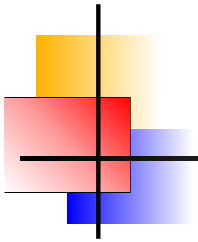


Algoritmos y Estructuras de Datos:

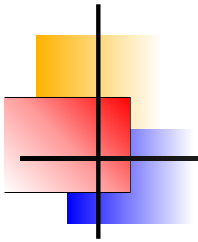
Introducción a los TAD y los Algoritmos

Guillermo Román Díez

groman@fi.upm.es

Universidad Politécnica de Madrid

Curso 2015-2016



Definiciones

Tipo Abstracto de Datos (TAD)

"*a systematic way of organizing and accessing data*"

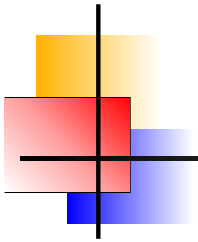
Algoritmo

"*a step-by-step procedure for performing some task in a finite amount of time*"

- Los lenguajes de programación ofrecen herramientas y mecanismos para implementar TADs (clases, interfaces, objetos, herencia, genéricos. . .)

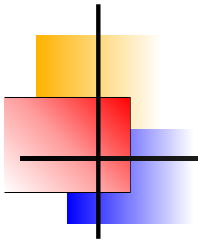
Ejemplo

Veamos un ejemplo de TAD: `ComplejoCart`



Observaciones ComplejoCart

- Los métodos `suma` y `resta` no modifican el estado del objeto sobre el que se invocan $\rightarrow$ **Objetos Inmutables**
  - Cada llamada devuelven un objeto nuevo con la suma de los atributos
- La clase `ComplejoCart` proporciona *getters* para acceder a los valores de los atributos
- La representación de los números complejos está restringida a una la representación cartesiana.
  - ¿y si queremos usar una representación polar (módulo y ángulo)?
  - ¿y si queremos combinar ambas representaciones?

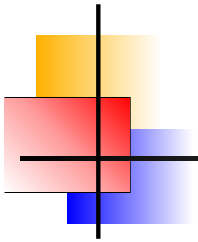


TADs en Java

- En Java un TAD consiste en una jerarquía de clases e interfaces
- Los **interfaces** especifican el **qué**
- Las **clases** implementan el **cómo**
- El interfaz no incluye constructores, ni métodos heredados de `Object` como `toString`, `equals`, etc.
- La descripción de lo que hace cada método del interfaz debe documentarse
  - Ya sea en lenguaje **natural** usando **Javadoc**
  - Ya sea en lenguaje **formal** usando **lógica formal**

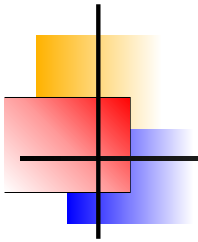
Ejercicio

Proponer un TAD para poder tener `ComplejoCart` y `ComplejoPolar`



Solución: Interfaz Complejo

```
public interface Complejo {
  /**  Devuelve la parte real */
  public double getRe();
  /** Devuelve la parte imaginaria */
  public double getIm();
  /** Devuelve el modulo */
  public double getMod();
  /** Devuelve el angulo */
  public double getAng();
  /** Devuelve un Complejo haciendo sumando 'c' */
  public Complejo suma(Complejo c);
  /** Devuelve un Complejo haciendo restando 'c' */
  public Complejo resta(Complejo c);
  /** Devuelve un Complejo haciendo conjugando 'c' */
  public Complejo conj(Complejo c);
}
```



Solución: Interfaz Complejo

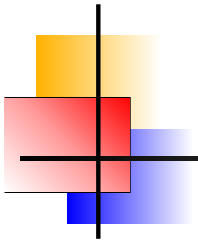
- Como hay dos formas de representar complejos ofrecemos *getters* para cada posible representación
- Los métodos `suma`, `resta` y `conj` reciben como parámetro cualquier clase que implemente el interfaz `Complejo`

```
public Complejo suma(Complejo c) {
  if (c==null) {throw new IllegalArgumentException();}
  return new ComplejoCart(this.re + c.getRe(),
                          this.im + c.getIm()) ;
}
```

- El uso de *getters* permite comparar con cualquier objeto que implemente el interfaz `Complejo`

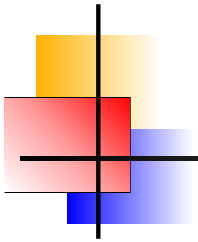
Ejemplo

Ver las clases `ComplejoCart2`, `ComplejoPolar` y el interfaz `Complejo`



Algunas Propiedades del Software

- **Acoplamiento**: Mide las dependencias (llamadas a métodos, acceso a datos, ...) entre las partes que constituyen un sistema de software
- **Cohesión**: Mide la relación entre las responsabilidades de las clases de un mismo módulo
- **Modularidad**: División de un programa en unidades que guardan una relación entre sí
- **Mantenibilidad**: Capacidad del software para ser corregido o incluir nuevas funcionalidades

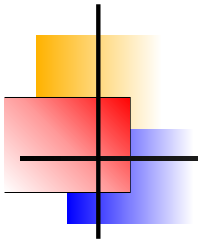


Abstracción

Abstracción

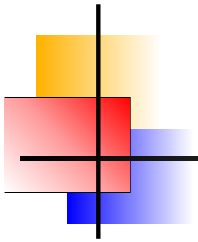
"Separar por medio de una operación intelectual las cualidades de un objeto para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción"

- **Abstracción Funcional**: Atender a la función ignorando la estructura/representación interna
- La **funcionalidad** se define mediante un **interfaz** y una especificación (formal o informal) del comportamiento
- La **representación interna** se describe en una **implementación**



Propiedades de la Abstracción Funcional

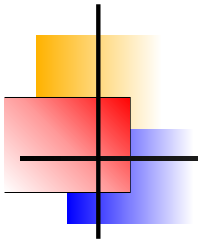
- **Encapsulación** (ocultación de información): sólo es necesario conocer la función, la estructura puede quedar oculta
  - El cliente de un TAD sólo necesita conocer los interfaces
- **Independencia de la representación**: la misma función puede ser realizada por diferentes representaciones
  - El interfaz puede ser implementado por distintas clases
- **Modularidad**: un módulo o cápsula realiza una funcionalidad
  - Debe tener alta cohesión (ser autocontenido) y bajo acoplamiento (no depender de cambios en otros módulos)
- Se hace en dos partes:
  - **Abstracción de Datos**: Tipos de datos, clases, paquetes...
  - **Abstracción de Control**: Métodos, funciones, comunicaciones, ...



Ventajas de la abstracción de datos

Pregunta

Enumerar algunas ventajas de la abstracción de datos

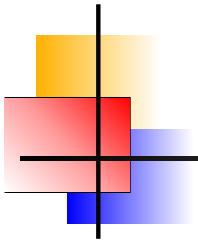


Ventajas de la abstracción de datos

Pregunta

Enumerar algunas ventajas de la abstracción de datos

- Idealmente la independencia de la implementación implica bajo acoplamiento ya que el interfaz es un "*contrato*"
  - Esto no es siempre del todo cierto: es necesario conocer los constructores y requiere un buen diseño del interfaz
- Los TADs tienen:
  - Alta cohesión
  - Bajo acoplamiento
  - Fuerzan un cierto diseño modular con refinamiento progresivo
- La labor de diseño es clave a la hora de decidir qué datos y métodos conforman un TAD



Resumen

- Un TAD consiste en uno o más interfaces y clases. Los interfaces especifican ("qué") y las clases implementan ("cómo")
- El "cliente" del TAD sólo tiene que conocer los interfaces y los constructores, para qué sirven, qué hacen los métodos y con qué eficiencia
- El "cliente" NO tiene que conocer los atributos ni el código de los métodos de las clases
- En lo posible, los interfaces y clases deben estar diseñados para poder combinar objetos de clases distintas pero que implementan el mismo interfaz
- Idealmente, el "implementador" de las clases puede cambiar los detalles sin que el código del "cliente" se vea afectado