Pràctica 1. `CircularQueue`: cues circulars, segments i streams orientats al disseny de protocols de transport
Laboratori d'Aplicacions i Serveis Telemàtics

Josep Cotrina, Marcel Fernández, Jordi Forga, Juan Luis Gorricho, Francesc Oller

Introducció

Les pràctiques de l'assignatura es plantegen com a exercicis de dificultat creixent orientats al disseny de protocols de transport. En els protocols de transport es disposa d'Emissors (Senders) i Receptors (Receivers) que es comuniquen a través de cues circulars. En aquesta primera pràctica programarem i provarem una cua circular de segments de transport que anomenarem `CircularQueue<TCPSegment>`—indica que la classe `CircularQueue` és paramètrica amb `TCPSegment` (segment de protocol de transport TCP)—. En aquesta primera pràctica el Sender i el Receiver estaran modelats seqüencialment, seran fragments de codi del `main`. En les restants seran threads executant-se concurrentment, seran classes.

L'API—**A**pplication **P**rogrammer's **I**nterface— d'accés a la capa de transport des de les aplicacions és una API orientada a *streams*—seqüències de bytes—. Per transportar els bytes, el Sender els agrupa en segments—classe `TCPSegment`—que s'envien a la capa de Xarxa—representada per el `Channel`—i en el Receiver es tornen a ensamblar en seqüències de bytes. Disposarem de dues classes, `TSocketSend`—connexió de transport per emissió— amb els mètodes `sendData` i `close`, i `TSocketRecv`—connexió de transport per recepció— amb el mètode `receiveData`, per enviar i rebre aquestes seqüències de bytes. Aquests mètodes estan definits com:

- `void sendData(byte[] data, int offset, int length)` envia `length` bytes del array `data` a partir de l'index `offset` al nivell de xarxa.
- `int receiveData(byte[] data, int offset, int length)` rep un màxim de `length` bytes en l'array `data` a partir de l'index `offset`. Retorna el nombre de bytes llegits del nivell de xarxa o -1—indicador de fi de dades—si el `Sender` ha tancat l'emissió.
- `void close()` tanca el protocol en emissió. EL receptor rebrà fi de dades.

El constructor de les dues classes anteriors te com a paràmetre el `Channel`.

Farem servir la classe `ast.protocols.tcp.TCPSegment.java` per modelar els segments. Aquesta classe representa un segment TCP i es troba a la biblioteca `lib/ast-protocols-1.3.1.jar`. Disposa, entre d'altres, dels següents mètodes:

```
public class TCPSegment {
  /**
   * set transport payload to data, starting at offset and
   * lasting length bytes
   */
  void setData(byte[] data, int offset, int length) { ... }

  /**
   * return the array that backs up the transport payload
   */
  public byte[] getData() { ... }
```

```
  /**
   * return the offset where transport payload begins
   */
  public int getDataOffset() { ... }

  /**
   * return the transport payload length
   */
  public int getDataLength() { ... }
}
```

Es demana:

1 La classe `CircularQueue<E>` (`E` és el tipus d'element)

Esta en el paquet `ast.util` i ha d'implementar la següent interfície:

```
package ast.util;

import java.util.Iterator;

public interface Queue<E> extends Iterable<E> {

    /**
     * Returns the number of elements in this queue
     */
    public int size();

    /**
     * Returns true if the space currently available in this queue is greater or equal
        than the given value.
     * Note: {@code hasFree(0)} always returns {@code true}.
     * @throws IllegalArgumentException if {@code n < 0}
     */
    public boolean hasFree(int n);

    /**
     * Returns true if this queue contains no elements. Equivals to {@code size() == 0}.
     */
    public boolean empty();

    /**
     * Returns true if no space is currently available in this queue (cannot put more
        elements).
     * Equivals to {@code !hasFree(1)}.
     */
    public boolean full();

    /**
     * Retrieves, but does not remove, the head (first element) of this queue, or returns
        null if this queue is empty.
     * @return the head of this queue
     */
```

```
    public E peekFirst();

    /**
     * Retrieves, but does not remove, the tail (last element) of this queue, or returns
         null if this queue is empty.
     * @return the tail of this queue
     */
    public E peekLast();

    /**
     * Retrieves and removes the head of this queue, or throws an exception if this queue
         is empty.
     * @return the head of this queue
     * @throws IllegalStateException if this queue is empty
    */
    public E get();

    /**
     * Inserts the specified element at the tail of this queue, or throws an exception if
         this queue is full.
     * @throws IllegalStateException if this queue is full
     */
    public void put(E e);

}
```

Per aquesta primera pràctica no cal `iterator`. Així que si es vol es pot llençar una excepció en la implementació:

```
...
public class CircularQueue<E> implements Queue<E> {
  ...
  public Iterator<E> iterator() {
    throw new UnsupportedOperationException("Unsupported method iterator()");
  }
  ...
}
```

Perquè és interessant la implementació de cua circular amb capacitat limitada? Podeu trobar una referència a una implementació de cua circular. No copieu literalment sinó mireu d'entendre la implementació. *IMPORTANT:* Haurieu de provar la cua afegint un `main` amb casos de prova de tots els mètodes a la classe: invocar una seqüència de puts i gets, comprovar que llença les excepcions correctament, etc. La compilació us dona algun *warning*?, tracteu d'eliminar-lo.

Hi ha una implementació feta de les classes `ast.util.Queue` i `ast.util.CircularQueue` a `lib/ast-protocols-1.3.1.jar`. Per provar les pròpies o les de la biblioteca simplement s'han de declarar primer aquelles que es vulguin provar. Des de la consola això es fa:

- `javac ... -cp src/java/:lib/ast-protocols-1.3.1.jar ...` per provar les pròpies. Estarien a `src/java/`
- `javac ... -cp lib/ast-protocols-1.3.1.jar:src/java/ ...` per provar les de la biblioteca.

Averigueu cóm s'especifiquen les preferències dels *bytecodes* en NETBEANS o en ECLIPSE.

2 La interficie `Channel`

A les següents pràctiques s'implementaran diverses versions de `Channel`. Resulta doncs convenient definit una interfície `Channel` on hi hauran implementacions concretes d'aquesta interfície en cada pràctica.

Definiu la interfície `Channel` segons la següent plantilla:

```
package ast.practica1;

public interface Channel {
  public void send(TCPSegment seg);
  public TCPSegment receive();
}
```

Definiu la classe `QueueChannel` que implementa l'anterior interfície. Usa la cua circular. Modela el nivell de Xarxa.

3 Les classes `TSocketSend` i `TSocketRecv`

Modelen el nivell de transport. Defineixen els mètodes `sendData`, `receiveData` i `close` anteriors.

4 La classe principal `Main`

Modela el nivell d'aplicació i obeeix a la següent plantilla:

```
package ast.practica1;
...

public class Main {
  ...

  // define line to send, two different buffers send and recv, and recv
  // buffer size
  ...

  // initialize BufferedReader attached to the keyboard,
  // Channel, TSocketSend and TSocketRecv
  ...

  while (there are keyboard lines) {
    // Sender
    // obtain line bytes and send them
    ...

    // Receiver
    // read bytes and print to the console
    ...
  }

  // (optional) close send and detect EOF at Receiver
  ...
}
```

Considerarem que els bytes de les línies del teclat es transmeten en un únic segment. Considerarem que el buffer de recepció és suficientment gran per encabir els bytes del segment.

Un concepte important de qualsevol stream és seva finalització. La detecten sempre els mètodes de lectura del stream, cadascun d'una forma particular. Així `readLine` retorna `null` quan es detecta EOF—**E**nd **O**f **F**ile—. En canvi `receiveData` retorna -1. Per iniciar la finalització s'ha d'invocar el mètode `close()` programat a la classe `TSocketSend`. l'EOF es detecta en el Receiver com un segment de payload `null` i longitud 0. Per tancar un stream de teclat i per tant indicar EOF s'ha de prémer `Ctrl-D`—en UNIX—. La consola de NETBEANS no detecta EOF correctament i per tant s'haurà d'acabar amb una comanda `"fi"` per exemple, en canvi si ho detecten tant la consola de ECLIPSE com el terminal. Opcionalment i a criteri del vostre professor de Laboratori, programeu la finalització.