Section: 1) UML Diagram Practice Test Question and 2) Threads in Java

1) Find the Objects, Responsibilities, and draw a UML class diagram

The college library has requested a system that supports a small library. The librarian allows a student to borrow certain books, return those borrowed books, and pay fees. Late fees and due dates have been established at the following rates:

	Late fee	Length of Borrow
books:	$0.50 per day	14 days

The due date is set when the borrowed item is checked out. A student with three (3) borrowed items, one late item, or late fees greater than $25.00 may not borrow anything new.

a) Create an initial list of candidate objects for this system and list its major responsibility

b) Draw a UML class diagram showing all of your candidate objects and any relationships between them. Show inheritance relationships, interface implementation, or general association such as dependency by drawing a line. Write any multiplicity adornment you can think of. You will likely have 1, and or * in a few places at least. The classes only need the class name and its major responsibility, which may be method names (no attributes needed for a perfect score).

2) Threads

Review extending `class Thread`, overriding `run`, and calling start

Run this program to show another example of a multiple threaded program. A sleep is inserted in the hopes that the scheduler will give the other thread some time

```
public class CountThread extends Thread {

  public static void main(String[] args) throws InterruptedException {
    // getName() will return the string in these constructors
    Thread t1 = new CountThread("t1");
    Thread t2 = new CountThread("t2");
    // Call start so the two Threads can share the CPU
    t1.start();
    t2.start();

    // Do not call start so the first Thread must finish before
    // the 2nd thread. This Mimics freezing the GUI.
    // t1.run();
    // t2.run();
  }

  public CountThread(String s) {
    super(s);
  }

  @Override
  public void run() {
    for (int i = 1; i <= 5; i++) {
      try {
        Thread.sleep(700);
      } catch (InterruptedException e) {
      }
      if(getName().equals("t2"))
        System.out.print("\n     ");
      System.out.print(getName() + ":" + i + " ");
      if(getName().equals("t2"))
        System.out.println();

    }
  }
}
```

Output (or something like this):

```
t1:1
     t2:1
t1:2
     t2:2
t1:3
     t2:3
t1:4
     t2:4
t1:5
     t2:5
```

An alternative to starting a new Thread

Another way of creating new threads in Java is to create a class that implements the Runnable Interface:

```
// Runnable is in java.lang
public interface Runnable {
  void run();
}
```

```
// We'll need an instance of this class passed to Thread's constructor
public class RunThreads implements Runnable {

  public void run() {
    // Put the code for the thread here
  }
}
```

Pass an instance of a Runnable to (a.k.a. a new RunThreads object here) to Thread's constructor:

```
RunThreads runner = new RunThreads();
Thread alpha = new Thread(runner);
Thread beta = new Thread(runner);
```

And as with extending Thread, start the new threads with start() messages

```
alpha.start();  // Both threads will doing the same thing:  see Run
beta.start();
```

```
/*
 * Attempt to show two threads sharing the processor.
 * We can't accurately predict when threads take control.
 * That's the job of the Thread scheduler.
 */
public class RunThreads implements Runnable {

  // Run this application several times to see that the
  // two threads are given processor time differently
  public static void main(String[] args) {

    RunThreads runner = new RunThreads();
    Thread alpha = new Thread(runner);
    Thread beta = new Thread(runner);
    alpha.setName("A");
    beta.setName("B");
    alpha.start();
    beta.start();
  }

  @Override
  public void run() {
    // Print the name of the current Thread 500 times
    for (int i = 1; i <= 400; i++) {
      if (i % 60 == 0)
        System.out.println(); // new line so we can see console output
      // Since we are not extending Thread, we can not just use getName.
      System.out.print(Thread.currentThread().getName() + " ");

      // If needed, slow down the loop a bit to give
      // the scheduler a reason to scheduler the other thread
//      try {
//        Thread.sleep(1); // millisecond
//      } catch (InterruptedException e) {
//
//      }
    }
  }
}
```

One output that happened. It will differ most times

```
A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
A B A B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B
B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B
B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B
B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B
B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B
B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B
B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B
B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B
B A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
```