Second Task:

We need to solve questions after reading and solving examples from the first task:

Some Of the tasks are easy after reading the wiki but the implementation is difficult:

Question:

How do you wait for the specific thread to complete its job?

Could not solve as:

thread.join() (TimeUnit.SECONDS.toMillis(10));

is not correct.

Also Tried,

thread.wait() (TimeUnit.SECONDS.toMillis(10));

Correct, answer to this question still not clear.

Question On Synchronisation(Easy):

Question: Make the following code below thread safe.

Hint: just synchronize the access to all the the class methods.

Answer is Quite Clear and Simple after reading the wiki material:

public synchroniz void withdraw(final int amount)

public synchronis void deposit(final int amount)

public synchronis int getBalance()

Answer: Synchronised Class is used to make the code Thread safe.

Some Questions Required doing some search in java manuals and google

Example:

Question:
How to get the number of processors (or more precisely logical threads) available to the Java virtual machine?
Answer:
Runtime.getRuntime().availableProcessors()
Research based Question: Informative Question

Question:

What it is the analogue in Java of the Go RWMutex ?

Answer: ReadWriteLock

Question:

What is the legal way to terminate the Java thread in the context of the program below?

Answer: isInterrupted()
Interrupt();

```
import java.util.concurrent.TimeUnit;
public class TerminateThread {
   public static void main(final String[] args) throws Exception {
      final Thread thread = new Thread(() -> {
         while (!Thread.currentThread(). isInterrupte ) {
            try {
               System.out.println("Running");
               TimeUnit.SECONDS.sleep(1);
            } catch (final InterruptedException ex) {
               break;
            }
         }
         System.out.println("Terminated");
      });
      thread.start();
      TimeUnit.SECONDS.sleep(5);
      System.out.println("Terminate a thread");
      thread.interrupt() ;
   }
}
```

Question:

Make the following code below thread safe.

Answer:

Timeunit is used as atomic package but I could not understand the second part of the code.

Private final and new

Example:

```
import java.util.concurrent.TimeUnit;
import java.util.concurrent.timeunit;
public class ThreadSafe2Quiz {
   public static void main(final String[] args) throws Exception {
      final ThreadSafe2 runnable = new ThreadSafe2();
      new Thread(runnable).start();
      TimeUnit.SECONDS.sleep(5);
      runnable.cancel();
   }
   public static class ThreadSafe2 implements Runnable {
      private final  done = new (false);
      @Override
      public void run() {
         while (!done.get()) {
            System.out.println("Running");
            try {
               TimeUnit.SECONDS.sleep(1);
            } catch (final InterruptedException ex) {
               // reset the interruption status
               Thread.currentThread().interrupt();
            }
         }
         System.out.println("Done");
      }
      public void cancel() {
         done.set(true);
      }
   }
}
```

Question:

What is the optimal number of threads for the IO intensive tasks?

Answer:

Number of available cores
Number of available cores + 1
Number of available cores / 2
Number of available cores * 2
Number of available cores / (1 - Blocking coefficient)

Question:
What is the analogue in Java of the following Go Lang concurrency primitive - *Once*, which according to the documentation?

Answer:

Java doesn't have one out of the box, but it can be modeled using static initializer or single value Enum.

Question:

What is the analogue in Java of the following Go lang concurrency primitive - *Pool*, which according to the documentation?

Answer:

Java doesn't have one out of the box, but it can be modeled using one of the concurrent bounded blocking queues, like ArrayBlockingQueue, for example.

Question:

What is the analogue in Java of the following Go lang concurrency primitive - *WaitGroup*, which according to the documentation*?*

Answer:

CountDownLatch

Question:

How to get the current thread in Java?

Answer:

currentThread()

```
public class CurrentThread {
   public static void main(final String[] args) {
      System.out.println(Thread. currentThr .getName());
   }
}
```

Question:

Make the corresponding change in the program below to allow the program to terminate even if not all the threads are completed (or if they are completed).

Answer:

Could Not Solve.

Most Possible Answers :

start()

Sleep()

Join()

Interrupt()

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
public class ExitMain2 {
   private static final int POOL_SIZE = 50;
   public static void main(final String[] args) throws Exception {
      final ExecutorService executor = Executors.newFixedThreadPool(POOL_SIZE, runnable -> {
         final Thread thread = new Thread(runnable);
         thread. ;
         return thread;
      });
      for (int i = 0; i < POOL_SIZE; i++) {
```

```
            executor.submit(() -> System.out.printf("Running %s%n", Thread.currentThread().getName()));
        }
    }
}
```

Question:

How do you wait for the specific thread to complete its job?

Answer:

Could not solve.

Most Probable answer:

Sleep()

Code snippet:

```
import java.util.concurrent.TimeUnit;
public class Waiting1 {
    public static void main(final String[] args) throws Exception {
        final Thread thread = new Thread(() -> {
            try {
                TimeUnit.SECONDS.sleep(5);
            } catch (final InterruptedException ex) {
                // reset the interruption status
                Thread.currentThread().interrupt();
            }
            System.out.println("Exiting");
        });
        thread.start();
        // it is recommended, whenever is appropriate and possible, to use timed out version of the
available JDK API around concurrency
        thread.sleep(TimeUnit.SECONDS.toMillis(10));
        System.out.println("Done");
    }
}
```

Question:

What is the optimal number of threads for the compute intensive tasks?

Answer: System Buggy. No answers showing correct.

Number of available cores

Number of available cores +1

Number of available cores /2

Number of available cores * 2