Nombre:..... ***Legajo:.....***

Segundo Parcial de Programación Orientada a Objetos

12 de Noviembre de 2011

Ej. 1	*Ej. 2*	*Ej. 3*	*Nota*

- **Condición de aprobación: Tener dos ejercicios calificados como B o B-.**
- **Los ejercicios que no se ajusten estrictamente al enunciado, no serán aceptados.**
- **No es necesario agregar las sentencias import.**
- **Además de las clases pedidas se pueden agregar las que se consideren necesarias.**

Ejercicio 1

Se tiene la siguiente jerarquía de clases que representan elementos de forma rectangular que pueden imprimirse en la consola:

```
public abstract class Element {
	public abstract int width();		// Devuelve el ancho del elemento
	public abstract int height();		// Devuelve el alto del elemento
	public abstract char contents(int row, int column);
	public void print() {
		for (int row = 0; row < height(); row++) {
			for (int col = 0; col < width(); col++) {
				System.out.print(contents(row, col));
			}
			System.out.println();
		}
	}
}
```

```
public class UniformElement extends Element {
	private int width, height;
	private char contents;
	public UniformElement(int width, int height, char contents) {
		this.width = width;
		this.height = height;
		this.contents = contents;
	}
	public int width() {
		return width;
	}
	public int height() {
		return height;
	}
	public char contents(int row, int column) {
		if (row < 0 || column < 0 || row >= height || column >= width) {
			throw new IllegalArgumentException();
		}
		return contents;
	}
	public void changeContents(char value) {
		contents = value;
	}
}
```

```
public class TextElement extends Element {
	private String value;
	public TextElement(String value) {
		setValue(value);
	}
	public void setValue(String value) {
		this.value = value;
	}
	public int height() {
		return 1;
	}
```

```
        public int width() {
                return value.length();
        }

        public char contents(int row, int column) {
                if (row != 0 || column < 0 || column >= value.length()) {
                        throw new IllegalArgumentException();
                }
                return value.charAt(column);
        }
}
```

Se quiere dar la posibilidad al usuario de formar nuevos elementos combinando otros ya existentes. Para esto se pueden agregar métodos y/o propiedades a las clases **Element, UniformElement** y/o **TextElement** o bien crear clases nuevas, de forma tal que el siguiente ejemplo compile e imprima lo que se indica a continuación (NO SE PUEDE MODIFICAR/ELIMINAR NINGUNA LINEA DEL CODIGO EXISTENTE, SOLO AGREGAR):

Ejemplo:

```
public class Test {

   public static void main(String[] args) {

      Element elem1 = new UniformElement(6, 2, '+');
      Element elem2 = new TextElement("hola");
      Element elem3 = new TextElement("mundo");

      Element elem4 = elem1.above(elem2);
      Element elem5 = elem3.below(elem2);

      System.out.println("elem4:");
      elem4.print();

      System.out.println("elem5:");
      elem5.print();

      ((UniformElement)elem1).changeContents('.');
      ((TextElement)elem3).setValue("adios");

      System.out.println("elem4:");
      elem4.print();

      System.out.println("elem5:");
      elem5.print();
   }
}
```

Salida del ejemplo:

```
elem4:
++++++
++++++
hola

elem5:
hola
mundo

elem4:
......
......
hola

elem5:
hola
adios
```

Ejercicio 2

La interface **`SpecialList<T>`** extiende a **`List<T>`** agregando un método que, dada cierta función que construye un objeto de tipo **S** en base a un objeto de tipo **`T`** (modelada a través de una nueva interface **`Function<T,S>`**), retorne un objeto **`Iterable<S>`**. Ese objeto permite obtener iteradores especiales que retornan los elementos de la lista original habiéndoles aplicado la función recibida.

```
public class MyTest {
        public static void main(String[] args) {
                SpecialList<String> list = new SpecialArrayList<String>();

                list.add("ABC"); list.add("X"); list.add("1234");list.add("HAZ");

                Function<String, Integer> f = new Function<String, Integer>() {
                        public Integer apply(String t) {
                                return t.length();
                        };
                };

                Iterable<Integer> iterable1 = list.iterable(f);

                for (Integer i: iterable1) {
                        System.out.println(i);        // Esto imprime en distintas líneas: 3 1 4 3
                }

                list.add("ZZZZZZZZ");

                Iterator<Integer> it = iterable1.iterator();
                while (it.hasNext()) {
                        Integer i = it.next();
                        System.out.println(i);        // Esto imprime en distintas líneas: 3 1 4 3 8
                        if ( i % 2 == 0) {
                                it.remove();
                        }
                }

                for (String s: list) {
                        System.out.println(s);        //Esto imprime en distintas líneas: ABC X HAZ
                }
        }
}
```

Se pide implementar `SpecialList.java`, `Function.java`, `SpecialArrayList.java` y cualquier otra clase que considere necesaria.

Ejercicio 3

Se quiere modelar una carrera y sus participantes para luego tener información sobre los resultados. Para esto se define la clase **Race**. Esta clase lleva un registro de todos los participantes de la carrera, permitiendo para cada uno establecer tiempos de partida y de llegada, y además permite imprimir los resultados (lista de todos los participantes con el tiempo total de carrera de cada uno).

Esta clase usa como clase auxiliar a la clase **Participant**, que permite definir un participante en base a su nombre y edad. Además permite definir el momento en que comenzó la carrera y el momento en que la finalizó (si es que logró llegar a la meta). Esta clase ofrece un método capaz de calcular la duración total de la carrera para dicho participante.

Los horarios de partida y de llegada se almacenan en variables de tipo long que representan unidades de tiempo arbitrarias, la resta entre la llegada y la partida es el tiempo total de carrera.

```
public class Race {

	private Map<String, Participant> participants = new HashMap<String, Participant>();

	/** Agrega a un participante a la carrera, especificando el nombre y la edad. */
	public void addParticipant(String name, int age) {
		if (participants.containsKey(name)) {
			throw new IllegalArgumentException("Duplicated participant!");
		}
		participants.put(name, new Participant(name, age));
	}

	/** Registra la hora en la que un participante pasa por la largada. */
	public void registerStartTime(String participant, long time) {
		getParticipant(participant).start(time);
	}

	/** Registra la hora en la que un participante cruza la meta.  */
	public void registerEndTime(String participant, long time) {
		getParticipant(participant).end(time);
	}

	/** Obtiene el tiempo total de carrera del participante. */
	public double getTotalTime(String participant) {
		return getParticipant(participant).getTotalTime();
	}

	/** Obtiene un participante existente a partir de su nombre.  */
	protected Participant getParticipant(String name) {
		Participant participant = participants.get(name);
		if (participant == null) {
			throw new IllegalArgumentException("Invalid participant name");
		}
		return participant;
	}

	/** Imprime la lista de todos los participantes inscriptos, indicando el nombre
	 * y el tiempo neto de cada uno. */
	public void printParticipants() {
		printParticipants(participants.values());
	}

	/** Imprime por consola el nombre y el tiempo neto de un grupo de participantes.  */
	protected void printParticipants(Iterable<Participant> part) {
		for (Participant p: part) {
			System.out.println(p.getName() + " " + (p.hasTime() ? p.getTotalTime() : "--"));
		}
	}
}
```

```
public class Participant {

	private String name;
	private int age;
	private Long startTime;
	private Long endTime;

	public Participant(String name, int age) {
		this.name = name;
		this.age = age;
	}
	public String getName() {
		return name;
	}
	public int getAge() {
		return age;
	}
	public void start(long time) {
		startTime = time;
	}
```

```
        public void end(long time) {
                endTime = time;
        }
        public boolean hasTime() {
                return startTime != null && endTime != null;
        }
        public long getTotalTime() {
                return endTime - startTime;
        }
}
```

Se pide implementar la clase **CategoryRace**, que representa una carrera subdividida en categorías por edad no necesariamente excluyentes (puede haber una categoría de 20 a 30 años y a su vez una de 25 a 35 años). Esta clase es capaz de listar a los participantes de cierta categoría ordenados según el tiempo total de carrera de cada uno y también genera un ranking completo (sin discriminar por categoría). Lo que sigue es un programa de ejemplo de uso de **CategoryRace**.

```
public class RankingRaceTest {

        public static void main(String[] args) {

                CategoryRace race = new CategoryRace();

                race.addCategory("Categoria1", 20, 40);
                race.addCategory("Categoria2", 30, 50);

                race.addParticipant("Persona A", 25);       // Etapa inscripción
                race.addParticipant("Persona B", 32);
                race.addParticipant("Persona C", 33);
                race.addParticipant("Persona D", 45);
                race.addParticipant("Persona E", 65);
                race.addParticipant("Persona F", 41);

                race.registerStartTime("Persona A", 1000);  // Se larga la carrera
                race.registerStartTime("Persona B", 1110);
                race.registerStartTime("Persona C", 1050);
                race.registerStartTime("Persona D", 1200);
                race.registerStartTime("Persona E", 1000);
                race.registerStartTime("Persona F", 1300);

                race.registerEndTime("Persona A", 2000);    // Comienzan a llegar a la meta
                race.registerEndTime("Persona B", 2600);
                race.registerEndTime("Persona C", 2240);
                race.registerEndTime("Persona D", 3100);
                race.registerEndTime("Persona E", 2100);

                System.out.println("Participantes:");       // Se obtienen resultados
                race.printParticipants();
                System.out.println("Ranking completo:");
                race.printGeneralRanking();
                System.out.println("Categoria 1");
                race.printCategoryRanking("Categoria1");
                System.out.println("Categoria 2");
                race.printCategoryRanking("Categoria2");

        }
}
```

En la salida estándar se muestra lo siguiente:

```
Participantes:
Persona B 1490
Persona C 1190
Persona A 1000
Persona F --
Persona E 1100
Persona D 1900
Ranking completo:
Persona A 1000
Persona E 1100
Persona C 1190
Persona B 1490
Persona D 1900
Categoria 1
Persona A 1000
Persona C 1190
Persona B 1490
Categoria 2
Persona C 1190
Persona B 1490
Persona D 1900
```