29.4 What is wrong in the following two programs? Correct the errors.

Program 1

```
public class Test implements Runnable {
     public static void main(String[] args) {
          new Test();
     }
     public Test(){
     Test task = new Test();
     new Thread(task).start();
     }
     public void run() {
          System.out.println("test");
     }
}
```

29.4 Answer 1

```
"Test task = new Test();" has a execution error that is a
StackOverflowError. This is because it is called recursively in its own
constructor.

I would correct the program as follows.

public class Test implements Runnable {
     public static void main(String[] args) {
          new Test();
     }
     public Test(){
     Thread task = new Thread(this);
     task.start();
     }
     public void run() {
          System.out.println("test");
     }
}
```

Program 2

```
public class Test implements Runnable {
     public static void main(String[] args) {
          new Test();
     }
     public Test(){
     Thread t = new Thread(this);
     t.start();
     t.start();
     }
     public void run() {
          System.out.println("test");
     }
}
```

29.4 Answer 2

```
The second "t.start();" has a execution that is a
IllegalThreadStateException. This is because the same thread cannot start
concurrently in more than one.

I would remove one of them as follows.
```

```
public class Test implements Runnable {
     public static void main(String[] args) {
          new Test();
     }
     public Test(){
     Thread t = new Thread(this);
     t.start();
     }
     public void run() {
          System.out.println("test");
     }
}
```

29.10 What are the benefits of using a thread pool?

29.10 Answer

By using a thread pool, tasks are added to a queue and are executed in order. Because of creating the minimum required tasks concurrently, you can avoid wasting resources.

29.14 How do you create a lock object? How do you acquire a lock and release a lock?

29.14 Answer

To create a lock object, you instantiate an object by using Lock interface as "Lock lock = new ReentrantLock();".

To acquire a lock, you use lock() method as "lockObject.lock();". To release a lock, you use unlock() method as "lockObject.unlock();".

29.15 How do you create a condition on a lock? What are the await(), signal() and signalAll() methods for?

29.15 Answer

If you created a Lock object named lock, you can create a condition by using Condition interface, which is bound in Lock interface, as "Condition condition = lock.newCondition();".

By using await(), current thread waits for it is signaled or interrupted. Or, if you set specified time to the argument, the thread waits until the time.

By using signal(), a waiting thread is started. By using signalAll(), all of waiting threads are started.

29.18 What is the possible cause for IllegalMonitorStateException?

29.18 Answer

If you use wait(), notify, or notifyAll methods outside of synchronized blocks, you will get an IllegalMonitorStateException.

29.21 What blocking queues are supported in Java?

29.21 Answer

You can block queues by using put(e), take(), offer(e, time, unit), or poll(time, unit), which are provided by BlockingQueue interface. Also, put(e) and take() are without timeouts, and offer(e, time, unit) and poll(time, unit) are with timeouts.

29.22 What method do you use to add an element to an ArrayBlockingQueue? What happens if the queue is full?

29.22 Answer

You can use add(e), offer(), offer(), or put() to add an element to an ArrayBlockingQueue.

When using add(e), and if the queue is full, the method throws IllegalStateException.

When using offer(e), and if the queue is full, the method returns false.

When using offer(e, time, unit), and if the queue is full, the method waits for the queue to become available until specified time

When using put(e), and if the queue is full, the method waits for the queue to become available.

29.23 What method do you use to retrieve an element from an ArrayBlockingQueue? What happens if the queue is empty?

29.23 Answer

You can use peek(), poll(), poll(time, unit), or take() to retrieve an element from an ArrayBlockingQueue.

When using peek() or poll(), and if the queue is empty, the methods throws null.

When using poll(time, unit), and if the queue is empty, the method waits for the queue become non-empty until specified time.

When using take(), and if the queue is empty, the method waits for the queue become non-empty.