Grado en Ingeniería Informática

Estructura de Datos y Algoritmos, Grupo 81M, 2014/2015

11 de Marzo de 2015

Nombre y Apellidos:

….....

PROBLEMA 1 (1 punto) – Programación Orientada a Objetos.

Un club deportivo solicita el desarrollo de una aplicación para gestionar los carnet de sus socios.
Hay dos clases de carnet de socios. Los carnet para socios individuales y los carnet de socio para familias. Los datos a almacenar en un carnet de socio son los siguiente:

- Código de Socio: tipo String.
- Nombre del titular: String.
- En el caso de un carnet de familia, además de almacenar el nombre del titular, es necesario guardar los nombres de los demás miembros de la familia que serán usuarios del club.
- Lista de las actividades: Array de String. Un socio podrá apuntarse a un máximo de 5 actividades. En los carnet de socios familiares, el número máximo de actividades será igual a 5 por el número total de miembros de la familia registrados en el carnet.
- Número total de actividades inscritas.

Cada vez que un socio (individual o familiar) se registra a una actividad, el nombre de la actividad se añade a la lista de actividades, almacenada en el carnet de socio. Si el socio ya ha alcanzado el número máximo de actividades permitido (5 para socios individuales y 5*numMiembros para familias), no se registrará la nueva actividad y se informará por mensaje que se ha alcanzado el máximo número de actividades permitidas.

El carnet se utiliza principalmente para acceder a las distintas salas dónde se imparten las actividades (natación, spinning, pilates, yoga, etc). Cuando un socio intenta entrar a una determinada actividad, en el carnet se comprobará si dicha actividad está en la lista de actividades inscritas por el socio.

En el caso de la sauna, el acceso a este servicio, dependerá de la clase de socio. Si el carnet es de clase individual, sólo

se permitirá el acceso a la sauna, cuando el socio al menos está inscrito a 3 actividades y una de ellas es natación. Si el carnet es de clase familiar, se permitirá acceso a la sauna siempre y cuando el número de actividades registradas es superior a 5, o bien el número de miembros de la familia es mayor o igual que 3.

Se pide desarrollar la(s) estructura(s) de datos necesaria(s) para implementar la solución. En concreto, será necesario:

- método(s) constructor(es) que reciban el código y nombre del titular del socio. En el caso de ser un carnet de clase familiar, se deberá recibir también como parámetro el array con la lista de miembros de la familia.
- registrarActividad-> recibe como parámetro el nombre de la actividad y guarda dicha actividad en la lista de actividades del socio.
- tieneAcceso-> recibe como parámetro el nombre de la actividad a la que quiere entre el socio con el carnet. Devuelve true si el socio está apuntado a dicha actividad y false en otro caso.
- tieneAccesoSauna-> devuelve true si el socio tiene acceso y false en otro caso.

La solución propuesta debe estar diseñada bajos los principios de la programación orientada a objetos, que permitan obtener software robusto, reutilizable y fácil de adaptar.

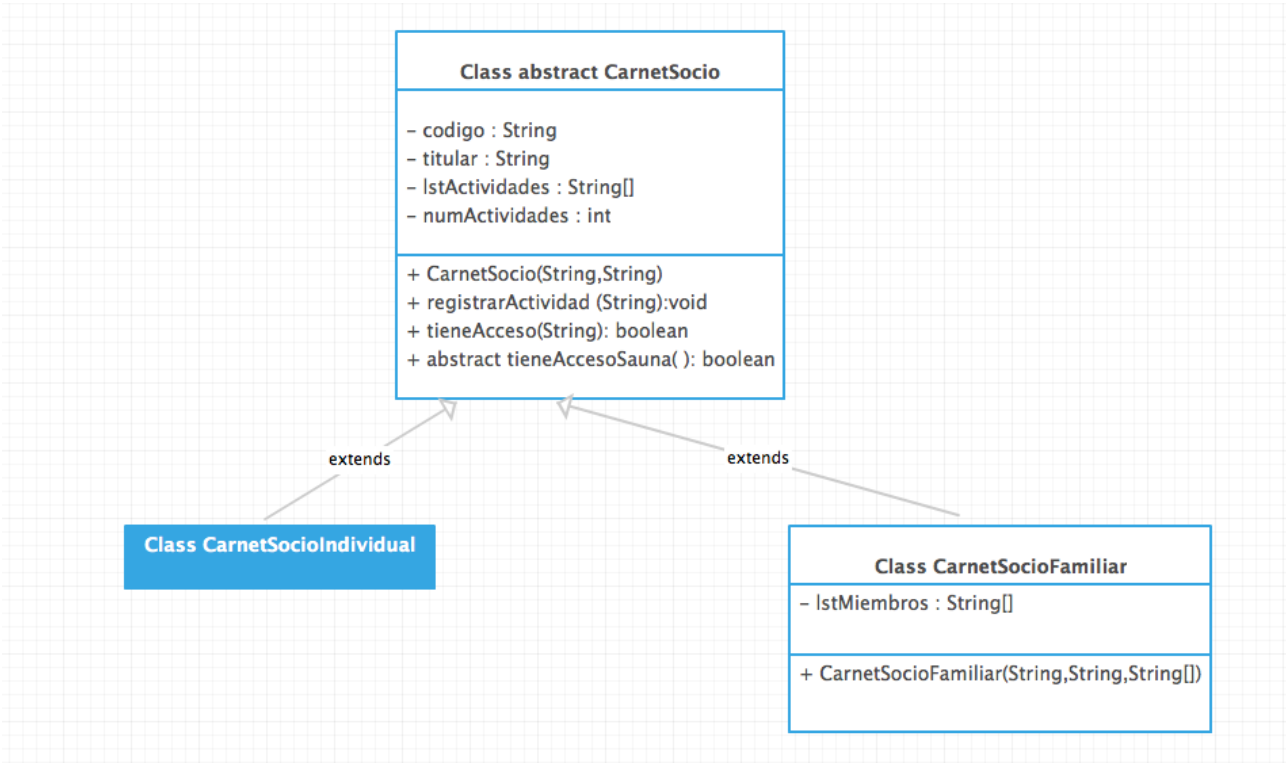
Criterios de evaluación:

- Diseño siga principios POO (0.25).
- Método constructores (0.20)
- registrarActividad() (0,15)
- tieneAcceso() (0,20)
- tieneAccesoSauna() (0,20)

Notas:

- No es necesario implementar los métodos getters and setters.
- No es necesario implementar un método main o una clase Test para probar la aplicación, es suficiente con diseñar la(s) clase(s).
- Para simplificar el problema, vamos a suponer que las actividades no se pueden eliminar de la lista de actividades del socio.

Solución:




```java
public abstract class CarnetSocio {
    String codigo;
    String titular;
    String[] lstActividades;
    int numActividades;

    public CarnetSocio(String codigo, String titular) {
        this.codigo=codigo;
        this.titular=titular;
        this.numActividades=0;
        lstActividades=new String[5];
    }

    public void registrarActividad(String sAct) {
        if (lstActividades.length==numActividades) {
            System.out.println("Ya se ha alcanzado el número "
                    + "máximo de actividades");
            return;
        }
        lstActividades[numActividades]=sAct;
        numActividades++;
    }
```

```
    public boolean tieneAcceso(String sAct) {
        for (int i=0;i<numActividades;i++) {
            if (lstActividades[i].equalsIgnoreCase(sAct)) return true;
        }
        return false;
    }

    public abstract boolean tieneAccesoSauna();
```

Clase CarnetSocioIndividual:

```
public class CarnetSocioIndividual extends CarnetSocio {

    public CarnetSocioIndividual(String codigo,String titular) {
        super(codigo,titular);
    }

    @Override
    public boolean tieneAccesoSauna() {
        if (this.numActividades<3) return false;
        for (int i=0;i<numActividades;i++) {
            if (lstActividades[i].equalsIgnoreCase("natación")) return true;
        }
        return false;
    }

}
```

Clase CarnetSocioFamiliar:

```
public class CarnetSocioFamiliar extends CarnetSocio {
    String[] miembros;
    int numMiembros;
    public CarnetSocioFamiliar(String codigo,String titular, String miembros[]) {
        super(codigo,titular);
        this.miembros=miembros;
        this.numMiembros=miembros.length;
        //En los carnet de socios familiares, el número máximo de actividades
        //será igual a 5 por el número total de miembros de la
        //familia registrados en el carnet.
        this.lstActividades=new String[5*(this.numMiembros+1)];
    }

    public boolean tieneAccesoSauna() {
        return (this.numActividades>5 ||this.numMiembros>=3);
    }

}
```

PROBLEMA 2 (1 punto)- Implementación de una lista de enteros, doblemente enlazada, sin usar nodos centinelas.

Dadas las siguientes clases:

```
package listaEntero;

public class ListaDobleInt {

DNodoInt prim;
DNodoInt ult;

public boolean esVacia() {
    return prim==null;
}

public void insertaPrincipio(int x) {
    DNodoInt nuevoNodo=new DNodoInt(x);
    if (esVacia()) {
        ult=nuevoNodo;
    } else {
        nuevoNodo.sig=prim;
        prim.ant=nuevoNodo;
    }
    prim=nuevoNodo;
}

public void borrarPrincipio() {
    if (!esVacia()) {
        prim=prim.sig;
        if (prim==null) ult=null;
        else prim.ant=null;
    }
}
```

Estas clases se corresponden con la implementación de una lista de enteros, doblemente enlazada, sin usar nodos centinelas. Dicho de otra forma, el atributo **prim** de la clase ListaDobleInt sería el primer nodo de la lista, mientras que el atributo **ult** sería el último nodo de la lista.

Se pide implementar los método insertaFinal(int x) y borraFinal().

Criterios de Evaluación:
- insertarFinal (0.5 puntos)
- borrarFinal (0.5 puntos)

Solución:

```
public void insertarFinal(int x) {
    DNodoInt nuevoNodo=new DNodoInt(x);
    if (esVacia()) {
        insertarPrincipio(x);
    } else {
        ult.sig=nuevoNodo;
        nuevoNodo.ant=ult;
        ult=nuevoNodo;
    }
}


public void borrarFinal() {
    if (!esVacia()) {
        ult=ult.ant;
        if (ult==null) prim=null;
        else ult.sig=null;
    }
}
```