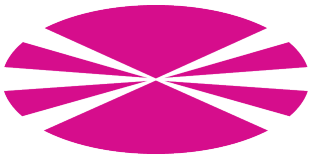

UNIVERSIDADE DA CORUÑA

Diseño Software

Boletín de Ejercicios 1 (2019-2020)

INSTRUCCIONES COMUNES A TODAS LAS PRÁCTICAS:

- **Estructura de los ejercicios**
  - Los ejercicios se realizarán por parejas de alumnos. Si algún alumno tiene algún problema para establecer una pareja que se lo comunique a su profesor de prácticas e intentará buscarle un compañero. Mantened el grupo de prácticas para todas las prácticas de la asignatura.
  - Cada pareja de alumnos deberá inscribirse en el wiki disponible en el campus virtual a tal efecto.
  - Los ejercicios serán desarrollados mediante la herramienta NetBeans (versión 8.2) que se ejecuta sobre Java 8.
- **Entrega de los ejercicios**
  - Los ejercicios se entregarán usando el sistema de control de versiones Subversión (SVN) instalado en el CeCaFI (`https://svn.fic.udc.es/grao2/ds/`) y donde existirá un repositorio para cada grupo.
  - Os explicaremos en un seminario el manejo básico de Subversion que utilizaremos a través del propio NetBeans o de clientes externos como TortoiseSVN en Windows o RapidSVN en Linux.
  - Para la evaluación de la práctica sólo tendremos en cuenta aquellas contribuciones hechas hasta la fecha de entrega de la misma, los envíos posteriores no serán tenidos en cuenta.
- **Evaluación**
  - **Importante: Todo ejercicio en el que haya sospechas fundadas de copias entre grupos implicará la anulación de su nota (tanto para el grupo que lo haya desarrollado originalmente como para el que lo haya copiado). Si la copia es flagrante y/o repetida puede llevar a la anulación de todo el boletín o práctica**.
  - **Criterios generales a evaluar son**: que el código compile correctamente, que no dé errores de ejecución, que se hayan seguido correctamente las especificaciones, que se hayan seguido las buenas prácticas de la orientación a objetos explicadas en teoría, que se hayan seguido las instrucciones para su realización, etc.

INSTRUCCIONES BOLETÍN 1:

Fecha límite de entrega: 11 de octubre de 2019 (hasta las 23:59).

- **Realización del boletín**
  - Se deberá crear un único proyecto para el boletín cuyo nombre será el nombre del grupo de prácticas más el sufijo -B1 (por ejemplo `DS-11-01-B1`).
  - Se creará un paquete por cada ejercicio del boletín con los nombres: `e1`, `e2`, etc.
  - Es importante que sigáis detalladamente las instrucciones del ejercicio, ya que persigue el objetivo de probar un aspecto determinado de Java y la orientación a objetos.
  - En el Campus Virtual existe un ejemplo de proyecto con varios ejercicios resueltos que os puede servir de plantilla para crear vuestro proyecto
- **Comprobación de la ejecución correcta de los ejercicios con JUnit**
  - En la asignatura usaremos el framework JUnit 4 para comprobar, a través de pruebas, que el funcionamiento de las prácticas es el correcto.
  - En este primer boletín os adjuntaremos los tests JUnit que deben pasar los ejercicios para ser considerados válidos.
  - **IMPORTANTE: No debéis modificar los tests que os pasemos**. Sí podéis añadir nuevos tests si consideráis que quedan aspectos importantes por probar en vuestro código (por ejemplo, que su cobertura sea baja en partes fundamentales). **La excepción es el ejercicio 4 que deja en blanco la definición de las variables para que lo hagáis vosotros.**
  - En el seminario de JUnit os daremos información detallada de como ejecutar los tests y calcular la cobertura de los mismos.
- **Evaluación**
  - Este boletín corresponde a 1/3 de la nota final de prácticas.
  - **Pasar correctamente nuestros tests es un requisito importante en la evaluación de este boletín**.
  - Aparte de criterios fundamentales habrá criterios de corrección específicos que detallaremos en cada ejercicio.
  - No seguir las normas aquí indicadas significará una penalización en la nota.

1. **Arrays**

   Crea una clase `MatrixFunctions` que incluya los siguientes métodos estáticos:

```java
package e1;

import java.util.*;

public class MatrixFunctions {

    // Returns the maximun value of a matrix
    public static int max(int[][] a) { }

    // Returns the sum of the values of a given row
    public static int rowSum(int[][] a, int row) { }

    // Returns the sum of the values of a given column
    public static int columnSum(int[][] a, int column) { }

    // Sums the value of each row and returns the results in an array.
    public static int[] allRowSums(int[][] a) { }

    // Sums the value of each column and returns the results in an array.
    // If a position does not exist because the array is "ragged" that position
    // is considered a zero value.
    public static int[] allColumnSums(int[][] a) { }

    // Checks if an array is "row-magic", that is, if all its rows have the same
    // sum of all its values.
    public static boolean isRowMagic(int[][] a) { }

    // Checks if an array is "column-magic", that is, if all its columns have
    // the same sum of all its values.
    public static boolean isColumnMagic(int[][] a) { }

    // Checks that a matrix is square, that is, it has the same number of rows
    // as columns and all rows have the same length.
    public static boolean isSquare(int[][] a) { }

    // Check if the matrix is a magic square. A matrix is magic square if it is
    // square, all the rows add up to the same, all the columns add up to the
    // same and the two main diagonals add up to the same. Also all these sums
    // are the same.
    public static boolean isMagic(int[][] a) { }

    // Checks if the given matrix forms a sequence, that is, it is square
    // (of order n) and contains all the digits from 1 to n * n, regardless of
    // their order.
    public static boolean isSequence(int[][] a) { }
}
```

Criterios:

- Manejo de estructuras típicas de control de Java.
- Manejo de arrays y matrices en Java.

2. **Polímeros**

Los polímeros son macromoléculas formadas por la unión mediante enlaces de una o más unidades simples llamadas monómeros. En nuestro laboratorio identificamos cada una de estas unidades por un carácter del alfabeto, de la `A` a la `Z` (solo caracteres ASCII, sin contar la `Ñ`).

Las unidades o monómeros pueden tener polaridad positiva, en cuyo caso se denotan por una letra minúscula (`a`), o polaridad negativa, en cuyo caso se denotan por una letra mayúscula (`A`).

Reacción de monómeros

Cuando en un polímero coinciden dos monómeros del mismo tipo pero con polaridades distintas, ambos reaccionan destruyéndose y dejando al polímero sin los mismos.

Por ejemplo, en el polímero `CaAdbb`, el par `aA` reacciona autoeliminándose y dejándo el polímero como `Cdbb`. `C` y `d` son de polaridades distintas pero al ser distintos monómeros no reaccionan entre sí. Las unidades `bb` son del mismo tipo pero al tener la misma polaridad no reacionan entre sí.

La eliminación de unidades o monómeros puede producir eliminaciones en cascada. Por ejemplo, el polímero `rFvdAaDVb` al reaccionar elimina sucesivamente el par `Aa`, el par `dD` y el par `vV` para dejar un resultado final de `rFb`. Una reacción en cadena puede eliminar por completo el polímero.

Eliminación del monómeros y polímeros mínimos

Es habitual en el laboratorio ver cuál sería el resultado de eliminar una determinada unidad de un polímero, tanto en su forma positiva como negativa, y ver las reacciones en cadena que se producirían. Por ejemplo, si del polímero `dedaDrERaeeChAHEcEF` eliminamos todas las "A", tanto en minúsculas (`a`) como en mayúsculas (`A`), el resultado es `dedDrEReeChHEcEF`. En este polímero los pares `dD` y `hH` reaccionan quedando como resultado final `derEReeCEcEF`, una cadena de 12 unidades.

Por otro lado, si de la misma cadena `dedaDrERaeeChAHEcEF` eliminamos las "E", en minúsculas (`e`) y mayúsculas (`E`), obtenemos `ddaDrRaChAHcF`. En este polímero el par `rR` reacciona obteniendo como resultado `ddaDaChAHcF`, una cadena de 11 unidades.

Finalmente, si en el mismo ejemplo `dedaDrERaeeChAHEcEF` quitamos las "F", que en este caso solo hay la versión en mayúsculas (`F`), el polímero quedaría entonces como `dedaDrERaeeChAHEcE` con 18 unidades y sin producirse ninguna reacción adicional más.

Los científicos están interesados en saber cuál sería la unidad que, una vez eliminada, deja el polímero con una menor longitud, en el caso anterior sería `e`.

Dado este contexto crea la clase `Polymer` que cumpla la especificación que se muestra en el siguiente código.

```java
/**
 * Class for handling polymers represented by Strings
 */
public class Polymers {
    /**
     * Given a polymer represented with the letters of the alphabet (uppercase
     * and lowercase excluding Ñ), it processes the chain reactions of said
     * polymer (due to adjoint units of the same type but different polarity,
     * for example: aA).
     * It returns the polymer without the reactive pairs (it can be an empty
     * string) and it throws an IllegalArgumentException if the String passed
     * as parameter is null.
     */
    public static String processPolymer(String polymer) { }

    /**
     * Given a polymer returns the monomer (existing in the polymer) whose
     * elimination (both in its positive and negative form) results, after the
     * consequent chain reactions, in the smallest polymer.
     * It returns the positive form (lowercase) of the monomer. In case of a
     * tie, the one with the lowest alphabetical order is returned.
     * An IllegalArgumentException is thrown if the original polymer is null or
     * if it is a empty string.
     */
    public static char minProcessedPolymer(String polymer) { }
}
```

Criterios:

- Manejo de estructuras típicas de control de Java.
- Manejo de la clase `String`, la clase `StringBuilder` y los caracteres en Java.

3. **Rectángulos**

   Escribe la clase `Rectangle` según la especificación que se muestra en el código. Ten en cuenta que la orientación del rectángulo es irrelevante a efectos de igualdad de objetos; es decir, un rectángulo se considera igual a sí mismo girado 90 grados. Ten en cuenta también que ciertos métodos pueden lanzar excepciones en su funcionamiento (los constructores o los métodos *set*).

```java
// Represents a rectangle
public class Rectangle {

    // Initializes a new rectangle with the values passed by parameter.
    // Throws IllegalArgumentException if a a negative value is passed to any of
    // the dimensions.
    public Rectangle(int base, int height) { }

    // Copy constructor
    public Rectangle(Rectangle r) { }

    // Getters
    public int getBase() { }
    public int getHeight() { }

    // Setters. Throw IllegalArgumentException if the parameters are negative.
    public void setBase(int base) { }
    public void setHeight(int height) { }

    // Return true if the rectangle is a square
    public boolean isSquare() { }

    // Calculate the area of the rectangle.
    public int area() { }

    // Calculate the perimeter of the rectangle.
    public int perimeter() { }

    // Calculate the length of the diagonal
    public double diagonal() { }

    // Turn this rectangle 90 degrees (changing base by height).
    public void turn() { }

    // Ensure that this rectangle is oriented horizontally (the base is greater
    // or equal than the height).
    public void putHorizontal() { }

    // Ensure that this rectangle is oriented vertically (the height is greater
    // or equal than the base).
    public void putVertical() { }

    // Two rectangles are equal if the base and the height are the same.
    // It does not take into account if the rectangle is rotated.
    public boolean equals(Object obj) { }

    // It complies with the hashCode contract and is consistent with the equals.
    public int hashCode() { }
}
```

Criterios:

- Instanciación de objetos.
- Encapsulamiento.
- Manejo de excepciones.
- Contratos del equals y el hashCode.

4. **Euros**

   Una moneda de Euro se caracteriza por tener un valor nominal (un euro, 50 céntimos, etc.), un color (oro, bronce u oro-plata), un país, un diseño (representado aquí por simplificación como un `String`, por ejemplo, `"Juan Carlos I"` o `"Felipe VI"`) y un año de acuñación.

   En la actualidad los siguientes países acuñan monedas de Euro, identificados por su nombre (en inglés) y su código ISO de dos letras: *Austria* (AT), *Belgium* (BE), *Cyprus* (CY), *Netherlands* (NL), Estonia (EE), Finland (FI), France (FR), Germany (DE), Greece (GR), Ireland (IE), *Italy* (IT), *Latvia* (LV), *Lithuania* (LT), *Luxembourg* (LU), *Malta* (MT), *Monaco* (MC), *Portugal* (PT), *San Marino* (SM), *Slovakia* (SK), *Slovenia* (SI), *Spain* (ES) y *Vatican City* (VA);

   Define la mejor estrategia para crear la clase `EuroCoin` haciendo uso de clases enumeradas como creas conveniente.

   Crea también una clase `EuroCoinCollection` que permita almacenar y mantener una colección de monedas de Euro. La clase nos permitirá introducir y borrar monedas de la colección, contar cuántas hay, saber su valor nominal total y comprobar si una moneda ya está en la colección.

   Si introducimos monedas repetidas en la colección estas no se añadirán. Se considera que dos monedas son repetidas si tienen el mismo valor, son del mismo país y tienen el mismo diseño visual (identificado este último como un `String`). El año de acuñación no se tiene en cuenta.

   Puedes ver su especificación en el código que se muestra a continuación. Para evitar trabajar con números en coma flotante usaremos como unidad los céntimos de euro, de esta forma el valor de una moneda de dos euros es de 200 céntimos de euro.

```
// Represents a Euro coin collection
public class EuroCoinCollection {
    // Inserts a coin in the collection. If the coin is already in the
    // collection (there is an equal coin inserted) then the coin is not inserted.
    // Returns true only if a new coin has been inserted in the collection.
    public boolean insertCoin(EuroCoin coin) { }

    // Checks if a coin has been already inserted in the collection
    public boolean hasCoin(EuroCoin coin) { }

    // Returns the nominal value of the entire collection in euro cents.
    public int value() { }

    // Counts the number of coins in the collection.
    public int numCoins() { }

    // Removes the specified coin from the collection
    public void removeCoin(EuroCoin coin) {  }
}
```

Criterios:

- Tipos enumerados. Definición de tipos simples o complejos según convenga utilizando todas la información suministrada.
- Colecciones de objetos. Uso y selección de la colección de datos más apropiada para el problema.