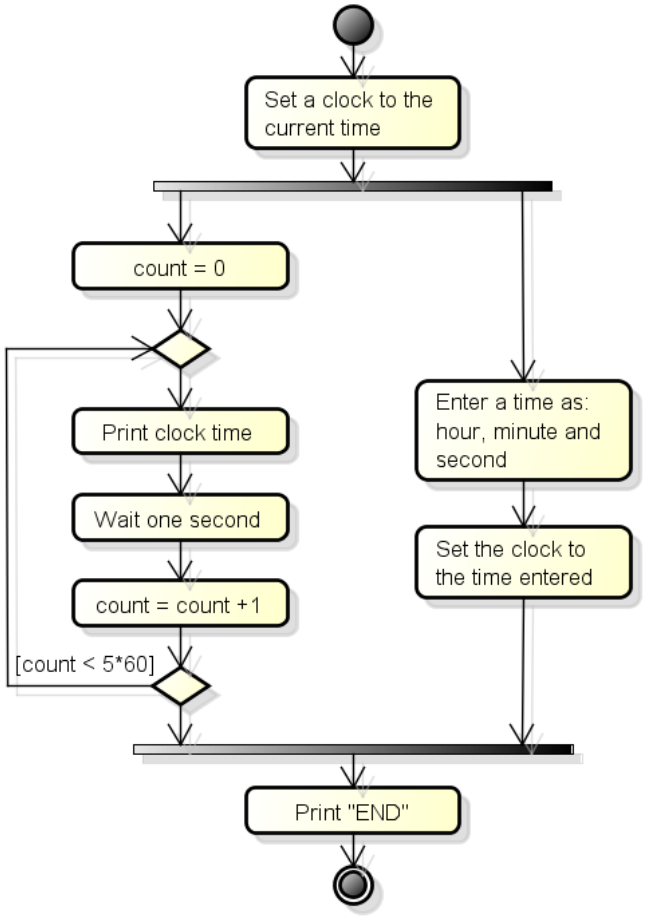
- Extending class Thread
  - extend the java.lang.Thread class and override its run() method
- Implementing interface Runnable
  - use a class that implements java.lang.Runnable

A simple GUI

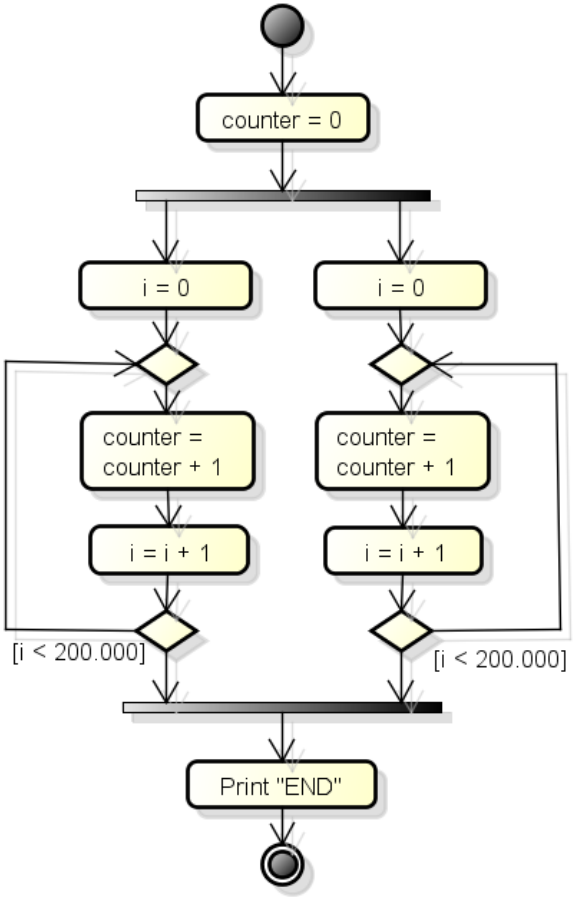
```
public class Main
{
    public static void main(String args[])
    {
        Temperature model = new Temperature();
        Clock clock = new Clock();
        TemperatureView view = new TemperatureView();
        view.startView(model, clock);
        System.out.println("MAIN ENDED");
    }
}
```



Activities in parallel



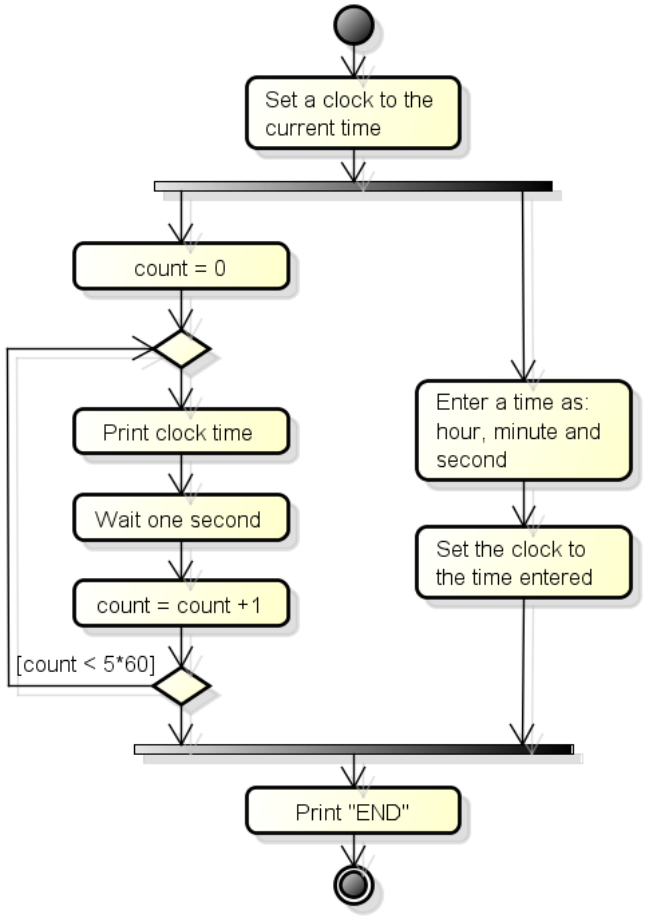
Similar activities in parallel

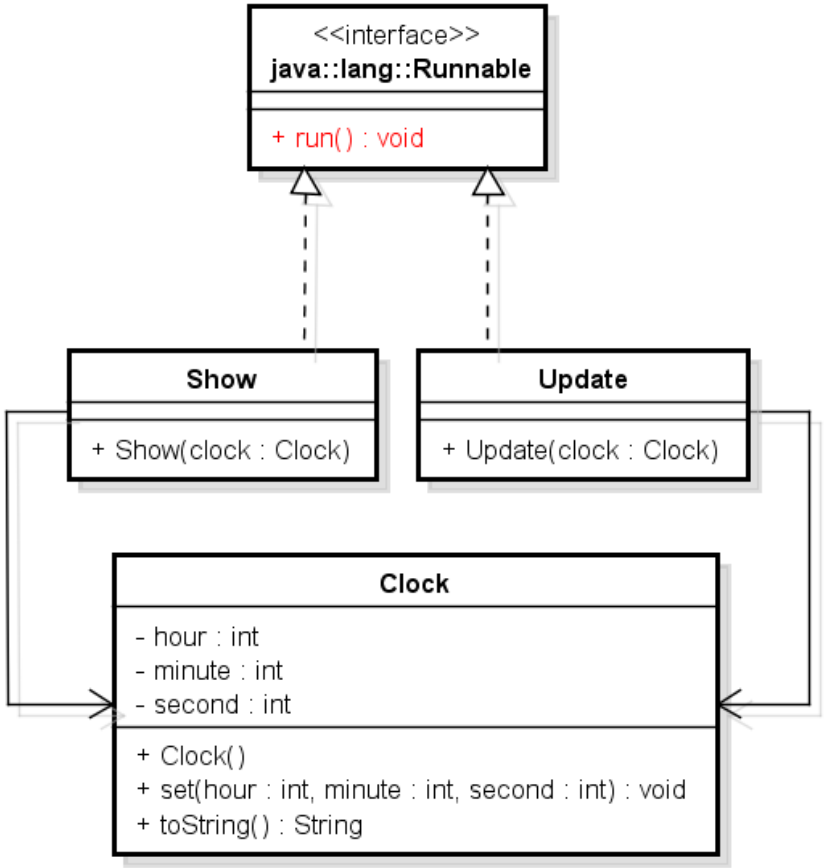


Creating a thread

```
public class MyRunnable implements Runnable
{
    //... Instance variables
    MyRunnable(/* parameters */)
    {
        //...
    }
    @Override
    public void run()
    {
        // operations to perform
    }
}
// Method starting the thread:
    Runnable myRunnable = new MyRunnable();
    Thread myThread = new Thread(myRunnable);
    myThread.start();
```

Clock example





Clock example (Show)

```
public class Show implements Runnable {
    private Clock clock;

    public Show(Clock clock)
    {
        this.clock = clock;
    }

    public void run() {
        for (int i = 0; i < 5 * 60; i++) {
            System.out.println(clock);
            // and some code to pause for one second
        }
    }
}
```

Clock example (Update)

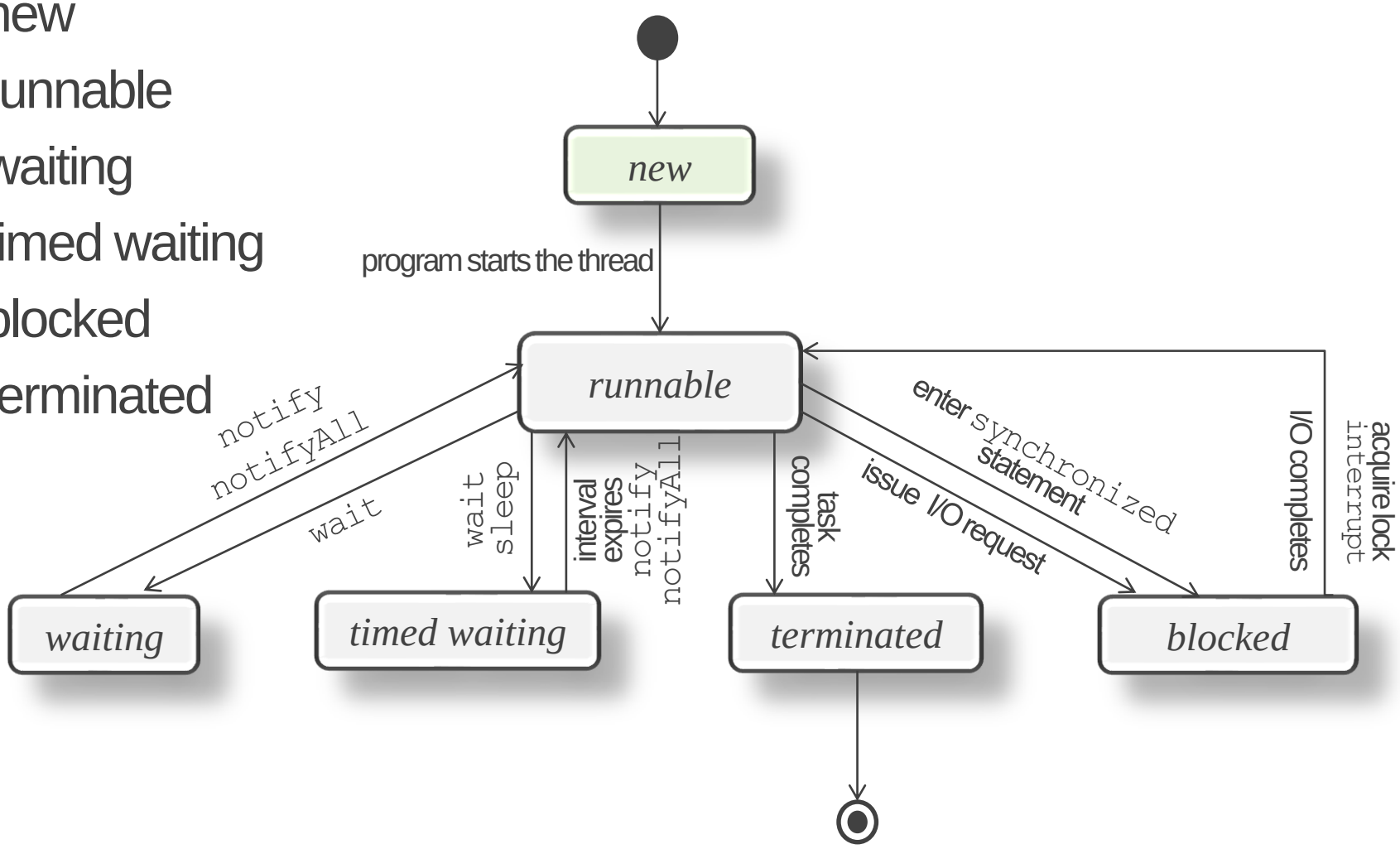
```
public class Update implements Runnable {
    private Clock clock;

    public Update(Clock clock)
    {
      this.clock = clock;
    }

    @Override
    public void run()
    {
        Scanner keyboard = new Scanner(System.in);
        int hour = keyboard.nextInt();
        int minute = keyboard.nextInt();
        int second = keyboard.nextInt();
        clock.set(hour, minute, second);
    }
}
```

Clock example (Main method)

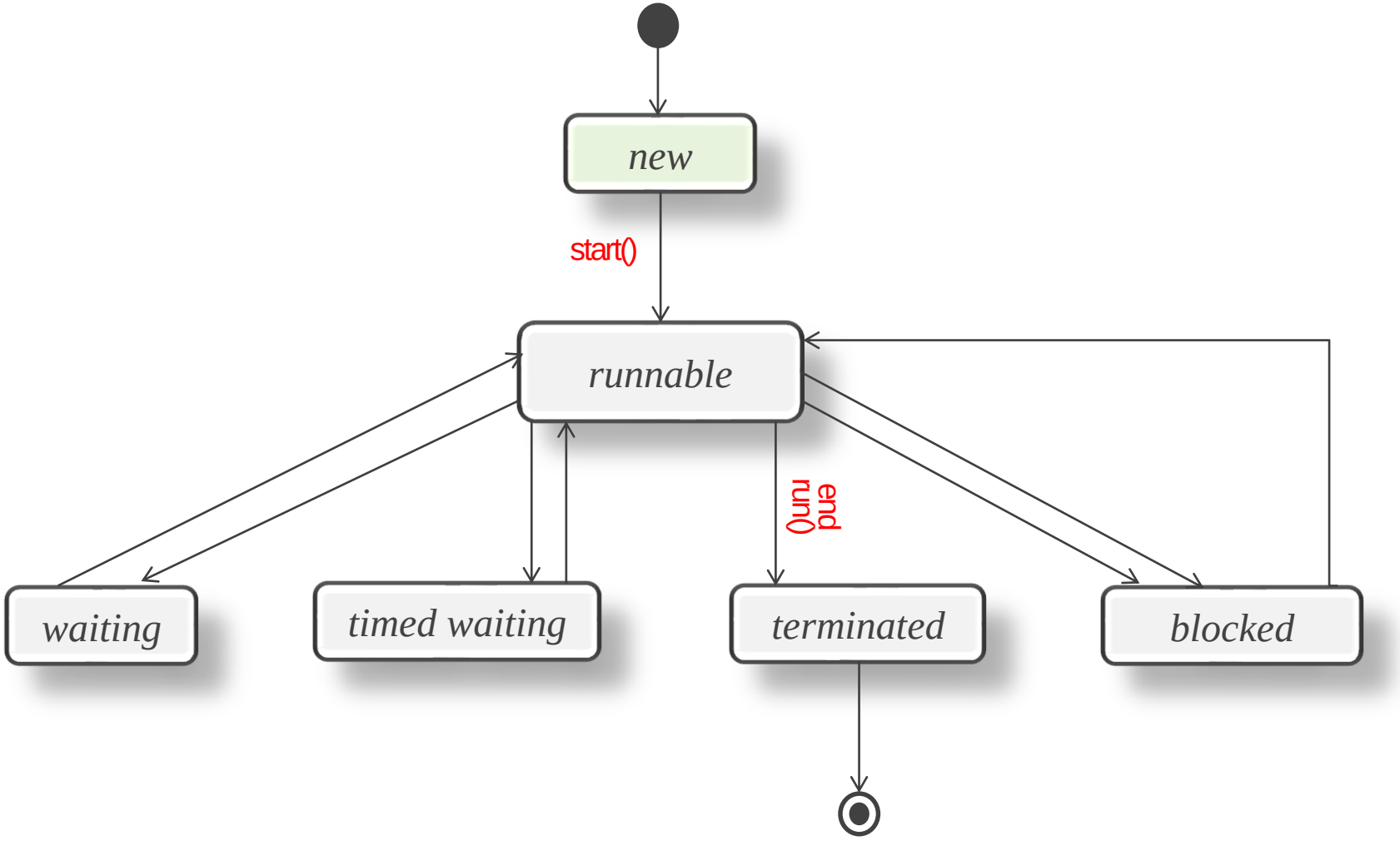
```
public class TestClock {
    public static void main(String[] args)
    {
        Clock clock = new Clock();
        Show showClock = new Show(clock);
        Update updateClock = new Update(clock);

        Thread showClockThread = new Thread(showClock);
        Thread updateClockThread = new Thread(updateClock);

        showClockThread.start();
        updateClockThread.start();

        System.out.println("MAIN ENDED");
    }
}
```

Thread States

- new
- runnable
- waiting
- timed waiting
- blocked
- terminated



Thread States – start & end run

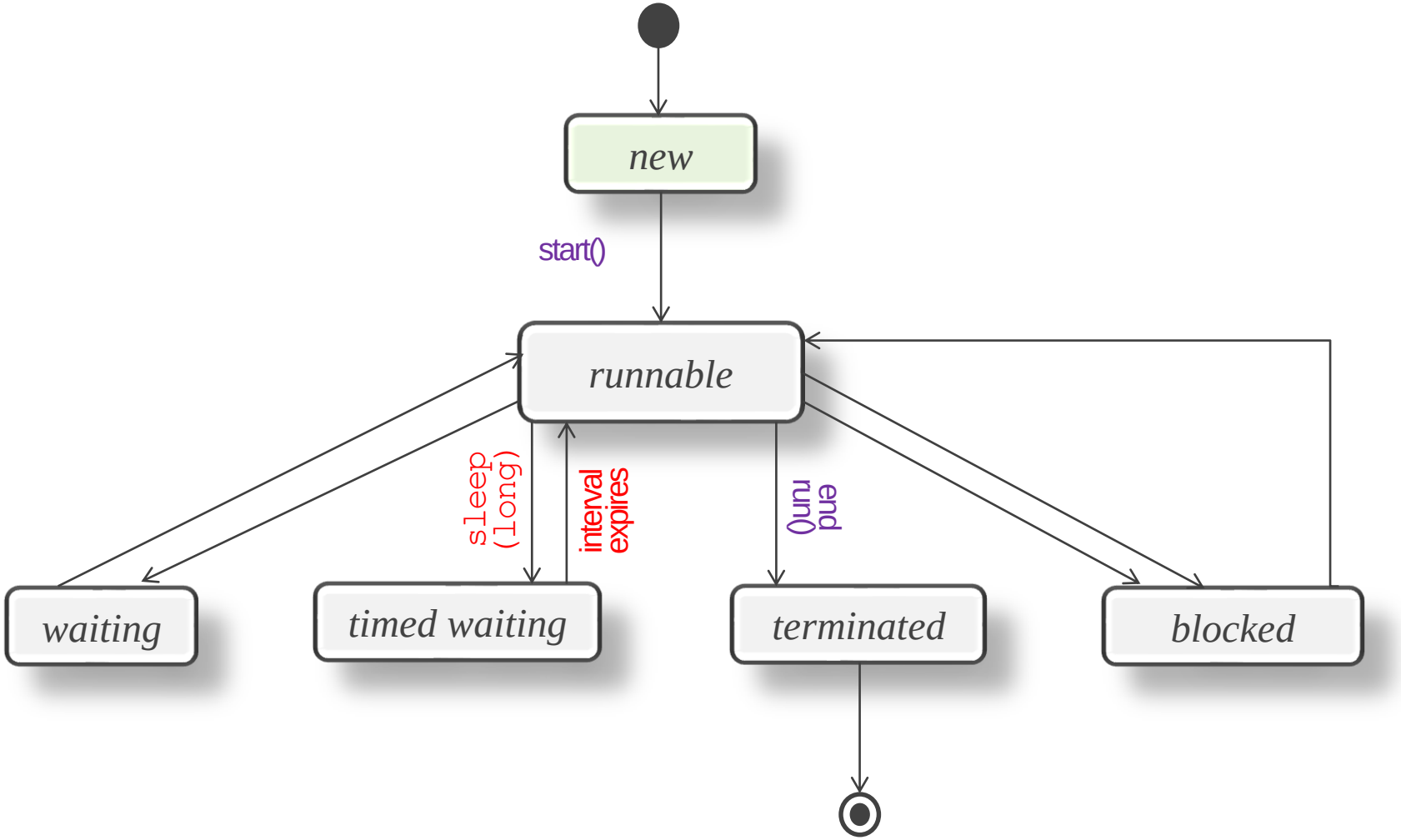


How to pause one second

```
try
{
   Thread.sleep(1000); // sleep for 1000 milliseconds
}
catch (InterruptedException e)
{
   // do nothing
}
```

Going from Runnable State to Timed Waiting State

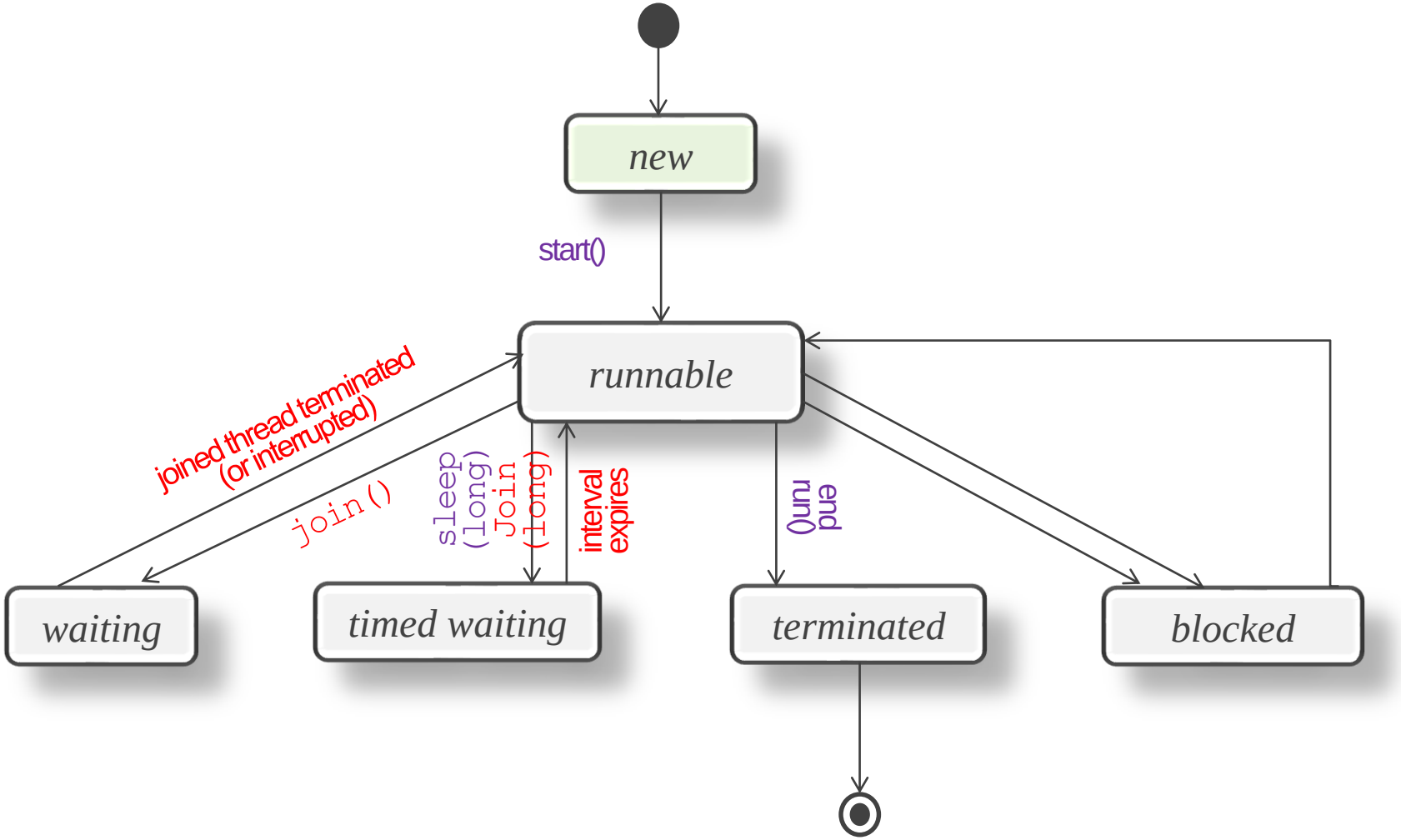
Thread States - sleep



Terminated state (synchronization)

- Pause until a thread is terminated (Wait state)
  - join(), join(long)
- Terminate when another thread terminates
  - setDeamon(true)

Thread States - join



Main thread in Wait state (join)

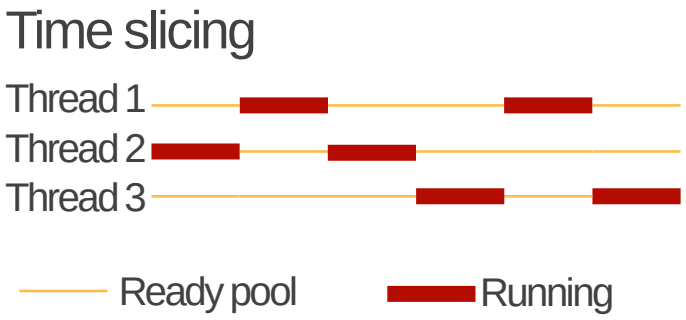
```
public class TestClock //the two clock threads controls when program ends
{
    public static void main(String[] args)
    {
        Clock clock = new Clock();
        Thread showClockThread = new Thread(new Show(clock));
        Thread updateClockThread = new Thread(new Update(clock));

        showClockThread.start();
        updateClockThread.start();

        try
        {
            showClockThread.join();
            updateClockThread.join();
        }
        catch (InterruptedException e) { /* nothing reinterrupt*/ }
    }
}
```

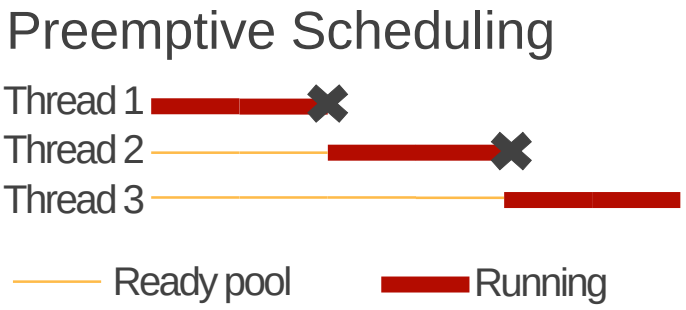
Deamon threads terminated when the Main thread terminates

```
public class TestClock // the Main thread controls when program ends
{
    public static void main(String[] args)
    {
        Clock clock = new Clock();
        Thread showClockThread = new Thread(new Show(clock));
        Thread updateClockThread = new Thread(new Update(clock));

        showClockThread.setDaemon(true);
        updateClockThread.setDaemon(true);

        showClockThread.start();
        updateClockThread.start();

        try
        {
            sleep(5000);
        }
        catch (InterruptedException e) { /* nothing */ }
    }
}
```

Runnable state

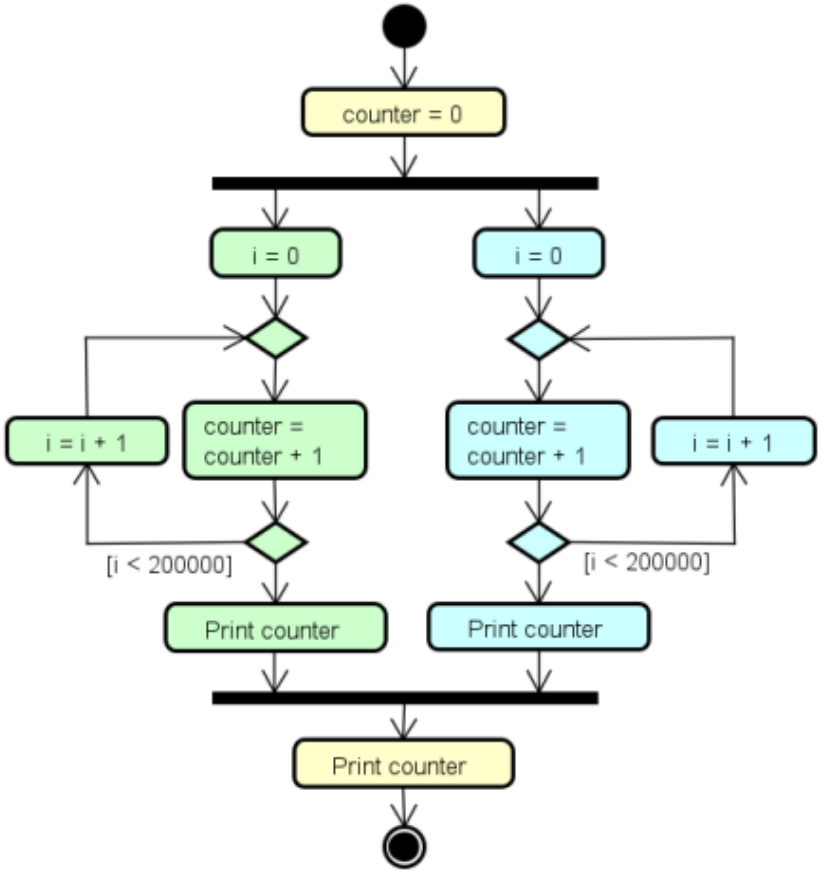
1. Running (scheduled CPU time)
   - Depends on thread priority, OS scheduling algorithm
2. Ready pool (ready for CPU time but not running)

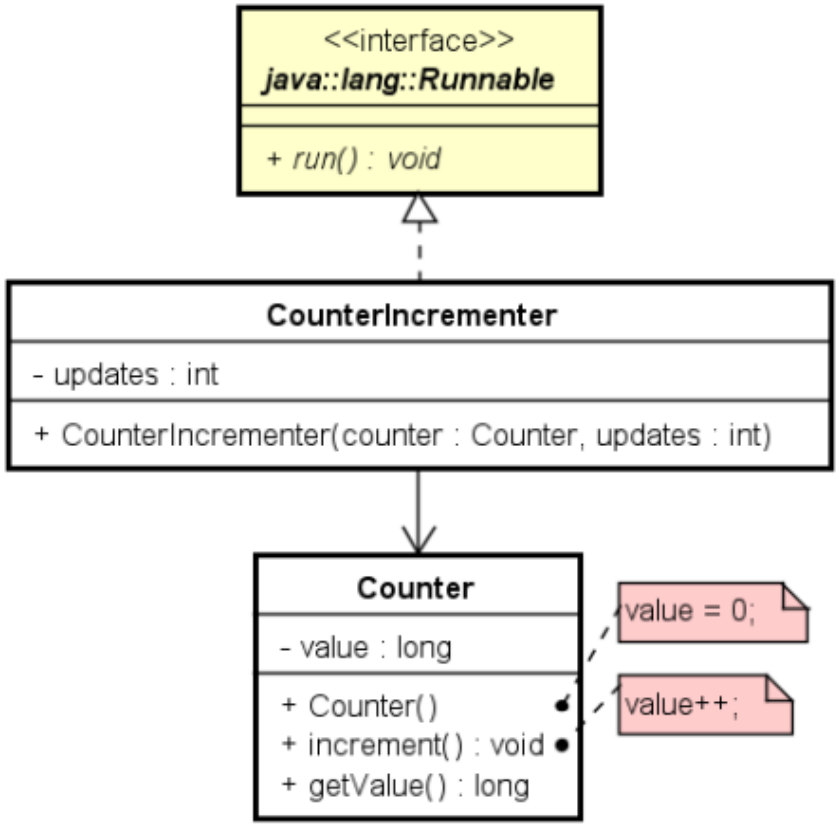




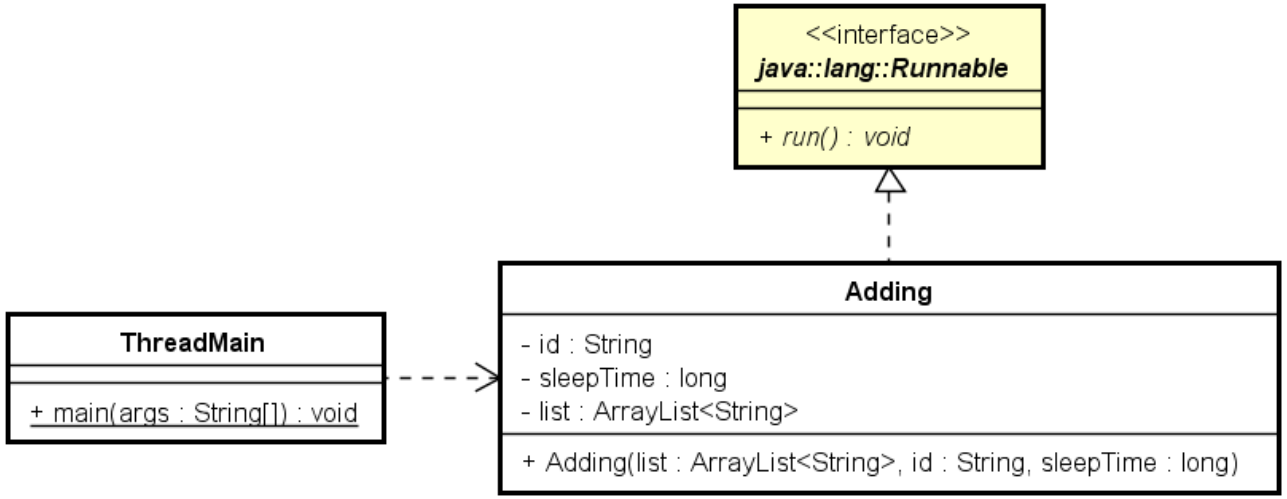
- Give away CPU time voluntarily
  - yield()

Exercises


counter = 0
i = 0
i = 0
i = i + 1
counter = counter + 1
counter = counter + 1
i = i + 1
[i < 200000]
[i < 200000]
Print counter
Print counter
Print counter

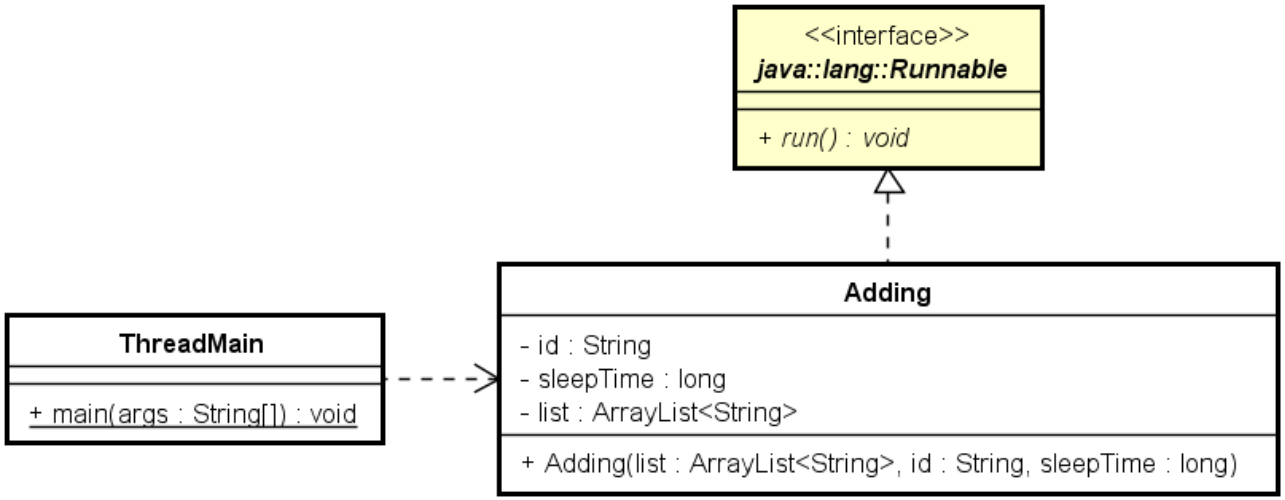

<<interface>>
java::lang::Runnable
+ run() : void
CounterIncrementer
- updates : int
+ CounterIncrementer(counter : Counter, updates : int)
Counter
- value : long
+ Counter()
+ increment() : void
+ getValue() : long
value = 0;
value++;

Exercises

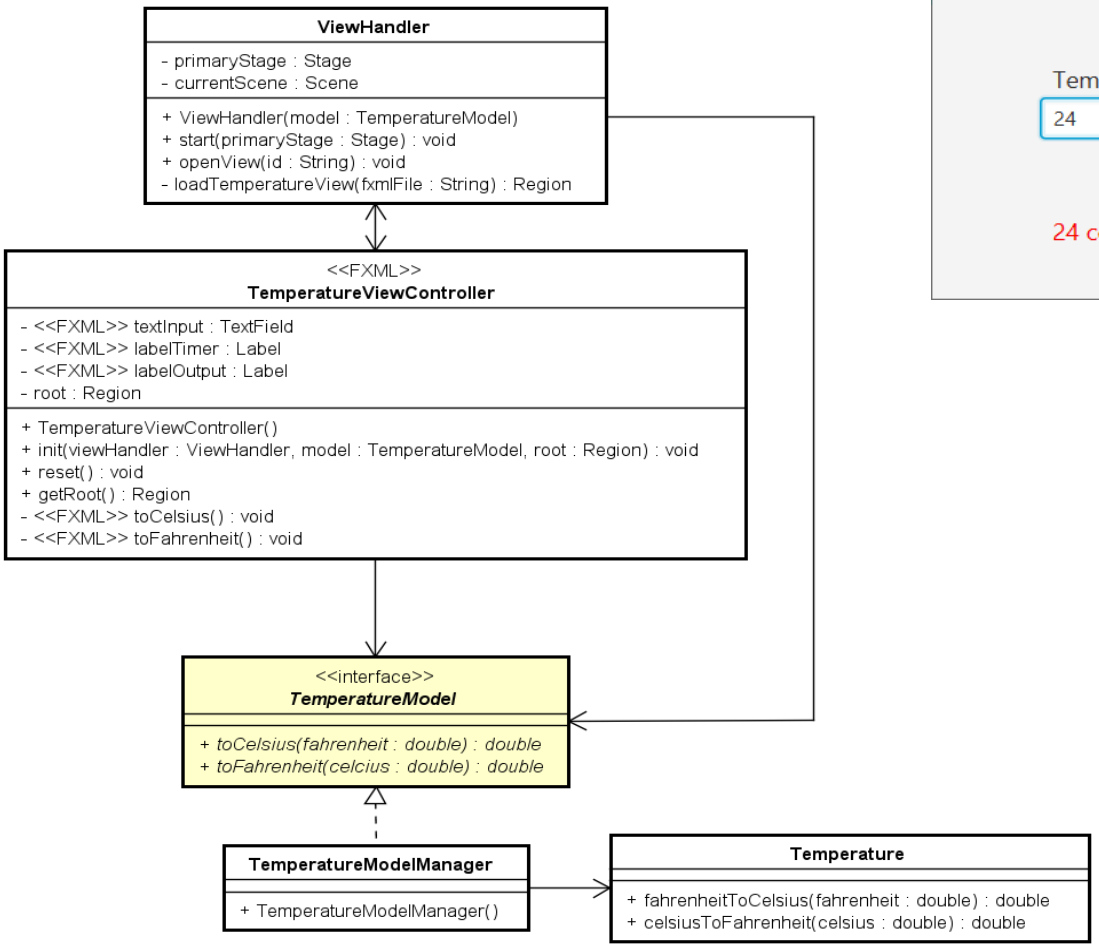

<<interface>>
java::lang::Runnable
+ run() : void
Adding
- id : String
- sleepTime : long
- list : ArrayList<String>
+ Adding(list : ArrayList<String>, id : String, sleepTime : long)
ThreadMain
+ main(args : String[]) : void

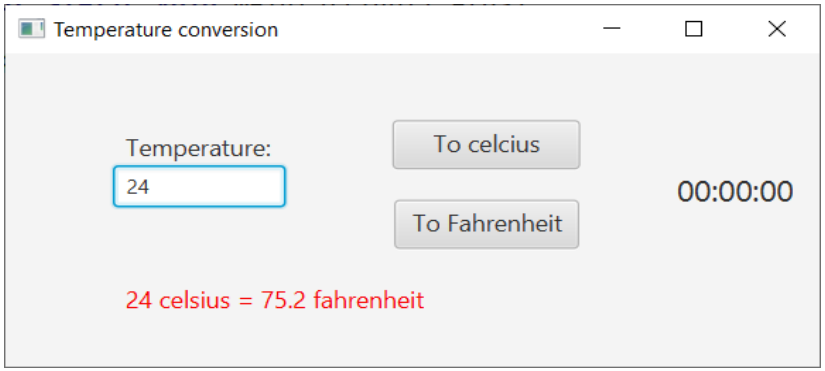
Exercises

```
id=A, list=[A#1]
id=B, list=[A#1, B#1]
id=A, list=[A#1, B#1, A#2]
id=C, list=[A#1, B#1, A#2, C#1]
...
[A#1, B#1, A#2, C#1, A#3, B#2, A#4, A#5, B#3, C#2, B#4, C#3, B#5, C#4, C#5]
count=15
```



Exercises


ViewHandler
- primaryStage : Stage
- currentScene : Scene
+ ViewHandler(model : TemperatureModel)
+ start(primaryStage : Stage) : void
+ openView(id : String) : void
- loadTemperatureView(fxmlFile : String) : Region
<<FXML>>
TemperatureViewController
- <<FXML>> textInput : TextField
- <<FXML>> labelTimer : Label
- <<FXML>> labelOutput : Label
- root : Region
+ TemperatureViewController()
+ init(viewHandler : ViewHandler, model : TemperatureModel, root : Region) : void
+ reset() : void
+ getRoot() : Region
- <<FXML>> toCelsius() : void
- <<FXML>> toFahrenheit() : void
<<interface>>
TemperatureModel
+ toCelsius(fahrenheit : double) : double
+ toFahrenheit(celcius : double) : double
TemperatureModelManager
+ TemperatureModelManager()
Temperature
+ fahrenheitToCelsius(fahrenheit : double) : double
+ celsiusToFahrenheit(celsius : double) : double


Temperature conversion
Temperature:
24
To celcius
To Fahrenheit
00:00:00
24 celsius = 75.2 fahrenheit

Exercises

```
---->Turning on the computer
Windows wants to update
New mail in your mailbox...
Skype wants to notify: a person logging in
Windows wants to update
Windows wants to update
New mail in your mailbox...
```