Tema 1. Introducción a la programación orientada a objetos

1. Paradigmas de programación
   - Programación imperativa
     - Programación estructurada
     - Programación modular
     - Programación orientada a objetos (OO)
2. Evolución de los enfoques de programación imperativa
3. Historia de la programación orientada a objetos
4. Un primer vistazo a la OO
   - Fecha en C++
   - Fecha en Java

1. Paradigmas de programación

- Un paradigma o modelo de programación es una forma de pensar acerca del proceso de descomposición de problemas y del desarrollo de soluciones
- Paradigmas clásicos
  - La programación lógica
  - La programación funcional
  - La programación imperativa
  - La programación orientada a objetos
- El importante saber que un paradigma de programación puede ser usado en diversos lenguajes de programación, y que los lenguajes de programación pueden permitir el uso de uno o más paradigmas simultáneamente
  - Java: imperativo, orientado a objetos
  - C++: imperativo, orientado a objetos y funcional

Programación imperativa

- Características de la programación imperativa:
  - Los algoritmos se escriben indicando paso a paso las instrucciones que la máquina debe ejecutar para resolver el problema.
  - La ejecución utiliza variables (estado del programa).
  - Las instrucciones actúan en base a ese estado y lo van modificando.
    - if, for, while, …
  - La programación imperativa es una traducción casi directa de las capacidades hardware de las máquinas
    - la memoria representa el estado del programa
    - las instrucciones máquina son las operaciones.

- Dentro de la programación imperativa, podemos hacer una división adicional: programación estructurada, programación modular y programación orientada a objetos.

Programación estructurada

- La programación estructurada es una forma de escribir programas de manera clara.
- Para ello utiliza únicamente tres estructuras de control que pueden ser combinadas para producir programas.
- Estas estructuras son: secuencia, selección e iteración.
- El uso de estas estructuras garantizan que los programas tengan exactamente una entrada y una salida, y que no existan bucles infinitos ni instrucciones que no se ejecutan.
- Por ello, este modo de programación evita el uso de instrucciones de transferencia incondicional como: goto, exit o múltiples return.

Programación modular

- La programación modular se presenta como una evolución de la programación estructurada para solucionar problemas de programación más grandes y complejos.
- Al aplicar la programación modular, un problema complejo debe ser dividido en varios subproblemas más simples, y estos a su vez en otros subproblemas más simples.
- Esto debe hacerse hasta obtener subproblemas lo suficientemente simples como para poder ser resueltos fácilmente con algún lenguaje de programación.

Programación modular

- Es el paradigma que se ha estado utilizando en FP.
- Sus características más relevantes son:
  - Los programas se dividen en funciones y procedimientos (subprogramas).
  - Las funciones y procedimientos se agrupan en módulos con funcionalidad similar.
  - Los datos y los procedimientos/funciones se consideran de forma separada.
  - Un subprograma recibe datos y (en el caso de las funciones) devuelve resultados en forma de datos.
  - Las funciones se transmiten datos entre ellas y operan sobre los mismos.
  - Mediante módulos permiten crear tipos abstractos de datos (TADs)
    - Entidades donde se reúnen un tipo de datos y las operaciones que manipulan los valores de ese tipo.
    - Encapsulamiento y ocultación de información
    - En EDA se tratan en profundidad
- El lenguaje C++ se ha utilizado en FP siguiendo el paradigma de programación modular, pero también se puede utilizar para realizar programación orientada a objetos.

Programación orientada a objetos (POO)

- Es un tipo de programación imperativa porque al usar este paradigma se indica la secuencia de pasos que debe seguir un programa para resolver un problema
- En la programación orientada a objetos surgen dos nuevos conceptos
  - las clases y los objetos.
- En vez de tener por un lado las funciones y procedimientos y por otro lado los datos, ambos se juntan para dar lugar a clases y objetos.
- Un objeto es la unión de una serie de datos y las funciones que operan sobre ellos, de forma que datos y operaciones se consideran una unidad.
  - En OO a las operaciones se les llama métodos.
  - Evidentemente esos métodos contendrán las instrucciones imperativas ya conocidas (bucles, condicionales, etc.).
- Los objetos (información que almacenan y operaciones que aceptan) se declaran típicamente por medio de clases.
- Una *clase* es como una plantilla de objetos, es decir, se podría decir que una clase es a un objeto lo que un tipo es a una variable en un lenguaje de programación imperativa.

Programación orientada a objetos (POO)

- Los programas orientados a objetos se construyen pues no como un conjunto de subprogramas, sino como un conjunto de objetos que interactúan entre ellos transmitiéndose órdenes e información, lo que se conoce como *envío de mensajes*.
- Las principales características de la POO son:
  - Encapsulamiento y ocultación
    - Agrupamiento de datos y funciones (encapsulación).
    - Privacidad de todo aquello que no se precise para utilizar un objeto (ocultación).
  - Herencia
    - Propiedad que permite construir una clase a partir de otra ya existente.
  - Polimorfismo
    - Propiedad que permite que distintos objetos respondan al mismo mensaje pero produciendo un comportamiento distinto.
  - Vinculación dinámica
    - El objeto que responde a un mensaje se determina en tiempo de ejecución.

2. Evolución de los enfoques de programación imperativa

- Código máquina
- Lenguaje ensamblador: Pequeña abstracción
- Primeros lenguajes de alto nivel: Tipo asociado a una variable
- Programación procedimental: Subprogramas
- Programación modular: Módulos que agrupan procedimientos y funciones
- Programación con TADs: Datos + operaciones
- Programación orientada a objetos: Clases y objetos

3. Historia de la programación orientada a objetos

- Simula-67 (finales de los años 60)
- Smalltalk (años 70)
- Década de los 80
  - Aparecen conferencias internacionales
  - Técnicas de ingeniería del software adaptadas a la OO
  - Aparecen nuevos lenguajes (Eiffel) o se extienden otros (C => C++)
- Década de los 90
  - Los objetos aparecen en todas partes
    - Sistemas operativos, entornos de desarrollo, ..
  - Aparece Java
- Hoy en día muchos lenguajes incluyen características orientadas a objetos:
  - PHP, LUA, C#, Python, …

4. Un primer vistazo a la OO

- Declaración del tipo de datos Fecha en C++

```
// Fichero Fecha.h
// Definición del tipo de datos

struct Fecha {
int dia;
int mes;
int anyo;
};

// Declaración de funciones relacionados con el tipo Fecha.
// Las implementaciones aparecerían en Fecha.cpp

// Construye una fecha
Fecha construye(int dia, int mes, int anyo);
// Dada uma fecha le suma el número de días pasado como parámetro
void suma(Fecha &fecha, int numDias);
// Dadas 2 fechas devuelve el número de días que hay entre ellas
int resta(const Fecha &fecha1, const Fecha &fecha2);
// Escribe por pantalla la fecha
void escribe(const Fecha &fecha);
```


Un primer vistazo a la OO

- Ejemplo de uso del tipo de datos Fecha en C++

```
int main() {
    // f1 y f2 son variables del tipo Fecha
    Fecha f1, f2;

    f1 = construye(12, 10, 1942);
    f2 = construye(11, 11, 1970);


    escribe(f1);
    escribe(f2);
    suma(f1, 3);
    std::cout << resta(f1, f2) << std::endl;

    return 0;
}
```

Clase Fecha en C++

- Declaración de la clase Fecha en C++

```
// Fichero Fecha.h
// Definición del tipo de datos

class Fecha {
private:
    int dia;
    int mes;
    int anyo;
public:
// Declaración de métodos relacionados con el tipo Fecha.
// Las implementaciones aparecerían en Fecha.cpp

// Construye una fecha
Fecha (int dia, int mes, int anyo);
// ..
void suma(int numDias);
// ...
int resta(const Fecha &fecha);
// ...
void escribe() const;

};
```


Clase Fecha en C++

- Ejemplo de uso de la clase Fecha en C++

```
int main() {
    // f1 y f2 son objetos de la clase Fecha
    Fecha f1(12, 10, 1492);
    Fecha f2(1, 1, 1970);


    f1.escribe();
    f2.escribe();
    f1.suma(3);
    std::cout << f1.resta(f2) << std::endl;

    return 0;
}
```

Clase Fecha en Java

- Declaración de la clase Fecha en Java

```
// Fichero Fecha.java
// Incluye la definición del tipo de datos y la implementación de los métodos

public class Fecha {

        private int dia;
        private int mes;
        private int anyo;

        public Fecha(int dia, int mes, int anyo) {
        ...
        }
        public void suma(int numDias) {
        ...
        }
        public void escribe() {
        ...
        }
        public int resta(Fecha fecha) {
        ...
        }
    };
```


Clase Fecha en Java

- Ejemplo de uso de la clase Fecha en Java

```
public class Ejemplo {
    public static void main(String [] args) {
        Fecha f1 = new Fecha(12, 10, 1492);
        Fecha f2 = new Fecha(1, 1, 1970);
        // . . .
        f1.escribe();
        f2.escribe();
        f1.suma(3);
        System.out.println(f1.resta(f2));
        }
    }
```