Nombre y Apellido: ..... N° Legajo: .....

Recuperatorio del Segundo Parcial de Programación Orientada a Objetos (72.33)

02/12/2019

Ejercicio 1	*Ejercicio 2*	*Ejercicio 3*	*Nota*	*Firma Docente*
/3	/2.5	/4.5		

- **Condición mínima de aprobación: SUMAR 5 PUNTOS.**
- **Las soluciones que no se ajusten al paradigma OO, no serán aceptadas.**
- **Las soluciones que no se ajusten estrictamente al enunciado, no serán aceptadas.**
- **Puede entregarse en lápiz.**
- **No es necesario escribir las sentencias `import`.**
- **Además de las clases solicitadas se pueden agregar las que consideren necesarias.**
- **Escribir en cada hoja Nombre, Apellido, Legajo, Número de Hoja y Total Hojas entregadas.**

Ejercicio 1

Se desea crear una colección que ofrezca el método **`toMap`**. Este método recibe una función que indica cómo se debe construir la clave que acompañará a cada valor de la colección en un nuevo mapa que retorna el método. En caso de que se obtengan dos o más claves iguales, en el mapa no quedará una cantidad de entradas igual a los elementos de la colección y no se debe validar.

Implementar <u>todo</u> lo necesario para que, con el siguiente programa de prueba, se obtenga la salida indicada en los comentarios.

```
public class SimpleListTester {

   public static void main(String[] args) {
       SimpleList<Integer> simpleList = new SimpleArrayList<>();

       simpleList.add(1);
       simpleList.add(3);
       simpleList.add(5);
       simpleList.add(7);

       System.out.println(simpleList.size()); // 4

       System.out.println(simpleList.contains(0)); // false

       Map<Integer, Integer> first = simpleList.toMap(element -> element * 2);

       System.out.println(first.get(2)); // 1
       System.out.println(first.get(6)); // 3
       System.out.println(first.get(10)); // 5
       System.out.println(first.get(14)); // 7

       Map<String, Integer> second = simpleList.toMap(element -> String.format("<%d>", element));

       System.out.println(second.get("<1>")); // 1
       System.out.println(second.get("<3>")); // 3
       System.out.println(second.get("<5>")); // 5
       System.out.println(second.get("<7>")); // 7

       Map<Integer, Integer> third = simpleList.toMap(element -> element % 2);

       System.out.println(third.get(0)); // null
       System.out.println(third.get(1)); // Podría ser 1, 3, 5 ó 7
   }

}
```

Ejercicio 2

Dada la siguiente jerarquía de clases, con los métodos de instancia indicados para cada una, se cuenta con tres instancias homónimas a la clase a la cual pertenecen.

Completar el cuadro de doble entrada (clase y mensaje) indicando qué se obtiene al enviar cada uno de los mensajes a instancias de cada una de las clases.

	m1	m2	m3
A			
B			
C			

```
class A {

 int m1() {
  return 7;
 }

 int m2() {
  return this.m3();
 }

 int m3() {
  return m1();
 }

}
```

```
class B extends A {

 int m1() {
  return 9;
 }

 int m3() {
  return super.m3();
 }

}
```

```
class C extends B {

 int m1() {
  return super.m1();
 }

 int m2() {
  return this.m2();
 }

 int m3() {
  return m3();
 }

}
```

Ejercicio 3

La clase `ExamFactory` permite crear exámenes con preguntas tomadas al azar de una lista de preguntas previamente cargada. Cada pregunta tiene dos posibles respuestas: falso o verdadero. Las preguntas están representadas mediante la clase `Question` y el examen mediante la clase `Exam`:

```
public class ExamFactory {

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

   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);
   }

   public Exam createExam(int quantity) {
       if (quantity > questions.size()) {
           throw new IllegalArgumentException("Not enough questions.");
       }
       Exam exam = getNewExam();
       List<Question> examQuestions = new ArrayList<>(questions);
       Collections.shuffle(examQuestions);
       for (int i = 0; i < quantity; i++) {
           exam.addQuestion(examQuestions.get(i));
       }
       return exam;
   }

   protected Exam getNewExam() {
       return new Exam();
   }

}
```

```
public class Exam {

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

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

```
    public double evaluate(List<Boolean> answer) {
        double result = 0;
        for (int i = 0; i < questions.size(); i++) {
            Question question = questions.get(i);
            if(evaluateQuestion(question, answer.get(i))) {
                result += question.getValue();
            }
        }
        return result;
    }

    protected boolean evaluateQuestion(Question question, boolean answer) {
        return question.evaluate(answer);
    }

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

}
```

```
public class Question implements Comparable<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 boolean equals(Object o) {
        if (this == o)
            return true;
        if (!(o instanceof Question))
            return false;
        Question question = (Question) o;
        return caption.equals(question.caption);
    }

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

    @Override
    public String toString() {
        return caption;
    }

    @Override
    public int compareTo(Question o) {
        return caption.compareTo(o.caption);
    }

}
```

Se quiere implementar una nueva clase `EnhancedExamFactory` que además de la funcionalidad ya ofrecida por `ExamFactory`, **permita iterar sobre un ranking de preguntas con distintos criterios**. Hay dos criterios posibles:

- **Por cantidad de respuestas (correctas o incorrectas)**
- **Por cantidad de respuestas correctas.**

  En ambos casos el orden es ascendente por cantidad. Y cuando los valores coinciden, el orden se determina alfabéticamente por el texto de la pregunta.

Implementar <u>todo</u> lo necesario para que, con el siguiente programa de prueba, se obtenga la salida indicada. **<u>No se pueden modificar las clases provistas.</u>**

```
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);

       // Suponemos que el azar elegirá las preguntas: "Pregunta 4", "Pregunta 5" y "Pregunta 1"
       Exam exam = factory.createExam(3);

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

       System.out.println("**********");

       // Suponemos que el azar elegirá las preguntas: "Pregunta 2" y "Pregunta 5"
       exam = factory.createExam(2);

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

       System.out.println("**********");

       for (Question q : factory.byAnswers()) {
           System.out.println(q);
       }
       System.out.println("**********");
       for (Question q : factory.byCorrectAnswers()) {
           System.out.println(q);
       }
   }

}
```

```
1.3
0.5
0.0
2.3
**********
1.0
0.0
**********
Pregunta 2
Pregunta 1
Pregunta 4
Pregunta 5
**********
Pregunta 2
Pregunta 5
Pregunta 4
Pregunta 1
```