COMPSCI 230 Threading Week11

“In computer science, a thread of execution is the smallest sequence of programmed instructions that can be managed independently by an operating system scheduler. The scheduler itself is a light-weight process. The implementation of threads and processes differs from one operating system to another, but in most cases, a thread is contained inside a process. Multiple threads can exist within the same process and share resources such as memory, while different processes do not share these resources”.

Figure 1 shows using multi-core processors to run multiple threads

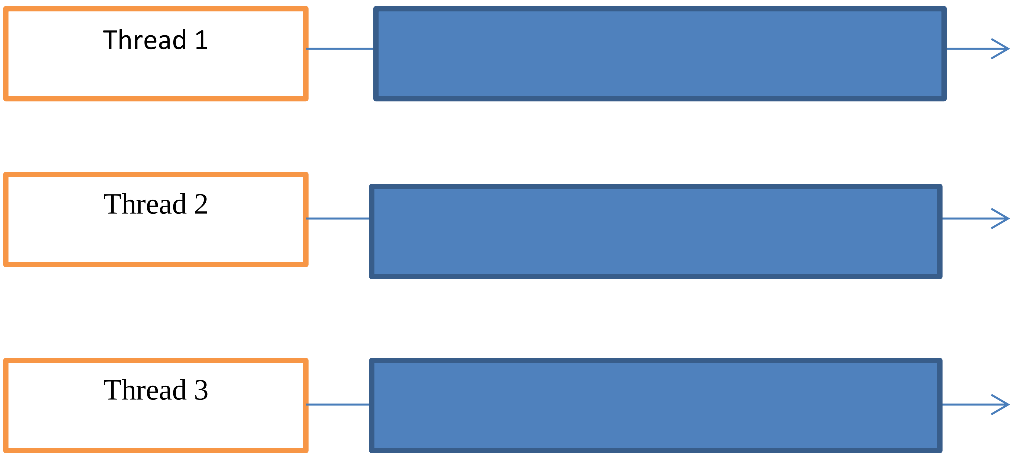
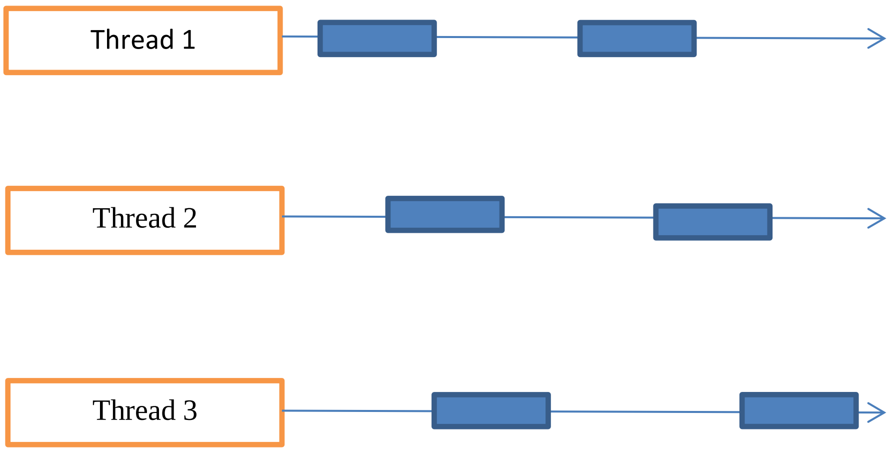


Figure 2 shows using a single processor to run multiple threads



Modify the testthread.java to create three threads, thread 1 will print the character 'a' 100 times, thread 2 will print the character 'b' 100 times and thread 3 will print the number from 1 to 100.
Q1: Please add the codes to create three threads and start threads

```
public class TestThread
{
  /**Main method*/
  public static void main(String[] args)
  {
    // Create threads

    PrintChar printA = new PrintChar('a', 100);
    PrintChar printB = new PrintChar('b', 100);
    PrintNum  print100 = new PrintNum(100);

    // Start threads
    print100.start();
    printA.start();
    printB.start();

  }
}
```

Q2: Please complete the blank part for the following line

```
class PrintNum extends  Thread

class PrintChar extends  Thread
```

Q3: Please override the run() method in both PrintChar and PrintNum classes

```
class PrintChar extends Thread
{
  private char charToPrint;  // The character to print
  private int times;  // The times to repeat

  /**Construct a thread with specified character and number of
     times to print the character
   */
  public PrintChar(char c, int t)
  {
    charToPrint = c;
    times = t;
  }

  /**Override the run() method to tell the system
     what the thread will do
   */
  public void run()
  {
    for (int i=0; i<times; i++)
      System.out.print(charToPrint);
  }
}
```

```
class PrintNum extends Thread
{
  private int lastNum;

  /**Construct a thread for print 1, 2, ... i*/
  public PrintNum(int n)
  {
    lastNum = n;
  }

  /**Tell the thread how to run*/
  public void run()
  {
    for (int i=1; i <= lastNum; i++)
      System.out.print(" " + i);
  }
}
```

Q2: In TestRunnable.java, use the Runnable interface to create three threads, thread 1 will print the character ‘a’ 100 times, thread 2 will print the character ‘b’ 100 times and thread 3 will print the number from 1 to 100.

```
public class TestRunnable {

        // main method
        public static void main(String args[]) {
                // Create threads
                Thread printA = new Thread (new PrintChar('a',
100));
                Thread printB = new Thread (new PrintChar('b',
100));
                Thread print100 = new Thread (new PrintNum(100));

                printA.start();
                printB.start();
                print100.start();

        } // main
} // TestRunnable

class PrintChar implements Runnable {
        private char charToPrint;      // the character to print
        private int times;             // The times to repeat

        // Construct a thread with specified character and number
        // of times to print the character
        public PrintChar(char c, int t) {
                charToPrint = c;
                times = t;
        } // printChar

        // Override the run() method to tell the system
        // what the thread will do
        public void run() {
                for (int i=0; i<=times; i++) {
                        System.out.print(" " + charToPrint);
                }
        } // run
} // PrintChar

class PrintNum implements Runnable {
        private int lastNum;           // the last number to print

        // Construct a thread with the last number
        public PrintNum(int n) {
                lastNum = n;
        } // printChar

        // Override the run() method to tell the system
        // what the thread will do
        public void run() {
                for (int i=0; i<=lastNum; i++) {
                        System.out.print(" " + i);
                }
        } // run
}// PrintNum
```

Q3: In TestRunnableSleep.java, modify the PrintNum class to allow the thread to sleep 1s, if the next print number is bigger than 50.

```
public void run()
   {
     for (int i=1; i <= lastNum; i++){
       System.out.print(" " + i);
       Try {
         If (i>50) thread.sleep(1000);
       }
       Catch (InterruptedException ex)
       {
       }
   }
}
```

Q4: When executing the programs from Q1,Q2, and Q3, the results are unpredictable. Why does this occur?

```
It occurs because we cannot ensure the order of thread execution.
```

Q5: What is a major benefit from using Runnable instead of Thread?

```
Thread is a class which means that, if you use it as a superclass, you
cannot extend from another class due to Java’s restriction on
extending from more than 1 class.

Implementing Runnable means you are still able to subclass.
```

Q6: What are the two types of threads and what is the difference?

```
Daemon and non-daemon threads.
```

Daemon: They are service or background threads.

Non-daemon: threads that are major part of the program execution. The main thread is one, as is the event dispatch thread in GUI apps. When all non-daemon threads terminate, then the program finishes.

Q7: Why shouldn't you execute time-consuming tasks in the Event-dispatch thread?

```
Because trying up the GUI thread with complex tasks will cause the GUI
itself to become unresponsive.
```

In MyForm.java in package Q4 we have a program that uses SwingWorker to complete background tasks. We're going to add our own background task that selects random letters from the alphabet and creates a string of length 10.

Q8: Create a new class called SwingAlphabet that extends SwingWorker. It should have a private instance variable of type String that is the just the alphabet. You must override the doInBackground method, the done method, and the process method. It should return a String at the end and print a Character as it's selected from the sequence.

```
class SwingAlphabet extends SwingWorker<String, Character>{

      Private String alpha = "abcdefghijklmnopqrstuvwxyz";

      @override
      protected String doInBackground(){
      }
      @Override
      protected void process(List<Character> chunks){
      }
      @Override
      protected void done(){
      }
}
```

Q9: Implement code in doInBackground() that: selects a random characters from variable alpha, publishes it so we can display it in the GUI, concatenates it to the end of a String, and finally sleeps for 1 second. Repeat this procedure 10 times then return the new random string.

```
@Override
             protected String doInBackground() {
                    System.err.format("... doInBackground isDaemon=%b%n",
Thread.currentThread().isDaemon());
                    String fakeString = "";
                    Random r = new Random();
                    for(int i = 0; i < 10; i++){
                           char t = alpha.charAt(r.nextInt(24));
                           super.publish(new Character(t));
                           fakeString += t;
                           try {
                                  Thread.sleep(500);
                           } catch (InterruptedException e) {
                                  // TODO Auto-generated catch block
                                  e.printStackTrace();
                           }
                    }
                    return fakeString;
             }
```

Q10: Implement code in done() that appends the completed string to the TraceBox.

```
@Override
             protected void done() {
                    try {
                           TraceBox.append("... done: 
str="+super.get()+"\r\n");
                    } catch (InterruptedException | ExecutionException e) {
                           // TODO Auto-generated catch block
                           e.printStackTrace();
                    }
                    AlphaButton.setEnabled(true);
             }
```

Q11: Implement code in process() that appends the currently selected letter to Tracebox.

```
@Override
             protected void process(List<Character> chunks) {
                    TraceBox.append("... process: ");
            for (Character i : chunks) {
                TraceBox.append(i+" ");
            }
            TraceBox.append("\r\n");
             }
```

Q12: Finally, add a button to start a new SwingALphabet and change the size of the JFrame to 800,520.

```
AlphaButton = new JButton("Start SwingAlpha");
        AlphaButton.setFont(font);
        AlphaButton.setPreferredSize(new Dimension(230,30));
        AlphaButton.addActionListener(new ActionListener() {

                    @Override
                    public void actionPerformed(ActionEvent arg0) {
                           AlphaButton_Click();

                    }
             });

……

this.setSize(800,520);

……..

public void AlphaButton_Click(){
       SwingAlphabet m = new SwingAlphabet();
       m.execute();
        AlphaButton.setEnabled(false);
    }
```

Q13: Look at the AccountWithoutSync.java in package Q5, the customer wants to create 100 threads, and using each thread to add one penny. After 100 threads, the balance should equal to 100.

```
package Q1;

public class AccountWithoutSync {

    private Account account = new Account();
    private Thread[] thread = new Thread[100];

    public static void main(String[] args) {
       AccountWithoutSync test = new AccountWithoutSync();
         System.out.println("What is balance ? " +  test.account.getBalance());
    }

   public AccountWithoutSync(){
           ThreadGroup g = new ThreadGroup("group");
           boolean done = false;
           for(int i =0 ; i< 100; i++)
           {
                   thread[i] = new Thread(g,new AddAPennyThread(), "t");
                   thread[i].start();
           }

           while (!done)
                   if(g.activeCount() == 0)
                           done = true;
   }
    // An inner class of task for adding a penny to the account
    class AddAPennyThread extends Thread {

        public void run() {
            account.deposit(1);
        }
    }
     class Account {

        private int balance = 0;

        public int getBalance() {
            return balance;
        }

        public void deposit(int amount) {

            int newBalance = balance + amount;

            try {
                Thread.sleep(5);
            } catch (InterruptedException ex) {
                // do nothing
            }

            balance = newBalance;
        }
    }
}
```

Q13-a: When you run `AccountWithoutSync.java`, what's the balance?

2

Q13-b: what's the problem?

The object is shared among multiple threads, which has been corrupted by different threads.

Q13-c: How to fix this problem?

```
Public synchronized void deposit(int amount)
```