Project 5-1
Multi-Thread Programming[*]

Jiashuo Wang
5100309436

April 27, 2013

Instructor: Ling Gong

I. Objective

Create a class named *Countdown* that uses an explicit *Thread* (don't use the *Timer* class for this homework) to write a countdown to the console, one number per second. You can either extend *Thread* or implement *Runnable*. Your solution is free to implement other classes, but main() should be in Countdown.

The program will not actually exit until the user hits the enter key. If the user hits the enter key before the countdown has finished, the program should print interrupted and stop immediately. See transcripts below:

```
java Countdown
10
9
8
7
6
5
4
3
2
1
0
user hits the Enter key
finished
java Countdown
10
9
8
7
6
5
4
```

*Designed by LaTeX

user hits the Enter key
interrupted
finished

Notes:

- The program should exit on exceptions
- Don't use *System.exit*() to stop your program when the user hits Enter (although you can put it in your Exception handlers)
- the *readLine*() and *sleep*() methods will be useful
- If you have any questions, send me an e-mail. You can also check out the Java tutorial on Threads.

Extra clarification/hint:

Since we are not using *System.exit*(), then a stoppable thread should check a boolean variable each time it goes around its loop to see if it should continue.

To stop the thread, one just sets the variable, and the loop will stop itself. Note that the thing stopping the thread will need a reference to the stoppable thread object. Also, the stoppable thread object must provide some public method for setting the value of the "should-I-continue" variable.

II. Algorithm

II.1 Create Another Thread

In general, there is only one thread in a java process, which is the one that contains the *main* method. In order to create another thread, you have to either extend *Thread* or implement *Runnable* when defining another class.

- **Extend Thread**
  If extending *java.lang.Thread*, I have to override *run*() method. I can simply use *start*() method to start a thread. And when a thread is started, *JVM* will execute *run*() automatically. However, if extending *Thread*, the class cannot extend other class, which is not extensible.
- **Implement Runnable**
  If implementing *java.lang.Runnable*, I also have to override *run*() method. But in order to start it, it seems a little complicated. Firstly, I need to instantiate the *Thread* with the constructor *Thread*(*Runnabletarget*). Then I can just use the *start*() method of *Thread* to start the thread.

In my code, I extend *Thread* to create a new thread.

II.2 Count Down

By using *sleep*(*millisecond*), I can easily get the interval of 1s. Then simply using *while* loop to count down is an easy way to finish the work.

II.3 End the Thread

If a variable is marked with *volatile*, this variable can be shared with the last modified value. After the Enter button is pressed, thread will be interrupted into the *InterruptedException* and the *volatile* variable "should-I-continue" will be changed, which leads to the stop of the counter and the exit of the thread.

II.4 More details

Use *wait*() to pend the thread to hand over the CPU to make CPU more efficient. And *wait*() method should be contained in the *synchronized*() method.

III. Results and Conclusions

III.1 Environment

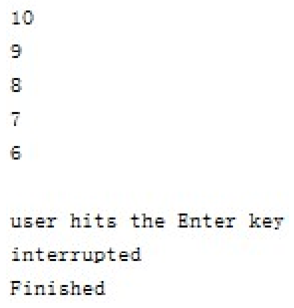
- Windows 8
- NetBeans IDE 7.3

III.2 Screenshots of the result

Use JVM to compile and execute the program in Figure 1 and Figure 2.

III.3 Thoughts

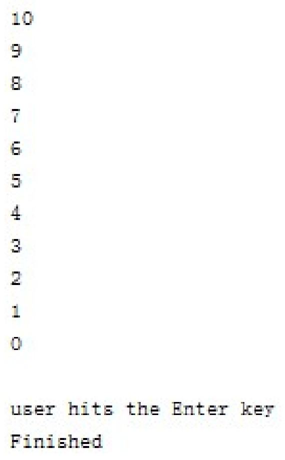
Multi-thread is useful and Java is a good way to implement it.



```
10
9
8
7
6

user hits the Enter key
interrupted
Finished
```

Figure 1: *Screenshots of Multi-Thread Programming(1)*



```
10
9
8
7
6
5
4
3
2
1
0

user hits the Enter key
Finished
```

Figure 2: *Screenshots of Multi-Thread Programming(2)*