Parallel Programming
0024

Thread Synchronization --- Examples

Assignment 2

http://www.lantersoft.ch/de/parallelprog/parallelprog.php

Danke für die Kommentare

Klassennamen gross schreiben

i++oder ++i ?

Exception im throw-Statement ? (ThrowNull)

Integer.parseInt() (ExploreParseInt)

Non-static, static und anonyme Klassen (ClassDemo)

Creating Threads

```
public class Main {
    public static void main(String[] args) {
        Buffer buffer = new UnsafeBuffer();
        new Thread(new Producer(buffer)).start();
        new Thread(new Consumer(buffer)).start();
    }
}
```

- Start a thread
  - thread.start() starts a new thread. A thread takes an object of type Runnable in the constructor.
  - subclass Thread and overwrite the run() method
- Note: thread.run() does not create a new thread

Putting a thread to sleep

```
try {
    //doze a random time (0 to 0.5 secs)
    //to simulate workload
    Thread.sleep((int)(Math.random()*500));
    } catch (InterruptedException e) { ... }
}
```

- Thread.sleep(long) puts the current thread to sleep for the specified time in milliseconds.
- An InterruptedException is thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the interrupt method in class Thread.

Synchronized

- Every class and every object has an intrinsic lock
- The synchronized keyword marks code blocks where a thread must acquire the lock before proceeding
- The synchronized keyword can be added to methods
- The “this” pointer is used as the lock for instance methods

```
public class Buffer {
    public synchronized void write(int i) {
...
    }

    public synchronized int read() {
...
    }
}
```

Mutually exclusive because both use “this” as the lock

Synchronized II

- The synchronized keyword can also be used to guard arbitrary blocks of code within a method, even in different classes
- It is important to use the correct object as the locks!

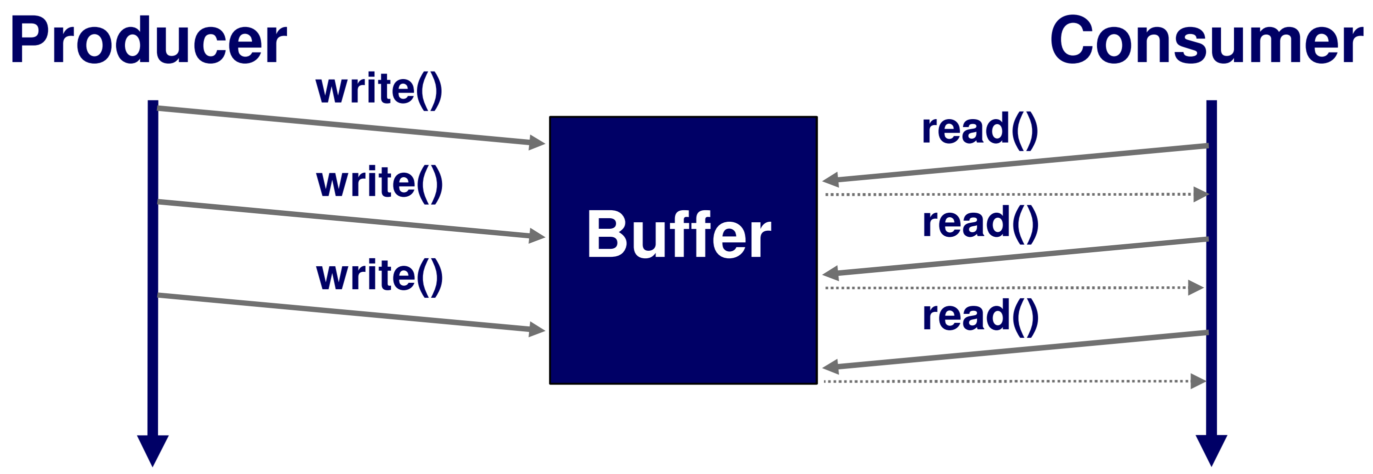
```
public void someMethod1() {
      //do something before
      synchronized(anObject) {
... }
      //do something after
}

public void someMethod2() {
      //do something before
      synchronized(anObject) {
... }
      //do something after
}
```

Mutually exclusive blocks because they use the same object instance as the lock

Questions

- Can static methods be synchronized?
- What is the lock “object”?
- What is a deadlock?
- How can a deadlock occur?

The producer/consumer example



- A producer thread constantly produces values and writes them into a **shared** buffer
- A consumer thread reads a value from the shared buffer and uses it
- Premise: <u>Every</u> value must be consumed <u>exactly once</u>
- Question: How to synchronize those two

Homework 3: The buffer interface

```
public interface Buffer {
    void write(int data) throws BufferFullException;
    int read() throws BufferEmptyException;
}
```

- And an implementation thereof (not threadsafe!):

```
public class UnsafeBuffer implements Buffer {
    private int data;
    public void write(int data) {
        this.data = data;
    }
    public int read() {
        return data;
    }
}
```

The producer

```
public class Producer implements Runnable {
    //shared instance
    private Buffer buffer;
    public Producer(Buffer buffer) {
        this.buffer = buffer; }
    public void run() {
        int counter = 0;
        while(counter < Integer.MAX_VALUE) {
        try {
            buffer.write(counter);
            System.out.println("Producer produced: " + counter);
            counter++;
           //do other work
        } catch(BufferFullException e) { /* try again next round*/}
```

The consumer

```
public class Consumer implements Runnable {
    //shared instance
    private Buffer buffer;
    public Consumer(Buffer buffer) { this.buffer = buffer; }
    public void run() {
        while(true) {
            try {
                int value = buffer.read();
                System.out.println("\t\t\tConsumer consumed: " + value);
                if(value == Integer.MAX_VALUE)
                    return;
                //do some work with the value
            } catch (BufferEmptyException e) { /*try again */}
```

Assignment 3

Verwendet Thread.sleep(long)