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

Segundo Parcial de Programación Orientada a Objetos

15 de Junio de 2011

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

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

Ejercicio 1

Se tiene la siguiente interfaz junto con una clase que la implementa:

```
public interface SimpleList<T> extends Iterable<T> {
	/** Agrega un elemento al final de la lista. */
	public void append(T elem);
}
```

```
public class SimpleArrayList<T> extends ArrayList<T> implements SimpleList<T> {
	public void append(T elem) {
		add(elem);
	}
}
```

Se quiere dar la posibilidad al usuario de concatenar dos listas. Implementar todo lo necesario para que el siguiente código de ejemplo compile e imprima lo que se indica en los comentarios. **No es posible modificar el código de SimpleList ni de SimpleArrayList.**

```
public class Test {
	public static void main(String[] args) {

		SimpleList<String> list1 = new SimpleArrayList<String>();
		SimpleList<String> list2 = new SimpleArrayList<String>();

		list1.append("A"); list1.append("B"); list1.append("C");
		list2.append("D"); list2.append("E");

		SimpleList<String> list3 = new ConcatSimpleList<String>(list1, list2);

		for (String s : list3) {		/* La salida de este for es: ABCDE */
			System.out.print(s);
		}

		list1.append("H");
		list2.append("I");

		for (String s : list3) {		/* La salida de este for es: ABCHDEI */
			System.out.print(s);
		}

		Iterator<String> it = list3.iterator();
		while (it.hasNext()) {
			String s = it.next();
			if (s.equals("B") || s.equals("D")) {
				it.remove();
			}
		}

		list3.append("J");

		for (String s : list1) {		/* La salida de este for es: ACH */
			System.out.print(s);
		}
		for (String s : list2) {		/* La salida de este for es: EIJ */
			System.out.print(s);
		}
	}
}
```

Ejercicio 2

Se tiene la interfaz **RankMap** que modela un mapa que lleva registro de la fecha de último acceso a cada clase y del ranking de cada una (cantidad de accesos):

```
public interface RankMap<K, V> {
	/**
	 * Agrega un par clave-valor al mapa con ranking 0. Si la clave ya existe, sobreescribe el
	 * valor y el ranking se matiene.
	 */
	public void put(K key, V value);

	/**
	 * Devuelve el valor asociado a la clave key. Incrementa el ranking. Si la clave no
	 * existe retorna null.
	 */
	public V get(K key);

	/**
	 * Devuelve el ranking de la clave key. Si la clave no existe lanza la
	 * excepción NoSuchKeyException.
	 */
	public int getRank(K key);

	/**
	 * Devuelve la fecha de último acceso a la clave key. Si la clave no existe lanza la
	 * excepción NoSuchKeyException.
	 */
	public Date getLastAccess(K key);

	/**
	 * Elimina el par clave-valor. Si la clave no existe, no hace nada.
	 */
	public void remove(K key);

	/**
	 * Devuelve una lista con las claves que tienen ranking mayor a cierto valor
	 */
	public List<K> getHigher(int rank);
}
```

Escribir una implementación de la interfaz. A continuación se muestra un posible programa de ejemplo con su correspondiente salida (considerar que el programa se ejecuta el día 15/6):

RankMapExample

```
public class RankMapExample {
	public static void main(String[] args) {
		RankMap<String, String> map = new RankHashMap<String, String>();
		map.put("google", "www.google.com");
		map.put("gmail", "gmail.google.com");
		map.put("iol", "www.iol.itba.edu.ar"); map.remove("iol");

		map.get("gmail"); map.get("gmail");
		map.get("google"); map.get("google"); map.get("google"); map.get("google");
		map.get("google"); map.get("google"); map.get("google"); map.get("google");

		Date access = map.getLastAccess("gmail");
		System.out.println("gmail fue consultado el " + access.getDate() +
			"/" + (access.getMonth() + 1));
		System.out.println("gmail fue consultado " + map.getRank("gmail") + " veces");

		for (String s: map.getHigher(4)) {
			System.out.println(s);
		}

		try {
			System.out.println("iol fue consultado " + map.getRank("iol") + " veces");
		} catch (NoSuchKeyException e) {
			System.out.println("No existe la clave iol");
		}
		try {
			System.out.println("iol fue consultado el" + map.getLastAccess("iol"));
		} catch (NoSuchKeyException e) {
			System.out.println("No existe la clave iol");
		}
	}
}
```

Salida

```
gmail fue consultado el 15/6
gmail fue consultado 2 veces
google
No existe la clave iol
No existe la clave iol
```

Ejercicio 3

La clase **ExamFactory** permite crear exámenes con preguntas tomadas al azar de una lista de preguntas previamente cargada. Las preguntas están representadas mediante la clase **Question** y el examen mediante la clase **Exam**:

```
public class ExamFactory {

	private Set<Question> questions = new HashSet<Question>();

	/** Agrega una posible pregunta de examen. */
	public void addQuestion(String caption, double value, boolean answer) {
		add(new Question(caption, value, answer));
	}

	protected void add(Question q) {
		if (questions.contains(q)) {
			throw new IllegalArgumentException("Duplicated question.");
		}
		questions.add(q);
	}

	/** Obtiene la lista completa de preguntas disponibles. */
	public Iterable<Question> getQuestions() {
		return questions;
	}

	/** Crea un nuevo examen, tomando la cantidad de preguntas recibida por parámetro al azar. */
	public Exam createExam(int quantity) {
		if (quantity > questions.size()) {
			throw new IllegalArgumentException("Not enough questions.");
		}
		Exam exam = new Exam();
		List<Question> examQuestions = new ArrayList<Question>(questions);
		for (int i = 0; i < quantity; i++) {
			int index = (int) (Math.random() * examQuestions.size());
			exam.addQuestion(examQuestions.remove(index));
		}
		return exam;
	}
}
```

```
public class Exam {

	private List<Question> questions = new ArrayList<Question>();

	void addQuestion(Question question) {
		questions.add(question);
	}

	public double evaluate(List<Boolean> answer) {
		double result = 0.0F;
		for (int i = 0; i<questions.size(); i++) {
			if (questions.get(i).evaluate(answer.get(i))) {
				result += questions.get(i).getValue();
			}
		}
		return result;
	}

	public void print() {
		for (Question q : questions) {
			System.out.println(q.getCaption());
		}
	}
}
```

```
public class Question {

	private String caption;
	private boolean answer;
	private double value;

	public Question(String caption, double value, boolean answer) {
```

```
                this.caption = caption;
                this.value = value;
                this.answer = answer;
        }

        public String getCaption() {
                return caption;
        }

        public double getValue() {
                return value;
        }

        public boolean evaluate(boolean answer){
                return this.answer == answer;
        }

        @Override
        public int hashCode() {
                return caption.hashCode();
        }

        @Override
        public boolean equals(Object obj) {
                if (obj == null || !(obj instanceof Question)) {
                        return false;
                }
                Question other = (Question)obj;
                return caption.equals(other.caption);
        }

}
```

Se quiere implementar una nueva clase **EnhancedExamFactory** que además de la funcionalidad ya ofrecida por **ExamFactory**, permita imprimir un ranking de preguntas. En este listado se muestran todas las preguntas que fueron respondidas al menos una vez, indicando para cada una de ellas el cociente entre la cantidad de veces que fue respondida correctamente y la cantidad total de veces que fue respondida. El listado debe estar ordenado por este valor, de menor a mayor.

```
public class ExamFactoryTest {

        public static void main(String[] args) {

                EnhancedExamFactory factory = new EnhancedExamFactory();

                factory.addQuestion("Pregunta 1", 0.5, true);
                factory.addQuestion("Pregunta 2", 1, false);
                factory.addQuestion("Pregunta 3", 0.2, true);
                factory.addQuestion("Pregunta 4", 0.8, true);
                factory.addQuestion("Pregunta 5", 1, true);

                Exam exam = factory.createExam(3);   /* Crea un examen con 3 preguntas al azar.
                                                        Para este ejemplo vamos a suponer que contiene:
                                                                    - Pregunta 2
                                                                    - Pregunta 3
                                                                    - Pregunta 5
                                                     */

                System.out.println(exam.evaluate(Arrays.asList(true, false, true)));      // 1.0
                System.out.println(exam.evaluate(Arrays.asList(false, false, true)));     // 2.0
                System.out.println(exam.evaluate(Arrays.asList(false, false, false)));    // 1.0
                System.out.println(exam.evaluate(Arrays.asList(true, true, true)));       // 1.2

                exam = factory.createExam(2);        /* Crea un examen con 2 preguntas al azar.
                                                        Para este ejemplo, contiene:
                                                                    - Pregunta 2
                                                                    - Pregunta 4
                                                     */

                System.out.println(exam.evaluate(Arrays.asList(false, false)));   // 1.0

                factory.printRanking();              /* Imprime:
                                                            Pregunta 4 (0.0)
                                                            Pregunta 3 (0.25)
                                                            Pregunta 2 (0.6)
                                                            Pregunta 5 (0.75)
                                                     */
        }
}
```