Examen de Metodología de la Programación
Curso 2013-2014, Convocatoria Febrero

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

- **Hay que tener las prácticas aprobadas para poder hacer el examen de teoría.**
- **La nota de teoría no se guarda. El resto de notas son válidas hasta Septiembre.**

Fórmula: Si (Teoria*50% + Sesiones Problemas*10%) >=3
Nota Acta = Sesiones Problemas *10% + Teoría*50% + Práctica*40%

1- (5.5 Puntos). C++. Un Club de Futbol utiliza las siguientes clases en un programa:

```
class Socio {
  const int num_socio;
  char *nombre;
  int edad;
  float cuota_anual;
  static float cuota[3];
public:
  Socio(char *nom, int edad); // constructor
  ~Socio(); //destructor
  int numSocio() { return num_socio; }
  static float getCuota(int i) { return cuota[i]; }
  static void setCuotas(float c1, float c2, float c3);
  //A RELLENAR POR EL ALUMNO

};

class Club {
  Socio **listaSocios; //inicialmente el nº máximo de socios será igual al indicado en el constructor
  int nls;             //nº elementos en listaSocios
  int nmaxls;          //nmaxls: capacidad de listaSocios
  int buscarSocio(int num_socio);     //devuelve la pos donde está en listaSocio, -1 si no está
public:
  Club(int cupoSocio); // constructor
  ~Club(); // destructor
  void alta();
  void baja(int num_socio);
  void ampliarCapacidad(int n);
  void verSocios();
};

int main() {
  Socio::setCuotas(10,20,15);
  Socio a("luis", 25), b("juan", 87), *c=new Socio("eva", 27);
  const Socio d("ana", 15);
  b=a; //copia los datos de a en b (excepto el nº socio)
  if (a==*c) cout << "a y c son iguales\n";
  cout << "d: " << d << "\nc: " << c->ver() << ", " << d.numSocio() << endl;
  if (Socio::joven(a, b)) cout << "a es mas joven que b\n";
  int diferencia_edad=restar(a,d);
  delete c;

  Club betis(2);
  betis.alta();
  betis.verSocios();
  betis.baja(1);
  betis.ampliarCapacidad(2);
  cout << "En el betis hay " << betis.nDiscapacitados(10,20) << " discapacitados entre 10 y 20 años\n";
  system("pause");  return EXIT_SUCCESS;
}
```

a) Implementa el constructor y destructor de la clase `Socio`, así como el método `setCuotas()` teniendo en cuenta que el número de socio debe ser único y generarse de forma automática, cada vez que un socio es dado de alta. El último campo (`cuota[3]`), de tipo float, indica la cuota que cada socio debe pagar según sea menor de edad, mayor de edad o jubilado (respectivamente). Estas cuotas deben estar establecidas antes de crearse algún socio, de forma que si no es así, deben ser pedidas como datos en el momento de crearse el primer socio. **(1.5 puntos)**

Nota: Si lo considera absolutamente necesario añade los atributos que estime oportunos a la clase Socio para la implementación del apartado a)

b) Dada la implementación de la clase `Socio`, corrige y completa la definición de dicha clase (enumerando aquellos métodos que son estrictamente necesarios) para que el `main()` pueda ejecutarse. **(1 punto)**

c) Implementa el método `ampliarCapacidad()` el cual debe permitir ampliar el cupo de Socios permitidos en tantas unidades como las indicadas por parámetro. **(1 punto)**

A fin de facilitar a las personas con algún tipo de discapacidad hacerse socios del club, los directivos del Club deciden crear una nueva modalidad de socio llamado SocioDiscapacitado cuya cuota anual tiene una reducción igual al porcentaje de discapacidad que presente la persona, aplicada sobre la cuota que tendría que pagar en función de su edad:

d) Haz que la clase Socio **tenga comportamiento polimórfico** (indica los cambios que hay que realizar) y crea una clase derivada `SocioDiscapacitado` con el atributo adicional `discapacidad`. Enumera los métodos de dicha clase e implementa el constructor y destructor. **(1 punto)**

e) Implementa un método nDiscapacitados(int edad1, int edad2) en la clase Club, que devuelva cuantos socios discapacitados de edad comprendida entre edad1 y edad2 (ambas inclusive) tiene el Club. **(1 punto)**

SOLUCION: En gris marcamos el por qué hay que usar const (hay que mirar el main()…)

```
int main() {
  Socio a("luis", 25), b("juan", 87), *c=new Socio("eva", 27);
  const Socio d("ana", 15); //el objeto d solo puede invocar métodos constantes al ser const
  ...
}
```

a) **(1.5 puntos)**

```
  Socio::Socio(const char *nom, int edad): num_socio(++n) {
    if (!(haycuota)) {
      float c1,c2,c3;
      cout << "Cuota menor edad: ";
      cin >> c1;
      cout << "Cuota mayor edad: ";
      cin >> c2;
      cout << "Cuota jubilado: ";
      cin >> c3;
      setCuotas(c1,c2,c3);
    }
    nombre=new char [strlen(nom)+1];
    strcpy(nombre, nom);
    this->edad=edad;
    if (edad<18) cuota_anual=cuota[0];
    else if (edad>65) cuota_anual=cuota[2];
    else cuota_anual=cuota[1];
  }

  virtual Socio::~Socio() { //virtual por la clase derivada SocioDiscapacitado
    delete [] nombre; // libera la memoria
  }

  void Socio::setCuotas(float c1, float c2, float c3) {
    cuota[0]=c1; cuota[1]=c2; cuota[2]=c3; haycuota=true;
  }
```

b) **(1 punto)**

```
class Socio {
  const int num_socio;
  char *nombre;
  int edad;
  float cuota_anual;
  static float cuota[3];
  static bool haycuota;
  static int n;
public:
  Socio(char *nom, int edad); // constructor
  ~Socio(); //destructor
  int numSocio() const { return num_socio; }
  static float getCuota(int i) { return cuota[i]; }
  static void setCuotas(float c1, float c2, float c3);
  //A RELLENAR POR EL ALUMNO
  //constructor copia de oficio NO funciona bien ya que hace copia binaria de los datos
  Socio(const Socio &s); //constructor de copia
  float getCuota_anual() const { return cuota_anual; }
  void setCuota_anual(float c) { cuota_anual=c; }
  int getEdad() const { return edad; }
  Socio& operator=(const Socio &s);
  bool operator==(const Socio &s) const;
  string ver() const;
  friend ostream& operator<<(ostream &st, const Socio &s);
  static bool joven(const Socio &a, const Socio &b);
};
int restar(const Socio &a, const Socio &b);

float Socio::cuota[3]; //lo declaro pero no lo inicializo a nada
int Socio::n=0;
bool Socio::haycuota=false;
```

c) **(1 punto)**

```
  void Club::ampliarCapacidad(int n) {
    Socio **aux=listaSocios;
    nmaxls+=n;
    listaSocios=new Socio*[nmaxls];
    for (int i=0; i<nls; i++)
      listaSocios[i]=aux[i];
    delete [] aux;
  }
```

d) **(1 punto)**

```
class Socio {
  const int num_socio;
  char *nombre;
  int edad;
  float cuota_anual;
  static float cuota[3];
  static bool haycuota;
  static int n;
public:
  Socio(char *nom, int edad); // constructor
  virtual ~Socio(); //destructor
  int numSocio() const { return num_socio; }
  static float getCuota(int i) { return cuota[i]; }
  static void setCuotas(float c1, float c2, float c3);
  //A RELLENAR POR EL ALUMNO
  float getCuota_anual() const { return cuota_anual; }
  void setCuota_anual(float c) { cuota_anual=c; }
  int getEdad() const { return edad; }
  Socio& operator=(const Socio &s);
  bool operator==(const Socio &s) const;
  virtual string ver() const;
  friend ostream& operator<<(ostream &st, const Socio &s);
  static bool joven(const Socio &a, const Socio &b);
};
int restar(const Socio &a, const Socio &b);

float Socio::cuota[3]; //lo declaro pero no lo inicializo a nada
int Socio::n=0;
bool Socio::haycuota=false;


class SocioDiscapacitado: public Socio {
  int discapacidad;
public:
  SocioDiscapacitado(char *nom, int edad, int d):Socio(nom, edad) {
    this->discapacidad=d;
    setCuota_anual(getCuota_anual()*(1-discapacidad/100.0));
  }

  ~SocioDiscapacitado(); //destructor (no es necesario implementarlo)
  SocioDiscapacitado (const SocioDiscapacitado &s); //constructor copia (no hace falta implementarlo)
  SocioDiscapacitado &operator=(const SocioDiscapacitado &s); //no hace falta implementarlo

  string ver() const {
    stringstream aux;
    aux << " " << discapacidad << "%";
    return Socio::ver()+aux.str();
  }
};
```

e) **(1 punto)**

```
  int Club::nDiscapacitados(int edad1, int edad2) const {
    int n=0;
    for (int i=0; i<nls; i++) {
      if (typeid(*listaSocios[i])==typeid(SocioDiscapacitado)) {
        int e=listaSocios[i]->getEdad();
        if (e>=edad1 && e<=edad2)
          n++;
      }
    }
    return n;
  }
```

2- (3 Puntos). Java. Codificar en Java reutilizando el máximo código posible en los métodos heredados, no rompiendo la encapsulación de las clases. Considerense implementados los get y sets de las variables que se quieran acceder. Ejemplo : variable int x, int getx();void setx(int x).

a) Cree la clase ***Asignatura*** con un código (un entero), una calificación (un real) y un estado Evaluada (boolean) que por defecto es false. Un metodo *toString()* como en el ejemplo: Salida → "Asignatura 01 >> 4.5"

b) Cree una clase ***Alumno*** que contenga *nombre* y *apellido1* (String) , *dirección* (String) e *id* (int) y una lista de asignaturas con un parámetro en el constructor que indica el número máximo de asignaturas (si se utiliza Array list u otra estructura dinamica omitir este número).

Contiene un método *String toString()* que devuelve id nombre apellido + la lista de asignaturas

c) Cree una lista de alumnos en la clase Alumno (variable de clase) e inicialícela a 10 posiciones. Cree un método de clase ***Alumno buscarAlumno(int posicion)***, que devuelva el alumno que se encuentra en la posición indicada de la lista.

d) Haga dos clases que hereden de Alumno, ***Erasmus*** , que añade *pais* (String) como atributo y ***Standard*** que añade apellido2 (String).

Eramus implementa *toString* como" [ERASMUS] pais id nombre apellido" + lista de asignaturas
Standard lo implementa como "[STANDARD] apellido2 id nombre apellido " + lista de asignaturas

e) Cree una Clase ***ExcepcionNoexiste*** que devuelva en el método *toString()* el mensaje "El alumno n no existe o se sale del rango" cuando no esté ocupada la posición o se salga del rango creado, siendo n el número del alumno que se busca. Ejemplo: se busca el alumno -1 → " El alumno -1 no existe o se sale del rango"

f) Hacer un programa de prueba que pruebe los puntos a, b, c ,d y e. Para el d) y el e) se debe buscar primero una posición correcta y después una no existente. Debe mostrar la descripción del alumno y Si no existe sacar un mensaje por pantalla indicado que la posición no está llena usando para ello la captura de la excepción creada