Análisis y diseño de software

dit
UPM

ejercicio 5

José A. Mañas

30.3.2016

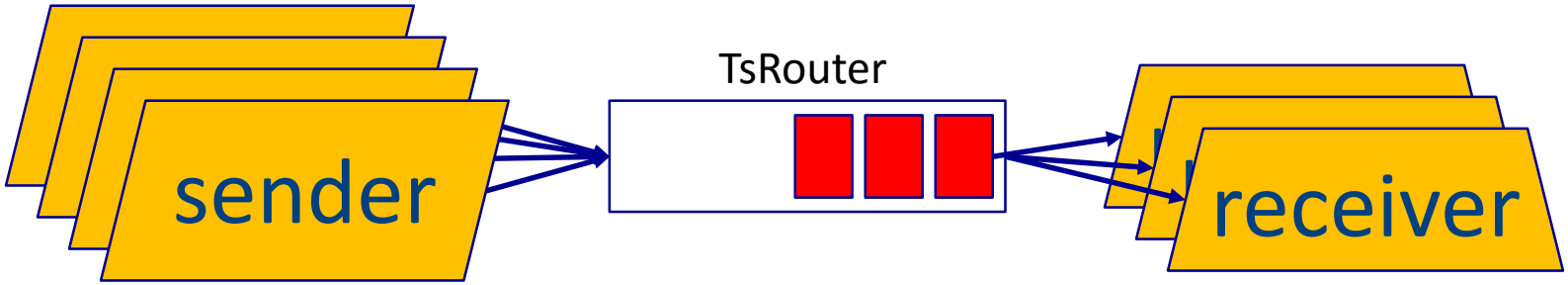
ETSIT UPM

temas

- concurrencia
  - lanzamiento de threads
  - zona de exclusión mutua
- programar y probar

tareas

- programar un thread-safe router
  - varias tareas le meten paquetes
  - varias tareas extraen paquetes
  - hay que evitar que la cola interna se corrompa



clases java

javadoc

- class Sender implements Runnable
- class Receiver implements Runnable
- class TsRouter

- class TsRouterTest
- class TsRouterSmokeTest

- class Packet
- enum Priority
- class Nap
- class Log

sender

en el método run()

1. prepara un paquete aleatorio
2. lo manda al router
   - si salta una excepción, la imprime y sigue
3. espera unos milisegundos
4. vuelve a 1

receiver

en el método run()

1. saca un paquete del router
   - si salta una excepción, la imprime y sigue
2. espera unos milisegundos
3. vuelve a 1

class TsRouter

```
/**
   * Agrega un paquete a la cola del router.
   * Si la cola está llena,
   * se hace sitio eliminando el paquete más antiguo de menor prioridad.
   *
   * @param packet el paquete a encolar.
   */
  public void send(Packet packet) {
```

```
/**
   * De los pendientes de entrega,
   * selecciona el de más prioridad y, dentro de la misma prioridad, el más antiguo.
   * Lo elimina de la cola y lo devuelve.
   *
   * @return el paquete seleccionado.
   */
  public Packet get() {
```

implementación

- el router tiene un estado interno: una cola
  - puede ser una lista:
    - private List<Packet> queue
  - puede ser un array:
    - private Packet[] queue
- sobre esta cola de paquetes hay que hacer las operaciones de envío, send(), y recepción, get(), de tal forma que la concurrencia no corrompa la estructura de datos

prueba - TsRouterSmokeTest

1. se crea un TsRouter con capacidad para 5 paquetes
2. se crean y lanzan varias threads de tipo Sender, por ejemplo 5
3. se crean y lanzan varias threads de tipo Receiver, por ejemplo 5

- si la zona exclusiva está bien, no deben saltar excepciones

pruebas unitarias: TsRouterTest

- se pueden hacer pruebas unitarias, JUnit,
  - sin concurrencia,
  - para validar que las operaciones de envío y recepción funcionan adecuadamente
  - class TsRouterTest

pruebas unitarias: TsRouterTest

```
private final Packet packetA1 = new Packet(ALTA, 1);
private final Packet packetM1 = new Packet(MEDIA, 1);
private final Packet packetM2 = new Packet(MEDIA, 2);
private final Packet packetB1 = new Packet(BAJA, 1);
```

```
@Test
public void ejemplo() {
   Packet[] seq_send = new Packet[]{
         packetM1, packetB1, packetA1, packetM2,};
   Packet[] seq_rec = new Packet[]{
         packetA1, packetM1, packetM2, packetB1,};
   TsRouter router = new TsRouter(5);
   for (Packet packet : seq_send)
      router.send(packet);
   for (Packet packet : seq_rec)
      assertSame(packet, router.get());
   assertNull(router.get());
}
```

pruebas

- corrección (correctness)
  - junit: los paquetes salen o son desechados según los criterios apuntados
- seguridad (safety)
  - el estado no se corrompe;
    un estado corrupto genera excepciones
- vivacidad (liveness)
  - entran y salen paquetes;
    el sistema no se queda congelado
- equidad (fairness)
  - todos los emisores mandan y todos los receptores reciben
  - use Log

NOTA: pasar todas las pruebas no implica que no haya errores;
pero unas buenas pruebas reducen la probabilidad de que queden.

Log: registro de actividad

- Si cada sender indica cuando manda algo y cada receiver cuando recibe algo entonces podemos observar
  - si todos están vivos
  - si se reparten el trabajo equitativamente

Methods

Modifier and Type	Method and Description
`static void`	`receiving(int id)` Le llama un Receiver cada vez que acaba de recibir un paquete.
`static void`	`sending(int id)` Le llama un Sender cada vez que acaba de enviar un paquete.

entrega

- package es.upm.dit.adsw.ej5
- todas las clases java que haya hecho
  - con los nombres que se han indicado