EXAMEN CON SOLUCIONES

Duración: 90 minutos
Puntuación máxima: 7 puntos
Fecha: 8 de mayo de 2018
Instrucciones para el examen:

- No se permite el uso de libros o apuntes, ni tener teléfonos móviles u otros dispositivos electrónicos encendidos. Incumplir cualquiera de estas normas puede ser motivo de expulsión inmediata del examen.
- Rellena tus datos personales antes de comenzar a realizar el examen.

APELLIDOS: _____

NOMBRE: _____ NIA: _____ GRUPO: _____

Problema 1. Listas (3 / 7 puntos)

Teniendo en cuenta las siguientes clases, que se encuentran completamente implementadas:

```
public class Point {
    private double x, y;
    public Point(double x, double y) { this.x = x; this.y = y; }
    public double distance(Point p) {
        return Math.sqrt(Math.pow(x - p.x, 2) + Math.pow(y - p.y, 2));
    }
    public double getX() { return x; }
    public double getY() { return y; }
}

public class Node<E> {
    private E info;
    private Node<E> next;
    public Node(E info, Node<E> next) { this.info = info; this.next = next; }
    public Node(E info) { this(info, null); }
    public Node() { this(null, null); }
    public E getInfo() { return this.info; }
    public void setInfo(E info) { this.info = info; }
    public Node<E> getNext() { return this.next; }
    public void setNext(Node<E> next) { this.next = next; }
}

public class LinkedList<E> {
    protected Node<E> first;
    public LinkedList() { this.first = null; }
    public void insert(E info) { /* inserta por el principio ... */ }
    public int size() { /* devuelve el tamaño de la lista ... */ }
}
```

Se pide:

1. [½ punto] Programa la clase `ListPoint`, y su constructor, sabiendo que `ListPoint` es una `LinkedList` de objetos de la clase `Point`.
2. [1,25 puntos] Programa en la clase `ListPoint` el método `public double sumDistancePoint()` que devuelve la suma de las distancias entre los puntos de la lista. Para una lista que tenga cero o un elemento devolverá el valor 0 y para una lista con más de un elemento irá sumando distancias entre un punto y el siguiente utilizando el método `public double distance(Point p)`.
3. [1,25 puntos] Programa en la clase `ListPoint` el método `public Point getXEquals(int value)` que devuelve el primer objeto `Point` de `ListPoint` cuya componente `x` sea igual al valor del parámetro `value`. No debe recorrerse el resto de la lista una vez encontrado el elemento buscado.

Solución:

```
public class ListPoint extends LinkedList<Point> {
    public ListPoint() {
        super();
    }
    public double sumDistancePoint() {
        if (this.size() <= 1) {
            return 0.0;
        } else {
            double totalDistance = 0.0;
            double distance = 0.0;
            Node<Point> current = this.first;

            /* Importante parar cuando el siguiente al actual sea null */
            while (current.getNext() != null) {
                Point point1 = current.getInfo();
                Point point2 = current.getNext().getInfo();
                distance = point1.distance(point2);
                totalDistance = totalDistance + distance;
                current = current.getNext();
            }
            return totalDistance;
        }
    }
    public Point getXEquals(int value) {
        Node<Point> current = this.first;
        while (current != null) {
            if(current.getInfo().getX() == value) {
                return current.getInfo();
            } else {
                current = current.getNext();
            }
        }

        return null;
    }
}
```

Criterios de corrección

- Apartado 1. Clase y constructor (máx 0,50):
  - (0,25) Extender correctamente de `LinkedList<Point>`.
  - (0,25) Implementar correctamente el constructor con llamada a `super();`
- Apartado 2. Método `public double sumDistancePoint()` (máx 1,25):
  - (0,25) Implementación correcta de los casos de 0 y 1 elemento en la lista.
  - (0,25) Creación e inicialización correcta de variables auxiliares (tanto contadores como referencias).
  - (0,25) Condición correcta de `while` (incluyendo el avance referencia `getNext()`).
  - (0,25) Recorrido correcto sobre `point1` y `point2`.
  - (0,25) Cálculo y devolución de distancia correcto.
- Apartado 3. Método `public Point getXEquals(int value)` (máx 1,25):
  - (0,25) Creación e inicialización correcta de variables auxiliares (tanto contadores como referencias.
  - (0,25) Recorrido correcto por la lista.
  - (0,50) Salir de la iteración cuando esté encontrado el elemento.
  - (0,25) Devolución del elemento encontrado correctamente.

Problema 2. Árboles (2,5 / 7 puntos)

Se pide implementar un evaluador de expresiones aritméticas para una calculadora. En este contexto, se define una expresión aritmética como:

a) Un valor `Double`

b) Una operación binaria (`SUMA`, `RESTA`, `MULTIPLICA` o `DIVIDE`) a realizar entre dos expresiones aritméticas.

Utilizando el código que se muestra:

```
public interface Expr {
    String SUMA = "+";
    String RESTA = "-";
    String MULTIPICA = "*";
    String DIVIDE = "/";
    Double eval();
}

public abstract class Nodo implements Expr {
    public String toString() { throw new UnsupportedOperationException("Not implemented yet."); }
}

public class NodoNumero extends Nodo {
    private Double numero;
    public NodoNumero(Double numero) { this.numero = numero; }
    public Double eval() { return this.numero; }
    public String toString() { return numero.toString(); }
}

public class NodoOperacion extends Nodo {
    private String operador;
    private Nodo left;
    private Nodo right;
    public NodoOperacion(String operador, Nodo left, Nodo right) {
        this.operador = operador;
        this.left = left;
        this.right = right;
    }
    public Double eval() { /* IMPLEMENTAR EN APARTADO 1 */ }
    public String toString() { /* IMPLEMENTAR EN APARTADO 2 */ }
}

public class Main {
    public static void main(String args[]) {
        Nodo nodo = new NodoOperacion(Expr.SUMA,
                new NodoOperacion(Expr.MULTIPICA, new NodoNumero(3.0), new NodoNumero(5.0)),
                new NodoNumero(7.0));
        System.out.println("El resultado de " + nodo + " es " + nodo.eval());
    }
}
```

En la salida de ejecución del programa se obtiene:

```
El resultado de ((3.0 * 5.0) + 7.0) es 22.0
```

Se pide:

1. 1 punto Implementar el método `eval()` de `NodoOperacion` sabiendo que debe calcular y devolver el valor correspondiente de aplicar la operación aritmética binaria representada por el atributo `operador` (cuyo valor será uno de las cuatro constantes definidas en la interfaz `Expr`) a los valores de las expresiones del primer operando (`left`) y el segundo operando (`right`).

2. 0,75 puntos Implementar el método `toString()` de `NodoOperacion` sabiendo que debe construir y devolver una representación textual de la expresión aritmética que contiene el nodo, de la misma manera que se observa en la salida de ejecución del programa.

3. 0,75 puntos Escribir el código necesario para construir una expresión aritmética utilizando los tipos `Nodo` adecuados para representar la siguiente expresión: $((3{,}0 * 5{,}0)/(2{,}0 - (8{,}0 + 4{,}0)))$

Solución:

```
public Double eval() {
    if (operador.equals(SUMA)) {
        return left.eval() + right.eval();
    } else if (operador.equals(RESTA)) {
        return left.eval() - right.eval();
    } else if (operador.equals(MULTIPICA)) {
        return left.eval() * right.eval();
    } else if (operador.equals(DIVIDE)) {
        return left.eval() / right.eval();
    } else {
        throw new IllegalArgumentException();
    }
}

public String toString() {
    return "(" + left + " " + operador + " " + right + ")";
}

new NodoOperacion(Expr.DIVIDE,
    new NodoOperacion(Expr.MULTIPICA, new NodoNumero(3.0), new NodoNumero(5.0)),
    new NodoOperacion(Expr.RESTA, new NodoNumero(2.0),
        new NodoOperacion(Expr.SUMA, new NodoNumero(8.0), new NodoNumero(4.0))));
```

Criterios de corrección

- Apartado 1. Método `public Double eval()` (máx 1,00):
  - (0,50) Comprobación de los cuatro condicionales del tipo de operación mediante `if` secuenciales y `String.equals()`; o bien mediante un `switch` moderno y los cuatro casos excluyentes (0,50 en total, 0,125 por cada caso).
  - (0,50) Realización de cada operación aritmética aplicada al hijo izquierdo y derecho, y devolución de su valor (0,50 en total, 0,125 por cada caso).
  - (-0,25) Penalización en total por realizar la comparación de cadenas en los `if` mediante el operador `==` de igualdad.
  - (-0,15) Penalización en total por usar literales de cadena para la comparación de los cuatro símbolos de los operadores en lugar de las constantes definidas en la interfaz `Expr`.
  - La comprobación y lanzamiento de error de si el operador no es ninguno de los cuatro posibles no es obligatoria, aunque se puede valorar positivamente (máx 0,10, y sin superar el máx 1,00 en este apartado).
- Apartado 2. Método `public String toString()` (máx 0,75):
  - (0,25) Añadir literales de paréntesis y espacios necesarios a la cadena.
  - (0,25) Construir la cadena con el operador en el centro y recursivamente la representación textual de la expresión izquierda a su izquierda, y la de la derecha a su derecha, concatenando todos los elementos.
  - (0,25) Devolver la cadena construida.
- Apartado 3. Construcción de expresión concreta (máx 0,75):
  - (0,30) Crear objetos `NodoNumero` y `NodoOperacion` necesarios.
  - (0,45) Combinarlos adecuadamente en los constructores de los demás nodos.

Problema 3. Ordenación (1,5 / 7 puntos)

Dada una clase `IntNode` con los métodos indicados en la misma y una clase que implementa una lista formada por `IntNode`.

```
public class IntNode {
    private int data;
    private IntNode link;
    public IntNode(int initialData, IntNode initialLink) {
        data = initialData;
        link = initialLink;
    }
    public IntNode(int initialData) { this(initialData, null); }
    public int getData() { return data; }
    public void setData(int newData) { data = newData; }
    public IntNode getLink() { return link; }
    public void setLink(IntNode newLink) { link = newLink; }
}

public class LList {
    private IntNode first;
    public LList() { first = null; }
    public LList(int info) {
        IntNode new_ele = new IntNode(info);
        new_ele.setLink(first);
        first = new_ele;
    }
    public void add(int info) {
        IntNode new_ele = new IntNode(info);
        new_ele.setLink(first);
        first = new_ele;
    }
    public IntNode getFirst() { return first; }
    public void selectionSort() {
        /* MÉTODO A IMPLEMENTAR */
    }
}
```

Se pide:

1. 1,50 puntos Implementar el método `public void selectionSort()` de `LList` sabiendo que debe ordenar la lista enlazada mediante el algoritmo *selection sort* de manera ascendente (de menor a mayor).

Solución:

```
public void selectionSort() {
    for (IntNode node1 = first; node1 != null; node1 = node1.getLink()) {
        IntNode min = node1;
        for (IntNode node2 = node1; node2 != null; node2 = node2.getLink()) {
            if (min.getData() > node2.getData()) {
                min = node2;
            }
        }
        int temp = node1.getData();
        node1.setData(min.getData());
        min.setData(temp);
    }
}
```

Criterios de corrección

- (0,40) Inicialización, recorrido y parada correcta de bucle externo.
- (0,40) Inicialización, recorrido y parada correcta de bucle interno.
- (0,20) Comprobación y actualización de nuevo mínimo.
- (0,50) Intercambio de los datos de los nodos (o intercambio correcto de los enlaces al siguiente).