**Exercise 1**

1. The time at which each task is added to a queue.

Representations of task priority, cost, price or urgency.

The number of tasks waiting for some condition.

Termination dependencies among tasks.

The number of tasks that have completed.

1. If a process continuously makes a request, eventually it will be granted. (weak)

If a process makes a request infinitely often, eventually it will be granted. (strong)

If a process makes a request, it will be granted before any other process is granted the request more than once. (linear)

1. Pass-Throughs: The host maintains a set of immutable reference to helper objects and simply relays all messages to them within unsynchronized methods.

Lock-Splitting: Instead of splitting the class, split the synchronization locks associated with subsets of the state.

1. Clients can tolerate either failure or retries.

You can avoid or cope with livelock.

You can undo actions performed before failure checks.

**Exercise 2**

Project file attached.

**Exercise 3**

1. In Thread.join(), waiting threads wait for specified executing Thread’s state to change to Terminated. After it’s terminated, they all get notified.
2. This code means that you will start execution of a new thread and wait for that thread’s execution to be finished. This is not different than linear programming, we can remove threading and code it as a sequential program. If we seperated this thread for calculations, than it would be better for us to use a Future instead of this join method.
3. Immutable object means that this object’s state cannot be modified after it’s created. According to this definiton and java documentation, Strings are immutable. However, if you mean immutability requires that every immutable object need to have it’s own instance, then that’s not the case with java. That’s because Java uses a string pool. So, if you created the same String before, than Java string pool matches this string and returns the reference of previously created string instance. But, still this doesn’t break the rule of an object’s state cannot be modified. Also, if you want to have some excitement, than you can use String Buffer to create mutable Strings.
4. No, that specified progress property enforce fairness to a degree, because it’s an or statement, which means if heads occur infinetely often or if tales occur infinetely often, then this property will hold. So, there’s a chance of having an infinitely often occuring heads and you can’t be aware of that actually tails are never occuring. If there’s a never occuring action, then we cannot call this as a fair program.