P5 - ADSOF - 2271

Daniel Birsan y Juan José Martínez Domínguez

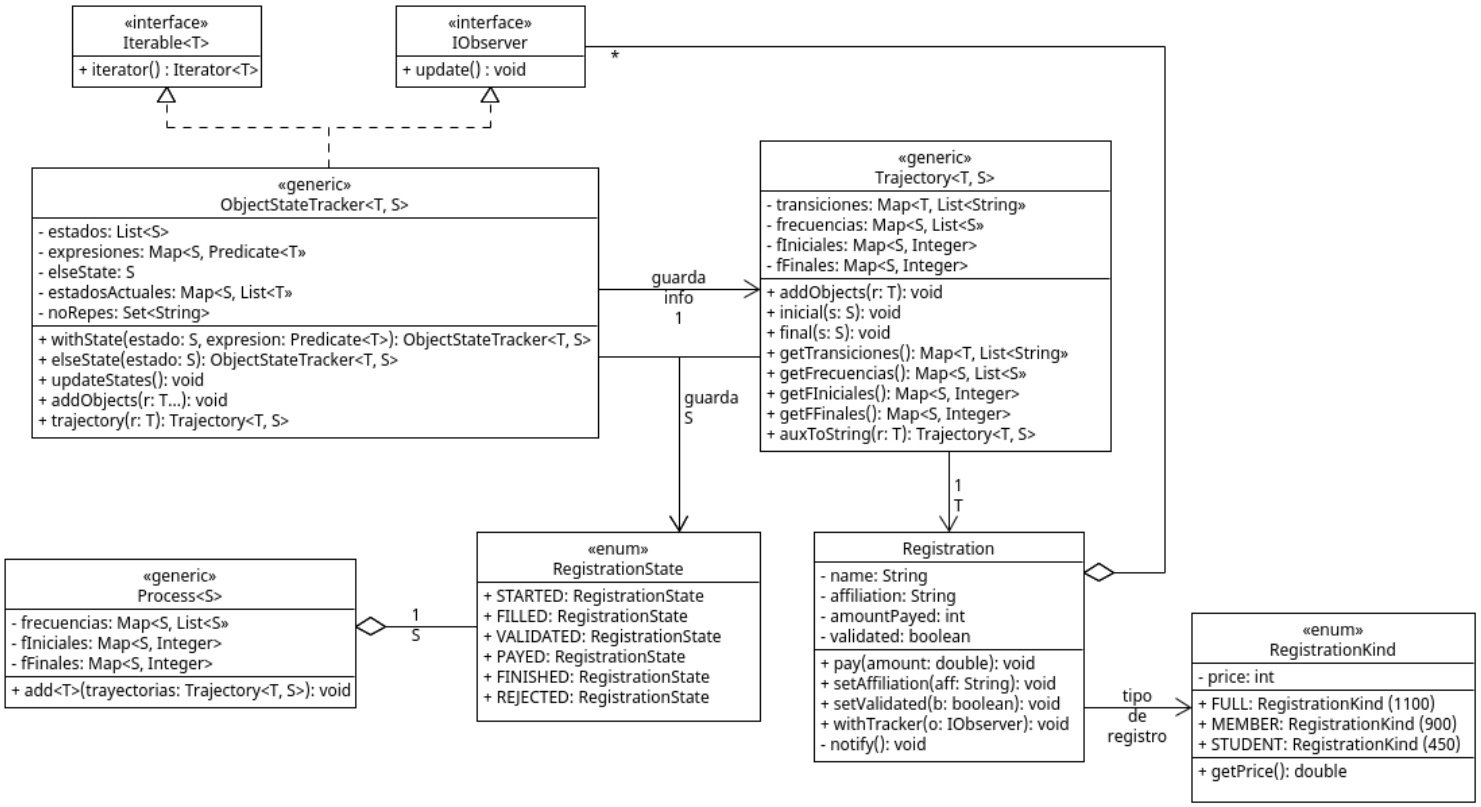


Diagrama de clases

Apartado 1

En este apartado empezamos con muchas dudas y problemas a la hora de recibir expresiones lambda por parámetro, con ayuda descubrimos que hay que usar la interfaz Predicate. Con esto asentado buscamos la forma de almacenar los estados en orden natural, decidimos usar una lista ya que esta mantiene el orden de inserción, sin embargo, tuvimos problemas a la hora de hacer esto porque no estábamos imprimiendo los valores correctos, con esto solucionado buscamos en las diapositivas una estructura que almacene pares clave-valor por orden de inserción, por eso, usamos LinkedHashMap.
El método updateStates nos costó pensarlo ya que teníamos que recorrer varias estructuras a la vez y evaluar la expresión lambda para comprobar si hay que actualizar el estado, pensábamos que lo habíamos conseguido pero tuvimos problemas al modificar el propio mapa que estábamos recorriendo, por eso, creamos un mapa auxiliar que después copiamos en el atributo de la clase.

```
❯  /usr/bin/env /usr/lib/jvm/java-11-openjdk-amd64/bin/java -cp /home/dani/.config/Co
de/User/workspaceStorage/eedb388c1380d4ee4421fcd4e2403941/redhat.java/jdt ws/P5 68649
cb5/bin testers.TesterStateChanges
{STARTED=[Reg. of: Ann Smith, Reg. of: John Doe, Reg. of: Lisa Martin], FILLED=[], VA
LIDATED=[], PAYED=[], FINISHED=[], REJECTED=[]}
{STARTED=[Reg. of: Lisa Martin], FILLED=[Reg. of: Ann Smith], VALIDATED=[], PAYED=[Re
g. of: John Doe], FINISHED=[], REJECTED=[]}
  ~/E/ADSOF/P5  main ?1                                                          ✔
```

Hemos creado una pequeña extensión al test que se nos proporciona, este simplemente modifica más estados y crea un nuevo registro para comprobar el funcionamiento del método elseState, para crear esta última prueba, tuvimos que pensar de qué forma podíamos hacer que no se cumpla ninguna expresión lambda. La forma que encontramos es validar un pago que se ha realizado por un importe incorrecto, de esta forma, al actualizar los estados, se rechaza.

```
❯  /usr/bin/env /usr/lib/jvm/java-11-openjdk-amd64/bin/java -cp /home/dani/.config/Co
de/User/workspaceStorage/eedb388c1380d4ee4421fcd4e2403941/redhat.java/jdt ws/P5 68649
cb5/bin mytests.Test1
{STARTED=[Reg. of: Ann Smith, Reg. of: John Doe, Reg. of: Lisa Martin, Reg. of: Rejec
ted], FILLED=[], VALIDATED=[], PAYED=[], FINISHED=[], REJECTED=[]}
{STARTED=[Reg. of: Lisa Martin, Reg. of: Rejected], FILLED=[Reg. of: Ann Smith], VALI
DATED=[], PAYED=[Reg. of: John Doe], FINISHED=[], REJECTED=[]}
{STARTED=[Reg. of: Rejected], FILLED=[Reg. of: Ann Smith], VALIDATED=[], PAYED=[Reg.
of: Lisa Martin], FINISHED=[Reg. of: John Doe], REJECTED=[]}
{STARTED=[], FILLED=[Reg. of: Ann Smith], VALIDATED=[], PAYED=[], FINISHED=[Reg. of:
Lisa Martin, Reg. of: John Doe], REJECTED=[Reg. of: Rejected]}
  ~/E/ADSOF/P5  main ?1                                                          ✔
```

Apartado 2

Este apartado por muy corto que fuese nos provocó dudas porque el enunciado nos pedía que modificasemos la clase Registration y ObjectStateTracker si fuese necesario, sin embargo, la única forma que hemos conseguido que funcione para este apartado es un Set de Strings (porque un registro se considera igual si los nombres son iguales), de forma que, al añadir un objeto, si ya está presente en el Set se ignora, por el lado contrario, se añade al Set y se procede con el registro.

```
❯  /usr/bin/env /usr/lib/jvm/java-11-openjdk-amd64/bin/java -cp /home/d
ani/.config/Code/User/workspaceStorage/eedb388c1380d4ee4421fcd4e2403941
/redhat.java/jdt ws/P5 68649cb5/bin testers.TesterRepeatedObjects
{STARTED=[Reg. of: Ann Smith, Reg. of: John Doe, Reg. of: Lisa Martin],
 FILLED=[], VALIDATED=[], PAYED=[], FINISHED=[], REJECTED=[]}
  ~/E/ADSOF/P5  main ?1                                          ✓
```

Apartado 3

Viendo el tester decidimos hacer que el método trajectories devuelva una Lista de Strings para que a la hora del formato se imprima automáticamente el formato [,,...]. Implementamos esto relativamente fácil, creando un map que relacionaba cada registro con su lista de trayectorias en el formato descrito antes:

```java
public class ObjectStateTracker<T, S extends Comparable<S>> implements Iterable<T> {
    private List<S> estados = new ArrayList<>();
    private Map<S, Predicate<T>> expresiones;
    private S elseState;
    private Map<S ,List<T>> estadosActuales;
    private Set<String> noRepes;
    private Map<T, List<String>> trayectoria;
```

Vemos el map de Registro-Lista de strings.

```java
if (!estadoAntiguo.equals(estado))
    this.trayectoria.get(r).add("(from: " + estadoAntiguo + " to " + estado + " at: " + LocalDateTime.now() + ")");


tadosActuales = nuevosEstadosActuales;


addObjects(T... r) {
reg : r) {
(this.noRepes.contains(reg.toString())) {
continue;
lse {
List<T> antiguosIniciales = this.estadosActuales.get(this.estados.get(0));
antiguosIniciales.add(reg);
this.estadosActuales.put(this.estados.get(0), antiguosIniciales);
List<String> trayectoriaIncial = new ArrayList<>();
trayectoriaIncial.add("(in: " + this.estados.get(0) + " at: " + LocalDateTime.now() + ")");
this.trayectoria.put(reg, trayectoriaIncial);
this.noRepes.add(reg.toString());
```

Vemos cómo actualizamos la lista de strings, dependiendo de la acción realizada.

Esta implementación nos generará dudas más adelante y optaremos por una más adecuada que no limite tanto la genericidad del programa.

```
❯  /usr/bin/env /usr/lib/jvm/java-11-openjdk-amd64/bin/java -cp
 /home/dani/.config/Code/User/workspaceStorage/eedb388c1380d4ee
4421fcd4e2403941/redhat.java/jdt ws/P5 68649cb5/bin testers.Tes
terTrajectories
Reg. of: Ann Smith: [(in: STARTED at: 2024-05-04T01:13:47.58967
9), (from: STARTED to FILLED at: 2024-05-04T01:13:47.622546)]
Reg. of: John Doe: [(in: STARTED at: 2024-05-04T01:13:47.622216
), (from: STARTED to PAYED at: 2024-05-04T01:13:47.645359), (fr
om: PAYED to FINISHED at: 2024-05-04T01:13:47.645500)]
Reg. of: Lisa Martin: [(in: STARTED at: 2024-05-04T01:13:47.622
312), (from: STARTED to FILLED at: 2024-05-04T01:13:47.645446)]
 ~/E/ADSOF/P5  main ?1                                      ✔
```

Implementamos, al igual que en el 1, una expansión del test que crea más actualizaciones y por tanto habrá más trayectorias.

```
❯  /usr/bin/env /usr/lib/jvm/java-11-openjdk-amd64/bin/java -cp
 /home/dani/.config/Code/User/workspaceStorage/eedb388c1380d4ee
4421fcd4e2403941/redhat.java/jdt ws/P5 68649cb5/bin mytests.Tes
t3
Reg. of: Ann Smith: [(in: STARTED at: 2024-05-04T01:15:28.02098
3), (from: STARTED to FILLED at: 2024-05-04T01:15:28.046456), (
from: FILLED to PAYED at: 2024-05-04T01:15:28.074056), (from: P
AYED to FINISHED at: 2024-05-04T01:15:28.074155)]
Reg. of: John Doe: [(in: STARTED at: 2024-05-04T01:15:28.046049
), (from: STARTED to FINISHED at: 2024-05-04T01:15:28.073944),
(from: FINISHED to REJECTED at: 2024-05-04T01:15:28.074095)]
Reg. of: Lisa Martin: [(in: STARTED at: 2024-05-04T01:15:28.046
170), (from: STARTED to FILLED at: 2024-05-04T01:15:28.074003)]
 ~/E/ADSOF/P5  main ?1                                      ✔
```

Apartado 4

Lo primero que hicimos fue implementar iterator(), tras una breve investigación, concluimos que este método debe devolver algo sobre lo que iterar, en nuestro caso, queríamos iterar sobre los registros, y como todos los registros tienen una trayectoria ligada, decidimos devolver el iterador de las claves del par registro-strings;

Como hemos comentado antes, vamos a cambiar la implementación del apartado 3 optando por crear una nueva clase genérica Trajectory parametrizada con el registro y los estados igual que ObjectStateTracker. Aunque primero, implementamos el apartado 4 según hicimos el 3, es decir, nuestro método add en Process recibiría la lista de strings que contenía todas las trayectorias asociadas a un registro y analizamos las cadenas para obtener la información que necesitábamos para crear el proceso:

```java
public void add(List<String> trayectorias) {
    int i = 0;
    for (String t : trayectorias) {
        if (!t.startsWith("(in:")) {
            for (S s : this.estados) {
                for (S s2 : this.estados) {
                    if (t.split("to")[0].contains(s.toString()) && t.split("to")[1].contains(s2.toString())) {
                        if (this.frecuencias.get(s) != null) {
                            this.frecuencias.get(s).add(s2);
                        } else {
                            this.frecuencias.get(s).add(s2);
                        }
                    }
                }
            }
        } else {
            for (S s3 : this.estados) {
                if (t.contains(s3.toString())) {
                    this.fIniciales.put(s3, this.fIniciales.get(s3) + 1);
                }
            }
        }
        if (i == trayectorias.size() - 1) {
            for (S s3 : this.estados) {
                if (t.split("to")[1].contains(s3.toString())) {
                    this.fFinales.put(s3, this.fFinales.get(s3) + 1);
                }
            }
        }
        i++;
    }
}
```

Como podemos deducir, la clase Process tiene entre los atributos 3 mapas que almacenan, las transiciones, las veces que un estado ha sido inicial y final. El toString sigue el formato que se nos pide y usamos el método que se nos presenta en la diapositiva 58 del 2.7 para contar las veces que un objeto, en este caso un estado, aparece en una colección, en este caso la lista de transiciones. Con Collections.frequency('lista de transiciones de cada estado', 'cada estado'), podemos obtener los datos que se piden.
El toString y los atributos se mantendrán para la futura implementación.

Como el método add no cuadraba para la genericidad debido a que si los estados no fuesen una enum sino una clase, muy probablemente la implementación fallaría, por ello, creamos la clase genérica Trajectory parametrizada igual que ObjectStateTracker, en esta nueva clase, almacenamos todas las cadenas para que el 3 siga funcionando, pero también la información que necesitamos en el 4 sin extraerla directamente desde una cadena, por ello, los atributos de Process y Trajectory son muy parecidos.

Ahora el problema surgía al dejar de devolver una lista de Strings en el método trajectory() de ObjectStateTracker, puesto que la salida ya no era correcta, para afrontar esto, creamos un "falso" porque nos dimos cuenta que se llamaba al toString iterativamente, entonces, queríamos que para cada iteración se devolviera el valor asociado a ese registro en formato string, para ello, creamos un atributo y cuando se llama a trajectory() se establece, de forma que al llamar a toString se imprime solo la información relacionada a ese registro.

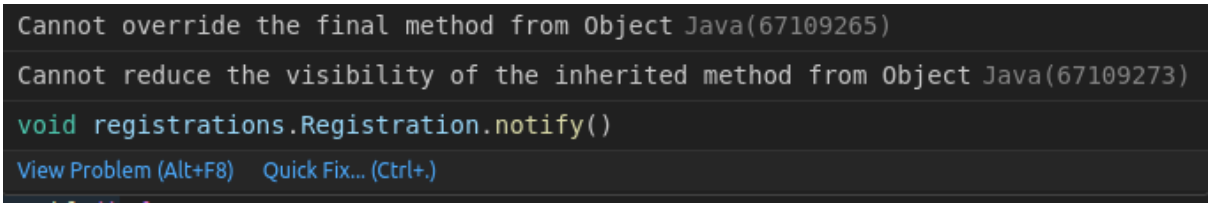
```
❯  /usr/bin/env /usr/lib/jvm/java-11-openjdk-amd64/bin/java -cp
 /home/dani/.config/Code/User/workspaceStorage/eedb388c1380d4ee
4421fcd4e2403941/redhat.java/jdt_ws/P5 68649cb5/bin testers.Tes
terProcess
Reg. of: Ann Smith: [(in: STARTED at: 2024-05-04T01:17:01.54809
5), (from: STARTED to FILLED at: 2024-05-04T01:17:01.578962)]
Reg. of: John Doe: [(in: STARTED at: 2024-05-04T01:17:01.578677
), (from: STARTED to PAYED at: 2024-05-04T01:17:01.611172), (fr
om: PAYED to FINISHED at: 2024-05-04T01:17:01.611367)]
Reg. of: Lisa Martin: [(in: STARTED at: 2024-05-04T01:17:01.578
752), (from: STARTED to FILLED at: 2024-05-04T01:17:01.611289),
 (from: FILLED to VALIDATED at: 2024-05-04T01:17:01.611431), (f
rom: VALIDATED to FINISHED at: 2024-05-04T01:17:01.611514)]
STARTED (initial 3 times, final 0 times):
FILLED (initial 0 times, final 1 times):
VALIDATED (initial 0 times, final 0 times):
PAYED (initial 0 times, final 0 times):
FINISHED (initial 0 times, final 2 times):
REJECTED (initial 0 times, final 0 times):

 ~/E/ADSOF/P5  main ?1                                        ✔
```

Apartado 5

Después de leer las diapositivas y vernos el vídeo relacionado al patrón Observer para entender su lógica y funcionamiento, decidimos que el Subject del patrón de ejemplo será nuestro Registration, después, nos damos cuenta que el Observador podría ser una Interfaz (IObserver) ya que no necesitaremos el constructor en nuestra implementación. Esta interfaz, define el método abstracto update() típico de este patrón. También decidimos que ObjectStateTracker va a implementar el observador para cumplir que puede haber observadores de diferentes implementaciones.

Con todo decidido empezamos a codificar basándonos en las diapositivas 78 y 79 del tema 3, de esta forma, hacemos que el método withTracker reciba una implementacion de IObserver y se añada a su lista de observadores (porque son a nivel de clase como se dice en el enunciado), al intentar implementar la notificación:



Por lo que le cambiamos el nombre a notifyy(), ahora cada set en Registration deberá llamar a este método para notificar a los observadores de la actualización de su estado.

Ahora queda implementar el método update definido en la interfaz de observadores, en este caso, en ObjectStateTracker usamos el mismo método que el que definimos en los apartados anteriores, updateStates, ya que la funcionalidad es la misma.