Concurrency

EECS 4315

www.eecs.yorku.ca/course/4315/

Java code

```
public static void main(String[] args) {
  Printer one = new Printer("1");
  one.run();
}
```

Executions

Question

Draw the state-transition diagram.

Executions

Java code

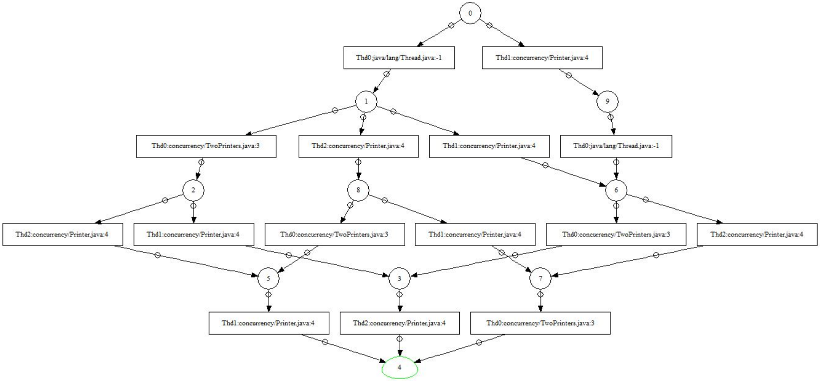
```
public static void main(String[] args) {
  Printer one = new Printer("1");
  Printer two = new Printer("2");
  one.start();
  two.start();
}
```

Executions

Question

Draw the state-transition diagram.

Executions


0
Thd0:java/lang/Thread.java:-1
Thd1:concurrency/Printer.java:4
1
9
Thd0:concurrency/TwoPrinters.java:3
Thd2:concurrency/Printer.java:4
Thd1:concurrency/Printer.java:4
Thd0:java/lang/Thread.java:-1
2
8
6
Thd2:concurrency/Printer.java:4
Thd1:concurrency/Printer.java:4
Thd0:concurrency/TwoPrinters.java:3
Thd1:concurrency/Printer.java:4
Thd0:concurrency/TwoPrinters.java:3
Thd2:concurrency/Printer.java:4
5
3
7
Thd1:concurrency/Printer.java:4
Thd2:concurrency/Printer.java:4
Thd0:concurrency/TwoPrinters.java:3
4

Counter class

Problem

Implement the class `Counter` with attribute `value`, initialized to zero, and the methods `increment` and `decrement`.

Counter class

Problem

Implement the class `Counter` with attribute `value`, initialized to zero, and the methods `increment` and `decrement`.

Question

Can multiple threads share a `Counter` object and use methods such as `increment` and `decrement` concurrently?

Counter class

Problem

Implement the class `Counter` with attribute `value`, initialized to zero, and the methods `increment` and `decrement`.

Question

Can multiple threads share a `Counter` object and use methods such as `increment` and `decrement` concurrently?

Answer

No, as before, if two threads invoke `increment` concurrently, the counter may only be incremented by one (rather than two).

Synchronized methods

Methods such as `increment` should be executed atomically. This can be accomplished by declaring the method to be `synchronized`.

A lock is associated with every object. For threads to execute a `synchronized` method on such the object, first its lock needs to be acquired.

Synchronized methods

Methods such as `increment` should be executed atomically. This can be accomplished by declaring the method to be `synchronized`.

A lock is associated with every object. For threads to execute a `synchronized` method on such the object, first its lock needs to be acquired.

```
public synchronized void increment() {
  this.value++;
}
```

Resource class

Problem

Implement the class `Resource` with attribute `available`, initialized to true, and the methods `acquire` and `release`.

Wait and notify

The `Object` class contains the following three methods:

- `wait`: causes the current thread to wait for this object's lock until another thread wakes it up.
- `notify`: wakes up a single thread waiting on this object's lock; if there is more than one waiting, an arbitrary one is chosen; if there are none, nothing is done.
- `notifyAll`: wakes up all threads waiting on this objects lock.

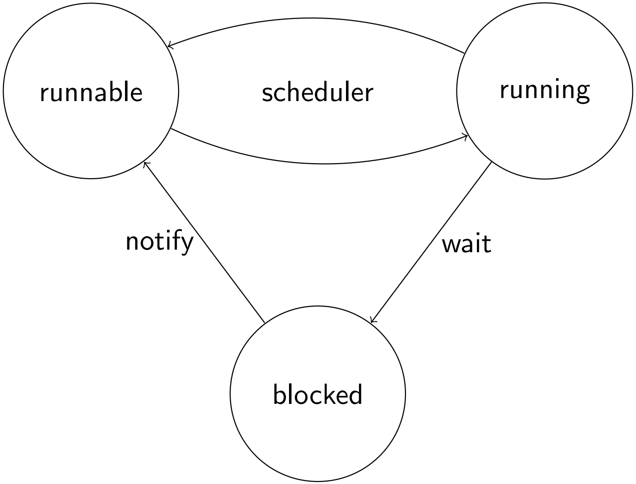
Wait and notify

The `Object` class contains the following three methods:

- `wait`: causes the current thread to wait for this object's lock until another thread wakes it up.
- `notify`: wakes up a single thread waiting on this object's lock; if there is more than one waiting, an arbitrary one is chosen; if there are none, nothing is done.
- `notifyAll`: wakes up all threads waiting on this objects lock.

Since every class extends the class `Object`, these methods are available to every object.

States of a thread



User class

```
public class User extends Thread {
  private Resource resource;

  public User(Resource resource) {
    super();
    this.resource = resource;
  }

  public void run() {
    super.run();
    this.resource.acquire();
    this.resource.release();
  }
}
```

Main method

```
final Resource resource = new Resource();
final int USERS = 2;
final User[] users = new User[USERS];
for (int i = 0; i < USERS; i++) {
  users[i] = new User(resource);
}
for (int i = 0; i < USERS; i++) {
  users[i].start();
}
```

Configuration file

```
target=Main
classpath=<folder that contains Main.class>
listener=listeners.StateSpaceWithThreadInfo
native_classpath=<folder that contains
  listener/StateSpaceWithThreadInfo.class>
```

State space