Examen de Metodología de la Programación
Curso 2015-2016, Convocatoria Febrero

Sección Informativa: **Duración del examen: 2 horas**

Fórmula: Si (Teoria*50% + Sesiones Problemas*10%) >=3
Nota Acta = Sesiones Problemas *10% + Teoría*50% + Práctica*30%+ Entrega Practica*10%

1- (6.50 Puntos). C++. Dada la clase Profesor y la clase Departamento:

```
class Profesor {
  char *nombre;
  static float sueldo_base;
public:
  ...
  const char *getNombre() const { return nombre; }
  float nomina() const { return sueldo_base; }
  static float getSueldoBase() { return sueldo_base; }
  static void setSueldoBase(float su) { sueldo_base = su; }
  void ver() const { cout << nombre << ", sueldo base: " << sueldo_base; }
};

class Departamento {
private:
  Profesor **lista;
  int nmax; //capacidad
  int n; //num elementos

  int busca(char *nombre) const; //devuelve la posición donde esta o -1 si no existe
public:
  Departamento(); // constructor
  ~Departamento(); // destructor
  void agregar();
  void ver() const;
};
```

1. Implementa los destructores y constructores de la clase Profesor, teniendo en cuenta que el último campo, de tipo float, indica el sueldo base común a todos los profesores, por lo que sólo debe ser pedido como dato por teclado cuando se cree el primer profesor. **(1.25 puntos)**
   Ayuda: debes inicializar el sueldo_base a un valor que te permita saber si has dado ya de alta a alguien

2. Implementa el método agregar el cual debe crear un objeto Profesor y añadirlo al Departamento. Para ello el método *agregar* pedirá el nombre del profesor y buscará si ya existe uno en el Departamento con dicho nombre, en caso de no existir lo añadirá, en caso contrario mostrará un mensaje indicando que ya existe. En caso que el Departamento esté completo, el tamaño del Departamento debe duplicarse dinámicamente para poder añadir al nuevo profesor. **(1.50 puntos)**
   Ayuda: suponed que los métodos busca() y ver() ya están implementados

Tras la firma del I Convenio Colectivo, los sindicatos consiguen que los profesores con contrato fijo perciban una retribución adicional en sus nominas de 40 euros extras por cada trienio que tengan, por lo que la Universidad necesita modificar el programa informático para tener esto en cuenta.

3. A fin de reutilizar el software, haz que la clase Profesor **tenga comportamiento polimórfico** y crea a partir de dicha clase crea una clase derivada ProfesorTC que represente a los profesores con contrato fijo. La clase ProfesorTC debe generar automáticamente un código único y fijo (no puede cambiar) a cada profesor de este tipo de forma automática. Implementa todos sus métodos, reutilizando el máximo código posible en los métodos heredados, no rompiendo la encapsulación de las clases. **(2.00 puntos)**

4. Implementa el operator= tanto en la clase profesor como en la clase ProfesorTC() teniendo en cuenta que en este último debe copiar todo excepto el código, que al ser constante no puede ser modificado Implementa el operador reutilizando el máximo código posible en los métodos heredados. **(1.75 puntos)**

SOLUCION: (en rojo viene indicado la puntuación de cada apartado)

1. Implementa los destructores y constructores de la clase Profesor, teniendo en cuenta que el último campo, de tipo float, indica el sueldo base común a todos los profesores, por lo que sólo debe ser pedido como dato por teclado cuando se cree el primer profesor. **(1.25 puntos)**
   Ayuda: debes inicializar el sueldo_base a un valor que te permita saber si has dado ya de alta a alguien

```
float Profesor::sueldo_base = -1; //0.20  // no puede incluir la palabra clave static

Profesor::Profesor(char *nom) {   //0.40
   nombre = new char[strlen(nom) + 1]; //crea espacio
   strcpy(nombre, nom);                //copia nombre en el objeto
   if (sueldo_base == -1) {
      cout << "Indica el sueldo base: ";
      cin >> sueldo_base;
   }
}

/*
Profesor::Profesor(const Profesor &p): Profesor(p.nombre) {  //ESTA ES OTRA SOLUCION VALIDA
  //Profesor(p.nombre);  //debe ponerse en inicializadores
                         //da error en tiempo ejecucion
}
*/

Profesor::Profesor(const Profesor &p) {  //0.40
  nombre = new char[strlen(p.nombre) + 1]; //crea espacio
  strcpy(nombre, p.nombre);                //copia nombre en el objeto
                                           //sueldo base ya debe tener valor valido
}

Profesor::~Profesor() {  //0.25
  delete [] nombre;  //libera la memoria
}
```

2. Implementa el método agregar el cual debe crear un objeto Profesor y añadirlo al Departamento. Para ello el método *agregar* pedirá el nombre del profesor y buscará si ya existe uno en el Departamento con dicho nombre, en caso de no existir lo añadirá, en caso contrario mostrará un mensaje indicando que ya existe. En caso que el Departamento esté completo, el tamaño del Departamento debe duplicarse dinámicamente para poder añadir al nuevo profesor. **(1.50 puntos)**
   Ayuda: suponed que los métodos busca() y ver() ya están implementados

```
void Departamento::agregar() {  //1.50
  char nombre[30], resp;
  Profesor *p;
  cout << "Nombre: ";
  cin >> nombre;
  if (busca(nombre)==-1) { //lo agrego si no esta aun
    if (n==nmax) {
        Profesor **aux=lista;
        nmax*=2;
        lista=new Profesor *[nmax];
        for(int i=0; i<n; i++)
            lista[i]=aux[i];
        delete [] aux;
    }
    cout << "El profesor tiene contrato fijo (s/n)?: ";
    cin >> resp;
    if (resp=='s') {
      int tri;
      cout << "Trienios: ";
      cin >> tri;
      p = new ProfesorTC(nombre, tri);
    }
    else
      p = new Profesor(nombre);
    lista[n] = p;
    n++;
  }
  else
    cout << nombre << " ya existe en el Departamento\n";
}
```

Tras la firma del I Convenio Colectivo, los sindicatos consiguen que los profesores con contrato fijo perciban una retribución adicional en sus nominas de 40 euros extras por cada trienio que tengan, por lo que la Universidad necesita modificar el programa informático para tener esto en cuenta.

3. A fin de reutilizar el software, haz que la clase Profesor **tenga comportamiento polimórfico** y crea a partir de dicha clase crea una clase derivada ProfesorTC que represente a los profesores con contrato fijo. La clase ProfesorTC debe generar automáticamente un código único y fijo (no puede cambiar) a cada profesor de este tipo de forma automática. Implementa todos sus métodos, reutilizando el máximo código posible en los métodos heredados, no rompiendo la encapsulación de las clases. **(2.00 puntos)**

```
class Profesor {
  char *nombre;
  static float sueldo_base;
public:
  Profesor(char *nom); // constructor
  Profesor(const Profesor &p); //constructor de copia
  virtual ~Profesor(); // destructor  //0.10
  const char *getNombre() const { return nombre; }
  virtual float nomina() const { return sueldo_base; } //0.10
  static float getSueldoBase() { return sueldo_base; }
  static void setSueldoBase(float su) { sueldo_base = su; }
  virtual void ver() const { cout << nombre << ", sueldo base: " << sueldo_base; } //0.10
};

class ProfesorTC: public Profesor {
  int trienios;
  static int n;
  const int codigo;
public:
  ProfesorTC(char *nom, int tri);  //constructor
  ProfesorTC(const ProfesorTC &p); //constructor de copia
  int getCodigo() const { return codigo; }   //0.05
  int getTrienios() const { return trienios; } //0.05
  void setTrienios(int tri) { trienios = tri; } //0.05
  float nomina() const { return (Profesor::nomina() + trienios * 40); } //0.30
  void ver() const {
    Profesor::ver();
    cout << " (" << codigo << "), trienios: " << trienios << ", nomina: " << nomina();   //0.30
  }
  ProfesorTC & operator=(const ProfesorTC &p); //para el apartado 4
};

int ProfesorTC::n = 1; // no puede incluir la palabra clave static      //0.15

ProfesorTC::ProfesorTC(char *nom, int tri): Profesor(nom), codigo(n)  { //0.40
  n++;
  trienios=tri;
}

ProfesorTC::ProfesorTC(const ProfesorTC &p): Profesor(p), codigo(n++)  { //0.40
   trienios=p.trienios;
}
```

4. Implementa el operator= tanto en la clase profesor como en la clase ProfesorTC() teniendo en cuenta que en este último debe copiar todo excepto el código, que al ser constante no puede ser modificado Implementa el operador reutilizando el máximo código posible en los métodos heredados. **(1.75 puntos)**

```
void Profesor::setNombre(const char *nom) {
  delete [] nombre;
  nombre=new char[strlen(nom)+1];
  strcpy(nombre,nom);
}

Profesor & Profesor::operator=(const Profesor &p) { //0.75
  if (this != &p)
    setNombre(p.getNombre());
  return *this;
}

ProfesorTC & ProfesorTC::operator=(const ProfesorTC &p) { //1.00
  if (this != &p) {
    Profesor::operator=(p);
    trienios=p.trienios;
  }
  return *this;
}
```

2- **(3.50 Puntos). Java.** Dado el siguiente código indica cual sería la definición de las clases. Implementa el **mínimo número de constructores posibles** (para el resto de métodos sólo indicar sus prototipos), para que el programa genere la salida indicada:

```
package libProfesor;

public class ClaseA {
  private int [] tabla;
  private int n;

  //A RELLENAR POR EL ALUMNO (2.3 puntos)
}
```

```
package libProfesor;

public class ClaseB extends ClaseA {
  String cadena;

  //A RELLENAR POR EL ALUMNO (1.2 puntos)
}
```

```
package libPruebas;

import libProfesor.*;

public class Prueba1 {
  public static void main(String[] args) {
    ClaseA a=new ClaseA(5), b=new ClaseA(), c=new ClaseA();//a con los 5 1° n° impares, b y c vacios
    ClaseA x=new ClaseB("BBVA");                           //x vacio y cadena BBVA
    ClaseB y=new ClaseB("AVE", 2), z, j=new ClaseB(y);     //y,j con los 2 1° n° impares y cadena AVE
    c=a;
    z=y;
    ((ClaseB)x).setCadena("OHL");                          //x con la cadena OHL
    if (ClaseA.mayor(a,b))                                 //true si n de a es mayor que n de b
      System.out.println("a:" + a);
    System.out.println("a:" + a + "\nb:" + b + "\nc:" + c);
    System.out.println(c.get(4) + "," + j.get(0));
    System.out.println("x contiene la cadena:" + ((ClaseB)x).getCadena());
  }

}
```

Salida:

```
a:1-3-5-7-9
a:1-3-5-7-9
b:
c:1-3-5-7-9
9,1
x contiene la cadena:OHL
```

SOLUCION: (en rojo viene indicado la puntuación de cada apartado)

```
package libProfesor;

public class ClaseA {
  private int [] tabla;
  private int n;

//EN JAVA LOS ATRIBUTOS NUMERICOS SE INICIALIZAN A 0 Y LAS REFERENCIAS A NULL (LOS BOOLEAN A FALSE)
//SI NO CREAMOS NINGUN CONSTRUCTOR JAVA CREA UNO DE OFICIO QUE NO HACE NADA
//SI CREAMOS AL MENOS UN CONSTRUCTOR (POR EJ. UNO CON PARAMETROS) ENTONCES JAVA NO CREA EL DE OFICIO

  public ClaseA() {               //0.3
    //this.n=0;    //NO ES NECESARIO (JAVA inicializa a 0 los atributos numericos) PERO ES MEJOR PONERLO
    //tabla=null; //NO ES NECESARIO (JAVA inicializa a null los atributos referencias) PERO MEJOR PONERLO
  }

  public ClaseA(int n) {          //0.7        //SI CREAMOS ESTE CONSTRUCTOR ENTONCES DEBEMOS CREAR
    this.n=n;                                  //EL ANTERIOR SIN PARAMETROS YA QUE JAVA SOLO LO CREA
    tabla=null;  //NO ES NECESARIO             //SI NOSOTROS NO CREAMOS NINGUNO
    if (n>0) {                                 //POR TANTO EL ANTERIOR HAY QUE PONERLO AUNQUE DENTRO
      tabla=new int[n];                        //NO TUVIERA NADA DE CODIGO
      for(int i=0; i<n; i++)
        tabla[i]=i*2+1;
    }
  }

  public int getN() { return n; }       //0.3  (porque no aparece explicitamente...)
  static public boolean mayor(ClaseA a, ClaseA b) { return (a.n>b.n); }       //0.3
  public int get(int i) { return tabla[i]; }       //0.1

  public String toString() {       //0.6
    String s="";
    if (n>0) {
      for(int i=0; i<n-1; i++)
        s = s+ tabla[i]+ "-";
      s=s+tabla[n-1];
    }
    return s;
  }

}
```

```
package libProfesor;

public class ClaseB extends ClaseA {
  String cadena;

  public ClaseB(String cad) {         //0.3
    //super(); //NO ES NECESARIO, si no lo llamamos Java por defecto llama a super()
    cadena=cad;
  }

  public ClaseB(String cad, int n) {  //0.3
    super(n);  //la llamada al constructor padre debe ser la 1ª instruccion del constructor hijo
    cadena=cad;
  }

  public ClaseB(ClaseB b) {           //0.4
    super(b.getN());  //la llamada al constructor padre debe ser la 1ª instruccion del constructor hijo
    cadena=b.cadena;
  }

  public String getCadena() { return cadena; }          //0.1
  public void setCadena(String s) { cadena=s; }         //0.1

}
```