Examen final – Convocatoria Ordinaria – Evaluación Continua

Análisis y Diseño de Software (2021/2022)

Entrega cada ejercicio en hojas separadas
No se permiten preguntas durante el examen

Ejercicio 1 ***(3,0 puntos)***

Se quiere desarrollar una aplicación informática para gestionar ítems de un periódico, en particular, notas de prensa y artículos de noticias. La aplicación recibe notas de prensa externas, cada una de las cuales compuesta por el nombre de la fuente, fecha y localización, título y texto, detalles de contacto, una o varias categorías, y tal vez una o varias imágenes.

Las categorías (con un nombre y una breve descripción) forman una taxonomía arbórea, en la que cada categoría puede tener una categoría padre (súper categoría) y/o varias categorías hijas (subcategorías). La taxonomía de categorías permitidas está construida en la aplicación, que la usa para verificar la suscripción de un periodista a una o más categorías.

Además de sus datos binarios, una imagen tiene asociadas una URL, un título y una fecha.

Cuando recibe una nota de prensa, la aplicación envía una notificación a los periodistas registrados (con identificadores internos y sus nombres y apellidos) que se han suscrito previamente al menos a una de las categorías de la nota de prensa.

Finalmente, los periodistas pueden crear artículos de noticias en la aplicación. Un artículo estará compuesto de un título, una fecha, uno o varios periodistas autores, un texto y un tipo de alcance (local, nacional o internacional). Asimismo, una noticia tendrá asociadas una o varias notas de prensa y tal vez imágenes a partir de las cuales fue creada.

La aplicación ha de almacenar tanto las notas de prensa recibidas desde el exterior como los artículos de noticias creados por los periodistas.

Se pide:

a. Realiza el diagrama de clases en UML con el diseño de la aplicación descrita arriba. Indica en el diagrama los **patrones de diseño** que uses, incluyendo el rol de los elementos de tu diagrama en dichos patrones. (**2 puntos).**

   En el diagrama no incluyas constructores, getters ni setters, a no ser que sean métodos necesarios para cumplir requisitos del enunciado.

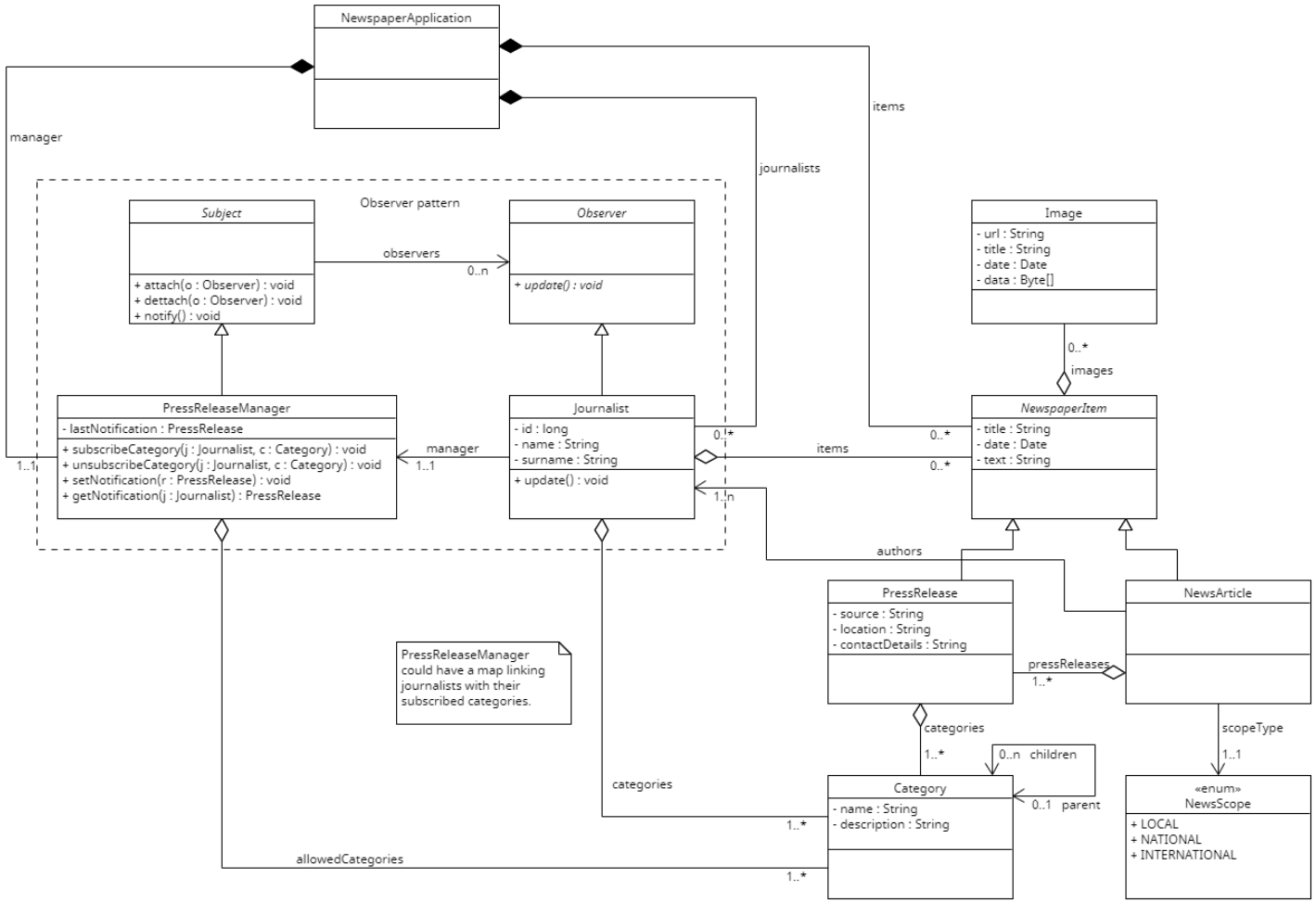
b. Incluye en el diagrama y proporciona el pseudocódigo de métodos con las siguientes funcionalidades: (**1 punto**).

   b.1. Suscribir un periodista a una categoría.

   b.2. Registrar la notificación de una nota de prensa en la aplicación.

Solución:

a)



b.1)

```
void suscribeCategory(Journalist j, Category c) {    // alternativa: gestionar un mapa journalist-category
   if ( observers.contains(j) && allowedCategories.contains(c) )
      j.setCategory(c);
}
```

b.2)

```
void setNotification(PressRelease r) {
   lastNotification = r;
   Category[] rCategories = r.getCategories();

   for ( Observer o: observers ) {
      Journalist j = (Journalist) o;
      Category[] jCategories = j.getCategories();
      for ( Category c: jCategories )
         if ( rCategories.contains(c) ) {
            j.update();
            break;
         }
   }
}
```

Esquema de corrección:

a) Un total de 2 puntos para los siguientes aspectos:

- NewsPaperItem: abstract + subclasses (0.25)
- Category: parent-children (0.25)
- Observer: attach/detach observers (0.25)
- Observer: notify/update (0.25)
- Journalist: category subscription (0.25)
- NewsArticle: aggregation of press releases (notas de prensa) (0.25)
- NewsScope: enum declaration (0.25)
- NewspaperApplication: declaration + aggregations (0.25)

Cada aspecto realizado de forma parcial: 0.1

b.1) Un total de 0.5 puntos para los siguientes aspectos:

- Argumentos de entrada: 0.25
- Chequeos de datos en la asignación: 0.25

b.2) Un total de 0.5 puntos para los siguientes aspectos:

- Argumentos de entrada y uso de la última press release: 0.25
- Bucle, chequeo de datos y llamada a update: 0.25

Ejercicio 2 ***(2.5 puntos)***

Se ha solicitado a los estudiantes de ADSOF el diseño e implementación de un sencillo simulador de naves de la saga de películas StarWars. Toda nave espacial (*Spaceship*) debe permitir subir al piloto (*ridePilot*), volar (*fly*), aterrizar (*land*) y bajar al piloto (*landPilot*), así como disponer de un método para consultar si en ese momento la nave puede volar (*canFly*). Ninguna nave espacial podrá volar sin piloto. Un piloto sólo podrá subir a la nave si no hay otro ocupándola y la nave no está volando. Del mismo modo, un piloto no podrá bajar de la nave si esta está en pleno vuelo. De los pilotos se debe conocer su nombre y su naturaleza (Humano, Droide, Criatura).

De momento, distinguiremos dos tipos de naves, a saber: los cargueros (*Freighter*) y las Alas X (*XWing*). Algunos cargueros, como el Halcón Milenario (*MillenniumFalcon*), son un poco especiales en su pilotaje, por lo que deben disponer de una lista con los nombres de los pilotos que pueden hacerla volar, aunque eso no impide que puedan subir si no hay ningún piloto ocupando la nave. Por su parte, para pilotar las Alas X se requiere que haya un copiloto droide.

El siguiente código muestra un ejemplo de programa de referencia:

```
public class Exercise2 {
        public static void main(String[] args) {
                Pilot lando = new Pilot("Lando Calrissian", Nature.HUMAN);
                Pilot jabba = new Pilot("Jabba the Hutt",   Nature.CREATURE);
                Pilot han   = new Pilot("Han Solo",         Nature.HUMAN);
                Pilot r2d2  = new Pilot("R2-D2",            Nature.DROID);
                Pilot luke  = new Pilot("SkyWalker",        Nature.HUMAN);

                XWing xwing0 = new XWing();     // XWing sin piloto ni copiloto
                XWing xwing1 = new XWing(r2d2); // XWing con r2d2 como copiloto
                xwing1.ridePilot(luke);         // asigna piloto
                System.out.println(xwing1);
                System.out.println("Can fly? " + xwing1.canFly());
                xwing1.fly();                   // Pone a volar la nave espacial

                // Carguero con la lista de personas que son capaces de hacerlo volar.
                Spaceship milleniumFalcon = new Freighter("Lando Calrissian", "Han Solo");
                milleniumFalcon.ridePilot(lando); // Lando puede montar porque no hay piloto en la nave espacial
                System.out.println(milleniumFalcon);
                System.out.println("Can fly? " + milleniumFalcon.canFly());
                milleniumFalcon.ridePilot(jabba); // Jabba no puede montar porque Lando sigue en la nave espacial
                milleniumFalcon.landPilot();
                milleniumFalcon.ridePilot(jabba); // Jabba puede montar porque no hay piloto en la nave espacial
                System.out.println("Can fly? " + milleniumFalcon.canFly());
                milleniumFalcon.landPilot();
                milleniumFalcon.ridePilot(han);  // Han puede montar porque no hay piloto en la nave espacial
                System.out.println("Can fly? " + milleniumFalcon.canFly());
                milleniumFalcon.fly();
                milleniumFalcon.land();
                milleniumFalcon.landPilot();     // Ahora, no hay ningún piloto en la nave espacial
                System.out.println("Can fly? " + milleniumFalcon.canFly());
        }
}
```

Salida esperada:

```
In this Spaceship rides SkyWalker (HUMAN). As copilot goes R2-D2
Can fly? true
In this Spaceship rides Lando Calrissian (HUMAN). Only [Lando Calrissian, Han Solo] are able to make it fly.
Can fly? true
Can fly? false
Can fly? true
Can fly? false
```

Se pide implementar el programa Java para obtener la salida de arriba con el código proporcionado.

1,0 puntos la clase Spaceship
0,5 puntos la clase Freighter
0,5 puntos la clase XWing
0,5 puntos la clase Pilot + enum
Total: 2.5 puntos

```
public abstract class Spaceship {
	private Pilot pilot;
	private boolean isFlying = false;

	public abstract boolean canFly();

	public void fly() {
		if (canFly()) {
			isFlying = true;
			flyingCount++;
		}
	}
	public void land() {
		if (isFlying) {
			isFlying = false;
			flyingCount--;
		}
	}

	public Pilot thePilot() {
		return this.pilot;
	}

	public void ridePilot(Pilot pilot) {
		if (thePilot() == null && !isFlying) {
			this.pilot = pilot;
		}
	}

	public void landPilot() {
		if (!isFlying)
			this.pilot = null;
	}

	@Override
	public String toString() {
		return "In this Spaceship rides " + thePilot() + ".";
	}
}

import java.util.List;

public class Freighter extends Spaceship {
	private List<String> whoCanPilot;

	public Freighter(String ... whoCanPilot) {
		this.whoCanPilot = List.of(whoCanPilot);
	}

	@Override
	public boolean canFly() {
		return thePilot() != null && whoCanPilot.contains(thePilot().name());
	}

	@Override
	public String toString() {
		return super.toString() + " Only " + whoCanPilot + " are able to make it fly.";
	}
}

public class XWing extends Spaceship{
	private Pilot copilot;

	public XWing() {
		this.copilot = null;
	}
```

```
	public XWing(Pilot p2) {
		if (p2 != null && p2.nature() == Nature.DROID)
			copilot = p2;
	}

	@Override
	public boolean canFly(){
		return thePilot() != null && theCopilot() != null;
	}

	private Pilot theCopilot() {
		return copilot;
	}

	@Override
	public String toString() {
		return super.toString() + " As copilot goes " + copilot.name();
	}
}

public class Pilot {
	private String name;
	private Nature nature;

	public Pilot(String name, Nature kind) {
		this.name = name;
		this.nature = kind;
	}

	public String name() {
		return this.name;
	}

	public Nature nature() {
		return this.nature;
	}

	@Override
	public String toString() {
		return this.name + " (" + nature + ")";
	}
}

public enum Nature {
	HUMAN, DROID, CREATURE
}
```

Ejercicio 3 ***(3 puntos)***

Se quiere construir un sistema de votación configurable para distintos eventos, como concursos de salto de trampolín, el concurso de la rosa de primavera de Madrid, o el festival de Eurovisión. Para ello, utilizaremos dos interfaces para caracterizar las opciones que se votan (p.ej. canciones en el caso de Eurovisión) y los votantes (p.ej., países en el caso de Eurovisión):

```
public interface IOption {    // Opción votable
       String getDescription();     // descripción
       String getKey();             // clave
}
```

```
public interface IVoter {     // Votante
       String getKey();      // clave
}
```

Tanto opciones como votantes tienen una clave (por ejemplo, el país), que se podrá utilizar para restringir los votos (en Eurovisión, un país no puede votar por una canción del mismo país).

Debes crear una clase `VotingSystem` que permita almacenar los votos recibidos por cada opción, así como los votos emitidos por cada votante. La clase debe lanzar una excepción `CannotVoteException` si el voto es inválido. Por defecto, será inválido si el votante o la opción no están almacenados en el sistema. Además – como puedes ver en el listado de más abajo – es posible sobreescribir el método `canVote(IVoter voter, IOption op)` para incluir condiciones adicionales, específicas del concurso (la condición sobre los países en el caso de Eurovisión). Finalmente, `VotingSystem` debe proporcionar un método `ranking` que devuelve una lista de las opciones ordenada de mayor a menor número de votos.

```
enum Country /** completar si es necesario **/{
        SPAIN, UK, UKRAINE;
        /** completar si es necesario **/
}

public class Main {
        public static void main(String[] args) {
                Singer chanel = new Singer("Chanel", Country.SPAIN),
                       sam    = new Singer("Sam Ryder", Country.UK),
                       ko     = new Singer("Kalush Orchestra", Country.UKRAINE);
                List<Singer> singers = List.of(chanel, sam, ko);

                VotingSystem euSongContest = new VotingSystem(singers, List.of(Country.values())) {
                        @Override public boolean canVote(IVoter voter, IOption op) {
                                return super.canVote(voter, op) && !voter.getKey().equals(op.getKey());
                        }
                };

                try {
                        euSongContest.vote(Country.SPAIN, new Singer("Sam Ryder", Country.UK), 12);
                        euSongContest.vote(Country.UK, chanel, 10);
                        euSongContest.vote(Country.SPAIN, chanel, 12);
                } catch(CannotVoteException e) {
                        System.out.println(e);
                }
                System.out.println(euSongContest);
                List<IOption> ranking = euSongContest.ranking();
                System.out.println("Final ranking: "+ranking);
        }
}
```

Salida esperada:

```
contest.CannotVoteException: Error: SPAIN cannot vote for Chanel
{Chanel=10, Sam Ryder=12, Kalush Orchestra=0}
{SPAIN={Chanel=0, Sam Ryder=12, Kalush Orchestra=0}, UK={Chanel=10, Sam Ryder=0, Kalush Orchestra=0}, UKRAINE={Chanel=0, Sam Ryder=0, Kalush Orchestra=0}}
Final ranking: [Sam Ryder, Chanel, Kalush Orchestra]
```

Se pide: Usando principios de orientación a objetos, construir las clases `VotingSystem` y `Singer`, la excepción `CannotVoteException`, y completar el enumerado `Country`, si fuese necesario.

Nota: Por simplicidad, no es necesario que controles otros errores en `VotingSystem` (p.ej, votos repetidos, etc.), más allá de los detectados por el método `canVote`.

```
enum Country implements IVoter{
	SPAIN, UK, UKRAINE;
	@Override public String getKey() { return this.toString(); }
}


public class Singer implements IOption {
	private String name;
	private Country country;

	public Singer(String name, Country country) {
		this.name = name;
		this.country = country;
	}

	@Override public String getDescription() { return this.name; }
	@Override public String getKey() { return this.country.toString(); }

	@Override public boolean equals(Object o) {
		if (o==this) return true;
		if (!(o instanceof Singer)) return false;
		Singer s = (Singer) o;
		return s.name.equals(this.name) && s.country.equals(this.country);
	}
	@Override public int hashCode() {
		return this.name.hashCode()+this.country.hashCode();
	}
	@Override public String toString() { return this.name; }
}


public class VotingSystem {
	private Map<IOption, Integer> scores = new LinkedHashMap<>();
	private Map<IVoter, Map<IOption, Integer>> votes = new LinkedHashMap<>();

	public VotingSystem(List<? extends IOption> options, List<? extends IVoter> voters) {
		for (IOption op : options)
			this.scores.put(op, 0);
		for (IVoter vot: voters)
			this.votes.put(vot, new LinkedHashMap<>(this.scores));
	}

	public boolean canVote(IVoter voter, IOption op) {
		return this.scores.keySet().contains(op) && this.votes.keySet().contains(voter);
	}

	public void vote(IVoter voter, IOption opt, int vote) throws CannotVoteException {  // no need to catch other errors
		if (!this.canVote(voter, opt)) throw new CannotVoteException(voter, opt);
		this.scores.put(opt, this.scores.get(opt)+vote);
		this.votes.get(voter).put(opt, vote);
	}

	public String toString() { return this.scores.toString()+"\n"+this.votes.toString(); }

	public List<IOption> ranking() {
		List<IOption> result = new ArrayList<>(this.scores.keySet());
		result.sort((op1, op2) -> this.scores.get(op2)-this.scores.get(op1));
		return result;
	}
}


public class CannotVoteException extends Exception {
	public CannotVoteException(IVoter voter, IOption opt) {
		super("Error: "+voter+" cannot vote for "+opt);
	}
}
```

Consideraciones de diseño:

- Almacenar los puntos en la clase Singer no es una buena opción. El manejo de los puntos otorgados a los IOption debería ser responsabilidad de la clase VotingSystem. Después de todo, ese es el servicio que la clase VotingSystem debería proporcionar, y no es razonable que los clientes de la clase necesiten implementar esa lógica.
- VotingSystem NO ES genérica. La adaptación a diferentes tipos de concursos se hace utilizando las interfaces IOption e IVoter, y probablemente sobreescribiendo el método canVote, como se muestra en el listado de ejemplo. El uso de interfaces es una forma de hacer que el código sea reutilizable en diferentes escenarios.

- Utilizar las clases Singer o Country en VotingSystem o en la excepción CannotVoteException es un grave error conceptual. VotingSystem y CannotVoteExeption pretenden ser reutilizables para muchos tipos diferentes de concursos, y no son específicas del concurso de Eurovisión.
- Proporcionar, por ejemplo, un String como argumento para la CannotVoteException requiere que los clientes de la excepción creen el String. Dado que la creación de una CannotVoteException puede ocurrir muchas veces en un programa, debería ser responsabilidad del constructor de la excepción construir el mensaje a partir de los parámetros. Por lo tanto, estas soluciones subóptimas han sido evaluadas como tales.
- VotingSystem debería mantener la puntuación de cada IOption, como la puntuación que cada IVoter asignó a cada IOption. Como se muestra en la salida esperada, las claves se almacenan en orden de inserción (de ahí los LinkedHashMaps).
- Como se muestra en el listado de ejemplo, el método canVote NO lanza ninguna excepción.
- No hay necesidad de modificar nada en el enum Country, más allá de definir el método requerido por IVoter. En particular, añadir atributos, definir otros métodos o nuevos constructores en el enum no es una solución óptima, que ha sido evaluada como tal.
- Singer debería implementar los métodos equals y hashCode, porque necesitamos considerar iguales los cantantes con el mismo nombre y país (ver la primera invocación al método vote en el listado de ejemplo).
- Una List<Singer> NO ES una List<IOpción>, y por tanto los argumentos del constructor de VotingSystem requieren el uso de comodines: List<? extends IOption> (y similar para List<Country>).

Esquema de corrección:

- Enum Country: 0,25
- Singer: 0,75
- VotingSystem: 1,75
- Exception: 0,25

Ejercicio 4 ***(1.5 puntos)***

Queremos desarrollar una clase genérica `ReducibleList` para almacenar elementos de una clase determinada (por ejemplo, objetos de tipo `Person`), y aplicarles una operación reductora que calcule un valor único a partir de ellos (por ejemplo, la suma de las edades de las personas). `ReducibleList` debe ser compatible con la interfaz `List`, y estará parametrizada con el tipo de los objetos a almacenar, y el tipo del resultado de la operación a realizar sobre estos objetos. El constructor recibe tres parámetros: una expresión lambda que mapea cada objeto a un valor; otra expresión lambda que toma dos valores y calcula un resultado; y un valor inicial (que se devuelve en caso de que la lista esté vacía).

El siguiente programa ilustra el uso de `ReducibleList` para almacenar una colección de objetos de tipo `Person`, y calcular la suma de sus edades, y la media.

```
class Person {
   private String name;
   private int age;
   public Person(String n, int a) {  // nombre y edad de la persona
        this.name = n;
        this.age = a;
   }
   public int getAge()               {  return this.age; }
   @Override public String toString() {  return this.name+" (age "+this.age+")";}
}

public class FoldMain {
   public static void main(String[] args) {
        ReducibleList<Person, Integer> ageAvg =          // reduce objetos Person a Integer
                        new ReducibleList<>( p -> p.getAge(),    // mapea personas a edades
                                             (n1, n2) -> n1+n2, // reduce sumando las edades…
                                             0);         // …con valor inicial 0

        List<Person> people = ageAvg;                            // Compatible con List
        people.addAll(List.of(new Person("Peter", 20),           // Llenamos la lista con objetos
                              new Person("Paul", 40),
                              new Person("Mary", 30)));

        Integer sum = ageAvg.reduce();                           // mapea objetos Person a Integers, y reduce
        System.out.println("The sum of ages of "+ageAvg+" is "+sum);

        Double avg = ageAvg.reduce(n -> n*1.0/ageAvg.size());    // Aplica una operación final después de la reducción
                                                                 // que puede producir un valor de otro tipo
        System.out.println("The average age of "+ageAvg+" is "+avg);
   }
}
```

Salida esperada:

```
The sum of ages of [Peter (age 20), Paul (age 40), Mary (age 30)] is 90
The average age of [Peter (age 20), Paul (age 40), Mary (age 30)] is 30.0
```

Una `ReducibleList` produce un valor mediante el método `reduce`, que recorre todos los objetos, mapeándolos a valores, y utilizando éstos en la operación reductora. El método `reduce` también puede ser invocado con una expresión lambda como argumento, que se aplica al resultado de la reducción, y que puede producir un valor de tipo arbitrario. En el listado anterior, se utiliza para calcular la media de las edades (un `Double`).

Se pide: Utilizando principios del diseño orientado a objetos, desarrolla la clase `ReducibleList`. Crea un diseño que permita la máxima generalidad en las expresiones lambda pasadas en el constructor y los métodos de `reduce`.

```java
public class ReducibleList<S, T> extends ArrayList<S>{
	private Function<? super S, ? extends T> func;
	private T init;
	// BinaryOperator instead BiFunction is OK, but would not admit this generality. You can also use your own interface
	private BiFunction<? super T, ? super T, ? extends T> operator;

	public ReducibleList( Function<? super S, ? extends T> func,
			      BiFunction<? super T, ? super T, ? extends T> oper,
			      T init) {
		this.func = func;
		this.operator = oper;
		this.init = init;
	}

	public <R> R reduce(Function<? super T, ? extends R> fold) {
		T acum = this.init;
		for (S elem: this)
			acum = this.operator.apply(this.func.apply(elem), acum);
		return fold.apply(acum);
	}

	public T reduce() {
		return this.reduce(n -> n);
	}
}
```

Consideraciones de diseño:

- ReducibleList debe ser compatible con la interfaz List. El enfoque más directo es extender desde una implementación particular de List, como ArrayList o LinkedList. Crear una clase que implemente List requeriría implementar TODOS los métodos de List.
- ReducibleList es una clase genérica con dos parámetros. No se espera nada de esos parámetros.
- Para conseguir la máxima generalidad de las expresiones lambda, hemos utilizado comodines. No utilizar comodines da lugar a soluciones más restringidas, y se han evaluado como tales.
- No es una buena opción repetir el código en ambos métodos reduce. Otros esquemas de reutilización (por ejemplo, invocar el reduce sin parámetros desde el otro) también son posibles.
- El método reduce con un argumento necesita ser un MÉTODO GENÉRICO, para considerar el tipo extra del resultado. Las soluciones que no han considerado la genericidad del método han sido evaluadas como incorrectas, o parcialmente correctas a lo sumo.

Esquema de corrección:

- Declaración de la clase: 0.4
- Constructor y atributos: 0.5
- Métodos reduce: 0.6