CSSE 374: Lab 4-2

Background

Assume that you work for a startup company called Blueberry Muffin. Blueberry Muffin is planning to launch a single-board computer similar to Raspberry Pi in Jan 2016. In contrast to Raspberry Pi's ARM processors, it is planning to use one of the Intel's low-power, high-performance, quad-core processors to beat its competitors.

Blueberry Muffin has decided to support Java as one of its main development languages. Java is inherently multi-threaded but there are some limitations with the architecture of this low-power version of Intel's quad-core processor. To avoid performance degradation due to context-switching of threads within a core, the quad-core processor only allows one thread to run in the core end-to-end. i.e., once a thread starts in a core, it will finish in that core running continuously without being context switched. So, at any given moment, there can only be a maximum of four executing threads, one per core, in this architecture.

You work for the team that is developing a threading support for the JVM that runs on Blueberry Muffin. The new JVM, so far, **only** has support for **java.lang.Thread** and **java.lang.Runnable** APIs for threading. It, however, does have support for other general purpose APIs such as List, Sets, Maps, etc. (Please read the following tutorial if you are not familiar with Java's threading APIs: http://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html.) At present, if more than four threads are created within any Blueberry Muffin Java applications, then the whole system crashes with an error.

Your product manager wants you to design and develop a solution for this problem. In particular, here are the feature requests:

F1 - Any Java application that runs on Muffin should be able to create more than four **virtual threads**. Internally, however, these virtual threads should be queued (First-Come-First-Served) such that only four **real Java threads** are doing the work on the cores. The library could use one of the cores (or thread) to just perform thread management and the other cores could be used for executing user defined jobs. As a proof of concept, for the purposes of this lab, you may use the **regular Thread class** and/or **Runnable interface** of your local JVM to simulate the JVM of Muffin with the assumption that your library cannot create more than four real threads including the main thread to do the work at any given time. Use **problem.blueberrymuffin** package to code the library.

F2- Code a sample usage of the library by creating more than four **virtual threads**. You could make each virtual thread execute the following, five times in a loop:

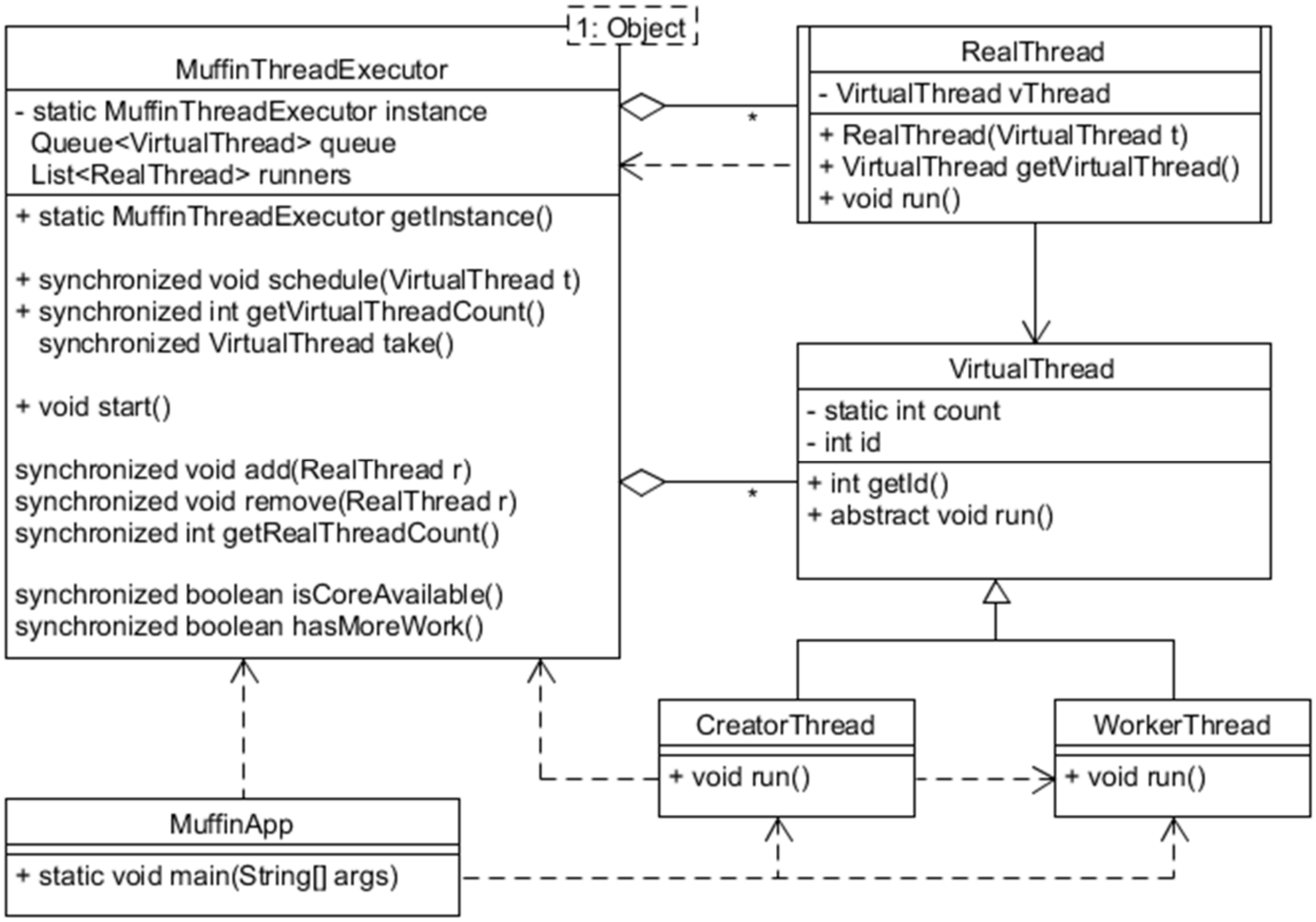
- Print "Hello from thread-x," where x is a unique virtual thread identifier.
- Sleep: 300 millis

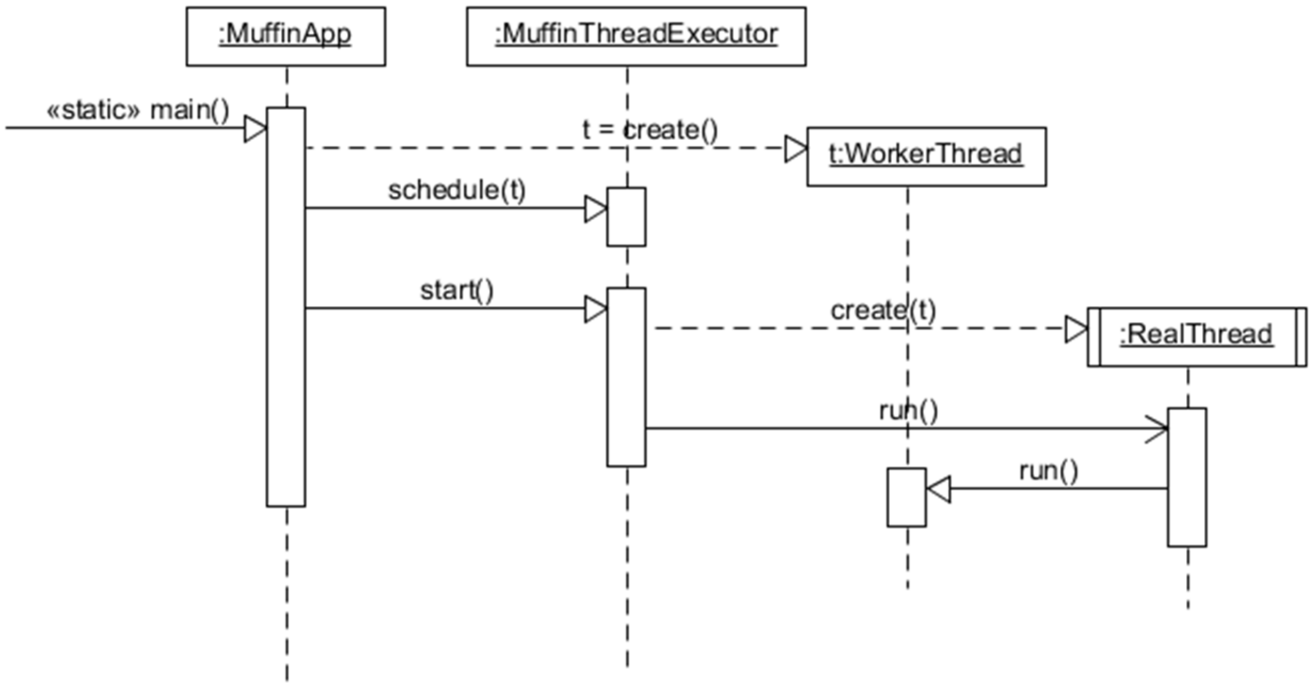
Use **problem.blueberrymuffinclient** package for F2.

Design

Create **Lab4-2/docs/Answer.pdf** with answers to the following problems:

Q1. Create a **UML Class Diagram** to present your design idea. Using a **Sequence Diagram**, depict the general sequence of method calls in your library. You should start from the **main** method of the client. Explain the design in a few lines. [**10 points**]


1: Object
MuffinThreadExecutor
- static MuffinThreadExecutor instance
Queue<VirtualThread> queue
List<RealThread> runners
+ static MuffinThreadExecutor getInstance()
+ synchronized void schedule(VirtualThread t)
+ synchronized int getVirtualThreadCount()
synchronized VirtualThread take()
+ void start()
synchronized void add(RealThread r)
synchronized void remove(RealThread r)
synchronized int getRealThreadCount()
synchronized boolean isCoreAvailable()
synchronized boolean hasMoreWork()
RealThread
- VirtualThread vThread
+ RealThread(VirtualThread t)
+ VirtualThread getVirtualThread()
+ void run()
*
VirtualThread
- static int count
- int id
+ int getId()
+ abstract void run()
*
CreatorThread
+ void run()
WorkerThread
+ void run()
MuffinApp
+ static void main(String[] args)


:MuffinApp
:MuffinThreadExecutor
«static» main()
t = create()
t:WorkerThread
schedule(t)
start()
create(t)
:RealThread
run()
run()

Short Explanation of the Design

We use Command pattern, where a Command is represented by a VirtualThread object. MuffinThreadExecutor is implemented as a Singleton and controls the number of real threads that executes these VirtualThread commands asynchronously. The MuffinThreadExecutor supplies each RealThread object a VirtualThread to run.

Implementation

Q2. Implement your code in the appropriate subpackages of the **Lab4-2/src/problem** package. [**F1- 20 points**, **F2-10 points**]

Testing

Q3: Implement necessary test cases in the appropriate subpackages of the **Lab4-2/test/problem** package. [**10 points**]

Deliverable

Bundle your project in the **zip** format [**not rar**] and turn it in on Moodle.