Prueba 2 de Evaluación Continua

Análisis y Diseño de Software (2020/2021)

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

Ejercicio 1 ***(3.5 puntos)***

Se va a desarrollar una aplicación para una compañía de seguros. La compañía ofrece seguros de hogar y de coches. Cada seguro debe tener el DNI del asegurado, un número de póliza que debe generarse automáticamente de forma incremental, un valor de póliza (el valor del producto asegurado), y un porcentaje para el cálculo de la prima. El seguro de hogar debe tener como identificador de la vivienda el número de catastro. Por su parte, el seguro de coches debe tener como identificador del vehículo la matrícula de este. La prima (valor que paga el usuario por el seguro) se calcula aplicando dicho porcentaje sobre el valor del producto asegurado. El porcentaje es fijo para hogares, establecido a 0.004 por la compañía. Para los coches, este porcentaje depende de la cobertura del vehículo (0.03 si está a terceros, 0.05 si está a terceros ampliado y 0.10 si está a todo riesgo).

Se pide: el código Java necesario para modelar el sistema descrito arriba, de forma que al ejecutar el siguiente código de prueba se produzca la salida mostrada abajo. No escribas métodos innecesarios para esta ejecución, pero presta especial atención al diseño de las clases necesarias y las relaciones entre ellas y entres sus métodos.

```
package uam.eps.acme.seguros;

public class Main {
  public static void main(String[] args) {
    Seguro[] seguros = {
        new SeguroHogar("3567498-H", "0987656788432", 450000.0),
        new SeguroCoche("3567498-H","9438-GBT",   4000.0, Cobertura.A_TERCEROS),
        new SeguroCoche("3567498-H","1412-FZH",  25000.0, Cobertura.TODO_RIESGO),
        new SeguroHogar("1234346-A", "9876545678987", 265472.0),
        new SeguroCoche("4498456-V", "4588-OTG",  16000.0, Cobertura.A_TERCEROS_AMPLIADOS)};

    for (Seguro s : seguros)
      System.out.println(s);
  }
}
```

```
Hogar-> Asegurado: 3567498-H, póliza: 1, valor: 450000.0, prima: 1800.0, catastro: 0987656788432
Coche-> Asegurado: 3567498-H, póliza: 2, valor: 4000.0, prima: 120.0, matrícula: 9438-GBT, cobertura: A_TERCEROS
Coche-> Asegurado: 3567498-H, póliza: 3, valor: 25000.0, prima: 2500.0, matrícula: 1412-FZH, cobertura: TODO_RIESGO
Hogar-> Asegurado: 1234346-A, póliza: 4, valor: 265472.0, prima: 1061.888, catastro: 9876545678987
Coche-> Asegurado: 4498456-V, póliza: 5, valor: 16000.0, prima: 800.0, matrícula: 4588-OTG, cobertura: A_TERCEROS_AMPLIADOS
```

```
package uam.eps.acme.seguros;

public abstract class Seguro {
	private String dniAsegurado;
	private static int generaIdPoliza = 0;
	private int idPoliza;
	private double valor;
	private double porcentajePrima;


	public Seguro(String dniAsegurado, double valor) {
		this.dniAsegurado = dniAsegurado;
		this.idPoliza = ++generaIdPoliza;
		this.valor = valor;
	}

	public int getIdPoliza() {
		return idPoliza;
	}

	protected void setPorcentajePrima(double porcentaje) {
		this.porcentajePrima = porcentaje;
	}

	public double getPrima() {
		return valor *  porcentajePrima;
	}

	@Override
	public String toString() {
		return "Asegurado: " + dniAsegurado + ", " +
				"póliza: " + idPoliza +  ", " +
				"valor: " + valor + ", " +
				"prima: " + getPrima();
	}
}

public class SeguroHogar extends Seguro{
	private String catastro;

	public SeguroHogar(String dniAsegurado, String catastro, double valor) {
		super(dniAsegurado, valor);
		this.catastro = catastro;
		setPorcentajePrima(0.004);
	}

	@Override
	public String toString() {
		return "Hogar-> " +
		super.toString() + ", " +
				"catastro: " + catastro;
	}
}
```

```
public class SeguroCoche extends Seguro{
	private String matricula;
	private Cobertura cobertura;

	public SeguroCoche(String dniAsegurado, String matrícula,
		 double valor, Cobertura cobertura) {
		super(dniAsegurado, valor);
		this.matricula = matrícula;
		this.cobertura = cobertura;
		setPorcentajePrima(cobertura.porcentaje());
	}

	@Override
	public String toString() {
		return "Coche-> " +
		super.toString() + ", " +
				"matrícula: " + matricula + ", " +
				"cobertura: " + cobertura;
	}
}

public enum Cobertura {
	A_TERCEROS(0.03), A_TERCEROS_AMPLIADOS(0.05), TODO_RIESGO(0.10);

	private double porcentaje;

	private Cobertura(double porcentaje) {
		this.porcentaje = porcentaje;
	}

	public double porcentaje() {
		return this.porcentaje;
	}
}
```

Prueba 2 de Evaluación Continua

Análisis y Diseño de Software (2020/2021)

Entrega cada ejercicio en hojas separadas

Ejercicio 2 ***(3,5 puntos)***

Considérese un servicio de películas bajo demanda desarrollado en Java, en el que se usa una clase `Movie` para guardar información sobre cada película: título, año de estreno, edad mínima de espectador recomendada, y lista de géneros.

En el servicio se quiere incluir catálogos de películas. Un catálogo (`MovieCatalogue`) ha de tener un conjunto de películas ordenadas que satisfacen un serie de filtros (restricciones). La ordenación de las películas se hará primero por título, alfabéticamente, y en caso de igualdad de títulos, por año de estreno, de menor a mayor.

Si se intenta añadir a un catálogo una película que no cumple alguno de sus filtros, se lanzará una excepción, subclase de `MovieException`.

```
public abstract class MovieException extends Exception {
    protected Movie movie;
    public MovieException(Movie movie) { this.movie = movie; }
}
```

Para ello, se define una interfaz genérica `MovieFilter`:

```
public interface MovieFilter {
    public void apply(Movie movie) throws MovieException;
}
```

Asumir dos tipos de filtros:

- `NotAllowedGenreMovieFilter`, que chequea si alguno de los géneros de una película es uno dado (no permitido). Su método `apply` lazaría una excepción de tipo `NotAllowedGenreMovieException`.
- `MinimumAgeMovieFilter`, que chequea si la edad mínima de espectador recomendada para una película excede un valor dado. Su método `apply` lazaría una excepción de tipo `MinimumAgeMovieException`.

Y el siguiente *tester* con su correspondiente salida por consola:

```
public class MovieCatalogTester {
    public static void addMovieToCatalogue(MovieCatalogue catalogue, Movie movie) {
        try {
            catalogue.addMovie(movie);
        } catch (MovieException e) { System.err.println(e); }
    }
    public static void main(String[] args) {
        MovieCatalogue c = new MovieCatalogue(new NotAllowedGenreMovieFilter("Horror"),
                                              new NotAllowedGenreMovieFilter("Crimen"),
                                              new MinimumAgeMovieFilter(12));
        Movie m1 = new Movie("Parque Jurásico", 1993, 12, Arrays.asList("Ciencia ficción"));
        Movie m2 = new Movie("Star Wars IV ", 1977, 13, Arrays.asList("Ciencia ficción", "Aventura"));
        Movie m3 = new Movie("El padrino", 1972, 16, Arrays.asList("Crimen"));
        Movie m4 = new Movie("High School Musical", 2006, 10, Arrays.asList("Comedia", "Musical"));
        MovieCatalogTester.addMovieToCatalogue(c, m1);
        MovieCatalogTester.addMovieToCatalogue(c, m2);
        MovieCatalogTester.addMovieToCatalogue(c, m3);
        MovieCatalogTester.addMovieToCatalogue(c, m4);
        System.out.println("Catálogo: " + c);
    }
}
```

```
Excepción: Star Wars IV (1977) filtrada. Edad mínima de 12 superada: 13
Excepción: El padrino (1972) filtrada. Género no permitido: Crimen
Catálogo: [High School Musical (2006), Parque Jurásico (1993)]
```

Se pide:

a) El código Java de los filtros `NotAllowedGenreMovieFilter` y `MinimumAgeMovieFilter`. No hay que dar el código Java de las excepciones, aunque habrá que invocarlas adecuadamente. (**1,5 puntos**)

b) El código Java de la clase `MovieCatalogue`. (**1,5 puntos**)

c) La declaración de la clase `Movie` (no proporcionar el código). (**0,5 puntos**)

Solución:

a)

```
public class NotAllowedGenreMovieFilter implements MovieFilter {
    private String genre;

    public NotAllowedGenreMovieFilter(String genre) { this.genre = genre; }

    public void apply(Movie movie) throws MovieException {
        if (movie.getGenres().contains(this.genre)) {
            throw new NotAllowedGenreMovieException(movie, this.genre);
        }
    }
}

public class MinimumAgeMovieFilter implements MovieFilter {
    private int minimumAge;

    public MinimumAgeMovieFilter(int minimumAge) { this.minimumAge = minimumAge; }

    public void apply(Movie movie) throws MovieException {
        if (movie.getMinimumAge() > this.minimumAge) {
            throw new MinimumAgeMovieException(movie, this.minimumAge);
        }
    }
}
```

b)

```
public class MovieCatalogue {
    private Set<Movie> movieSet;
    private List<MovieFilter> filterSet;

    public MovieCatalogue(MovieFilter... filters) {
        this.movieSet = new TreeSet<>();

        this.filterSet = new ArrayList<>();
        for (MovieFilter filter : filters) {
            this.filterSet.add(filter);
        }
    }

    public void addMovie(Movie movie) throws MovieException {
        for (MovieFilter filter : this.filterSet) {
            filter.apply(movie);
        }
        this.movieSet.add(movie);
    }

    public String toString() {
        return this.movieSet.toString();
    }
}
```

c)

```
public class Movie implements Comparable<Movie>
```

Prueba 2 de Evaluación Continua

Análisis y Diseño de Software (2020/2021)

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

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

Se quiere desarrollar una clase genérica `DataStore` para el almacenamiento de medidas asociadas a *propiedades* de los nodos de una estructura en forma de árbol. En particular, usaremos `DataStore` para guardar los votos de las elecciones a rector en la UAM, desagregados por sus centros y departamentos. Para modelar estructuras en árbol utilizaremos la siguiente interfaz genérica:

```
public interface ITree<T> {
	List<ITree<T>> getChildren();
	T getValue();
}
```

Donde `getChildren` devuelve los nodos hijos, y `getValue` el valor del nodo. El siguiente enumerado, **que debes completar**, contiene la estructura jerárquica de la UAM (muy simplificada), donde la raíz es `General`, que tiene `EPS` y `Medicine` como hijos directos.

```
public enum UAMCenter _____________ /* completar si es necesario*/ {
	ComputerScience, Teleco, EPS(ComputerScience, Teleco), Medicine, General(EPS, Medicine);

	UAMCenter(UAMCenter...centers) { /* completar */ }
	private List<ITree<UAMCenter>> centers = new ArrayList<>();
	// completar
}
```

El siguiente programa ejemplifica el uso de `DataStore`. La clase se parametriza con el tipo de la estructura en forma de árbol (`UAMCenter`) y el tipo de los elementos de los que se almacena la medida (`String`, ya que se utiliza el nombre de los candidatos). En este caso, el constructor recibe dos elementos, pero podría recibir un número arbitrario de ellos.

```
public static void main(String[] args) {
  DataStore<UAMCenter, String> ds = new DataStore<>(UAMCenter.General, "rector B", "rector A");

  ds.sum(UAMCenter.Teleco, "rector A", 5);   // Teleco da 5 votos a A, deben sumarse en EPS y General
  ds.sum(UAMCenter.Medicine, "rector B", 10); // Medicine da 10 votos a B, deben sumarse en General

  System.out.println(ds);

  List<UAMCenter> centers = ds.childrenWith("rector A", p -> p==0);
  System.out.println("Centers with 0 votes for rector A: "+centers);
}
```

Un `DataStore` debe guardar las medidas (votos) asociadas a los elementos (rectores) **por el orden natural** de estos últimos. Además, debe almacenar los `DataStores` asociados a los hijos del nodo (p.ej., a los departamentos dentro de un centro), para poder presentar el detalle de las medidas de manera desagregada. El resultado del programa anterior sería:

```
General: {rector A=5, rector B=10} [EPS: {rector A=5, rector B=0} [ComputerScience: {rector A=0, rector B=0},
Teleco: {rector A=5, rector B=0}], Medicine: {rector A=0, rector B=10}]
Centers with 0 votes for rector A: [ComputerScience, Medicine]
```

Como puedes ver, el método `sum` suma votos a un candidato en un centro, que deben propagarse hasta la raíz de la jerarquía. El método `childrenWith` permite obtener los elementos del árbol cuyas medidas sobre un elemento cumplan la condición que se pasa como parámetro. En el ejemplo, se obtienen los centros con 0 votos al rector A.

Se pide: Usando principios de orientación a objetos, desarrolla la clase `DataStore` y completa el enumerado `UAMCenter`.

Solución

```
public enum UAMCenter implements ITree<UAMCenter>{
	ComputerScience, Teleco,
	EPS (ComputerScience, Teleco),
	Medicine,
	General(EPS, Medicine);

	UAMCenter(UAMCenter...centers) { this.centers.addAll(List.of(centers)); }
	private List<ITree<UAMCenter>> centers = new ArrayList<>();

	@Override public List<ITree<UAMCenter>> getChildren() { return this.centers; }
	@Override public UAMCenter getValue() { return this; }
}

public class DataStore<T extends ITree<T>, E extends Comparable<E>> {
	private T node;
	private TreeMap<E, Integer> data = new TreeMap<>();
	private List<DataStore<T, E>> children = new ArrayList<>();
	private DataStore<T, E> parent;

	public DataStore(T root, E... elements) {
		this.node = root;
		for (E el : elements)
			this.data.put(el, 0);
		for (ITree<T> ch: root.getChildren()) {
			DataStore<T, E> ds = new DataStore<>(ch.getValue(), elements);
			ds.parent = this;
			this.children.add(ds);
		}
	}

	@Override public String toString() {
		return this.node+": "+this.data+(this.children.size()==0?"":" "+this.children.toString());
	}

	public boolean sum(T node, E element, Integer value) {
		if (this.node.equals(node)) {
			this.update(element, value);
			return true;
		} else
			for (DataStore<T, E> ds: this.children)
				if (ds.sum(node, element, value)) return true;

		return false;
	}

	private void update(E element, Integer value) {
		this.data.put(element, this.data.get(element)+value);
		if (this.parent!=null) this.parent.update(element, value);
	}

	public List<T> childrenWith(E element, Predicate<Integer> condition) {
		List<T> result = new ArrayList<>();
		if (condition.test(this.data.get(element)))
			result.add(this.node);
		for (DataStore<T, E> t : this.children)
			result.addAll(t.childrenWith(element, condition));
		return result;
	}
}
```