NOMBRE:
APELLIDOS:
NIA:
GRUPO:

Convocatoria extraordinaria

2ª Parte: Problemas (7 puntos sobre 10)

Duración: 150 minutos
Puntuación máxima: 7 puntos
Fecha: 29 junio 2022

Instrucciones para el examen:

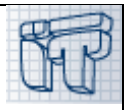
- No se permite el uso de libros o apuntes, ni tener teléfonos móviles u otros dispositivos electrónicos encendidos. Incumplir cualquiera de estas normas puede ser motivo de expulsión inmediata del examen.
- Rellena tus datos personales antes de comenzar a realizar el examen.

Problema 1 OO y testing (3 puntos)

Una empresa líder en el sector de recursos humanos le ha pedido que desarrolle un programa para la gestión del personal. La clase empleado `(Employee)` tiene los atributos y constantes indicados en la siguiente imagen. NOTA: Puedes asumir que todos los métodos get y set están implementados. Dada la clase Employee se pide:

```
public class Employee {
    private String name;
    private String dni;
    private int salary;
    private char type; /* CONSTANTS p permanent,  t temporary employment,  i intern */
    /* Define constants */
    public static final char PERMANENT = 'p';
    public static final char TEMPORARY = 't';
    public static final char INTERN = 'i';

    /* Constuctors */
    public Employee(String name, String dni, char type) throws Exception {
    public Employee(String name, String dni, char type, int salary) throws Exception {

    public String toString() {

    public void incrementSalary() {

    /* setters and getters */
    public void setType(char type) throws Exception{
        if(type==Employee.PERMANENT || type==Employee.TEMPORARY || type==Employee.INTERN) {
            this.type = type;
        }else {
            throw new Exception("The type invalid");
        }
    }
```

Apartado 1.1 (0,5 puntos)



Implementar el método `public String toString()` de la clase Employee. Debe usar los métodos get y set para acceder o modificar los parámetros. Un ejemplo del método sería:

`"NAME: Juan Pérez, DNI: 512355X, TYPE: t, SALARY: 1800”`

Apartado 1.2 (0,75 puntos)

La gerencia de la empresa ha decidido incrementar el salario a todos los empleados con las siguientes consideraciones: 20% para los empleados permanentes o fijos, 15% a los empleados temporales y 10% para los becarios. Implemente el método `public void incrementSalary()`.

Apartado 1.3 (1 puntos)

Debido a la gran cantidad de trabajo la empresa necesita contratar a un becario. Para ello se pide implementar una nueva clase `(Intern)` que herede de empleado `(Employee)`, que además de contener la información de cualquier empleado recibe tres atributos: `department` indica el nombre del departamento, `hoursWorked` indica las horas trabajadas y `basePay` indica el valor por hora a pagar al becario, estos dos últimos atributos sirven para calcular el salario del becario. `NOTA`: La clase debe contener un constructor que reciba los parámetros necesarios.

Apartado 1.4 (0,75 puntos)

Implemente un método de testing llamado `testSetType()` para el método `setType` de la clase `Employee`. Dicho método debe comprobar que se lanza la excepción en caso de asignar a un empleado un tipo incorrecto.

Problema 2 (3 puntos)

Una empresa líder en el sector de la logística le ha pedido que desarrolle un programa para la gestión de un almacén. Las clases de este programa son las siguientes (suponga que todos los métodos `set` y `get` para todos los atributos están ya implementados):

- Persona (`Person`). Permite identificar a cualquier persona relevante para la gestión de un almacén; puede ser un cliente del almacén o el empleado que gestiona el pedido. Los atributos no son relevantes para el problema.

- Pedido (`Order`). Representa el documento que contiene cada uno de los pedidos que se realizan en el almacén. El pedido tiene un atributo de tipo double que refleja el beneficio del pedido (`benefit`). También tiene una referencia al cliente que realiza el pedido (`client`).

- Gestor del almacén (`StoreManager`). Es el cerebro de la aplicación y contendrá toda la lógica del programa. Esta clase tendrá dos atributos:

  - Una cola de pedidos (`LinkedQueue<Order>`), que almacena los pedidos que están pendientes de ser procesados (`ordersToProcess`). La cola de pedidos implementa la siguiente interfaz:

```
public interface Queue<E> {
```

```
    boolean isEmpty();
    int size();
    void enqueue (E info);
    E dequeue();
    E front();
}
```

Nota: Para manejar la cola debe utilizar **exclusivamente** los métodos incluidos en la anterior interfaz. No se admitirán soluciones que no cumplan este requisito.

- Un árbol binario (`LBTree<Order>`) con los pedidos que han obtenido un mayor beneficio (`topOrders`). El árbol tiene las siguientes declaraciones:

```
public class LBNode<E> {
   private E info;
   private LBTree<E> left;
   private LBTree<E> right;


/* Todos los métodos get y set ya
implementados */


   public E getInfo() {...}
   public LBTree<E> getLeft() {...}
   public LBTree<E> getRight {...}

   ...

}
```

```
public class LBTree<E> {
   private LBNode root;


/* Todos los métodos get y set ya
implementados */


   public boolean isEmpty() {...}
   public E getInfo() {...}
   public LBTree<E> getLeft() {...}
   public LBTree<E> getRight() {...}
   public int size() {...}
   public int height() {...}

   ...
}
```

Apartado 2.1 (1,5 puntos)

Codifique el método `public Queue<Order> ordersWithHigherBenefit(double benefit, Person customer)` en la clase `StoreManager`. Dicho método debe devolver una cola de pedidos que contenga los pedidos existentes en la cola `ordersToProcess` que cumplan estas dos condiciones (deben cumplirse ambas simultáneamente):

- El pedido debe pertenecer al cliente referenciado por el parámetro `customer`.
- El beneficio del pedido debe ser mayor al beneficio especificado por el parámetro `benefit`.

La cola `ordersToProcess` debe quedar con los elementos en el mismo orden que estaban tras la ejecución del método.

Apartado 2.2 (1,5 puntos)

Codifique el método `public double benefitTopCustomer(Person customer)` en la clase `LBTree`. Este método debe devolver el beneficio correspondiente al cliente referenciado por el argumento `customer` de los pedidos que se encuentran almacenados en el árbol de pedidos. El recorrido del árbol debe realizarse de forma recursiva, **no admitiendo soluciones que no cumplan con este requisito**.

Problema 3 (1 punto)

Dado el siguiente código de ordenación de un array de elementos enteros:

```
public static void sorting(int[] elements){
 for (int i = 0; i<elements.length; i++){
  int tmp = elements[i];
  int j = i;
  while (j>0 && tmp < elements[j-1]){
   elements[j] = elements[j-1];
   j--;
  }
  elements[j] = tmp;
 }
}
```

Se pide:

1. Indica qué método de ordenación implementa.
2. Transforma el código para que en vez de ordenar sobre un array `int [] elements` se haga sobre un ArrayList `ArrayList<Integer> elements`.
3. Asegura que la solución ordena de mayor a menor.

Nota: La clase ArrayUtil implementa, además de los métodos vistos en clase, el siguiente método:

`E set(int index, E element)`: Replaces the element at the specified position in this list with the specified element.

SOLUCIONES DE REFERENCIA (varias soluciones son posibles)

PROBLEMA 1

Apartado 1.1 (0,5 puntos)

```
public String toString() {
     return "NAME: " + getName() + ", DNI: " + getDni() +  ", TYPE: " + getType() + ", SALARY: " + getSalary();
}
```

- 0 si no tiene sentido.
- 0,1 por concatenar los diferentes string
- 0.3 por obtener los cuatro parámetros usando get
- 0,1 por devolver correctamente el valor
- Los errores significativos y el no usar los getters están sometidos a penalizaciones adicionales.

Apartado 1.2 (0,75 puntos)

```
public void incrementSalary() {
     if(type==Employee.PERMANENT) {
         setSalary((int) (getSalary()*1.20));
     }else if(type==Employee.TEMPORARY) {
         setSalary((int) (getSalary()*1.15));
     }else if(type==Employee.INTERN) {
         setSalary((int) (getSalary()*1.10));
     }
}
```

- 0 si no tiene sentido.
- 0,25 valida el condicional para cada tipo
- 0,5 por calcular correctamente el valor del nuevo salario para cada tipo usando set y get.
- Los errores significativos están sometidos a penalizaciones adicionales.

Apartado 1.3 (1 punto)

```
public class Intern extends Employee {
    private int hoursWorked;
    private int basePay;
    private String department ;

    public Intern(String name, String dni, int basePay,int hoursWorked, String department ) throws Exception {
        super(name,dni,Employee.INTERN);
        super.setSalary(basePay*hoursWorked);
        setDepartment(department );
    }
```

- 0 si no tiene sentido.
- 0,2 por declarar la clase correctamente heredando de Employee y los atributos propios de la clase Intern.

- 0,2 por declarar correctamente el constructor.
- 0,3 por invocar correctamente a super() con los parámetros correctos, teniendo en cuenta que el tipo es una constante que no debe ser argumento.
- 0,2 asignar el atributo salary calculando el valor multiplicando los dos atributos como argumentos.
- 0.1 asignar el atributo departamento
- Los errores significativos están sometidos a penalizaciones adicionales.

Apartado 1.4 (0,75 puntos)

```
@Test
void testSetType() throws Exception {

    Employee employee = new Employee("Juan Perez","512355X ",Employee.TEMPORARY,1800);

    // Solution 1 JUNit5
    assertThrows(Exception.class, ()->{employee.setType('d');});

    // Solution 2
    try {
        employee.setType('d');
        fail("Failed test");
    } catch (Exception e) {
        e.printStackTrace();
    }

}
```

- 0 si no tiene sentido.
- 0,15 por usar la anotación @test.
- 0,25 por instanciar el objeto Product con los valores correctos.
- 0,35 por usar correctamente assertThrows o el try/catch llamando a setCategory con un valor no permitido para que salte la excepción.
- Los errores significativos están sometidos a penalizaciones adicionales.

PROBLEMA 2

Apartado 2.1 (1,5 puntos)

```
public Queue<Order> ordersWithHigherBenefit(double benefit, Person customer) {

	LinkedQueue<Order> result = new LinkedQueue<Order>();

	for (int i = 0; i < ordersToProcess.size(); i++) {

		Order order = ordersToProcess.dequeue();

		if ((order.getClient().equals(customer))
```

```
                        && (order.getBenefit() > benefit))

                result.enqueue(order);

            ordersToProcess.enqueue(order);

        }

        return result;

}
```

Criterios de evaluación

- 0 si el código no tiene sentido.
- 0,15 si se inicializa correctamente la cola a devolver.
- 0,3 si la cola se recorre correctamente.
- 0,2 por desencolar y encolar los elementos en la cola ordersToProcess.
- 0,3 por la condición correcta de comparar correctamente los elementos de tipo Person y el beneficio.
- 0,2 por encolar en la cola a devolver.
- 0,25 si consigue que la cola no quede modificada al terminal método (encolando o creando una cola auxiliar).
- 0,1 si se devuelve la cola correctamente.
- Si no se usan los métodos de la interfaz la calificación máxima es 0,75.
- Los errores significativos están sujetos a penalizaciones adicionales.

Apartado 2.2 (1,5 puntos)

```
public double benefitTopCustomer(Person customer) {

        double result = 0;

        if (!isEmpty()) {

            if (((Order) root.getInfo()).getClient().equals(customer))

                result = ((Order) root.getInfo()).getBenefit();

            result += getLeft().benefitTopCustomer(customer) +

            getRight().benefitTopCustomer(customer);

        }

        return result;

}
```

Criterios de evaluación

- 0 si el código no tiene sentido o no se resuelve recursivamente.
- 0,25 por la condición correcta del caso base.
- 0,25 si se compara correctamente con el parámetro customer.

- 0,7 si las llamadas recursivas son correctas (0,35 cada llamada recursiva).
- 0,3 si se acumula y devuelve el resultado correctamente.
- -0,1 si no hace el casting a Order.
- Los errores significativos están sujetos a penalizaciones adicionales.

PROBLEMA 3 (1 punto)

```
public static void sortingArrayList(ArrayList<Integer> elements){
  for (int i=0; i<elements.size(); i++){
    int tmp = elements.get(i);
    int j=i;
    while (j>0 && tmp > elements.get(j-1)){
      Integer value = elements.get(j-1);
      elements.set(j,value);
      j--;
    }
    elements.set(j,tmp);
  }
}
```

Rúbrica:

1. (0,2 puntos) Método de ordenación indicado correctamente:
InsertionSort.

2. (0,1 puntos) Parámetro en el método correcto.
`public static void ordenationArrayList(ArrayList<Integer> elements){

3. (0,1 puntos) Uso de size() correcto.
```
for (int i=0; i<elements.size(); i++){
```

4. (0,1 puntos) tmp asignada correctamente con get.
```
int tmp = elements.get(i);
```

5. (0,3 puntos) Ordena de mayor a menor correctamente.
```
while (j>0 && tmp > elements.get(j-1))
```

6. (0,1 puntos) Asignación correcta:

```
Integer value = elements.get(j-1);
elements.set(j,value);
j--;
```

7. (0,1 puntos) Asignación correcta:
```
elements.set(j,tmp);
```