POO

Características:
- Mantención
- Extensión
- Reutilización

Conceptos fundamentales:
- Clase
- Objeto
- Instancia
- Atributos
- Métodos

Un objeto consta de:
- Tiempo de vida
- Estado (definido por sus atributos)
- Comportamiento (definido por sus métodos)

Cada objeto es responsable de inicializarse y destruirse correctamente

Interfaz: Mecanismo mediante el cual un objeto se comunica con el medio, se materializa a través de los métodos públicos de la clase.

Encapsulamiento: Proceso mediante el cual se ocultan las estructuras de datos y los detalles de implementación.
- Public: métodos de cualquier clase
- Protected: métodos de la clase y sus derivadas
- Private: métodos de la clase

Destructores:
- ~NombreClase
- No tiene ningún retorno ni parámetros
- No pueden ser sobrecargados
- Se emplean para liberar los recursos solicitados por el constructor
- Solo se requiere si se usa almacenamiento dinámico
- Se invocan implícitamente cuando finaliza el bloque declarado por el objeto

Sobrecarga de operadores

Características:
- Se pueden redefinir algunos operadores existentes en C++ para los objetos de una clase determinada
- Se utiliza para simplificar el código a escribir

- La definición de la clase será más compleja pero más sencilla de usar

Operadores sobrecargables:
- Casi todos los operadores unitarios o binarios
- El operador de llamado a funcion ()
- No se pueden sobrecargar: (.) (?:) (sizeof) (::) (*.)

No se puede modificar:
- La gramática de un operador
- La cardinalidad de un operador

Herencia

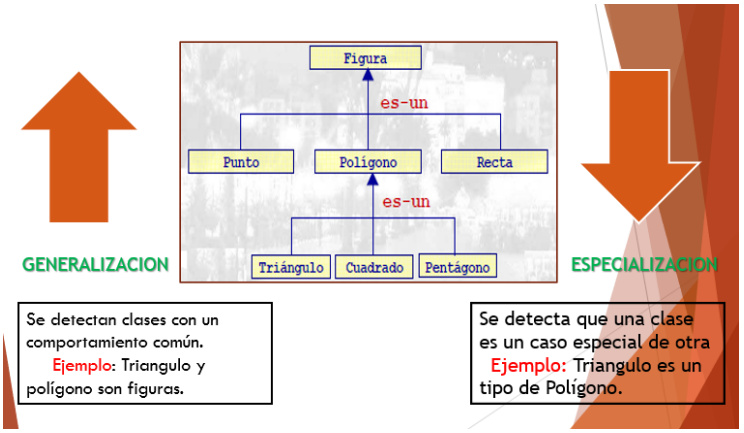
Características:
- Se expresa como una relación de descendencia
- Consiste en definir una nueva clase a partir de una existente
- Es transitiva (se puede heredar sucesivamente)
- La nueva clase se denomina **subclase** o **clase derivada**
- La clase existente se denomina **superclase** o **clase base**
- Es la propiedad que permite a los ejemplares de una subclase acceder a los miembros de la superclase
- Los métodos heredados se ejecutan más lentamente que el código especializado
- Puede ser **simple** o **múltiple**

Características de las subclase:
- Heredan tanto los métodos como los atributo de la superclase
- Tienen todas las propiedades de la superclase y otras más (extensión)
- Constituye una especialización de la superclase (reducción)
- Un método de superclase es anulado por un método con el mismo nombre definido en la subclase
- Un **constructor** de la subclase siempre invoca primero al de la superclase
- Un **destructor** de la subclase se ejecuta antes del destructor de la superclase

Los elementos de la superclase que no pueden ser heredados son:
- Constructores
- Destructores
- Funciones y datos estáticos de la clase
- Operador de asignación sobrecargado

Constructores de las clases derivadas:
- Debe llamarse al constructor de la clase base
- Se debe especificar un inicializador base
- Se puede omitir si la clase base cuenta con un constructor por defecto

```
C_CuentaJoven(const char *unNombre, int
  laEdad, double unSaldo=0.0, double
  unInteres=0.0): C_Cuenta(unNombre,
  unSaldo, unInteres)
```



Polimorfismo

Características:

- Son funciones distintas con el mismo nombre declaradas de forma virtual en la superclase (ligadura dinámica)
- Las funciones convencionales se invocan en el tiempo de compilación
- Las funciones virtuales se resuelven en tiempo de ejecución

Funciones virtuales

```
class A {
   public:
       virtual void mostrar();
}

class B: public A {
   public:
       void mostrar();
}
```

```
A objA;
B objB;

A* ptrA1;
A* ptrA2;

ptrA1 = &objA;
ptrA2 = &objB;

ptrA2->mostrar();
```

Clases abstractas:
Son superclases que no se instancian directamente, su función es darle estructura o forma a sus subclases

Composición de clases

Características:

- Se manifiesta como una relación de pertenencia
- Consiste en declarar objetos de una clase A como atributos de otra clase B
- El constructor de una clase que contiene objetos de otra llamará a los correspondientes constructores

- Un constructor por defecto de la clase llamara implícitamente a los constructores por defectos de los objetos declarados como atributos

```
class Curso {
   public:
      Curso(int t=30);
      void Inscribir(Alumno&);
      void Listar();
      double Promedio();
      int Aprobados();
      int Reprobados();
   private:
      int N;
      char nomCur[25];
      char codCur[7];
      Alumno v[50];
};
```

```
Alumno::Alumno() {
   k = 0;
   t = 0;
}
Alumno::Alumno(char *n, char *r, int m, int c) {
   nom = new char[strlen(n)+1];
   rut = new char[strlen(r)+1];
   strcpy(nom, n);
   strcpy(rut, r);
   mat = m;
   carrera = c;
   k = 0;
   t = 0;
}
```

```
void Curso::Listar() {
   for (int i=0; i<N; i++)
      v[i].Listar();
}
```

```
int main() {
   Curso C;
   C.Listar();
   return 0;
}
```

Clases genéricas o parametrizadas

Un template es un patrón para crear funciones y clases

Propósito:

- Evitar escribir múltiples versiones de la misma función para llevar a cabo la operación con distintos tipos de datos.
- El parámetro T representa un tipo de dato

Funciones Templates:

```
template <class TParam>
void Swap( TParam & X, TParam & Y )
{
  TParam temp = X;
  X = Y;
  Y = temp;
}
```

Clases Templates:

- Los templates de clases permiten crear nuevas clases con tipos de datos no definidos
- Usamos plantillas de clases cuando tenemos que crear múltiples clases con los mismos atributos y operaciones

```
template <class T>
class vector{
  private:
     T *V;
     int tam;
  public:
   vector(int);
   vector(const vector&);
   vector operator+(vector x);
   vector operator-(vector x);
   void operator=(vector x);
   T& operator[](int i);
   ~vector();
};
```

```
template<class T>
vector<T>::vector(int t){
 tam=10;
 V=new T [tam];
}

template<class T>
T &vector<T>::operator[](int i){
  return V[i];
}
```

```
int main(){
 vector<int> a(10),b(10),c(10);
...
 vector<float> af(10),bf(10),cf(10);
...
 vector<racional> r1(10),r2(10),r3(10);
```

Manejo de excepciones

Las excepciones son errores o situaciones anómalas que se producen durante el tiempo de ejecución, si ocurren y no se ha implementado el manejo de excepciones el programa terminará abruptamente. La implementación de esto aumentará la calidad del programa.