11. El modelo de objetos

Índice

- Referencias
- Introducción
- Complejidad del software
- Características de sistemas complejos
- Descomposición, abstracción y jerarquía
  - Introducción.
  - Descomposición.
  - Abstracción.
  - Jerarquía

Índice

- Diseño de sistemas complejos
  - Diseño.
  - Modelos.
- El modelo de objetos
  - Introducción
  - Programación orientada a objetos.
  - Diseño orientado a objetos.
  - Análisis orientado a objetos.

Índice

- Elementos del modelo de objetos
  - Introducción.
  - Abstracción.
  - Encapsulamiento.
  - Modularidad.
  - Jerarquía.
  - Tipos.
  - Concurrencia.
  - Persistencia.

Índice

- Aplicación del modelo de objetos
  - Beneficios del modelo de objetos.
  - Problemas planteados.
- La naturaleza de los objetos
  - Definición.
  - Estado.
  - Comportamiento.
  - Identidad.

Índice

- Relaciones entre objetos
  - Tipos de relaciones.
  - Enlaces.
  - Agregación
- La naturaleza de las clases
  - Clases y categorías de clases.

Índice

- Relaciones entre clases
  - Tipos de relaciones.
  - Herencia.
  - Asociación.
  - Agregación.
  - Dependencia.
  - Instanciación.
  - Metaclases.

Índice

- El papel de clases y objetos en análisis y diseño
- Construcción de clases y objetos de calidad
- Conclusiones

Referencias

- Booch G., *Análisis y diseño orientado a objetos con aplicaciones*, Segunda edición, Addison-Wesley/Díaz de Santos, 1996
- Booch G., Rumbaugh J., Jacobson I., *El Lenguaje Unificado de Modelado. 2ª Edición.* Addison Wesley, 2006

Referencias

- Jacobson I., Booch G., Rumbaugh J., *El proceso unificado de desarrollo de software*. Addison-Wesley 2000

- Deitel H.M., Deitel P.J. *Java. How to program*. 7ª edición. Prentice Hall, 2006

Introducción

- Hasta ahora habéis visto el modelo de objetos desde la perspectiva de programadores
- Este tema introduce el modelo de objetos desde la perspectiva de analistas/diseñadores
- Intenta por tanto haceros evolucionar en vuestra concepción del modelo de objetos

Introducción

- Además, muestra al modelo de objetos per se, con independencia de lenguajes de modelado o lenguajes de programación concretos

Complejidad del software

- La construcción de software es una tarea eminentemente compleja
- Software de *dimensión industrial* vs. software de *dimensión personal*
- Causas complejidad software:
  - Complejidad dominio problema.
  - Dificultad gestión proceso desarrollo.
  - Flexibilidad del software.
  - Problemas para caracterizar sistemas discretos.

Complejidad del software

- Consecuencias complejidad:
  - Retrasos en los proyectos.
  - Dificultad en el mantenimiento.

Características de sistemas complejos

- Los sistemas complejos, por lo general presentan cinco características:

  - La complejidad toma forma de *jerarquía*: sistemas divididos en subsistemas hasta componentes primitivos.

  - Elección de componentes primitivos arbitraria.

  - Enlaces internos entre componentes más fuertes que enlaces entre sistemas.

Características de sistemas complejos

- La descomposición jerárquica de los sistemas complejos tiende a ajustarse en una serie de *patrones*.

- Un sistema complejo que funciona ha evolucionado a partir de un sistema más simple que funcionaba.

Descomposición, abstracción… Introducción

- Descomposición, abstracción y jerarquía son factores claves para entender sistemas complejos
- Por lo tanto, también son factores claves a la hora de diseñar sistemas complejos

Descomposición, abstracción ... Descomposición

- Durante el diseño es necesario *descomponer* un sistema en partes más pequeñas
- e.g.: el cuerpo humano.
- A estas partes se les aplica el mismo criterio
- Llegamos hasta los elementos primitivos de la jerarquía

Descomposición, abstracción ... Abstracción

- La *abstracción* sirve para enfrentarse a la complejidad
- Ignoramos los detalles no esenciales de una entidad, tratando con un modelo suyo generalizado e idealizado
- e.g.: profesor/alumno
- La visión OO facilita la abstracción al ver los objetos como abstracciones del mundo real

Descomposición, abstracción ... Jerarquía

- La descomposición divide un sistema en subsistemas
- La *jerarquía* caracteriza las relaciones existentes entre subsistemas
- La jerarquía facilita la comprensión de un sistema complejo
- e.g.: Ford Focus vs. Seat León

Descomposición, abstracción ... Jerarquía

- Hay dos tipos de estructuras fundamentales:

  - De objetos.

  - De clases.

- Estructura de objetos:

  - Ilustra cómo diferentes objetos colaboran entre si a través de patrones de interacción denominados *mecanismos*.

- Estructura de clases

  - Resalta la estructuración y componentes comunes en el interior de un sistema.

Descomposición, abstracción ... Jerarquía

- Arquitectura de un sistema:

  - Estructura de clases + estructura de objetos.

  - Estructura organizativa de un sistema que incluye su descomposición en partes, conectividad, mecanismos de interacción y principios de guía que proporcionan información sobre el diseño del mismo.

Diseño de sistemas complejos
Diseño

- La ingeniería es ciencia, no arte
- *Diseño*: aproximación disciplinada que proporciona una solución para un problema determinado, suministrando así un camino desde los requisitos hasta la implementación
- En última instancia el diseño proporciona un *modelo* del sistema

Diseño de sistemas complejos
Modelo

- Un *modelo* es una abstracción semánticamente completa de un sistema
- Un *modelo* es una representación abstracta de un sistema
- No confundir con *modelo* de objetos en la acepción de *paradigma* de objetos

Diseño de sistemas complejos
Modelo

- Ventaja fundamental de disponer de un modelo: ataca el problema del desarrollo de software desde los principios de descomposición, abstracción y jerarquía
- Utilidades del modelo:

  - Capta y enumera exhaustivamente los requisitos y el dominio de conocimiento, de forma que todos los implicados pueden entenderlos y estar de acuerdo con ellos.

Diseño de sistemas complejos
Modelo

- Ayuda a pensar el diseño de un sistema.
- Permite capturar decisiones de diseño en una forma complementaria a partir de los requisitos.
- Permite generar productos aprovechables para el trabajo.
- Permite organizar, encontrar, filtrar, recuperar, examinar, y corregir la información en grandes sistemas.

Diseño de sistemas complejos
Modelo

- Permite explorar económicamente múltiples soluciones.

- Permite *domesticar* sistemas complejos.

- Usos de un modelo:

  - Guía al proceso de pensamiento.

  - Especificación abstracta de la estructura esencial de un sistema.

  - Especificación completa de un sistema final.

Diseño de sistemas complejos
Modelo

- Ejemplo de sistema típico o posible.
- Descripción completa o parcial de un sistema.

- Elementos necesarios para definir un modelo:
  - Notación.
  - Semántica.
  - Contexto.

Diseño de sistemas complejos
Modelo

- Significado de un modelo:

  - Un modelo es un *generador* de potenciales configuraciones de sistemas.

  - Un modelo es una *descripción* de la estructura genérica y del significado de un sistema.

  - Un modelo es una *abstracción* de un sistema.

Diseño de sistemas complejos
Modelo

- En sistemas complejos, el modelo puede estar formados por más de un *submodelo*
- e.g.: modelo de persona =
  modelo fisiológico + modelo psicológico
- En sistemas software nosotros modelaremos:

1. La relación de los usuarios con el sistema.
2. El comportamiento del sistema, con independencia de las clases y/u objetos implicados.

Diseño de sistemas complejos
Modelo

3. La estructura de clases.
4. La estructura e interacción de objetos.
5. Los posibles estados de los objetos de una clase.
6. Las relaciones que existen entre las declaraciones de comportamiento y sus implementaciones físicas.
7. La arquitectura *física* del sistema y su relación con los elementos de la arquitectura *lógica* (3+4)

Diseño de sistemas complejos
Modelo

- Precisamente, un modelo que utilice varios modelos para representar la información descrita de 1 a 7 es un modelo del sistema según el modelo de objetos

El modelo de objetos Introducción

- El modelo de objetos es una *visión* de la construcción de software que abarca principios de:
  - Abstracción.
  - Encapsulación.
  - Modularidad.
  - Jerarquía.

El modelo de objetos
Introducción

- Tipos.
- Concurrencia.
- Persistencia.

- *Rápidamente* podemos decir que la programación OO se basa en:
  - Encapsulación.
  - Herencia.
  - Polimorfismo.

El modelo de objetos Programación OO

- La *programación orientada a objetos* POO es un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son, todas ellas, miembro de una jerarquía de clases unidas fundamentalmente mediante relaciones de herencia.

El modelo de objetos Programación OO

- Un lenguaje *orientado a objetos*:

  - Soporta objetos, que son abstracciones de datos con una interfaz de operaciones, con un nombre, y con un estado local oculto.

  - Los objetos tienen un tipo asociado.

  - Los tipos pueden heredar atributos de los supertipos

El modelo de objetos
Diseño OO

- El *diseño orientado a objetos* DOO es un método de diseño que abarca el proceso de descomposición OO, y una notación para describir los modelos de los sistemas que se diseña

El modelo de objetos
Análisis OO

- El *análisis orientado a objetos* AOO es un método de análisis que examina los requisitos desde la perspectiva de las clases y objetos que se encuentran en el vocabulario del dominio del problema

Elementos del modelo de objetos Introducción

- Los veremos *rápidamente* ahora y en profundidad cuando estudiemos la notación visual que soporta a estos elementos (i.e. UML)

Elementos del modelo de objetos Introducción

- Hay cuatro elementos fundamentales en el modelo de objetos:
  - Abstracción.
  - Encapsulamiento.
  - Modularidad.
  - Jerarquía
- Un modelo que carezca de alguno de estos elementos no es OO

Elementos del modelo de objetos Introducción

- Hay tres elementos secundarios en el modelo de objetos:
  - Tipos.
  - Concurrencia.
  - Persistencia.

Elementos del modelo de objetos
Abstracción

- Ya hemos comentado que la abstracción es un elemento fundamental para combatir la complejidad
- Una *abstracción* denota las características principales de un objeto que lo distinguen de todos los demás tipos de objetos y proporciona así fronteras conceptualmente nítidas

Elementos del modelo de objetos
Abstracción

- e.g. ejemplar vs. usuario
- Tipos de abstracción:

  - *De entidades*. Representa un modelo de una entidad del dominio.

  - *De acciones*. Proporciona un conjunto generalizado de operaciones llevadas a cabo por una entidad.

  - De *maquinas virtuales o interfaces*. Agrupa operaciones, todas ellas virtuales.

Elementos del modelo de objetos
Abstracción

- *Abstracción de coincidencia*. Almacena un conjunto de operaciones sin relación entre sí.

- La abstracción en sus últimas consecuencias lleva a un *modelo contractual de programación:*

  Cada objeto está caracterizado por el contrato que ofrece a otros objetos y que a su vez puede ser llevado a cabo por contratos de otros objetos

Elementos del modelo de objetos
Abstracción

- El *contrato* declara las *responsabilidades* del objeto
- De esta forma se puede caracterizar a los objetos por los servicios que presta a otros objetos, así como las operaciones que puede realizar sobre otros objetos
- Así el *cliente* es un objeto que utiliza los recursos de otro objeto llamado *servidor*

Elementos del modelo de objetos
Abstracción

- e.g.

```
public class A {
    B b;
    public A() {b.f();}
};
......
```

`A a;` → `a` es cliente de `b`

Elementos del modelo de objetos
Abstracción

- Los contratos de un objeto están definidos por sus operaciones
- Booch denomina al conjunto de operaciones que puede realizar un cliente sobre un objeto su *protocolo*

Elementos del modelo de objetos
Encapsulamiento

- La abstracción y el encapsulamiento son conceptos complementarios:

  - La abstracción se centra en el comportamiento observable de un objeto.

  - El encapsulamiento se centra en la implementación que da lugar al comportamiento.

Elementos del modelo de objetos
Encapsulamiento

- El encapsulamiento se logra mediante la *ocultación de información*, proceso por el cual se ocultan los detalles de un objeto que no contribuyen a sus características esenciales, i.e., estructura del objeto e implementación de sus operaciones
- Definimos:

  - *Operación*: especificación de una transformación o consulta que puede tener que ejecutar un objeto.

Elementos del modelo de objetos
Encapsulamiento

- *Método*: Implementación de una operación. Especifica el algoritmo o procedimiento que da lugar a los resultados de una operación.

- e.g.

  operación: `String saludo();`

  método 1:
  ```
  String saludo()
      {return “hola”;}
  ```

  método 2:
  ```
  String saludo()
      {return “hi there”;}
  ```

Elementos del modelo de objetos
Encapsulamiento

- El encapsulamiento representa la separación existente en ciertos lenguajes como C++ entre interfaz e implementación:

  - *Interfaz*: vista externa de la clase que considera la abstracción que se ha hecho del comportamiento común de todas las instancias de la clase.

  - *Implementación*: representación de la abstracción, así como los mecanismos que consiguen el comportamiento deseado.

Elementos del modelo de objetos Encapsulamiento

- e.g.

```
// saludohola.h
class SaludoHola {
 public:
    String saludo();
 private:
    String saludo= “hola”; };
 //saludo.cpp
 #include “saludohola.h”
 String SaludoHola::saludo() {return saludo;}
```

Elementos del modelo de objetos
Encapsulamiento

- En lenguajes como Java el encapsulamiento puede entenderse como la separación entre interfaces y clases

  - *Interfaz*: Conjunto de operaciones que posee un nombre y que caracteriza el comportamiento de un elemento.

  - *Clase*: Descriptor de un conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y comportamiento.

Elementos del modelo de objetos Encapsulamiento

- e.g.

```
interface Saludar{
String saludo();
}
public class SaludoHola implements
Saludar{
String saludo= “hola”;
public String saludo()
{ return saludo;}
 };
```

Elementos del modelo de objetos
Encapsulamiento

- Discusión: ¿Son equivalentes ambas aproximaciones?
- *Encapsulación*: almacenamiento en un mismo compartimiento los elementos de una abstracción que constituyen su estructura y su comportamiento; sirve para separar el interfaz contractual de una abstracción de su implementación

Elementos del modelo de objetos
Modularidad

- En los programas OO las clases y objetos forman la *estructura lógica* de un sistema
- Estas abstracciones se sitúan en los *módulos* para producir la *arquitectura física* del sistema
- En última instancia, un módulo no es más que un conjunto de clases relacionadas

Elementos del modelo de objetos
Modularidad

- En C++ se incluyen las clases relacionadas en un mismo archivo de cabecera
- En Java se utilizan los *packages*
- La *cohesión* de un módulo es una medida de la solidez de las relaciones entre sus componentes

  - Un módulo debe capturar una abstracción clave en el sistema.

Elementos del modelo de objetos
Modularidad

- Si los módulos incluyen partes relacionadas con una abstracción distinta aparece baja cohesión.
- e.g. entre `javax.swing.JFrame` y `javax.swing.JButton` hay una alta cohesión.
- e.g. entre `javax.swing.JFrame` y `java.net.Socket` la cohesión es baja

Elementos del modelo de objetos
Modularidad

- El *acoplamiento* es una medida de la fuerza de interconexión entre módulos

  - El acoplamiento es alto si intercambian datos e información de control.

  - e.g. entre `javax.swing.*` y `java.awt.*` el acoplamiento es alto.

  - e.g. entre `javax.swing.*` y `java.net.*` el acoplamiento es bajo.

Elementos del modelo de objetos
Modularidad

- La *modularidad* es la propiedad que tiene un sistema que ha sido descompuesto en módulos cohesivos débilmente acoplados

Elementos del modelo de objetos
Jerarquía

- Como ya hemos comentado, la *jerarquía* es una clasificación u ordenación de abstracciones
- e.g.:
  - Herencia.
    - Simple.
    - Múltiple.
  - Agregación.

Elementos del modelo de objetos
Tipos

- Un *tipo* es una caracterización precisa de propiedades estructurales o de comportamiento que comparten una serie de entidades
- En lenguajes como Java un objeto sólo puede tener una clase de implementación, pero muchos tipos

Elementos del modelo de objetos
Tipos

- e.g.

```
interface A {...};
interface B {...};
class AB implements A, B {...};
...
```

`AB ab= new AB();` → `ab` es de clase `AB`, pero tiene tipo `A` y `B`

Elementos del modelo de objetos
Tipos

de esta forma podemos hacer:

```
A aa;
B bb;
aa= ab;
bb= ab;
```

pero no

```
aa= bb;
```

Elementos del modelo de objetos
Tipos

- En esta asignatura solamente se podrá preguntar por el tipo de un objeto (`instanceof`, `typeid()`, `...`) cuando éste se haya perdido *porque no había otro remedio*, por ejemplo:
  - Al trabajar con estructuras de datos Java
  - Al cargar dinámicamente objetos
  - Al saltar capas

Elementos del modelo de objetos
Tipos

- En otras palabras, el polimorfismo es ideal cuando un cliente tiene que hacer un tratamiento similar con distintos objetos, pero dependiente de su tipo particular
  - Por ejemplo, el cálculo de la nómina en un empleado de tiempo completo o tiempo parcial
  - En vez de preguntar el tipo y hacer un cálculo u otro, el propio empleado es responsable del cálculo de su nómina

Elementos del modelo de objetos
Tipos

- En este caso, el cliente, un objeto responsable de calcular la nómina de una empresa, puede delegar en el objeto empleado a través del polimorfismo porque es razonable que “un empleado sepa calcular su nómina”

Elementos del modelo de objetos
Tipos

- Si saltamos una capa, y el cliente, por ejemplo, es la interfaz de usuario, y tiene que modificar un empleado a tiempo completo o uno de tiempo parcial, no es razonable delegar en empleado
- En este caso, sí que tendrá que preguntar por su tipo

Elementos del modelo de objetos
Tipos

- Por último, nótese que es equivalente usar operadores del tipo `instanceof` que incluir una función del estilo `String dameTipo()` en el objeto correspondiente
- Lo único que cambia es el responsable de calcular el tipo

Elementos del modelo de objetos
Concurrencia

- Podemos caracterizar la concurrencia como la ejecución de un programa en varios hilos de control
- Procesos pesados vs. procesos ligeros
- La concurrencia en POO no difiere en exceso de la concurrencia en lenguajes tradicionales

Elementos del modelo de objetos
Concurrencia

- En POO a los objetos que representan un hilo separado de control se le denomina *objeto activo*
- De esta forma, *concurrencia* es la propiedad que distingue a un objeto activo de uno que no está activo

Elementos del modelo de objetos Persistencia

- Un objeto de software ocupa una cierta cantidad de espacio y existe durante una cierta cantidad de tiempo
- Tipos de persistencia de objetos:

  - Resultados transitorios de la evaluación de expresiones.

  - Variables locales en la activación de procedimientos.

Elementos del modelo de objetos Persistencia

- Variables globales y elementos del montículo.
- Datos que existen entre ejecuciones de un programa.
- Datos que existen entre varias versiones de un programa.
- Datos que sobreviven al programa.

- A nosotros nos interesa los tres últimos tipos, y en especial, el cuarto tipo

Elementos del modelo de objetos
Persistencia

- La persistencia implica algo más que la duración de los datos, ya que no solo hay que guardar el *estado*, sino la *clase* del objeto
- De esta forma los programas pueden interpretar los datos
- En algunos lenguajes (e.g. Java) es automático
- En otros (e.g. C++) no lo es

Elementos del modelo de objetos
Persistencia

- Definimos *persistencia* como la propiedad de un objeto por la que su *existencia trasciende* el *tiempo* (es decir, el objeto continúa existiendo después de que su creador deja de existir) y/o el *espacio* (es decir, la posición del objeto varía con respecto al espacio de direcciones en el que fue creado)

Aplicación del modelo de objetos
Beneficios del modelo de objetos

- El modelo de objetos presenta los siguientes beneficios:

  - Está interrelacionado con los lenguajes OO.

  - Reutilización del software y de diseños (e.g. APIs*).

  - Los sistemas OO son más pequeños que los no OO → mayor productividad → beneficios de coste y planificación.

*API: Application Program Interface

Aplicación del modelo de objetos
Beneficios del modelo de objetos

- Sistemas OO más flexibles ante el cambio.
- El modelo de objetos resulta atractivo para la cognición humana.

Beneficios del modelo de objetos Problemas planteados

- Para aplicar el modelo de objetos *correctamente* debemos resolver varios *problemas*:

  - ¿Qué son exactamente las clases y los objetos?

  - ¿Cómo se identifican correctamente las clases y objetos relevantes de una aplicación concreta?

  - ¿Cómo sería una notación adecuada para expresar el diseño de un sistema OO?

Beneficios del modelo de objetos
Problemas planteados

- ¿Qué proceso puede conducir a un sistema OO bien estructurado?

- ¿Cuáles son las implicaciones en cuanto a gestión que se deriva del uso del modelo de objetos?

La naturaleza de los objetos Definición

- Un *objeto* es una entidad que tiene estado, comportamiento e identidad
- La estructura y comportamiento de objetos similares están definidos en su *clase* común
- Los términos objetos e *instancia* son intercambiables

La naturaleza de los objetos
Estado

- El *estado* de un objeto abarca todas las propiedades (normalmente estáticas) del mismo, más los valores actuales (normalmente dinámicos) de cada una de esas propiedades

La naturaleza de los objetos
Estado

- Una *propiedad* es una característica inherente o distintiva, un rasgo o cualidad que contribuye a que un objeto sea ese, y no otro
- Las propiedades tienen algún *valor*
- Este valor puede ser una cantidad o denotar a otro objeto

La naturaleza de los objetos
Estado

- Nótese que preservar el estado de un objeto requiere *espacio*
- En la práctica, las propiedades de un objeto son sus *atributos*
- En consecuencia, el estado de un objeto está *encapsulado*

La naturaleza de los objetos Comportamiento

- Por lo general, los objetos no están aislados
- Reciben acciones y actúan sobre otros objetos
- El *comportamiento* es cómo actúa y reacciona un objeto, en términos de sus cambios de estado y paso de mensajes

La naturaleza de los objetos Comportamiento

- Es decir, el comportamiento de un objeto representa su actividad visible y comprobable exteriormente
- En lenguajes OO puros (e.g. Smalltalk) se dice que un objeto *pasa un mensaje* a otro
- En lenguajes OO mixtos (e.g. C++) se dice que un objeto *invoca una función miembro* de otro

La naturaleza de los objetos Comportamiento

- El comportamiento de un objeto es función de su estado y de la operación que se realice sobre él
- Algunas operaciones modifican el estado de un objeto
- Teniendo en cuenta esto, refinamos la definición de estado

La naturaleza de los objetos Comportamiento

- El *estado* de un objeto representa los resultados acumulados de su comportamiento
- En última instancia, el comportamiento del objeto queda capturado en sus *operaciones*
- Una operación denota un servicio que una clase ofrece a sus clientes

La naturaleza de los objetos Comportamiento

- Podemos clasificar las funciones en:

  - *Modificador*. Altera el estado del objeto.

  - *Selector*. Accede al estado sin modificarlo.

  - *Iterador*. Accede a todas las partes de un objeto en un orden preestablecido.

  - *Constructor*. Crea un objeto e inicializa su estado.

  - *Destructor*. Libera el estado de un objeto y/o destruye al propio objeto

La naturaleza de los objetos Comportamiento

- Además, pueden existir *subprogramas libres*

  - Éstos son procedimientos o funciones que sirven como operaciones no primitivas sobre un objeto u objetos de la misma o distintas clases.

  - Se suelen agrupar en utilidades de clases o en clases `static` sin atributos.

La naturaleza de los objetos Comportamiento

- Todos los métodos y subprogramas libres de un objeto forman su *protocolo*
- El protocolo define la envoltura/declaración del comportamiento admisible de un objeto
- Muchas veces, no vemos todo el protocolo de un objeto, sino la parte que *nos interesa*

La naturaleza de los objetos Comportamiento

- Precisamente, si la abstracción es no trivial, el protocolo puede dividirse en grupos lógicos de comportamiento
- Estos grupos lógicos denotan los *papeles* que el objeto puede desempeñar
- Por lo tanto un papel define un *contrato* entre una abstracción y sus clientes

La naturaleza de los objetos Comportamiento

- Discusión: ¿qué concepto de programación representa un *papel*?

La naturaleza de los objetos Comportamiento

- Unificando estado y comportamiento, obtenemos las *responsabilidades* del objeto: conocimiento que un objeto mantiene y acciones que puede llevar a cabo
- Por tanto, las responsabilidades de un objeto son todos los servicios que proporciona para los contratos que soporta

La naturaleza de los objetos Comportamiento

- En otras palabras: el estado y comportamiento de un objeto definen el conjunto de papeles que puede representar el objeto en el mundo, los cuales cumplen las responsabilidades de la abstracción

La naturaleza de los objetos Comportamiento

- En lo concerniente a la concurrencia, al tener estado un objeto, el orden de invocación de acciones sobre el mismo es crucial
- Dicho de otra forma, un objeto es como una pequeña máquina de estados
- En base a esto, los objetos pueden ser:
  - Activos.
  - Pasivos.

La naturaleza de los objetos Comportamiento

- Un objeto *activo* es aquel que comprende su propio hilo de control, mientras que un objeto pasivo no tiene su propio hilo de control
- Por lo tanto, los objetos activos pueden tener comportamiento sin que ningún otro objeto opere sobre ellos

La naturaleza de los objetos Comportamiento

- En contraposición, los objetos pasivos sólo pueden sufrir un cambio de estado cuando se actúa explícitamente sobre ellos
- Bueno, esto es un poco relativo porque a los objetos activos hay que activarlos, y esto es una actuación explícita sobre ellos

La naturaleza de los objetos
Identidad

- La *identidad* es aquella propiedad de un objeto que lo distingue de todos los demás objetos
- ¿Cuándo son dos objetos iguales?:

  - ¿Cuándo referencian al mismo objeto? (compartición estructural)

  - ¿Cuándo tienen el mismo estado?

La naturaleza de los objetos
Identidad

- Para comprobar la igualdad:

  - Misma dirección de memoria: utilizar operador de igualdad (C++ y Java).

  - Mismo estado:

    - Sobrecargar el operador de igualdad en C++.

    - Sobrecargar el operador `equals()` en Java.

- En la práctica está relacionado con el concepto de *asignación*

La naturaleza de los objetos Identidad

- La asignación puede provocar compartición estructural
- Igual sucede con el paso de parámetros por referencia de objetos

  - En Java, siempre.

  - En C++ podemos usar parámetros por valor para los objetos.

    - Esto es desaconsejable ya que produce:

      - Gasto de memoria.
      - Gasto de tiempo en duplicación de estado (llamada al constructor).

La naturaleza de los objetos Identidad

- Podemos hablar de:

  - *Copia superficial* (*shallow copy*). Se copia el objeto compartiendo estado si hay miembros dinámicos.

  - *Copia profunda* (*deep copy*). Se copia el objeto y su estado de forma recursiva.

- El operador de asignación en C++ y Java implementan la copia superficial

La naturaleza de los objetos Identidad

- Si se desea implementar la copia profunda:

  - Sobrecargamos el operador de asignación en C++.

  - Proporcionamos una función de copia en Java (e.g. el propio constructor o implementamos la función `clone()`).

La naturaleza de los objetos
Identidad

- Por último cabe destacar que el objeto tiene un *espacio de vida*
- El mismo abarca desde que se crea el objeto, consumiendo espacio, hasta que ese espacio se recupera
- Nótese que en lenguajes como C++, no es lo mismo un objeto, que un puntero o referencia a objeto

La naturaleza de los objetos
Identidad

- Una vez que el objeto ya no es necesario:

  - En C++ hay que hacer recolección explícita de basura.

  - En Java y Smalltalk no.

- Nótese que si el objeto es persistente, la destrucción tiene una semántica distinta

Relaciones entre objetos
Tipos de relaciones

- Los objetos contribuyen al comportamiento de un sistema colaborando con otros objetos
- Hay dos tipos de relaciones entre objetos de especial interés:
  - Enlaces.
  - Agregación

Relaciones entre objetos
Enlaces

- Un *enlace* es una conexión física o conceptual entre objetos
- Denota una asociación específica por al cual un objeto (el cliente) utiliza los servicios de otro objeto (el suministrador o servidor)
- Si existe un enlace entre objetos `a` y `b`, para que `a` envíe un mensaje a `b`, `b` debe ser visible para `a` de algún modo

Relaciones entre objetos
Enlaces

- En análisis la cuestión de la visibilidad se puede obviar, pero en diseño/implementación hay que resolverlo
- Las formas de tener visibilidad son:

  - El objeto servidor es global para el objeto cliente.

  - El objeto servidor es parámetro de alguna operación del cliente.

Relaciones entre objetos
Enlaces

- El objeto servidor es parte del objeto cliente.

- El objeto servidor es un objeto declarado localmente en alguna operación del cliente.

- Además de la visibilidad, la sincronización es fundamental en el paso de mensajes
- Cuando un objeto pasa un mensaje a otro, deben estar sincronizados

Relaciones entre objetos
Enlaces

- En una aplicación secuencial, esta sincronización se lleva a cabo simplemente mediante el paso de mensajes
- En una aplicación con múltiples hilos de control, los objetos requieren un paso de mensajes más sofisticado con el fin de tratar problemas de exclusión mutua

Relaciones entre objetos
Agregación

- Los enlaces denotan relaciones cliente-servidor
- La *agregación* es un tipo especializado de asociación que denota una relación todo/parte
- De esta forma siempre se puede llegar del objeto agregado a su atributo

Relaciones entre objetos
Agregación

- El camino contrario solo es posible si se incluye una referencia explícita
- La agregación puede denotar una *vinculación* entre la vida del objeto agregado y sus componentes
  - Si no es así: agregación *simple* o *agregación*.
  - Si es así: agregación *compuesta* o *composición*.

Relaciones entre objetos
Agregación

- En lenguajes como C++:

  - Agregación simple: contención por referencia.

  - Agregación compuesta: contención por valor.

- En lenguajes como Java:

  - Agregación simple: contención.

  - Agregación compuesta: contención + asignación a `null / finalize() + System.gc()` (supuesto que no lo referencie otro objeto).

La naturaleza de una clase Clases y categorías de clases

- Las clases y los objetos están estrechamente relacionados, ya que todo objeto pertenece a una clase
- El objeto es una entidad concreta que existen en el tiempo y el espacio
- La clase representa la abstracción, la esencia del objeto

La naturaleza de una clase
Clases y categorías de clases

- Así decimos que una *clase* es un conjunto de objetos que comparten una estructura y comportamientos comunes
- De esta forma, el interfaz y la implementación de las abstracciones se corresponden con el interfaz y la implementación de las clases

La naturaleza de una clase
Clases y categorías de clases

- En ciertas clases (no todas) es conveniente capturar su ciclo de vida (comportamiento interno)
- Decimos que un objeto es una *instancia* de una clase
- La clase es un vehículo necesario pero no suficiente para la descomposición

La naturaleza de una clase
Clases y categorías de clases

- A veces es necesario un conjunto cooperativo de clases para dar soporte a una abstracción compleja (e.g. IGU), es necesario una *categoría de clases*
- También se conocen como *frameworks* o *APIs*

Relaciones entre clases
Tipos de relaciones

- Las relaciones que se pueden dar entre clases son:
  - Herencia.
  - Asociación.
  - Agregación.
  - Dependencia.
  - Instanciación.
  - Metaclase.
  - *Realización* no la ponemos al ser interfaz y no clase

Relaciones entre clases
Herencia

- La *herencia* es una relación entre clases en la que una clase comparte estructura y/o comportamiento definidos en una (h. simple) o más clases (h. múltiple)
- Una clase de las que otras heredan se denomina *superclase*
- Una clase que hereda de otra o más clases se denomina *subclase*

Relaciones entre clases
Herencia

- La clase más generalizada en una estructura de clases se denomina *clase base*
- Además, las clases pueden ser:

  - *Abstractas*: no instanciables.

  - *Concretas*: instanciable.
- Discusión: ¿cuándo debemos definir una clase como abstracta?

Relaciones entre clases
Herencia

- En C++ conseguiremos clases abstractas incluyendo alguna función virtual pura
- En Java podemos definir una clase como `abstract` e incluir una función `abstract` sin código, aunque lo más normal es no definir ninguna función y definir un interfaz

Relaciones entre clases
Herencia

- La *herencia múltiple* es una cualidad *peligrosa*

  - En C++ se resuelve poniendo el calificador `virtual` en la superclase.

  - En Java se resuelve no permitiéndola.

    - Aunque sí permite la herencia múltiple entre interfaces.

    - Además, una misma clase puede implementar más de un interfaz.

Relaciones entre clases
Herencia

- Por *último* en algún lugar de esta asignatura debería constar una definición de polimorfismo
- El *polimorfismo* es la capacidad de objetos de clases diferentes relacionados mediante herencia de responder de forma distinta a una misma llamada de una función

Relaciones entre clases
Asociación

- La *asociación* denota una dependencia semántica entre clases
- Es la relación:
  - Más general.
  - De mayor debilidad semántica.
- Denota:
  - Conexión semántica bidireccional.
  - Relación sin refinar.

Relaciones entre clases
Agregación

- La relación de agregación entre clases representan a mayor nivel de abstracción la relación de agregación entre objetos
- Recordemos que puede ser:
  - Simple, o agregación.
  - Compuesta, o composición.

Relaciones entre clases
Dependencia

- Una *dependencia* denota una relación entre dos elementos en los cuales, un cambio en un elemento (el proveedor) puede afectar al otro elemento (el cliente), pero no necesariamente a la inversa
- Las dependencias se dan con profusión entre las clases, donde un objeto requiere la presencia de otro objeto para su funcionamiento

Relaciones entre clases
Dependencia

- Básicamente la dependencia se debe a que un objeto:

  - Es global a otro objeto.

  - Es parámetro de una función de otro objeto.

  - Está definido dentro de una función de otro.

  - Es un objeto estático, accesible sin necesidad de crearlo.

Relaciones entre clases
Instanciación

- Por *instanciación* entendemos concretar clases a partir de otras más genéricas, denominadas clases *parametrizadas* o *plantilla*
- A partir de Java J2SE 1.5.0 hay *generics*

Relaciones entre clases
Metaclases

- Una *metaclase* es una clase cuyas instancias son clases
- Concepto de Smalltalk
- En C++/Java se puede simular incluyendo miembros `static` (e.g. hombre y mujer)

El papel de clases y objetos en análisis y diseño

- Las clases y objetos son conceptos distintos pero muy relacionados:
  - Todo objeto es instancia de una clase.
  - Una clase tiene cero o más instancias.
- Las clases son estáticas y los objetos dinámicos.

  Esto tiene una fuerte repercusión en los diagramas de modelado.

El papel de clases y objetos en análisis y diseño

- Durante el análisis y las primeras etapas de diseño, el desarrollador tiene dos tareas principales:

  - Identificar las clases y objetos que forman el vocabulario del problema.

  - Idear las estructuras por las que conjuntos de objetos trabajan juntos para lograr comportamientos que satisfacen los requisitos del problema

El papel de clases y objetos en análisis y diseño

- En conjunto a esas clases y objetos se les denomina *abstracciones claves* del problema, y a las estructuras cooperativas los *mecanismos* de la implementación
- Las etapas de análisis y las primeras etapas de diseño se centran en obtener una vista externa de clases y objetos

El papel de clases y objetos en análisis y diseño

- Esta estructura externa se plasma en diagramas de clases (o interfaces) y objetos, el *modelo lógico*
- En las últimas etapas de diseño se centran en obtener una representación física del sistema
- Esta representación física se plasma en los diagramas de componentes y despliegue, el *modelo físico* del sistema

Construcción de clases y objetos de calidad

- Hay cinco factores que determinan si una clase* está *bien* diseñada:
  - Acoplamiento.
  - Cohesión.
  - Suficiencia.
  - Compleción.
  - Ser primitivo.

*clase o categoría de clases

Construcción de clases y objetos de calidad

- Del *acoplamiento* y *cohesión* hablamos en el apartado de modularidad
- Una clase es *suficiente* si captura suficientes características de la abstracción como para permitir una interacción significativa y eficiente
- Lo contrario produce componentes inútiles (e.g. una clase pila sin función apilar)

Construcción de clases y objetos de calidad

- Una clase es *completa* si su interfaz es suficientemente general para ser utilizado de forma común por cualquier cliente
- Podemos decir que la suficiencia proclama una interfaz mínima, y la completitud una de lujo

Construcción de clases y objetos de calidad

- Es una cuestión subjetiva que puede exagerarse, ya que muchas operaciones de alto nivel pueden construirse sobre otras de bajo nivel
- Esto sugiere la necesidad de que los clases sean primitivas

Construcción de clases y objetos de calidad

- Una clase *primitiva* es aquella cuyas operaciones solo pueden implementarse si tienen acceso a la representación subyacente de la abstracción (apilar vs. iterar)
- Minimizaremos las operaciones no primitivas
- Una operación que implementada sobre operaciones primitivas tiene una alto coste computacional, puede ser también primitiva

Conclusiones

- Modelo de objetos
- Independiente de:
  - Lenguajes de programación.
  - Lenguajes de modelado.
- IS rentable en software de dimensión industrial
- Importancia de modelar

Conclusiones

- Modelo de objetos:
  - Abstracción.
  - Encapsulación.
  - Modularidad.
  - Jerarquía.
  - Tipos.
  - Concurrencia.
  - Persistencia.

Conclusiones

- POO *rápidamente*:
  - Encapsulación.
  - Herencia.
  - Polimorfismo.
- Análisis OO vs. diseño OO vs. programación OO
- Modelo de objetos: beneficios e inconvenientes

Conclusiones

- Objeto:
  - Estado.
  - Comportamiento.
  - Identidad.
- Relaciones entre objetos:
  - Enlaces.
  - Agregación.

Conclusiones

- Clases: aglutinadores de objetos
- Categorías de clases
- Relaciones entre clases:
  - Herencia.
  - Asociación.
  - Agregación.
  - Dependencia.
  - Instanciación.
  - Metaclase.

Conclusiones

- Papeles jugados por clases y objetos en análisis y diseño
- Clases de calidad