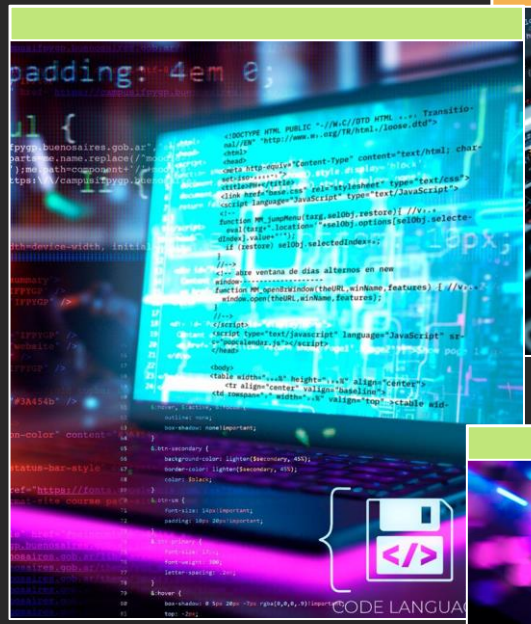
Programacion II [PRACTICA]

1 0 0 1 0 1 0 1 0 1 0 1 0 0

1 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0

Programación Orientada a Objetos [P.O.O]



1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 1 1 0

1 0 1 0 1 0 1 1 1 0 0 0 1 1 1 0

POLIMORFIMOS - FACTS

El polimorfismo es una característica de la POO que permite que un objeto se comporte de diferentes maneras dependiendo del contexto. En C++, esto se logra mediante clases base y clases derivadas

Con el polimorfismo podemos diseñar e implementar sistemas que puedan extenderse con facilidad; pueden agregarse nuevas clases con sólo modificar un poco (o nada) las porciones generales del programa, siempre y cuando las nuevas clases sean parte de la jerarquía de herencia que el programa procesa en forma genérica.

POLIMORFISMO - BENEFICIOS

Flexibilidad: Resalta cómo el polimorfismo hace que el código sea más flexible, permitiendo que se añadan nuevas clases derivadas sin necesidad de cambiar la clase base.

Reutilización de código: Puedes reutilizar el mismo código en la clase base mientras permites que las clases derivadas personalicen su comportamiento.

EJEMPLO – CLASE Animal

```
1  class Animal {
2  public:
3      virtual void hacerSonido() {
4          cout << "El animal hace un sonido." << endl;
5      }
6  };
7
```

EJEMPLO CLASE HIJA Perro

```
1  class Perro : public Animal {
2  public:
3      void hacerSonido() override {
4          cout << "El perro ladra." << endl;
5      }
6  };
```

EJEMPLO CLASE HIJA Gato

```
1  class Gato : public Animal {
2  public:
3      void hacerSonido() override {
4          cout << "El gato maúlla." << endl;
5      }
6  };
```

POLIMORFISMO – PALABRA CLAVE

```
    virtual void hacerSonido() {
        cout << "El animal hace un sonido." << endl;
    }
};
```

Virtual: El uso de la palabra clave virtual en el método hacerSonido() indica que este método es **virtual** o "sobrescribible" en clases derivadas. En otras palabras, cuando se usa un puntero o referencia a la clase base (en este caso, Animal), y se llama a hacerSonido(), C++ determinará cuál es la implementación adecuada de este método según el tipo real del objeto al que apunta el puntero (polimorfismo en tiempo de ejecución).

POLIMORFISMO – PALABRA CLAVE

```
void hacerSonido() override {
    cout << "El perro ladra." << endl;
}
```

```
void hacerSonido() override {
    cout << "El gato maúlla." << endl;
}
```

El uso de la palabra clave override en el método hacerSonido() indica que ese método está sobrescribiendo un método virtual de la clase base. Es una forma explícita de decirle al compilador que esta función en la clase derivada está diseñada para reemplazar una función virtual definida en una clase base.

EJEMPLO USO POLIMORFISMO

```
1   int main() {
2       Animal* animal;
3       Perro perro;
4       Gato gato;
5
6       // Animal apunta a un perro
7       animal = &perro;
8       animal->hacerSonido(); // Imprime "El perro ladra."
9
10      // Animal apunta a un gato
11      animal = &gato;
12      animal->hacerSonido(); // Imprime "El gato maúlla."
13  }
```

Retos de Programacion

01 Animales y Sonidos:

Crea una clase base llamada Animal que tenga un método virtual llamado hacerSonido().
Luego, crea tres clases derivadas: Perro, Gato, y Vaca, que sobrescriban el método hacerSonido() con los sonidos que corresponden a cada animal.

Objetivo:

- Utilizar punteros o referencias de tipo Animal para almacenar objetos de diferentes clases derivadas.

- Implementar el polimorfismo para invocar el método hacerSonido() y que cada objeto haga el sonido adecuado.

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

02 Figuras Geométricas

Diseña una clase base Figura que tenga un método virtual puro calcularArea(). Luego, crea clases derivadas como Circulo, Rectangulo y Triangulo. Cada clase derivada debe implementar el método calcularArea() según la fórmula de área correspondiente.

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

03 Vehículos en Movimiento

Define una clase base Vehiculo que tenga un método virtual llamado mover(). Luego, crea clases derivadas como Coche, Bicicleta y Avion, donde cada una implemente el método mover() con su propio comportamiento.

Objetivo:

- Implementar polimorfismo para simular el movimiento de diferentes tipos de vehículos.

04 Sistema de Pago

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

Crea una clase base Empleado que tenga un método virtual puro calcularPago(). Luego, crea clases derivadas como EmpleadoFijo y EmpleadoPorHoras, donde cada una implemente el método calcularPago() según su forma de pago (por salario fijo o por horas trabajadas).

Objetivo:

- Implementar polimorfismo para calcular el pago de una lista de empleados, donde algunos sean de salario fijo y otros de pago por horas.

- Usar un vector de punteros a Empleado y demostrar cómo se puede calcular el pago de cualquier tipo de empleado usando polimorfismo.

1 0 1 0 0 1 0 1 0 0 1 1 0 1 1 1