Análisis y Diseño de Software
Curso 2024-2025

Práctica 4: Herencia, Interfaces y Excepciones

Inicio: A partir del 17 de Marzo.
Duración: 3 semanas.
Entrega: En Moodle, una hora antes del comienzo de la siguiente práctica según grupos (semana del 7 de Abril)
Peso de la práctica: 30%

El objetivo de la práctica es aprender técnicas de orientación a objetos más avanzadas que en las prácticas anteriores. En concreto, se hará énfasis en los siguientes conceptos:

- herencia
- interfaces
- excepciones
- colecciones

Para ello, diseñaremos e implementaremos una aplicación de participación ciudadana, donde los ciudadanos, asociaciones ciudadanas y fundaciones pueden proponer proyectos para que el ayuntamiento los lleve a cabo, así como apoyar proyectos existentes, y recibir notificaciones respecto a los distintos eventos, de acuerdo a distintas estrategias de distribución de mensajes.

Apartado 1. Proponentes de proyectos (2.75 puntos)

Comenzaremos creando clases para representar a los ciudadanos, asociaciones y fundaciones, y registrarlos en el sistema.

Los *ciudadanos* tienen un nombre, un NIF y una contraseña, que deben proporcionar al registrarse en el sistema. El formato de los NIFs debe tener 8 dígitos seguidos de una letra (p. ej. 12345678A).

Las *asociaciones* agrupan un conjunto de ciudadanos con un interés común, así como otras asociaciones (que a su vez podrán agrupar otros ciudadanos y asociaciones, de manera recursiva). Además, cada asociación tiene un nombre, una contraseña, y un ciudadano registrado que la representa. Estos datos se deben especificar al registrar la asociación en el sistema. Un ciudadano debe poder inscribirse a cualquier asociación registrada, así como

darse de baja de las asociaciones a las que pertenece (de una en una, o de todas las que pertenece). Un ciudadano puede pertenecer a varias asociaciones, pero una asociación no puede tener inscrito dos veces al mismo ciudadano (esto es, con el mismo NIF) ni directa ni indirectamente en las asociaciones que contiene. Por ejemplo, si la asociación *"conservemos el mazacarnés"* contiene la asociación *"amigos de los pájaros"*, un ciudadano podrá formar parte de una de ellas, pero no de ambas. Si una asociación agrupa otras asociaciones, todas deben tener el mismo representante. Por simplicidad, no se podrá añadir una asociación a otra, en caso que la primera esté vacía.

Por último, las *fundaciones* tienen un nombre, un CIF y una contraseña, que deben proporcionar al registrarse en el sistema. El formato de los CIFs debe tener una letra, seguida de 7 dígitos, seguida de una letra.

El sistema debe controlar que no hay NIFs ni CIFs repetidos en el sistema, que su formato es correcto, que la inscripción de ciudadanos en asociaciones cumple las condiciones mencionadas previamente, y que si una asociación agrupa a otras todas deben tener el mismo representante. De no ser así, se deberán lanzar excepciones que permitan identificar el tipo de error producido, contengan información de los detalles del error, y permitan su consulta. Debes diseñar una API adecuada para el sistema, que permita, por ejemplo, recuperar un usuario a partir de su nombre, u obtener todos los usuarios registrados.

Se pide: Usando principios de orientación a objetos, crea todo el código necesario para el registro de ciudadanos, asociaciones y fundaciones, y la inscripción y baja de ciudadanos en asociaciones. Crea tests que prueben la funcionalidad que has implementado. Como ejemplo, la siguiente salida resulta de imprimir la lista de usuarios registrados en un sistema con 3 ciudadanos, 2 asociaciones y 1 fundación. La asociación *"conservemos el mazacarnés"* agrupa la asociación *"amigos de los pájaros"*, Juan pertenece a *"conservemos el mazacarnés"*, Ana y Luisa a *"amigos de los pájaros"*. Se pueden añadir salidas adicionales que justifiquen el diseño usado por claridad.

[Juan Bravo NIF (01234567K) <usuario>,
Ana López NIF (01234567L) <usuario>,
Luisa Gómez NIF (01234567G) <usuario>,
conservemos el manzanares <asociación con 3 ciudadanos>,
amigos de los pájaros <asociación con 2 ciudadanos>,
Fundación Canal CIF (A1234567B) <fundación>]

Apartado 2. Proyectos participativos (1 punto)

Los ciudadanos, asociaciones y fundaciones podrán proponer proyectos participativos. Un proyecto tiene un título, una descripción y un proponente (ciudadano, asociación o fundación). Además, en el caso de proyectos propuestos por una fundación, deberán indicar un presupuesto estimado, y el porcentaje del presupuesto que asumiría la fundación. Al crear un nuevo proyecto, el sistema le asignará automáticamente un código identificativo único, y guardará la fecha y hora en que se propuso.

Se debe controlar que el presupuesto estimado sea mayor que cero, y que el porcentaje sea un número entre 1 y 100, lanzándose excepciones en caso contrario. Debes diseñar una API adecuada para el sistema, con métodos que permitan, por ejemplo, recuperar todos los proyectos registrados, o recuperar un proyecto dado su nombre o su código identificativo.

Se pide: Usando principios de orientación a objetos, crea el código necesario para permitir la propuesta de proyectos por ciudadanos, asociaciones y fundaciones. Crea *testers* que prueben la funcionalidad que has implementado. A modo de ejemplo, la siguiente salida es el resultado de imprimir la lista de proyectos que devuelve el ejemplo de sistema del apartado 1. En el sistema se han registrado dos proyectos, uno propuesto por una asociación, y otro por una fundación. Nótese que en la salida impresa se han introducido saltos de línea por claridad.

yaml
CopiarEditar

```
[0: Limpieza del manzanares. Proponente: conservemos el manzanares
<asociación con 3 ciudadanos>,
1: Gastemos menos agua. Proponente: Fundación Canal CIF (A1234567B)
<fundación>. Presupuesto: 1000000.0€. Porcentaje: 80.0% /proyecto de
fundación/]
```

Apartado 3. Apoyo a proyectos (2.5 puntos)

Los ciudadanos y asociaciones registradas podrán apoyar los proyectos participativos propuestos en el sistema. No obstante, si intentan apoyar un proyecto propuesto por ellos, que han apoyado previamente, o que se propuso hace más de 60 días, el sistema lanzará una excepción y el apoyo será inválido. Si una asociación apoya un proyecto, posteriormente ningún ciudadano inscrito en esa asociación (ni en las asociaciones que contiene, o que la contienen) podrá apoyar ese proyecto (se considera que el ciudadano ya lo está apoyando con su pertenencia a la asociación). Para evitar inconsistencias, cuando una asociación A apoya un proyecto P, el sistema eliminará todos los apoyos a P que provengan de ciudadanos y asociaciones pertenecientes (directa o indirectamente) a A, manteniéndose únicamente el apoyo de A al proyecto. Se entiende que, si una asociación o ciudadano propone un proyecto, automáticamente lo apoya.

Diseña una API adecuada para poder consultar los apoyos a proyectos. Por ejemplo, el sistema deberá incluir un método para obtener un mapa de pares de proyectos y su número de apoyos, ordenados de mayor a menor número de apoyos, y en caso de tener igual número de apoyos, aparecerán en primer lugar los proyectos apoyados más recientemente. El apoyo de un ciudadano a un proyecto se suma al 'n total' de apoyos al proyecto. El apoyo de una asociación suma tantas unidades como ciudadanos pertenezcan a la asociación, directa o indirectamente. Se debe tener en cuenta que, si un ciudadano se da de baja de una asociación, se elimina su voto de todos los apoyos realizados por la asociación hasta ese momento. Igualmente, si un ciudadano se da de alta en una asociación, su voto a todos los proyectos será eliminado por la asociación hasta ese momento. Crea también un método para obtener un mapa con los proyectos y la colección de ciudadanos que los apoyan.

Se pide: Usando principios de orientación a objetos, crea el código necesario para que ciudadanos y asociaciones puedan apoyar los proyectos registrados, y consultar los apoyos. Crea *testers* que prueben la funcionalidad implementada, creando y apoyando varios proyectos, e imprimiendo por pantalla el mapa ordenado de proyectos con el número de apoyos recibidos, y la lista de ciudadanos que los apoyan. A modo de ejemplo, la siguiente salida muestra el resultado de imprimir el mapa con los apoyos por proyecto (donde Juan apoya el proyecto *"Gastemos menos agua"*, y *"Limpieza del manzanares"* viene apoyado por los ciudadanos miembros de la asociación proponente):

```
{0: Limpieza del manzanares. Proponente: conservemos el manzanares
<asociación con 3 ciudadanos>=3,
1: Gastemos menos agua. Proponente: Fundación Canal CIF (A1234567B)
<fundación>. Presupuesto: 1000000.0€. Porcentaje: 80.0% /proyecto de
fundación/=1}
```

De manera similar, el resultado de imprimir el mapa con los ciudadanos que apoyan cada proyecto es:

```
{0: Limpieza del manzanares. Proponente: conservemos el manzanares
<asociación con 3 ciudadanos>=[Juan Bravo NIF (01234567K) <usuario>,
Ana López NIF (01234567L) <usuario>, Luisa Gómez NIF (01234567G)
<usuario>],
1: Gastemos menos agua. Proponente: Fundación Canal CIF (A1234567B)
<fundación>.
```

Apartado 4. Seguidores (2.25 puntos)

Los ciudadanos y las asociaciones pueden seguir a proyectos, asociaciones y fundaciones. Los seguidores (ciudadanos y asociaciones) recibirán anuncios de interés relacionados con las entidades (proyectos, asociaciones y fundaciones) a las que siguen. Por defecto, todos los ciudadanos miembros de una asociación la siguen. Si un ciudadano sigue a una asociación, recibirá anuncios tanto de las entidades que él sigue, como de las entidades a las que sigue la asociación, sin repeticiones. Para que la aplicación pueda extenderse fácilmente con nuevos tipos de seguidores (p. ej. personal del ayuntamiento, una newsletter), los seguidores deben implementar la interfaz `Follower` que se muestra a continuación. El método `receive` se encargará de recibir y procesar anuncios: si es un ciudadano, almacenará el anuncio en su lista de mensajes en orden de recepción, y si es una asociación, lo redistribuirá a sus miembros (ciudadanos y asociaciones).

java
CopiarEditar

```
public interface Follower {
  // puedes modificar esta interfaz si lo crees conveniente,
  // pero debe incluir al menos el siguiente método:
  public void receive(Announcement t);
}
```

También se quiere poder añadir fácilmente al sistema nuevas entidades a las que seguir. Para ello, cualquier entidad a seguir debe implementar la interfaz `FollowedEntity` que se muestra a continuación. Los métodos `follow` y `unfollow` permiten que un seguidor siga o deje de seguir a una entidad, respectivamente. El método `announce` se encargará de emitir los anuncios (objetos de tipo `Announcement`) a todos los seguidores. En el caso de proyectos, anunciará cada nuevo apoyo que reciba (informando del número total de apoyos obtenidos hasta la fecha). En el caso de fundaciones, anunciará cada proyecto que proponga (indicando su título y descripción). En el caso de asociaciones, anunciará cada proyecto que proponga (incluyendo su creación), cada proyecto que apoya (indicando su título y descripción), y cada nueva inscripción (indicando el número total de miembros inscritos).

```
public interface FollowedEntity {
  // puedes modificar esta interfaz si lo crees conveniente,
  // pero debe incluir al menos los siguientes métodos:
  public boolean follow(Follower f);
  public boolean unfollow(Follower f);
  public void announce(Announcement t);
}
```

Se pide: Usando principios de orientación a objetos, crea el código necesario para que ciudadanos y asociaciones puedan seguir a proyectos, asociaciones y fundaciones. Crea *testers* que prueben la funcionalidad que has implementado, mostrando cómo los ciudadanos y las asociaciones pueden seguir a una entidad, y cómo se le anuncian los nuevos apoyos, proyectos e inscripciones. A modo de ejemplo, se muestra un posible resultado de imprimir los anuncios para los 3 ciudadanos usados en los ejemplos anteriores, donde Juan ha seguido a la Fundación Canal antes de que ésta cree el proyecto. En la salida se han añadido algunos saltos de línea por claridad:

Anuncios para Juan Bravo:

```
[ Alta de Juan Bravo en conservemos el manzanares (1 miembros),
  conservemos el manzanares propone el proyecto Limpieza del
manzanares: "Limpiar el manzanares para recuperar su flora y fauna",
  conservemos el manzanares da apoyo al proyecto Limpieza del
manzanares (3 apoyos),
  Fundación Canal propone el proyecto Gastemos menos agua: "Mejora
de las infraestructuras de distribución y captación de agua"]
```

Anuncios para Ana López:

```
[Alta de Ana López en amigos de los pájaros (1 miembros),
 Alta de Luisa Gómez en amigos de los pájaros (2 miembros),
 conservemos el manzanares propone el proyecto Limpieza del
manzanares: "Limpiar el manzanares para recuperar su flora y fauna",
 conservemos el manzanares da apoyo al proyecto Limpieza del
manzanares (3 apoyos)]
```

Anuncios para Luisa Gómez:

```
[Alta de Luisa Gómez en amigos de los pájaros (2 miembros),
 conservemos el manzanares propone el proyecto Limpieza del
manzanares: "Limpiar el manzanares para recuperar su flora y fauna",
 conservemos el manzanares da apoyo al proyecto Limpieza del
manzanares (3 apoyos)]
```

Apartado 5. Estrategias de distribución de anuncios (1.5 puntos)

Extiende la interfaz `FollowedEntity` para permitir el uso de estrategias de distribución de anuncios (objetos de tipo `AnnouncementStrategy`) cuando un `Follower` sigue a una `FollowedEntity`. Por ejemplo, la interfaz extendida podría contener el siguiente método:

```
public boolean follow(Follower f, AnnouncementStrategy ns);
```

Esta nueva variante del método `follow` recibe un objeto de tipo `AnnouncementStrategy`. Si no se usa, al seguidor se le anunciará cada nuevo apoyo, proyecto e inscripción de la entidad seguida, tal y como se explica en el apartado 4. Por el contrario, si se utiliza, debes realizar un diseño que permita aplicar otras estrategias para el envío de anuncios a un seguidor. Se considerarán dos estrategias: (i) enviar al seguidor uno de cada *n* anuncios, con *n* configurable para cada seguidor (p. ej. enviarle uno de cada 10); (ii) si se sigue a un proyecto, enviar al seguidor un único anuncio cuando supere *n* apoyos, con *n* configurable para cada seguidor.

Se pide: Usando principios de orientación a objetos, crea el código necesario para permitir estrategias de distribución de anuncios. Crea *testers* que prueben la funcionalidad que has implementado, mostrando cómo actúan las dos estrategias descritas. A modo de ejemplo, se muestra un posible resultado de imprimir los anuncios para Juan Bravo, con una estrategia que muestra uno de cada dos anuncios de las fundaciones que sigue y las asociaciones en las que está inscrito:

```
[Alta de Juan Bravo en conservemos el manzanares (1 miembros),
 conservemos el manzanares da apoyo al proyecto Limpieza del
manzanares (3 apoyos),
 Fundación Canal propone el proyecto Gastemos menos agua: "Mejora de
las infraestructuras de distribución y captación de agua"]
```

Comentarios adicionales

- No olvides que, además del correcto funcionamiento de la práctica, un aspecto fundamental en la evaluación será la **calidad del diseño**. Tu diseño debe utilizar los principios de orientación a objetos, además de ser claro, fácil de entender, extensible y flexible. Presta atención a la calidad del código, evitando redundancias.

- Organiza el código en **paquetes**.

- Crea **programas de prueba** para ejercitar el código de cada apartado, y entrégalos con tu código.

Normas de entrega

- Se debe entregar el **código Java** de los apartados, la **documentación** generada con *javadoc*, los **programas de prueba** creados, y un **diagrama de diseño** de toda la práctica (en PDF) junto con una breve explicación.

- El nombre de los alumnos debe ir en la cabecera *javadoc* de todas las clases entregadas.

- La entrega la realizará uno de los alumnos de la pareja a través de Moodle.

- Se debe entregar un único fichero ZIP / RAR con todo lo solicitado, que debe llamarse de la siguiente manera:
  `GR<numero_grupo>_<nombre_estudiantes>.zip`
  Por ejemplo, Marisa y Pedro, del grupo 2261, entregarían el fichero:
  `GR2261_MarisaPedro.zip`