dit
UPM

ejercicio 6 /versión 0
(ejercicio de clase: no entregar)

José A. Mañas

10.4.2016

ETSIT UPM

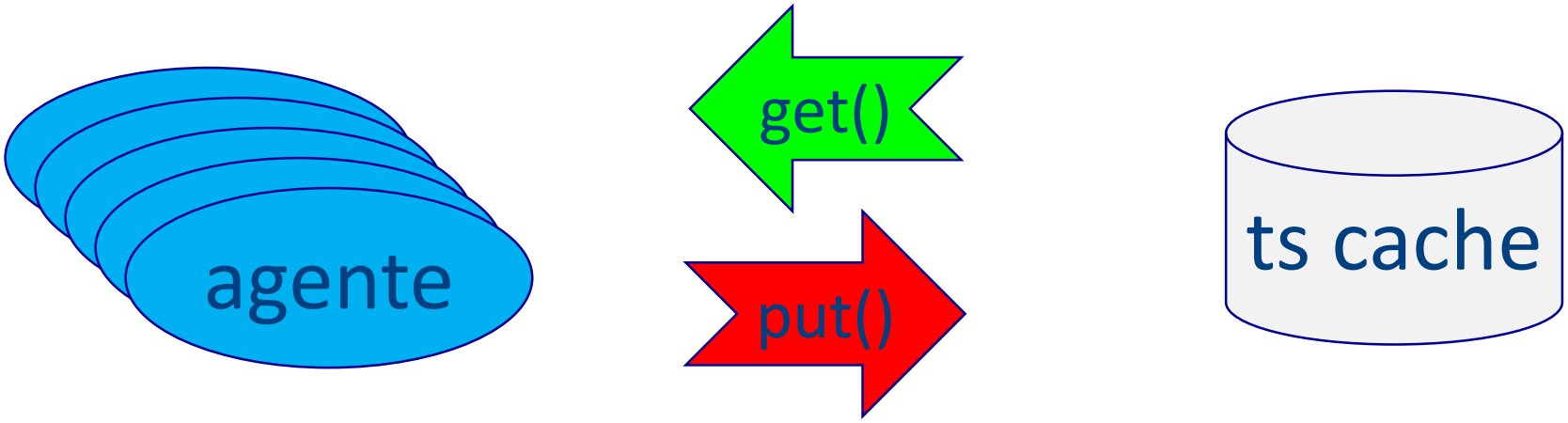
temas

- concurrencia
  - zonas de exclusión mutua
- programar y probar

tareas

- programar una cache thread-safe
  - una caché es un diccionario
    - ej. url → página web
  - las tareas miran si está en la cache
    - si está usan el valor almacenado
    - si no, descargan la página web y la almacenen
  - varias tareas recorren la web concurrentemente
    - hay que evitar que la cache se corrompa
    - hay que evitar que varias tareas lean o escriban a la vez

tareas



clases java

javadoc

- class TsCache
- class TsList

- class TsCacheSmokeTest
- class TestAgent implements Runnable

- class CV
- class Nap
- class LogViewer

agente

```
@Override
  public void run() {
     while (true) {
        try {
           String key = String.valueOf(random.nextInt(1000));
           if (cache.get(key) == null) {
              String val = "{" + key + "}";
              Nap.random(10, 20);
              cache.put(key, val);
           }
           Nap.sleep(10);
        } catch (Exception e) {
           e.printStackTrace();
        }
     }
  }
```

implementación

- TsCache es una tabla hash con listas de desbordamiento (TsList)
  - la identificación de la ranura puede hacerse concurrentemente, sin zonas de exclusión
  - cada ranura tiene su TsList
- TsList es una lista clásica
  - las operaciones get(), put(), remove() y clear() deben cuidar que no se corrompa la lista

prueba - TsCacheSmokeTest

1. se crea una TsCache con 10 ranuras
2. se lanzan 50 TestAgent

- si las zonas de exclusión están bien, no deben saltar excepciones

pruebas unitarias: TsCacheTest

- se pueden hacer pruebas unitarias, JUnit,
  - sin concurrencia,
  - para validar que las operaciones get() y put() funcionan adecuadamente
  - reuse las del ejercicio 3

pruebas

- corrección (correctness)
  - junit:
    salen o son desechados según los criterios apuntados
- seguridad (safety)
  - el estado no se corrompe;
    un estado corrupto genera excepciones
- vivacidad (liveness)
  - se guardan y se recuperan datos;
    el sistema no se queda congelado
- equidad (fairness)
  - todos los agentes progresan
  - use LogViewer

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

LogViewer

- Si cada operación pinta cuando empieza y termina, podemos observar
  - si todos están vivos
  - si se reparten el trabajo equitativamente

dump

```
public void dump(java.lang.Object id,
         int nReaders,
         int nWriters)
```

Pintor.

Parameters:

`id` - el objeto que lo llama.

`nReaders` - numero de lectores en este momento.

`nWriters` - numero de escritores en este momento.

LogViewer

- ejemplo de uso

```
public class TsList {
   private LogViewer viewer = LogViewer.getInstance();


   public String remove(String clave) {
      viewer.dump(this, 0, 1);
      ...
      try {
         ....
      } finally {
         viewer.dump(this, 0, 0);
      }
   }
```

entrega

- este ejercicio no se entrega
- el que se entrega es el ejercicio 6