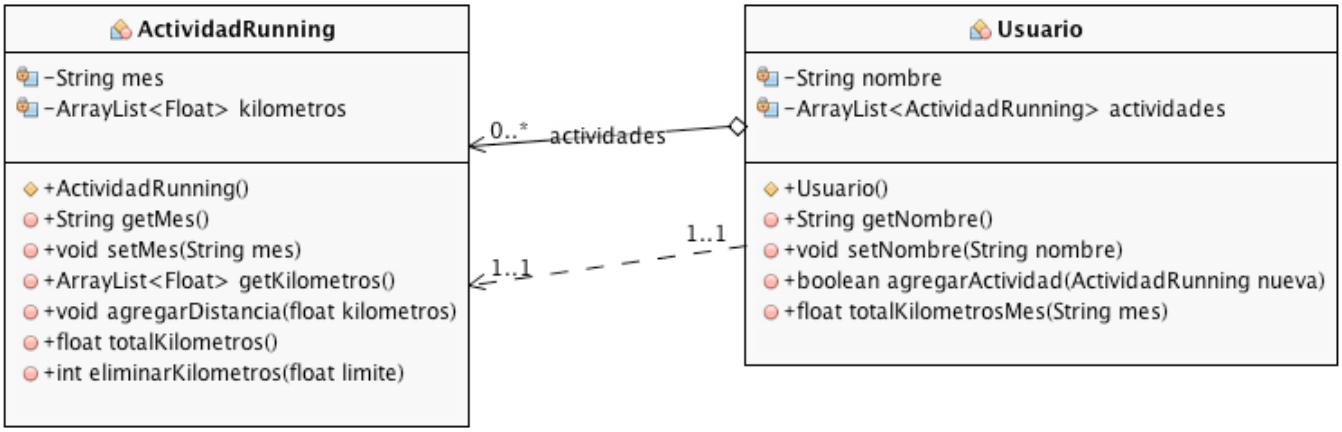
SOLUCIÓN EVALUACIÓN N° 1 - PRÁCTICA
PROGRAMACIÓN ORIENTADA A OBJETOS

Docente Jazna Meza Hidalgo
Fecha. Noviembre 2018

FORMA A

EJERCICIO – IMPLEMENTACIÓN DE CLASES

DIAGRAMA DE CLASES



CONTEXTO DEL NEGOCIO

Una actividad de running contiene todos los kilómetros (distancias) que se corren durante un determinado mes. Los usuarios registran actividades de running una por cada mes.

REGLAS DE NEGOCIO

- Los kilómetros recorridos (distancias) debe ser un valor mayor a cero.
- El nombre del mes debe ser válido considerando el nombre del mes (Enero, Febrero, …)
- Se permite solo una actividad de running por mes para un usuario.

MÉTODOS ESPECIALIDADOS

MÉTODO	DESCRIPCIÓN	VALOR DE RETORNO
agregarActividad(ActividadRunning nueva)	Permite agregar una actividad al usuario	True en caso de que logre agregar la actividad (mes no repetido) y false en caso contrario.
totalKilometros(String mes)	Calcula el total de kilómetros corridos durante el mes consultado.	Retorna el total de kilómetros.
agregarDistancia(float kilometros)	Agrega kilómetros recorridos a la actividad del mes.	Sin valor de retorno
totalKilometros()	Calcular el total de kilómetros de la actividad.	Retorna el total de kilómetros corridos durante la actividad.
eliminarKilometros(float limite)	Elimina todos las distancias recorridas menores al límite indicado.	Retorna el total de distancias eliminadas.

REQUERIMIENTOS

Considerando el diagrama de clases y las reglas de negocio, se pide:

- Implementar las clases del diagrama
- Implementar la clase de prueba MainE1A que permita cumplir con los requerimientos:
  - Crea un usuario y asignarle el nombre del estudiante que está rindiendo la evaluación.
  - Agregar las siguientes actividades de running:
    - Octubre, corre 2 veces: 2.1 kilómetros y 1.7 kilómetros.
    - Noviembre, corre 2 veces: 1.5 y 0.9 kilómetros.
  - Agregar las actividades de running al usuario creado.
  - Mostrar en pantalla, usando el método especializado, el total de kilómetros que el usuario corrió en Noviembre.
  - Eliminar, usando el método especializado, todas las distancias recorridas en el mes de Noviembre que sean inferiores a 1.0 kilómetros.
  - Mostrar en pantalla el número de distancias eliminadas.
  - Mostrar en pantalla, usando el método especializado, el total de kilómetros recorridos en Noviembre (para comprobar que funciona la eliminación).

NOTA. Recuerde que debe respetar el diagrama de clases que se entrega.

SOLUCIÓN FORMA A

```java
package e1;

import java.util.ArrayList;
import java.util.Arrays;

/**
 *
 * @author Jazna
 */
public class ActividadRunning {
    private String mes;
    private ArrayList<Float> kilometros;

    public ActividadRunning(){
        this.kilometros = new ArrayList<>();
    }

    public String getMes() {
        return mes;
    }

    public void setMes(String mes) {
        String meses[] = {"Enero","Febrero","Marzo","Abril","Mayo","Junio","Julio","Agosto",
            "Septiembre","Octubre","Noviembre","Diciembre"};
        if (Arrays.asList(meses).contains(mes)){
            this.mes = mes;
        }
    }
    public ArrayList<Float> getKilometros() {
        return kilometros;
    }

    public void agregarDistancia(float kilometros){
        if (kilometros > 0){
            this.kilometros.add(kilometros);
        }
    }

    public float totalKilometros(){
        float total = 0;
        for(float d : this.kilometros){
            total+=d;
        }
        return total;
    }
```

```
53      /**
54       * Elimina los kilómetros menores al límite
55       * @param limite, parámetro para eliminar
56       * @return total de kilómetros eliminados
57       */
58      public int eliminarKilometros(float limite){
59          ArrayList victimas = new ArrayList();
            for(float d : this.kilometros){
61              if (d < limite){
62                  victimas.add(d);
63              }
64          }
65          this.kilometros.removeAll(victimas);
66          return victimas.size();
67      }
68  }
```

```
6   package e1;
7
8   import java.util.ArrayList;
9
10  /**
11   *
12   * @author Jazna
13   */
14  public class Usuario {
15      private String nombre;
        private ArrayList<ActividadRunning> actividades;
17
18      public Usuario(){
19          this.actividades = new ArrayList<>();
20      }
21
22      public String getNombre() {
23          return nombre;
24      }
25
26      public void setNombre(String nombre) {
27          this.nombre = nombre;
28      }
```

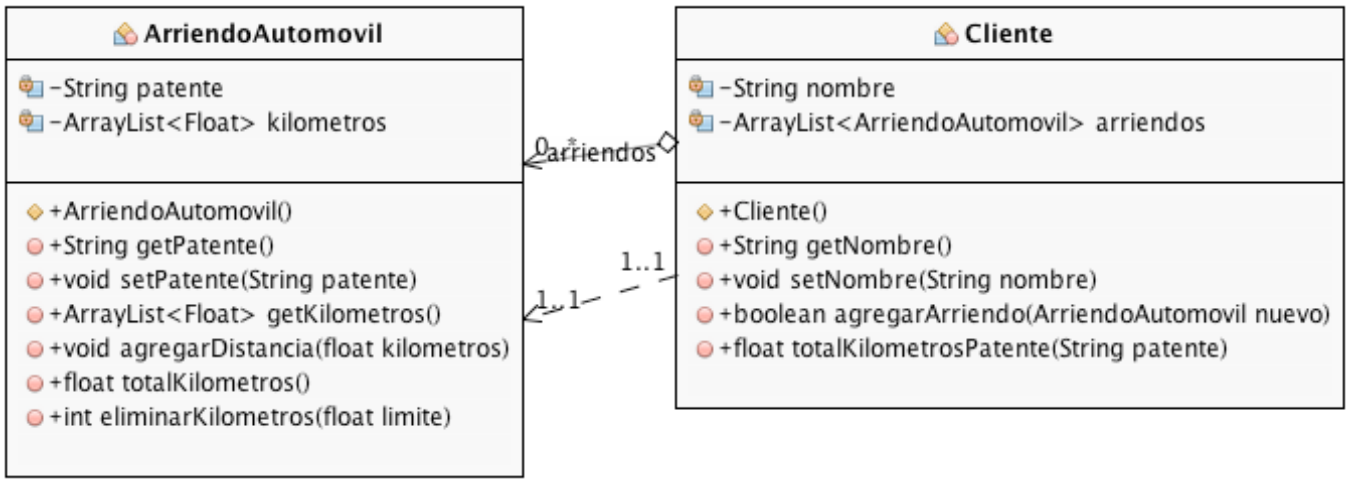
```
30      /**
31       * Agrega una nueva actividad a la lista considerando que NO puede ver 2
32       * actividades con el mismo mes
33       * @param nueva, nuevo objeto que pretende insertar
34       * @return true en caso de inserción exitosa y false en caso contrario
35       */
36      public boolean agregarActividad(ActividadRunning nueva){
38          for(ActividadRunning a : this.actividades){
39              if (a.getMes().equalsIgnoreCase(nueva.getMes())){
40                  return false;
41              }
42          }
43          this.actividades.add(nueva);
44          return true;
45      }
46
47      public float totalKilometrosMes(String mes){
48          for(ActividadRunning a : this.actividades){
49              if (a.getMes().equalsIgnoreCase(mes)){
50                  return a.totalKilometros();
51              }
52          }
53          return 0;
54      }
  }
```

```
6   package e1;
7
8   /**
9    *
10   * @author Jazna
11   */
12  public class MainE1A {
13      public static void main(String[] args) {
14          Usuario u = new Usuario();
15          u.setNombre("Carla");
16          ActividadRunning ax = new ActividadRunning();
17          ax.setMes("Noviembre");
18          ax.agregarDistancia(1.5f); ax.agregarDistancia(0.9f);
19          u.agregarActividad(ax);
20          ActividadRunning ay = new ActividadRunning();
21          ay.setMes("Octubre");
22          ay.agregarDistancia(2.1f); ay.agregarDistancia(1.7f);
23          u.agregarActividad(ay);
24
25          System.out.println(u.totalKilometrosMes("Octubre"));
26          System.out.println("Antes de eliminar : " + u.totalKilometrosMes("Noviembre"));
27          System.out.println("Elimina " + ax.eliminarKilometros(10.0f) + " distancias");
28          System.out.println("Despues de eliminar : " + u.totalKilometrosMes("Noviembre"));
29
30      }
31  }
```

FORMA B

EJERCICIO – IMPLEMENTACIÓN DE CLASES

DIAGRAMA DE CLASES



CONTEXTO DEL NEGOCIO

Un arriendo de automóvil contiene todos los kilómetros (distancias) que se utiliza en un determinado automóvil con determinada patente. Los clientes registran arriendos de automóviles, pero no se permiten 2 arriendos con la misma patente, en ese caso se debe agregar los kilómetros a la lista de kilómetros existentes.

REGLAS DE NEGOCIO

- Los kilómetros recorridos (distancias) debe ser un valor mayor a cero.
- La patente debe terminar con 2 dígitos.
- Se permite solo un arriendo de automóvil por patente para un cliente.

MÉTODOS ESPECIALIDADOS

MÉTODO	DESCRIPCIÓN	VALOR DE RETORNO
agregarArriendo(ArriendoAutomovil nuevo)	Permite agregar un arriendo al cliente	True en caso de que logre agregar el arriendo(patente no repetida) y false en caso contrario.
totalKilometrosPatente(String patente)	Calcula el total de kilómetros recorridos por el cliente con una determinada patente.	Retorna el total de kilómetros.
agregarDistancia(float kilometros)	Agrega kilómetros recorridos a la patente asociada al arriendo.	Sin valor de retorno
totalKilometros()	Calcular el total de kilómetros del arriendo.	Retorna el total de kilómetros corridos durante el arriendo.
eliminarKilometros(float limite)	Elimina todos las distancias recorridas menores al límite indicado.	Retorna el total de distancias eliminadas.

REQUERIMIENTOS

Considerando el diagrama de clases y las reglas de negocio, se pide:

- Implementar las clases del diagrama
- Implementar la clase de prueba MainE1B que permita cumplir con los requerimientos:
  - Crea un cliente y asignarle el nombre del estudiante que está rindiendo la evaluación.
  - Crear los siguientes arriendos de automóviles:
    - CXPR12, es arrendado 2 veces y se recorren: 34.6 kilómetros y 50.3 kilómetros.
    - FGRT29, es arrendado 2 veces y se recorren: 246.7 y 120.7 kilómetros.
  - Agregar los arriendos al cliente creado.
  - Mostrar en pantalla, usando el método especializado, el total de kilómetros que el usuario recorrió con el vehículo patente FGRT29.
  - Eliminar, usando el método especializado, todas las distancias recorridas en los arriendos con el vehículo patente FGRT29 que sean inferiores a 150 kilómetros.
  - Mostrar en pantalla el número de distancias eliminadas.
  - Mostrar en pantalla, usando el método especializado, el total de kilómetros recorridos durante los arriendos del vehículo patente FGRT29 (para comprobar que funciona la eliminación).

NOTA. Recuerde que debe respetar el diagrama de clases que se entrega.

SOLUCIÓN FORMA B

```java
package e1;

import java.util.ArrayList;
import java.util.Arrays;

/**
 *
 * @author Jazna
 */
public class ArriendoAutomovil {
    private String patente;
    private ArrayList<Float> kilometros;

    public ArriendoAutomovil(){
        this.kilometros = new ArrayList();
    }

    public String getPatente() {
        return patente;
    }

    public void setPatente(String patente) {
        String digitos[] = {"0","1","2","3","4","5","6","7","8","9"};
        String ultimos = patente.substring(patente.length()-2);
        String ultimo = ultimos.substring(1, 2);
        String penultimo = ultimos.substring(0, 1);
        if (Arrays.asList(digitos).contains(ultimo) && Arrays.asList(digitos).contains(penultimo)){
            this.patente = patente;
        }
    }
    public ArrayList<Float> getKilometros() {
        return kilometros;
    }

    public void agregarDistancia(float kilometros){
        if (kilometros > 0){
            this.kilometros.add(kilometros);
        }
    }

    public float totalKilometros(){
        float total = 0;
        for(float d : this.kilometros){
            total+=d;
        }
        return total;
    }
```

```
55      /**
56       * Elimina las distancias de la lista inferiores al límite
57       * @param limite, parámetro de eliminación
58       * @return total de distancias eliminadas de la lista
59       */
60      public int eliminarKilometros(float limite){
61          ArrayList victimas = new ArrayList();
            for(float d : this.kilometros){
63              if (d < limite){
64                  victimas.add(d);
65              }
66          }
67          this.kilometros.removeAll(victimas);
68          return victimas.size();
69      }
70  }
```

```
6   package e1;
7
8   import java.util.ArrayList;
9
10  /**
11   *
12   * @author Jazna
13   */
14  public class Cliente {
15      private String nombre;
        private ArrayList<ArriendoAutomovil> arriendos;
17
18      public Cliente(){
19          this.arriendos = new ArrayList<>();
20      }
21    public String getNombre() {
22          return nombre;
23      }
24
25      public void setNombre(String nombre) {
26          this.nombre = nombre;
27      }
```

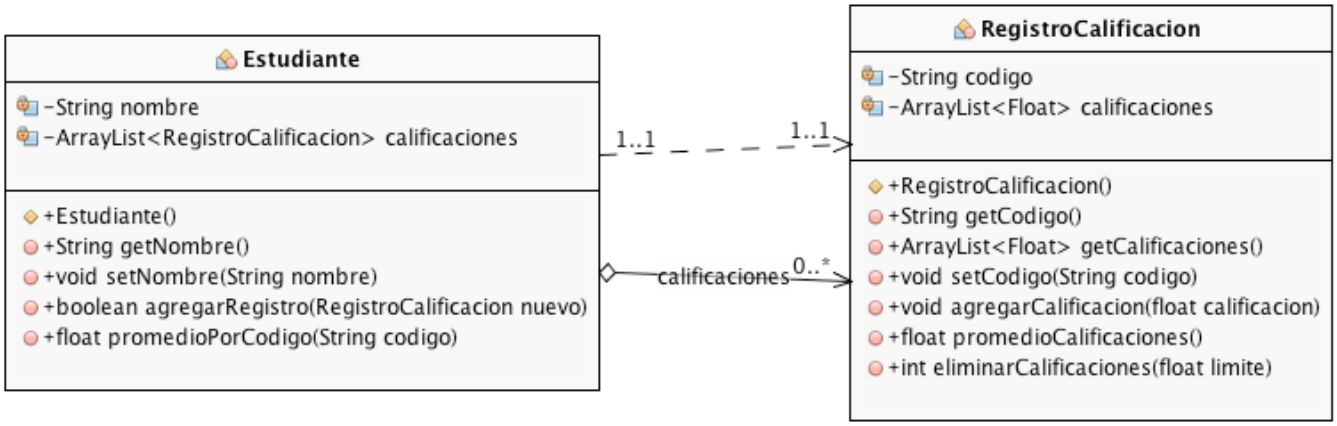
```java
    /**
     * Agrega un nuevo arriendo a la lista, sabiendo que no puede
     * tener 2 arriendo del mismo mes
     * @param nuevo, objeto que será inserto en la lista
     * @return true en caso de éxito en la inserción y false en caso contrario
     */
    public boolean agregarArriendo(ArriendoAutomovil nuevo){
        for(ArriendoAutomovil a : this.arriendos){
            if (a.getPatente().equalsIgnoreCase(nuevo.getPatente())){
                return false;
            }
        }
        this.arriendos.add(nuevo);
        return true;
    }

    public float totalKilometrosPatente(String patente){
        for(ArriendoAutomovil a : this.arriendos){
            if (a.getPatente().equalsIgnoreCase(patente)){
                return a.totalKilometros();
            }
        }
        return 0;
    }
}
```

```java
package e1;

/**
 *
 * @author Jazna
 */
public class MainE1B {
    public static void main(String[] args) {
        Cliente c = new Cliente();
        c.setNombre("Carla");
        ArriendoAutomovil ax = new ArriendoAutomovil();
        ax.setPatente("CXPR12");
        System.out.println(ax.getPatente());
        ax.agregarDistancia(34.6f); ax.agregarDistancia(50.3f);
        ArriendoAutomovil ay = new ArriendoAutomovil();
        ay.setPatente("FGRT29");
        System.out.println(ax.getPatente());
        ay.agregarDistancia(246.7f); ay.agregarDistancia(120.7f);

        c.agregarArriendo(ay); c.agregarArriendo(ax);

        System.out.println(c.totalKilometrosPatente("CXPR12"));
        System.out.println("Antes de eliminar : " + c.totalKilometrosPatente("FGRT29"));
        System.out.println("Se han eliminado : " + ay.eliminarKilometros(150f));
        System.out.println("Despues de eliminar : " + c.totalKilometrosPatente("FGRT29"));

    }
}
```

FORMA C

EJERCICIO – IMPLEMENTACIÓN DE CLASES

DIAGRAMA DE CLASES



CONTEXTO DEL NEGOCIO

Un registro de calificación contiene un código de asignatura y una lista de calificaciones. Los estudiantes tienen varios registros de calificaciones, pero no se permiten 2 registros de calificación con el mismo código, en ese caso corresponde agregar la calificación a la lista de calificaciones existentes asociadas a ese código.

REGLAS DE NEGOCIO

- Las calificaciones deben ser valores dentro del rango [1.0; 7.0].
- El código debe contener sólo dígitos.
- Se permite solo un registro de calificación por código para un mismo estudiante.

MÉTODOS ESPECIALIDADOS

MÉTODO	DESCRIPCIÓN	VALOR DE RETORNO
agregarRegistro(RegistroCalificacion nuevo)	Permite agregar un registro al estudiante	True en caso de que logre agregar el registro (código no repetido) y false en caso contrario.
promedioPorCodigo(String codigo)	Calcula el promedio de calificaciones para un determinado código.	Retorna el promedio.
agregarCalificacion(float calificacion)	Agrega una calificación al registro.	Sin valor de retorno
promedioCalificaciones()	Calcular el promedio de calificaciones.	Retorna el promedio de calificaciones del registro.
eliminarCalificaciones(float limite)	Elimina todas las calificaciones menores al límite indicado.	Retorna el total de calificaciones eliminadas.

REQUERIMIENTOS

Considerando el diagrama de clases y las reglas de negocio, se pide:

- Implementar las clases del diagrama
- Implementar la clase de prueba MainE1C que permita cumplir con los requerimientos:
  - Crea un cliente y asignarle el nombre del estudiante que está rindiendo la evaluación.
  - Crear los siguientes registros de calificaciones:
    - 620344, con 2 calificaciones: 3.6 y 5.3.
    - 620431, con 2 calificaciones: 6.7 y 3.4.
  - Agregar los registros al estudiante creado.
  - Mostrar en pantalla, usando el método especializado, el promedio de calificaciones del registro con código 620431.
  - Eliminar, usando el método especializado, todas las calificaciones del registro con código 620431 que sean inferiores a 4.0.
  - Mostrar en pantalla el número de calificaciones eliminadas.
  - Mostrar en pantalla, usando el método especializado, el promedio de calificaciones del registro con código 620431 (para comprobar que funciona la eliminación).

NOTA. Recuerde que debe respetar el diagrama de clases que se entrega.

SOLUCIÓN FORMA C

```
6   package e1;
7
8   import java.util.ArrayList;
9   import java.util.Arrays;
10
11  /**
12   *
13   * @author Jazna
14   */
15  public class RegistroCalificacion {
16      private String codigo;
        private ArrayList<Float> calificaciones;
18
19      public RegistroCalificacion(){
20          this.calificaciones = new ArrayList();
21      }
22
23      public String getCodigo() {
24          return codigo;
25      }
26
27      public ArrayList<Float> getCalificaciones() {
28          return calificaciones;
29      }
```

```
31      public void setCodigo(String codigo) {
32          String digitos[] = {"0","1","2","3","4","5","6","7","8","9"};
33          for(int i = 0; i < codigo.length(); i++){
34              if (!Arrays.asList(digitos).contains(codigo.substring(i, i+1))){
35                  return;
36              }
37          }
38          this.codigo = codigo;
39      }
40
41      /**
42       * Agrega una nueva calificación
43       * @param calificacion
44       */
45      public void agregarCalificacion(float calificacion){
46          if (calificacion >= 1 && calificacion <= 7){
47              this.calificaciones.add(calificacion);
48          }
49      }
50
51      public float promedioCalificaciones(){
52          float total = 0;
54          for(float d : this.calificaciones){
55              total+=d;
56          }
57          return (total == 0?0:total/this.calificaciones.size());
58      }
59      /**
60       * Elimina las calificaciones de la lista inferiores al límite
61       * @param limite, parámetro de eliminación
62       * @return total de calificaciones eliminadas de la lista
63       */
64      public int eliminarCalificaciones(float limite){
65          ArrayList victimas = new ArrayList();
67          for(float d : this.calificaciones){
68              if (d < limite){
69                  victimas.add(d);
70              }
71          }
72          this.calificaciones.removeAll(victimas);
73          return victimas.size();
74      }
    }
```

```java
package e1;

import java.util.ArrayList;

/**
 *
 * @author Jazna
 */
public class Estudiante {
    private String nombre;
    private ArrayList<RegistroCalificacion> calificaciones;

    public Estudiante(){
        this.calificaciones = new ArrayList<>();
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
```

```
29      /**
30       * Agrega un nuevo arriendo a la lista, sabiendo que no puede
31       * tener 2 arriendo del mismo mes
32       * @param nuevo, objeto que será inserto en la lista
33       * @return true en caso de éxito en la inserción y false en caso contrario
34       */
35      public boolean agregarCalificacion(RegistroCalificacion nuevo){
37          for(RegistroCalificacion r : this.calificaciones){
38              if (r.getCodigo().equalsIgnoreCase(nuevo.getCodigo())){
39                  return false;
40              }
41          }
42          this.calificaciones.add(nuevo);
43          return true;
44      }
45
46      public float promedioPorCodigo(String codigo){
47          for(RegistroCalificacion r : this.calificaciones){
48              if (r.getCodigo().equalsIgnoreCase(codigo)){
49                  return r.promedioCalificaciones();
50              }
51          }
52          return 0;
53      }
```

```
6   package e1;
7
8   /**
9    *
10   * @author Jazna
11   */
12  public class MainEC {
13      public static void main(String[] args) {
14          Estudiante e = new Estudiante();
15          e.setNombre("Carla");
16          RegistroCalificacion rx = new RegistroCalificacion();
17          rx.setCodigo("620433");
18          System.out.println(rx.getCodigo());
19          rx.agregarCalificacion(3.6f); rx.agregarCalificacion(5.3f);
20          RegistroCalificacion ry = new RegistroCalificacion();
21          ry.setCodigo("620431");
22          System.out.println(ry.getCodigo());
23          ry.agregarCalificacion(6.7f); ry.agregarCalificacion(3.4f);
24
25          e.agregarCalificacion(ry); e.agregarCalificacion(rx);
26
27          System.out.println(e.promedioPorCodigo("620433"));
28          System.out.println("Antes de eliminar : " + e.promedioPorCodigo("620431"));
29          System.out.println("Se han eliminado : " + ry.eliminarCalificaciones(4.0f));
30          System.out.println("Despues de eliminar : " + e.promedioPorCodigo("620431"));
31
32      }
33  }
```