Nombre: .....

Fecha: /11/2019 Grupo: 1 □ 2 □ 3 □ 4 □

PRÁCTICA 04.01

HERENCIA ENTRE CLASES Y POLIMORFISMO DE MÉTODOS (II)

Objetivos
Los objetivos de la práctica son: - repasar la noción de polimorfismo; - recuperar algunos métodos de la librería (API) de Java e intentar reprogramarlos en nuestras propias clases; - definiremos en C++ métodos similares, pero a falta de una API y una jerarquía similares a las de Java, simplemente los definiremos como nuevos métodos de nuestras clases; - introducir una noción de “persistencia” en nuestro sistema. Para ello vamos a recuperar el sistema de información que ya programamos en la Práctica 03.02, aunque con ligeras modificaciones.
Código de la práctica
El código de partida para la práctica lo puedes encontrar en varios enlaces que se proporcionan a lo largo de este guion.
Material adicional
En el aula virtual podrás encontrar un test con preguntas sobre la práctica, es conveniente que lo hagas ya que te ayudará a profundizar en distintos conceptos que se ven en la misma.

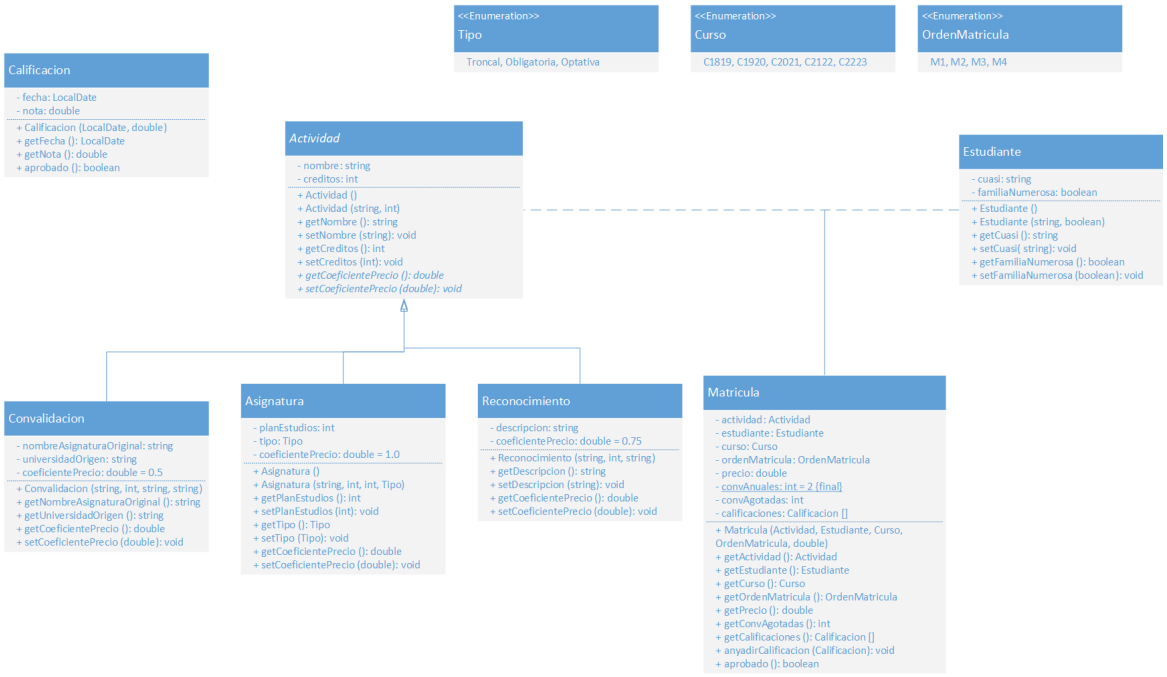
Generalmente, en el diseño de un sistema de información de cierta complejidad, muchas veces la estructura del mismo se divide en tres capas; las mismas responden a los nombres de persistencia (o datos), capa de lógica, y capa de presentación (http://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas); en los sistemas complejos, las mismas suelen estar estructuradas de forma que las diferencias entre ellas son patentes; por ejemplo, es muy común encontrar sistemas de información en que la persistencia se encuentra en una Base de Datos, la lógica está programada en Java, y la presentación se hace por medio de páginas web; a veces, un mismo lenguaje, como Java, puede servir de unión para esas tres capas.

En nuestro caso, la labor de capa de presentación se hace a través de la consola ("cmd"), y en esta práctica la persistencia la vamos a simular por medio de estructuras de datos (listas) que nos permitan alojar objetos de cada una de las entidades (o clases) que aparecen en nuestro sistema de información.

Parte 01. Relaciones de herencia y polimorfismo de métodos.

00. Para C++ acepta la tarea disponible en https://classroom.github.com/a/f7lUsMBv, y para Java acepta la tarea disponible en https://classroom.github.com/a/dYxw0tl7.

01. En primer lugar vamos a revisar el sistema de información que creamos en las Prácticas 02.02 y 03.02. Recupera los ficheros de las Prácticas 02.02 y 03.02. Reprograma las clases tanto en Java como en C++ para que el sistema se corresponda con el siguiente diagrama de clases.



En particular, ten en cuenta las siguientes modificaciones:

Nueva clase **Calificacion**:

- Esta clase va a almacenar información sobre la nota obtenida por un estudiante y la fecha correspondiente. La clase **LocalDate** es propia de la librería de Java (https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html); en C++ la reemplazaremos por enteros de tipo **long**.

- El método “aprobado (): boolean” devolverá “true” cuando la nota sea mayor o igual que 5.

Clase (abstracta) ***Actividad**:*

- Aparecen los métodos abstractos “*getCoeficientePrecio(): double*” y “*setCoeficientePrecio (double): void*”.
- La clase pasa a ser abstracta.

En la clase **Matricula**:

- En cada matrícula, un estudiante puede tener 0 o varias calificaciones (a lo sumo dos, pero obviaremos esta limitación), por lo que la clase **Matricula** contiene un vector de objetos de tipo **Calificacion**.
- Observa que debes reprogramar el constructor de la clase **Matricula** para que se ajuste al nuevo diseño de la clase (al vector “calificaciones” le debes asignar un tamaño de “convAnuales:int=2”).
- El atributo “convAgotadas:int” es un contador del número de elementos que contiene el vector “calificaciones”, debes iniciar este valor en el constructor.
- El método “getConvAgotadas():int” se comporta del modo esperado.
- El método “getCalificaciones (): Calificacion []” devuelve el vector de calificaciones.
- El método “anyadirCalificacion (Calificacion): boolean” comprueba si el valor de “convAgotadas:int” es menor que 2, en cuyo caso añade una nueva calificación al vector “calificaciones”, actualiza el atributo “convAgotadas:int”, y devuelve “true”; en caso contrario devuelve “false”.
- El método “aprobado (): boolean” devuelve “false” si el estudiante no tiene calificaciones o ninguna de las calificaciones está aprobada (deberías hacer uso del método de la clase **Calificacion**); devuelve “true” en otro caso.
- Con respecto a prácticas anteriores desaparecen el atributo “calificacion:double”, y los métodos “getCalificacion():double”, “anyadirCalificacionInicial(double):boolean” y “sobreescribirCalificacion(double):boolean”.

Comprueba los tests (los tests correspondientes a “toString” y “equals” no pasarán debido a que no se han definido todavía estas funciones) y realiza un commit and push.

02. Vamos a ver ahora en Java cómo podemos redefinir el comportamiento de algunos de los métodos de la clase “Object” en las clases que hemos creado; en particular, declara y define un método “toString(): String”, una vez compruebes su comportamiento por defecto (https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#toString--).

El mismo debe devolver:

En la clase ***Actividad***, el método debe devolver la cadena:

```
"Actividad: " + this.getNombre ()
"Creditos: "  + this.getCreditos ()
"Coeficiente: " + this.getCoeficientePrecio ()
```

En las clases **Convalidacion**, **Asignatura** y **Reconocimiento**, el método debe devolver la cadena (la siguiente cadena corresponde al método “toString (): String” en la clase **Convalidacion**, modifícala adecuadamente para las clases **Asignatura** y **Reconocimiento**):

```
Llama a "toString(): String" de la clase Actividad
"Asignatura de origen: " + this.getNombreAsignaturaOriginal ()
"Universidad de origen: " + this.getUniversidadOrigen ()
```

En la clase **Calificacion** el método debe devolver la cadena:

```
"Fecha: " +  this.getFecha ().toString ()
"Nota: " + this.getNota ()
```

En la clase **Estudiante**, el método debe devolver la cadena:

```
"CUASI: " + this.getCuasi()
"Pertenece a familia numerosa: " + this.getFamiliaNumerosa ()
```

En la clase **Matricula**, el método debe devolver la cadena (incluye los saltos de línea que consideres adecuados):

```
"Estudiante:" + this.getEstudiante ().toString()
"Actividad:" + this.getActividad ().toString()
"Curso:" + this.getCurso ().toString()
"Matricula: " +     this.getOrdenMatricula ().toString()
"Precio:" + this.getPrecio ()
"Convocatorias Agotadas:" + this.getConvAgotadas ()
```

"Calificaciones:" Llama al método `"toString ()"` de cada objeto del vector "calificaciones"

```
"Aprobado:" + this.aprobado ()
```

Comprueba los tests (los tests de “equals” no pasarán debido a que no se han definido todavía estas funciones) y realiza un commit and push.

03. Repite el ejercicio anterior en C++, creando un método “mostrar(): void” en cada una de tus clases con un comportamiento similar al que hemos definido en Java; nota que el método, en lugar de devolver las cadenas de caracteres, simplemente debe mostrarlas por pantalla cuando sea invocado. Realiza un commit and push.

04. Otro de los métodos de la librería de Java, también propio de la clase Object, que nos interesa reprogramar en nuestro sistema es el método “equals (Object): boolean” (https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#equals-java.lang.Object-). Comprueba el comportamiento del mismo en el enlace anterior.

Vamos a ver cómo la propia librería de Java lo redefine para algunas de sus clases; en particular, en el siguiente fragmento de código podemos ver cómo es la definición en OpenJDK (http://openjdk.java.net/, código fuente en http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/tip/src/share/classes/java/lang/String.java#l949) del método “equals(Object): boolean” para la clase “String” (https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#equals(java.lang.Object)):

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence {
    /** The value is used for character storage. */
    private final char value[];
    …

public boolean equals (Object anObject) {
       if (this == anObject) {
           return true;
       }
       if (anObject instanceof String) {
           String anotherString = (String) anObject;
           int n = value.length;
           if (n == anotherString.value.length ) {
              char v1[] = value;
              char v2[] = anotherString.value;
              int i = 0;
              while (n-- != 0) {
                  if (v1[i] != v2[i])
                     return false;
```

```
                i++;
            }
            return true;
        }
    }
    return false;
}
```

El método tiene el interés intrínseco de ser una implementación válida del código de la API de Java para el método “equals (Object): boolean” de la clase “String”, pero, en realidad, lo que nos interesa del anterior método es su estructura, que trataremos de entender:

```
public boolean equals (Object anObject) {
      // Comparamos this con el parámetro anObject; si son el mismo objeto en memoria,
      // ya podemos devolver true sin más comprobaciones
      if (this == anObject) {
          return true;
      }
      // Si no es el mismo objeto, pero ambos objetos son del mismo tipo, en este caso
      // de tipo String, le hacemos un “cast” al tipo correspondiente (String)
      if (anObject instanceof String) {
          String anotherString = (String) anObject;
              // Con el objeto “this” y el parámetro del mismo tipo,
              // podemos hacer la comparación de objetos “atributo a atributo”;
              // Importante: esa comparación debería ser hecha por el método
              // “equals (Object): boolean” de las clases de los atributos
              boolean aux = …;
              return aux;
      }
      // Si el objeto anObject ni siquiera es una instancia de esta clase
      // (en este caso String), podemos devolver false
      else {
              return false;
      }
  }
```

Define en Java el método “equals (Object): boolean” para las siguientes clases del anterior sistema de información:

- **Calificacion**: dos calificaciones serán iguales si sus fechas y notas lo son;
- ***Actividad***: dos actividades serán iguales si sus nombres y créditos lo son;
- **Convalidacion**: dos convalidaciones serán iguales si son iguales como actividades y además los nombres y las universidades de la asignatura de origen coinciden;
- **Asignatura**: dos asignaturas serán iguales si son iguales como actividades y además coinciden sus planes de estudios y sus tipos;
- **Reconocimiento**: dos reconocimientos son iguales si son iguales como actividades y además sus descripciones son iguales;
- **Estudiante:** dos estudiantes son iguales si tanto sus cuasis como su atributo “familiaNumerosa” son iguales;
- **Matricula:** dos matrículas son iguales si coinciden en actividad, estudiante y curso.

Para comparar los enum puedes consultar el siguiente enlace https://stackoverflow.com/questions/1750435/comparing-java-enum-members-or-equals.

05. Comprueba que tus programas pasan los tests tanto en Java como en C++. Realiza un commit and push.

06. Construye en C++ un cliente del sistema de información, llamado *“Principal.cpp”*, que compruebe que se pueden construir objetos (o punteros a objetos) de los tipos **Estudiante**, **Asignatura**, **Reconocimiento**, **Convalidacion**; crea objetos de tipo **Matricula** a partir de los objetos creados. Comprueba que el método “mostrar (): void” de las distintas clases se comporta de forma adecuada (en particular, comprueba el valor de “getPrecio (): double”, que debería enlazarse dinámicamente). Realiza un commit and push.