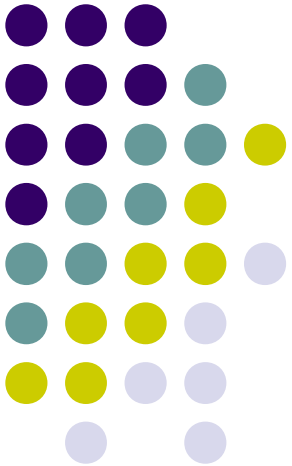
Presentation of Practical Exercise 2

Thread synchronization in Java

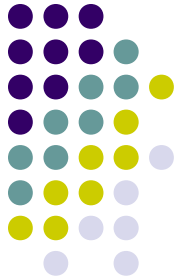
The Producer/Consumer model

The Barbershop example



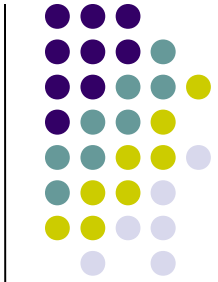
TDT4186, H2011

What is thread synchronization?



- All the threads of a process share the same address space and other resources.
- The various activities of the threads must be coordinated (synchronized) so that they do not interfere with each other or corrupt data structures.
- Threads also use synchronization to signal each other to coordinate actions (e.g. the Producer/Consumer model).

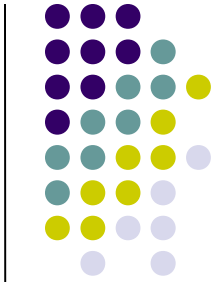
Synchronization in Java



- In Java, threads are synchronized using a modified monitor concept.

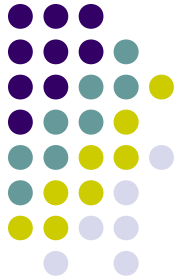
- Traditional monitor characteristics:
  - A monitor is equivalent to a Java object, but:
  - The local data variables are accessible only by the monitor's procedures and not by any external procedure (read: All member variables are private).
  - A process (read: Thread) enters the monitor by invoking one of its procedures (read: By invoking a method on the object).
  - Only one process may be executing in the monitor at a time; any other process that has invoked the monitor is suspended, waiting for the monitor to become available (read: All methods are synchronized).

How to make a monitor



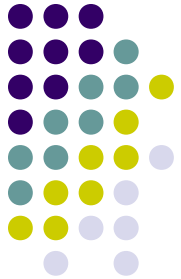
- All java objects can serve as monitors.
- If an object has only private member variables, and all methods are synchronized, the object is a traditional (Lampson & Redell) monitor.
- Not all methods of a Java object need be synchronized. This makes hybrid solutions possible.

Thinking about monitors



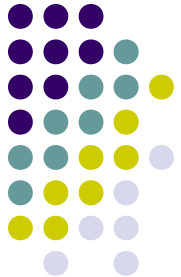
- A monitor protects data that are vulnerable to concurrent modifications.
- Methods accessing the data should be synchronized to ensure mutual exclusion.
- The mutual exclusion is implemented by a lock associated with the monitor. This lock is held by the thread currently executing the monitor.
- Threads accessing the protected data can synchronize their actions using the wait() and notify() methods.

wait() and notify()



- wait() and notify() are methods of the class Object that can only be invoked inside a synchronized block.
- These methods are used for message passing between threads.
- wait() causes a thread to release the monitor's lock and suspend itself, waiting for a notify() message.
- notify() is used to notify a suspended thread that it's time to "wake up".

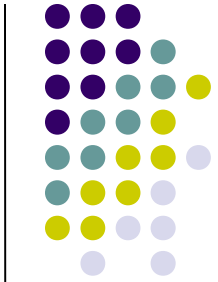
Nitty-gritty details



- Threads who invoke the wait() method…
  - release the monitor…
  - are suspended…
  - and are placed in the monitor's condition queue.

- Threads who invoke the notify() method…
  - cause one (unspecified) thread in the condition queue to wake up…
  - and continue their execution without releasing the monitor.

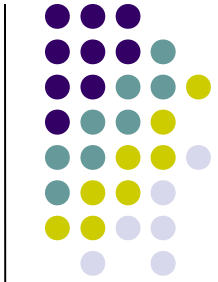
More details



- Threads in the condition queue that wake up..
  - try to reacquire the monitor
  - continue their execution after they have acquired the monitor.

- All this ensures mutual exclusion. Threads may wait for each other, but only one thread is executing ”within” the monitor at any time.

The notifyAll() method



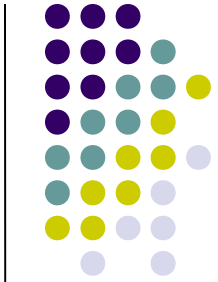
- If a thread calls the notifyAll() method, all threads in the condition queue wake up.
- These threads must then compete for the monitor's lock, so only one of them is allowed to enter the monitor at a time.
- This method can be useful in the Producer/Consumer model (and is equivalent to the cbroadcast primitive).

Synchronized blocks

- A synchronized block is a block of code protected with mutual exclusion by a monitor.
- A synchronized method constitutes a synchronized block protected by the monitor that the method belongs to.
- A thread must be inside a synchronized block to be able to invoke the wait() and notify() methods (otherwise an IllegalMonitorStateException is thrown).
- A synchronized block may be placed in another class than the monitor's, using the following construct:

```
synchronized(theMonitorObject) {
  // Protected code
}
```

The synchronized() construct

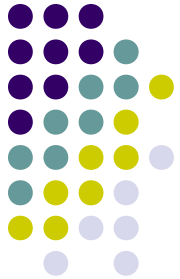


- The synchronized() construct can also be used to make only part of a method synchronized:

```
public void someMethod() {
   // Some non-critical code
   synchronized(this) {
     // Protected code
   }
   // Some more non-critical code
}
```

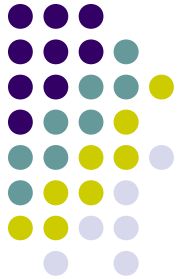
- The synchronized() construct is rarely used but is mentioned for completeness.

When to synchronize



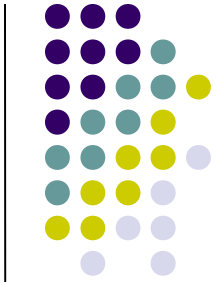
- Use the synchronized keyword to perform ”atomic operations”.
  - Data structure modifications
    - Adding an element to a linked list
    - Shifting the contents of an array
    - etc.
  - ”One at a time” operations
    - Outputting a status report
    - Other I/O
- Use the wait() and notify() methods to coordinate the activites of threads accessing synchronized blocks.
- Be aware that the use of synchronized methods can lead to deadlocks.

The producer / consumer model



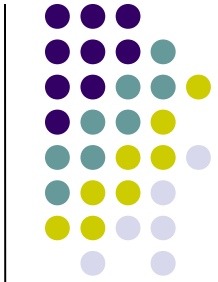
- The P/C model is a classic example of when threads need to coordinate their actions.
- The producer is producing items needed by the consumer. The items are stored in a temporary buffer. The consumer is consuming items stored in the buffer.
- If the buffer is full the producer must wait.
- If the buffer is empty, the consumer must wait.
- Only one of them may access the buffer at any time.

Typical P/C situations



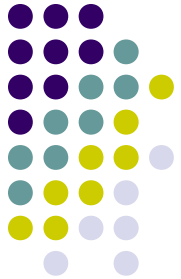
- Network communication.
- I/O management in general
- Streams in Java.
- Classical example:
  - The Barbershop example

The exercise



- Implement a version of the Barbershop example.
- A doorman produces customers.
- Customers are buffered in a ring of chairs.
- Three barbers consume customers.
- A handed-out GUI visualizes the proceedings.

Hints



- Your buffer of customers contains data that need protection!
- Hence, this buffer should probably function as a monitor.
- You can run the handed out code by running the P2.bat batch file, but the simulation doesn't do anything until you've completed your part.