EXPLORE DESIGN PERFECTION

Introduction to Java

Carlos Kavka

ESTECO SpA

EXPLORE DESIGN PERFECTION

Introduction to Java

Part V - Concurrency

esteco.com

Threads

- ✓ It is possible to run concurrently different tasks called threads.
- ✓ The threads can communicate between themselves
- ✓ Their access to shared data can be synchronized
- ✓ Two implementation possibilities: extend thread or implement runnable interface

Sub-classing the Thread class

```
class CharThread extends Thread {
  char c;
  CharThread(char aChar) {
    c = aChar;
  }
  public void run() {
    while (true) {
      System.out.println(c);
      try {
        sleep(100);
      } catch (InterruptedException e) {
        System.exit(0);
      }
    }
  }
}
```

```
class TestThreads {
  public static void main(String[] args) {
    CharThread t1 = new CharThread('a');
    CharThread t2 = new CharThread('b');

    t1.start();
    t2.start();
  }
}
```

```
$ java TestThreads
a
b
a
b
...
```

Runnable interface

```
class CharThread implements Runnable {
  char c;
  CharThread(char aChar) {
    c = aChar;
  }
  public void run() {
    while (true) {
      System.out.println(c);
      try {
        Thread.sleep(100);
      } catch (InterruptedException e) {
        System.out.println("Interrupted");
      }
    }
  }
}
```

Now the class can extend other classes

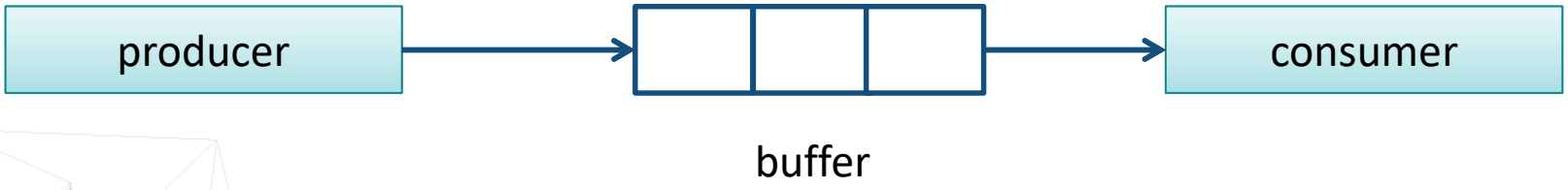
Note that sleep is not inherited any more!

An example

```
class ProducerConsumer {
  public static void main(String[] args) {
    Buffer buffer = new Buffer(10);
    Producer prod = new Producer(buffer);
    Consumer cons = new Consumer(buffer);

  prod.start();
  cons.start();
  }
}
```

The producer and the consumer are implemented with threads

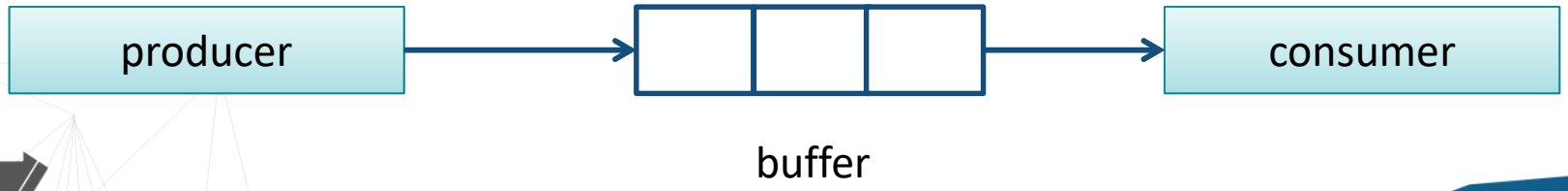
The buffer is shared between the two threads



An example: the producer and consumer

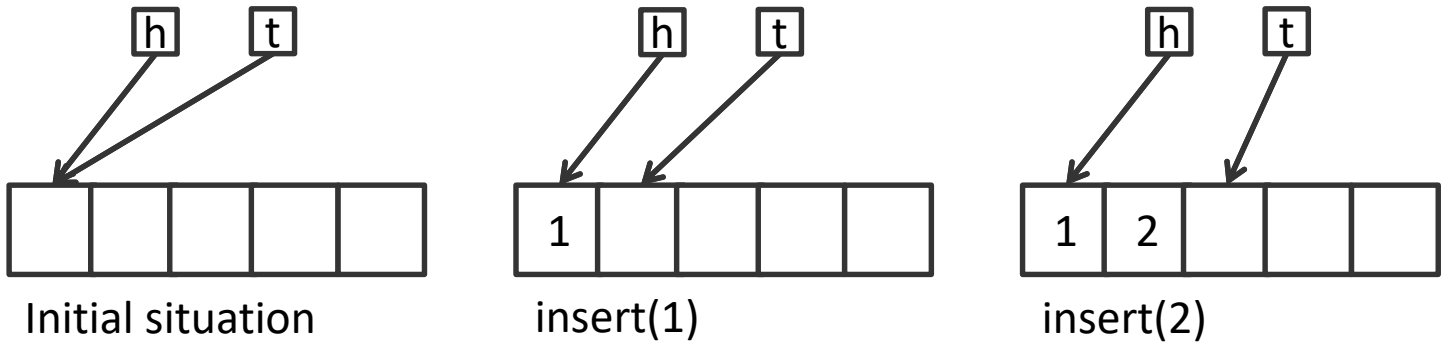
```
class Producer extends Thread {

  Buffer buffer;

  public Producer(Buffer b) {
    buffer = b;
  }

  public void run() {
    double value = 0.0;
    while (true) {
      buffer.insert(value);
      value += 0.1;
    }
  }
}
```

```
class Consumer extends Thread {

Buffer buffer;

  public Consumer(Buffer b) {
    buffer = b;
  }

  public void run() {
    while(true) {
      double element = buffer.delete()
      System.out.println(element);
    }
  }
}
```

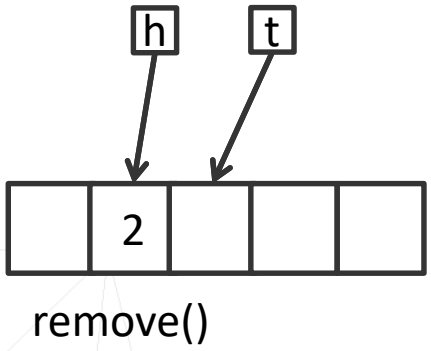


An example: the circular buffer

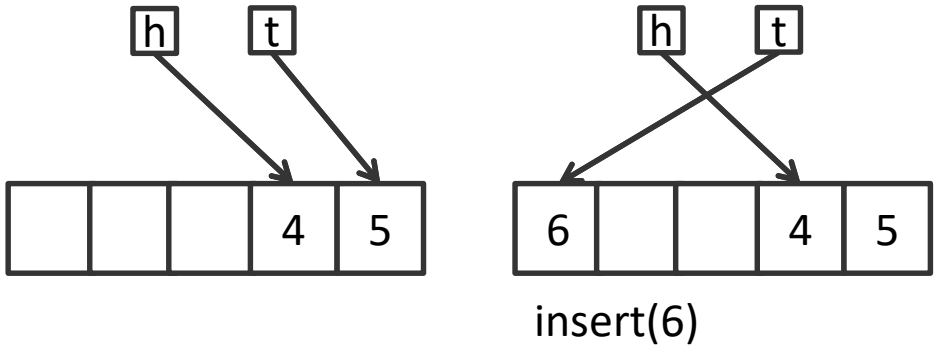
Insertion of elements in the buffer:



Remove one element:



Going beyond the limit of the buffer:



An example: the buffer

```
class Buffer {
  private double []buffer;
  private int head = 0,tail = 0,size = 0,numElements = 0;

  public Buffer(int s) {
    buffer = new double[s];
    size = s;
  }
  public void insert(double element) {
    buffer[tail] = element; tail = (tail + 1) % size;
    numElements++;
  }
  public double delete() {
    double value = buffer[head]; head = (head + 1) % size;
    numElements--;
    return value;
  }
}
```

An example: problems

However, the implementation does not work!.

- The methods insert() and delete() operate concurrently over the same structure.
- The method insert() does not check if there is at least one slot free in the buffer
- the method delete() does not check if there is at least one piece of data available in the buffer.

There is a need for synchronization

Synchronization

- ✓ Synchronized access to a critical resource can be achieved with synchronized methods
- ✓ Each instance has a lock, used to synchronize the access.
- ✓ Synchronized methods are not allowed to be executed concurrently on the same instance.

An example: synchronized methods

```
public synchronized void insert(double element) {

while (numElements == size) {
   try {
     wait();
   } catch(InterruptedException e) {
     System.out.println("Interrupted");
   }
 }

 buffer[tail] = element;
 tail = (tail + 1) % size;
 numElements++;
 notify();
}
```

The methods goes to sleep (and release the lock) if buffer is full

At the end, it awakes producer(s) which can be sleeping waiting for the lock

Synchronized access to the critical resource is achieved with a synchronized method:

An example: synchronized methods

```
public synchronized double delete() {

  while (numElements == 0) {
    try {
      wait();
    } catch(InterruptedException e) {
      System.out.println("Interrupted");
    }
  }

  double value = buffer[head];
  head = (head + 1) % size;
  numElements--;
  notify();
  return value;
}
```

Synchronized access to the critical resource is achieved with a synchronized method:

Thank you for your attention!

EXPLORE DESIGN PERFECTION