Objetos en C++

1. Definiciones.

Programación Orientada a Objetos: Es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas informáticos.

Clase: Es la definición de una entidad real o abstracta que reúne aquellas características (propiedades) y comportamientos (métodos) que han sido considerados como útiles para el propósito con el cual se ha concebido la clase.

Objeto: Es una instancia de una clase, es la materialización de una definición de una clase.

Método: Corresponde a las acciones que ejecuta un objeto cuando recibe un mensaje. En lenguaje C, un método corresponde a una función miembro de la clase.

Propiedad: Es una característica que se a considerado como útil en la definición de una clase. En lenguaje C corresponde a una variable.

Mensaje: Es la forma de comunicarse con un objeto. En lenguaje se, corresponde a la llamada a una función miembro de la clase.

Miembros: Son las propiedades y métodos declarados para una clase.

Encapsulamiento: En líneas generales, cuando se define una clase se está encapsulando (incorporando) en ella aquellas propiedades y métodos considerados como útiles de definir. El encapsulamiento permite restringir el acceso a sus miembros, de tal forma proporcionar al medio externo (programas que hace uso de la clase) de un conjunto limitado de propiedades y métodos.

Herencia: Es la capacidad que tiene una clase de adoptar características y comportamientos pertenecientes a otra clase.

Polimorfismo: Es la capacidad que tiene un objeto de comportarse de forma diversa de acuerdo a ciertas circunstancias.

2. Declaración de una Clase.

El siguiente código, define la clase Rectángulo, declarando las propiedades `Alto` y `Ancho` y los métodos `Area()` y `Perimetro()`.

```
01 class Rectangulo {
02   public:
03     int Alto, Ancho;
04     int Area();
05     int Perimetro();
06 };
07
08 int Rectangulo::Area() {
09     return Alto * Ancho;
10 }
11
12 int Rectangulo::Perimetro() {
13     return Alto * 2 + Ancho * 2;
14 }
```

Predeterminadamente, los miembros de la clase (propiedades y métodos) son declarados como privados; es decir, el alcance que estos tienen se limita al bloque de código que compone la declaración de la clase (líneas 01 a la 06). Es por ello que en la línea 02 se ha añadido la cláusula **`public:`** la cual hace que todos los miembros declarados a continuación sean visibles desde el exterior de la clase.

```
01 #include <stdio.h>
02 #include <conio.h>
03
04 class Rectangulo {
05   public:
06     int Alto, Ancho;
07     int Area();
08     int Perimetro();
09 };
10
11 int Rectangulo::Area() {
12   return Alto * Ancho;
13 }
14
15 int Rectangulo::Perimetro() {
16   return Alto * 2 + Ancho * 2;
17 }
18
19 int main() {
20   Rectangulo Forma;
21   Forma.Alto  =  5;
22   Forma.Ancho = 10;
23   printf("Area: %d \n", Forma.Area());
24   printf("Perímetro: %d \n", Forma.Perimetro());
25 }
```

Como se puede observar, la declaración del objeto Forma (línea 19) se realiza como si se declarara una variable cualquiera, anteponiendo el tipo de dato al nombre de la variable (en este caso el nombre de la clase). Como todos los miembros de la clase son públicos, es posible acceder a las propiedades y ejecutar los métodos de forma directa (Líneas 21 a la 24).

3. Encapsulamiento.

Establecer como públicas las propiedades y método hace que el programa que esté haciendo uso de la clase pueda realizar operaciones inapropiadas, como asignar un valor negativo a la altura del rectángulo. Es por ello que el encapsulamiento permite proteger aquellos miembros (propiedades y métodos) que sean de uso interno y/o que se requiera validar el valor que contengan. El encapsulamiento ofrece una coraza que oculta los miembros del medio exterior. Sin embargo, una clase con miembros ocultos no permiten mantener una comunicación (mensaje) entre el objeto y el medio externo. En el ejemplo anterior, fue necesario declarar los miembros de la clase como públicos mediante el uso de la palabra **`public`**: Para declarar miembros ocultos o privados, se emplea la palabra **`private`**:

```
01  #include <stdio.h>
02  #include <conio.h>
03
04  class Rectangulo {
05    public:
06      int Area();
07      int Perimetro();
08    private:
09      int Alto, Ancho;
10  };
11
12  int Rectangulo::Area() {
13    return Alto * Ancho;
14  }
15
16  int Rectangulo::Perimetro() {
17    return Alto * 2 + Ancho * 2;
18  }
19
20  int main() {
21    Rectangulo Forma;
22    clrscr();
23    Forma.Alto  = 5;
24    Forma.Ancho = 10;
25    printf("Area: %d \n", Forma.Area());
26    printf("Perímetro: %d \n", Forma.Perimetro());
27    getch()
28  }
```

En el programa anterior, se ha añadido la palabra **`private`** en la línea 08. Esto cierra la declaración de los miembros públicos declarados anteriormente y señala que los miembros declarados a continuación son privados (visibles solamente dentro de la clase). Por lo tanto, las líneas 23 y 24 generan un error, debido a que las propiedades (variables) Alto y Ancho no son visibles desde el programa principal. Sin embargo, en las líneas 13 y 17 las propiedades Alto y Ancho sí son visibles, ya que están incluidas dentro de los métodos que componen la clase.

En el caso de las propiedades, para poder acceder a ellas desde el exterior, es necesario crear métodos que cumplan la función de interfaz. Es por ello que el siguiente código añade 4 métodos que permiten asignar valores a las propiedades y obtener los valores almacenados en las ellas:

```
01 #include <stdio.h>
02 #include <conio.h>
03
04 class Rectangulo {
05   public:
06     int  Area();
07     int  Perimetro();
08     void SetAlto(int Valor);
09     void SetAncho(int Valor);
10     int  GetAlto();
11     int  GetAncho();
12   private:
13     int Alto, Ancho;
14 };
15
16 int Rectangulo::Area() {
17   return Alto * Ancho;
18 }
19
20 int Rectangulo::Perimetro() {
21   return Alto * 2 + Ancho * 2;
22 }
23
24 void Rectangulo::SetAlto(int Valor) {
25   if (Valor < 0)
26       Alto = 0;
27   else
28       Alto = Valor;
29 }
30
31 void Rectangulo::SetAncho(int Valor) {
32   if (Valor < 0)
33       Ancho = 0;
34   else
35       Ancho = Valor;
36 }
37
38 int Rectangulo::GetAlto() {
39   return Alto;
40 }
41
42 int Rectangulo::GetAncho() {
43   return Ancho;
44 }
45
46 int main() {
47   Rectangulo Forma;
48   clrscr();
49   Forma.SetAlto(5);
50   Forma.SetAncho(10);
51   printf("Área: %d \n", Forma.Area());
52   printf("Perímetro: %d \n", Forma.Perimetro());
53   printf("Alto: %d \n", Forma.GetAlto());
54   printf("Ancho: %d \n", Forma.GetAncho());
55   getch()
56 }
```

En el programa anterior se añadieron los métodos SetAlto(), SetAncho(), GetAlto y GetAncho(). En el léxico informático, generalmente se utiliza la palabra Set cuando se asigna un valor a una propiedad y Get cuando se obtiene un valor desde una propiedad, es por ello que el nombre de estos métodos hace uso de estas palabras (aunque usted puede llamarlos como desee). Además se modificaron las líneas en donde se asignaban los valores a las propiedades (líneas 49 y 50) y se añadieron 2 líneas para mostrar los valores almacenados en las propiedades (líneas 53 y 54).

Ejercicio 1.

Añada a la clase Rectángulo las propiedades Fila y Columna las que corresponden a la posición en pantalla donde se desplegará el rectángulo (en el rectángulo corresponderá a la esquina superior izquierda de éste). Además, se deberá implementar el método Dibujar().

- **Fila**: Debe contener un valor entre 0 y 25.
- **Columna**: Debe contener un valor entre 0 y 79.

El método Dibujar() deberá desplegar en pantalla el rectángulo, de acuerdo al alto y ancho especificado en las propiedades respectivas. Si el rectángulo sobrepasa los límites establecidos (25 y 79) se deberá recortar el rectángulo para que quede visible dentro de la pantalla. Por ejemplo: Si el rectángulo se despliega en la fila 20, columna 70 y tiene un alto de 10 (caracteres) y un ancho de 20 (caracteres), se deberá desplegar un rectángulo de 6 de alto por 10 de ancho.

Ejercicio 2.

Añadir al Ejercicio 1 las propiedades Color_Linea, Estilo_Linea, Color_Relleno y Estilo_Relleno:

- **Color_Linea:** Define el Color de la línea definida para el trazo del contorno del rectángulo, puede tener uno de los siguientes valores.

  0 = Negro
  1= Azul
  2= Verde
  3= Aguamarina
  4= Rojo
  5= Púrpura
  6= Amarillo
  7= Blanco
  8= Gris
  9= Azul Claro
  A= Verde Claro
  B= Aguamarina Claro
  C= Rojo Claro
  D= Púrpura Claro
  E= Amarillo Claro
  F= Blanco Brillante

  Para aplicar el color, es necesario emplear la biblioteca: stdio.h. Además, se debe emplear la función system como system("color fondoletra"). Por ejemplo: system("color 06") indica que el color de fondo es negro "0" y el color de la letra es amarillo "6".

- **Estilo_Linea**: Define qué carácter se empleará en el trazo del contorno del rectángulo, en donde:

  0: Sin líneas.
  1: Línea Simple. (Ver tabla Ascii)
  2: Línea Doble. (Ver tabla Ascii)
  3: Doble Horizontal, Simple Vertical (Ver tabla Ascii)
  4: Simple Horizontal, Doble Vertical (Ver tabla Ascii)
  5: Asterisco.

- **Color_Relleno:** Define el Color de relleno o fondo del rectángulo.
- **Estilo_Relleno:** Define el carácter empleado como relleno del rectángulo. Éste puede ser cualquier carácter.

4. Constructores.

Los constructores son métodos miembros de la clase que se ejecutan automáticamente cuando se genera una instancias de la clase (se crea el objeto), esto es: cuando se declara la variable objeto. Tienen por propósito realizar cualquier operación necesaria inmediatamente creado el objeto, generalmente para inicializar las propiedades (variables).

Un constructor se declara de forma muy similar a un método, pero no se especifica tipo de dato a retornar y lleva por nombre el de la clase, además de declararlo con alcance público (public:).

El siguiente fragmento de programa muestra cómo declarar un constructor (se han omitido la escritura del resto de los métodos miembros):

```
01  #include <stdio.h>
02  #include <conio.h>
03
04  class Rectangulo {
05    public:
06      int  Area();
07      int  Perimetro();
08      void SetAlto(int Valor);
09      void SetAncho(int Valor);
10      int  GetAlto();
11      int  GetAncho();
12      Rectangulo();
13    private:
14      int Alto, Ancho;
15  };
16
17  Rectangulo::Rectangulo() {
18    Alto  = 0;
19    Ancho = 0;
20  }
21
22  int main() {
23    Rectangulo Forma;
24    clrscr();
25    Forma.SetAlto(5);
26    Forma.SetAncho(10);
27    printf("Area: %d \n", Forma.Area());
28    printf("Perímetro: %d \n", Forma.Perimetro());
29    printf("Alto: %d \n", Forma.GetAlto());
30    printf("Ancho: %d \n", Forma.GetAncho());
31    getch()
32  }
```

Observe que se ha declarado el método constructor en la línea 12 ocupando el mismo nombre de la clase y sin anteponer tipo de dato alguno. Luego, en las líneas 17 a 20, se implementa el método constructor. Finalmente, cuando se genera una instancia de la clase (se crea el objeto) en la línea 23, se ejecuta automáticamente el método constructor.

5. Destructores.

Un destructor es un método que se ejecuta automáticamente cuando se elimina el objeto y tiene por propósito, entre otros, liberar memoria de estructuras creadas dinámicamente (por ejemplo: listas enlazadas con punteros).

Un destructor se declara similar a un constructor, pero anteponiendo el símbolo ~

Ejemplo: El siguiente fragmento de programa muestra cómo declarar un constructor y un destructor para que pueda verificar su funcionamiento:

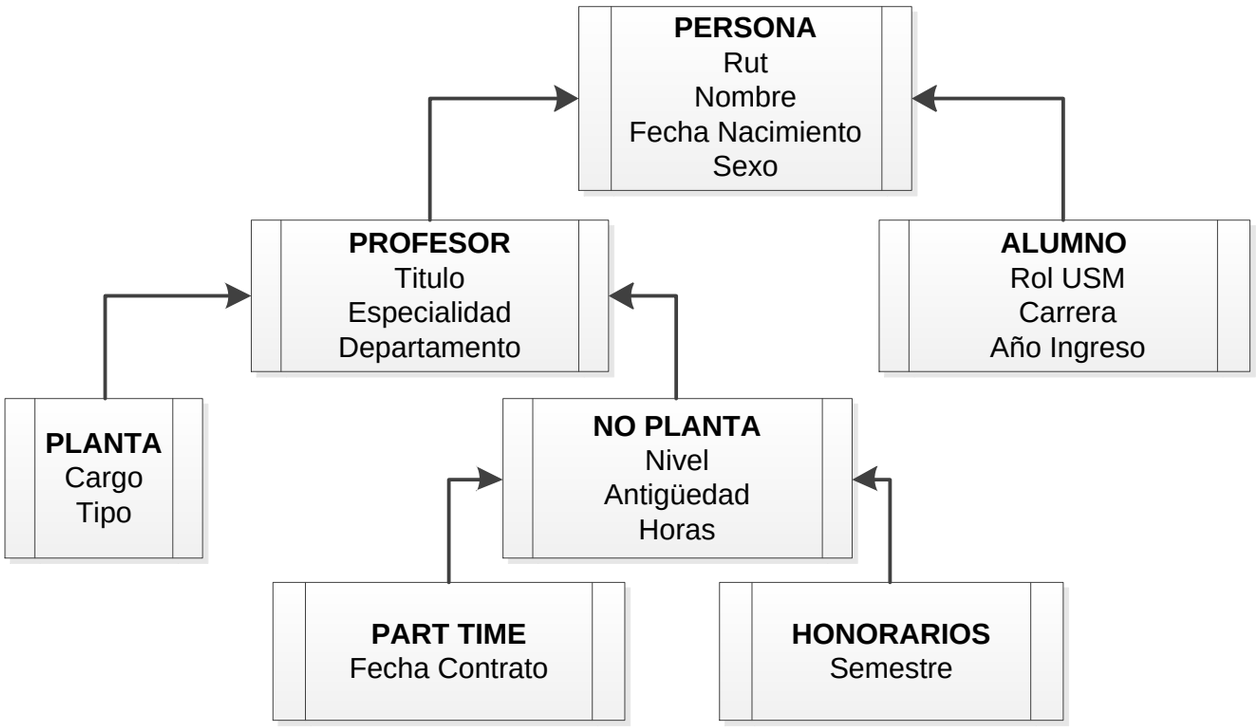
```
01  #include <stdio.h>
02  #include <conio.h>
03
04  class Ejemplo {
05    public:
06      Ejemplo();
07     ~Ejemplo();
08  };
09
10  Ejemplo::Ejemplo () {         // Constructor.
11    printf("Se ha ejecutado el método Constructor \n");
12    getch();
13  }
14
15  Ejemplo::~Ejemplo () {        // Destructor.
16    printf("Se ha ejecutado el método Destructor \n");
17    getch();
18  }
19
20  int main() {
21    Ejemplo X;
22  }
```

En el código anterior se muestra un ejemplo de la utilización de un constructor (líneas 10 a 13) y un destructor (líneas 15 a la 18).

El programa principal (líneas 20 a 22) solamente declara el objeto llamado X como una instancia de la clase Ejemplo. Como se puede apreciar, el resultado es el mensaje a desplegar en ambos casos.

6. Herencia.

La herencia consiste en adoptar características de una clase (clase base) para ser incorporadas por la clase que la adopta (clase derivada). El siguiente diagrama ilustra la herencia a partir de la entidad PERSONA:



Para implementar este esquema de clases, es necesario aclarar en más detalle el alcance de los miembros de una clase. Hasta ahora hemos visto 2 tipos de alcance (visibilidad) de los miembros de una clase: el alcance Público (**`public`**:) y Privado (**`private`**:). Recordemos que los miembros públicos son visibles desde el exterior de la clase (programa y módulos que hacen uso de los objetos declarados como instancias de la clase, esto es muy parecido a las variables globales) y en el caso de los miembros privados su alcance se limita al interior de la clase (muy parecido a las variables locales). Ahora bien, para poder heredar características de una clase base, es necesario declarar los miembros como protegidos (**`protected`**:) el cual tiene un alcance parecido al privado con la diferencia que este último no es visibles en las clases derivadas, en cambio los que son declarados como protegidos sí lo son.

Por otra parte, los miembros heredados pueden pasar a conservar el alcance con el que se habían declarado en la clase base o pueden heredarse para pasar a ser miembros privados de la clase derivada.

Los constructores y destructores de las clases derivadas se ejecutan posteriormente a la ejecución de los constructores y destructores de la clase base.

En un programa escrito en lenguaje C++, el diagrama anterior quedaría como se muestra a continuación:

```
01  #include <stdio.h>
02  #include <conio.h>
03
04  class PERSONA {
05    public:    PERSONA();
06               void SetSexo(char xSexo);
07               char GetSexo();
08    protected: char Rut[13];
09               char Nombre[30];
10               char Sexo;
11               int  Fec_Nac;
12    private:   int Fec_Defuncion;
13  };
14
15  PERSONA::PERSONA() {
16    Fec_Nac       = 0;
17    Fec_Defuncion = 0;
18  }
19
20  void PERSONA::SetSexo(char xSexo) {
21    Sexo = xSexo;
22  }
23
24  char PERSONA::GetSexo() {
25    return Sexo;
26  }
27
28  class ALUMNO : public PERSONA {
29    protected: char Rut_USM[13];
30               int  Cod_Carrera;
31               int  Anno_Ingreso;
32  };
33
34  class PROFESOR : public PERSONA {
35    protected: char Titulo[60];
36               char Especialidad[60];
37               int  Cod_Depto;
38  };
39
40  class PLANTA : public PROFESOR {
41    protected: char Cargo[20];
42               char Tipo[10];
43  };
44
45  class NO_PLANTA : public PROFESOR {
46    protected: int Nivel;
47               int Antiguedad;
48               int Horas;
49  };
50
51  class PART_TIME : public NO_PLANTA {
52    protected: int Fec_Contrato;
53  };
54
55  class HONORARIO : public NO_PLANTA {
56    protected: int Semestre;
57  };
58
59  int main() {
60    PART_TIME PartTime;
61    PartTime.SetSexo('M');
62    printf("Sexo: %c",PartTime.GetSexo());
63    getch();
64  }
```