Laboratorio A.E.D. — ejercicio Laboratorio 7

Guillermo Román
guillermo.roman@upm.es
Lars-Åke Fredlund
lfredlund@fi.upm.es
Manuel Carro
manuel.carro@upm.es
Marina Álvarez
marina.alvarez@upm.es
Julio García
juliomanuel.garcia@upm.es
Tonghong Li
tonghong@fi.upm.es
José Ramón Sánchez
joseramon.sanchezp@fi.upm.es

Normas

- Fechas de entrega y penalización aplicada a la puntuación obtenida:

Hasta el Lunes 18 de diciembre, 23:59 horas	0 %
Hasta el Martes 19 de diciembre, 23:59 horas	20 %
Hasta el Miércoles 20 de diciembre, 23:59 horas	40 %
Hasta el Jueves 21 de diciembre, 23:59 horas	60 %
Después la puntuación máxima será 0	

- Se comprobará plagio y se actuará sobre los detectados
- Usad las horas de tutoría para preguntar sobre programación – son oportunidades excelentes para aprender

Entrega

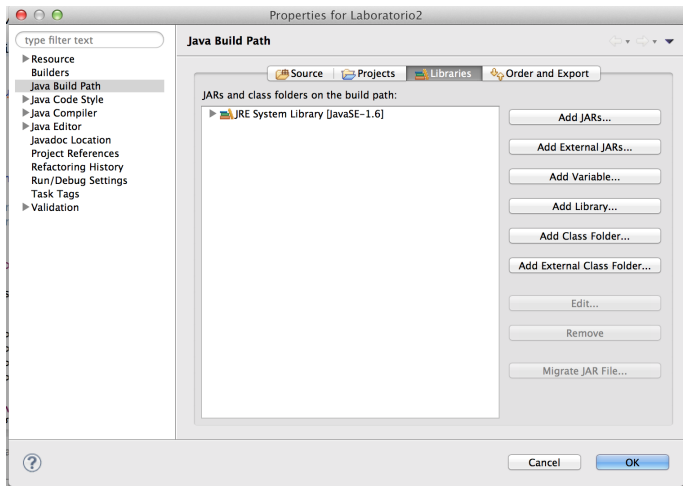
- Todos los ejercicios de laboratorio se deben entregar a través de la web `http://lml.ls.fi.upm.es/~entrega`.
- Los ficheros que hay que entregar son: `Cache.java`.

Configuración previa

- Arrancad Eclipse
- Si trabajáis en portátil, podéis utilizar cualquier versión relativamente reciente de Eclipse. Debería valer cualquier versión a partir de la versión 3.7. Es suficiente con que instaléis la *Eclipse IDE for Java Developers*
- Cambiad a "Java Perspective".
- Cread un proyecto Java llamado `aed`:
  - Seleccionad separación de directorios de fuentes y binarios
- Cread un *package* `aed.cache` en el proyecto `aed`, dentro de `src`
- Aula Virtual → AED → Laboratorios y Entregas Individuales → Laboratorio 7 → `Laboratorio7.zip`; descomprimidlo
- Contenido de `Laboratorio7.zip`:
  - `Cache.java`, `CacheCell.java`, `Storage.java`, `RunOneTest.java`, `TesterLab7.java`

Configuración previa al desarrollo del ejercicio.

- Importad al paquete `aed.cache` las fuentes que habéis descargado (`Cache.java`, `CacheCell.java`, `Storage.java`, `RunOneTest.java`, `TesterLab7.java`)
- Añadid al proyecto `aed` la librería `aedlib.jar` que tenéis en Moodle (en Laboratorios y Entregas Individuales). Para ello:
- Project → Properties → Java Build Path. Se abrirá una ventana como esta:



- Usad la opción “Add External JARs...”.
- Intentad ejecutar `TesterLab7.java`

Documentación de la librería `aedlib.jar`

- La documentación de la API de la librería aedlib.jar esta disponible en `http://lml.ls.fi.upm.es/~entrega/aed/docs/aedlib/`
- También se puede añadir la documentación de la librería a Eclipse (*no es obligatorio*): en el "Package Explorer": "Referenced Libraries" $\rightarrow$ `aedlib.jar` y elige la opción "Properties". Se abre una ventana donde se puede elegir "Javadoc Location" y ahí se pone como "javadoc location path:" `http://lml.ls.fi.upm.es/~entrega/aed/docs/aedlib/` y presionar el botón "Apply and Close"

Tarea: implementar una memoria Caché

- Una memoria caché es un complemento a otra memoria M:
  - Una caché contiene copias de algunos datos de M
  - Leer y escribir en la caché es **mucho** más rápido que en M
  - Por tanto, es preferible acceder a la caché en lugar de a M
  - Pero el tamaño de la caché es mucho menor que el de M
  - Por tanto hay que elegir qué datos se guardan en la caché
- En informática se usan memorias caché en muchas ocasiones: en procesadores (CPUs), para permitir acceso rápido a páginas web, ...

Ejemplo de caché

Memoria M

key	value
8938343U	Pamela Martínez
2377373L	José González
X3773478	Stina Karlsson
X6553279	Jim Smith
2382882I	Noelia Martín
8232839G	Clara Fernández
3727347Z	George Romero
2388282H	Francisco García
3634639U	Susana Díaz

Caché (tamaño 3)

key	value
2377373L	José González
2382882I	Noelia Martín
2388282H	Francisco García

- La caché contiene un subconjunto de los elementos (clave y valor) contenidos en la memoria M
- La pregunta esencial es: cuando la caché está llena y queremos guardar un valor nuevo en la caché, ¿qué elemento quitamos de la caché?

Implementación de la caché

Interfaz

Queremos implementar los algoritmos de una memoria caché simple con solo dos operaciones:

- acceder a un valor (get)
- recuperar un valor (put)

```
public class Cache<Key,Value> {
  // Constructor de la cache. Especifica el tamano maximo
  // y la memoria M que se va a utilizar
  public Cache(int maxSize, Storage<Key,Value> storage) { ... }

  // Devuelve el valor que corresponde a una clave "Key"
  public Value get(Key key) { ... }

  // Establece un valor nuevo para la clave en la memoria cache
  public void put(Key key, Value value) { ... }
}
```

Implementación de la caché

Memoria principal

La memoria M está implementada en la clase `Storage`

```
public class Storage<Key,Value> {
  public Value read(Key key) { ... }
  public void write(Key key, Value value) { ... }
}
```

NOTA: La implementación no es realista en términos de eficiencia, pero es funcionalmente correcta

Implementación de la caché

Política de reemplazo LRU

- Si se intenta a recuperar el valor asociado a una clave que no está en la caché (por ejemplo, con `cache.get("3727347Z")`):
  - Se accede al valor M usando `storage.read("3727347Z")`
  - Se guarda el par $\langle$clave,valor$\rangle$ en la caché.
- Si no hay espacio para un valor nuevo, ¿qué valor antiguo quitamos?
  - Vuestra implementación debe eliminar de la caché el dato con clave k que lleve maś tiempo sin ser usado, es decir, los demás datos han sido accedidos posteriormente al último acceso a k
- Esta política se llama LRU ("**L**east **R**ecently **U**sed")
- Después veremos el algoritmo concreto

Implementación de la caché

Detalles internos

- La clase Caché contiene los siguientes atributos:

```
// Para acceder a la memoria M
private Storage<Key,Value> storage;

// Un 'map' que asocia una clave con un ``CacheCell''
private Map<Key,CacheCell<Key,Value>> map;

// Un PositionList que guarda las claves en orden de
// uso -- la clave mas recientemente usado sera el lru.first()
private PositionList<Key> lru;
```

- Un `CacheCell` contiene:

```
public class CacheCell<Key,Value> {
  private Value value;          // El valor de la clave
  private boolean dirty;        // Necesita copiar al storage?

  private Position<Key> pos;    // La posicion del lru
                                // donde esta la clave

  // y getters y setters
}
```

Implementación de `cache.get(key,value)`

1. Si `key` esta en la caché (e.d., en `map`), se mueve la posición (en `lru`) con la clave `key` a la primera posición de `lru`
2. Si no está:
   - se lee el valor de `storage`,
   - se guarda la clave `key` en la primera posicíon de `lru`,
   - se crea un nuevo `CacheCell` ((apuntando a la nueva `Position` creada en `lru`)),
   - se guarda el `CacheCell` en el `map`.
   - Si la memoria caché está llena, hay que eliminar un valor antiguo:
     - se borra la entrada de `map` correspondiente a la última posicion de `lru`,
     - se borra el último nodo de `lru`,
     - si el dato a borrar está marcado como *dirty*, el valor en la memoria caché se copia en la memoria `storage`
3. Devuelve el valor solicitado

Implementación de `cache.put(key,newValue)`

1. Si `key` está en la caché (e.d., en `map`), se mueve la posición (en `lru`) con la clave `key` a la primera posición de `lru`
2. Si no está:
   - se guarda la clave `key` en la primera posicíon de `lru`,
   - se crea un nuevo `CacheCell` (apuntando a la nueva `Position` creada en `lru`),
   - se guarda el `CacheCell` en el `map`.
   - Si la memoria caché está llena, hay que eliminar un valor antiguo:
     - se borra la entrada de `map` correspondiente a la última posicion de `lru`,
     - se borra el ultimo nodo de `lru`,
     - si el dato a borrar está marcado como *dirty*, el valor en la memoria caché se copia en la memoria `storage`.
3. Se cambia el valor de la `CacheCell` asociado con `key` al `newValue`
4. Se marca que la `CacheCell` está sucia usando `setDirty` de `CacheCell`

Ejemplo

```
// <k,(v,pk,d)> significa que el map contiene un entry con clave k, y tiene
// como valor un CacheCell con value 'v', 'pk' es el Position de lru que
// contiene la clave 'k' y 'd' indica el valor de dirty

M = [<k1,v1>, <k2,v2>, <k3,v3>, <k4,v4>, <k5,v5>]
Cache c = new Cache(3,M);

c.get(k2);  // c.map = [<k2,(v2,pk2,F)>]
            // c.lru = [k2]
            // return v2

c.get(k1);  // c.map = [<k2,(v2,pk2,F)>, <k1,(v1,pk1,F)>]
            // c.lru = [k1,k2]
            // return v1

c.get(k4);  // c.map = [<k2,(v2,pk2,F)>, <k1,(v1,pk1,F)>, <k4,(v4,pk4,F)>]
            // c.lru = [k4,k1,k2]
            // return v4

c.get(k1);  // c.map = [<k2,(v2,pk2,F)>, <k1,(v1,pk1,F)>, <k4,(v4,pk4,F)>]
            // c.lru = [k1,k4,k2]
            // return v1

c.get(k5);  // c.map = [<k1,(v1,pk1,F)>, <k4,(v4,pk4,F)>, <k5,(v5,pk5,F)>]
            // c.lru = [k5,k1,k4]
            // return v5
```

Ejemplo

```
M = [<k1, v1>, <k2, v2>, <k3, v3>, <k4, v4>, <k5,v5>]
// c.map = [<k1,(v1,pk1,F)>, <k4,(v4,pk4,F)>, <k5,(v5,pk5,F)>]
// c.lru = [k5,k1,k4]

c.put(k3,v8);   // c.map = [<k1,(v1,pk1,F)>, <k3,(v8,pk3,T)>, <k5,(v5,pk5,F)>]
                // c.lru = [k3,k5,k1]
                // k3 esta dirty!!

c.get(k3);      // c.map = [<k1,(v1,pk1,F)>, <k3,(v8,pk3,T)>, <k5,(v5,pk5,F)>]
                // c.lru = [k3,k5,k1]
                // return v8

c.put(k2,v7);   // c.map = [<k2,(v7,pk2,T)>, <k3,(v8,pk3,T)>, <k5,(v5,pk5,F)>]
                // c.lru = [k2,k3,k5]
                // k3 y k2 estan dirty!!

c.get(k4);      // c.map = [<k2,(v7,pk2,T)>, <k3,(v8,pk3,T)>, <k4,(v4,pk4,F)>]
                // c.lru = [k4,k2,k3]

c.get(k1);      // c.map = [<k2,(v7,pk2,T)>, <k1,(v1,pk1,F)>, <k4,(v4,pk4,F)>]
                // c.lru = [k1,k4,k2]
                // Quitamos <k3,(v8,pk3,T)> y como estaba sucio
                // actualizamos M llevando v8 a la clave k3
                // M = [<k1, v1>, <k2, v2>, <k3, v8>, <k4, v4>, <k5,v5>]
```

Notas

- Se puede asumir que el tamaño de la memoria caché es ≥ 2
- El proyecto debe compilar sin errores y debe cumplirse la especificación de los métodos a completar
- Debe ejecutar `TesterLab7.java` correctamente y sin mensajes de error
  - Nota: una ejecución sin mensajes de error no significa que el método sea correcto (es decir, que funcione bien para cada posible entrada)
- Todos los ejercicios se comprueban manualmente antes de dar la nota final