Guías de diseño

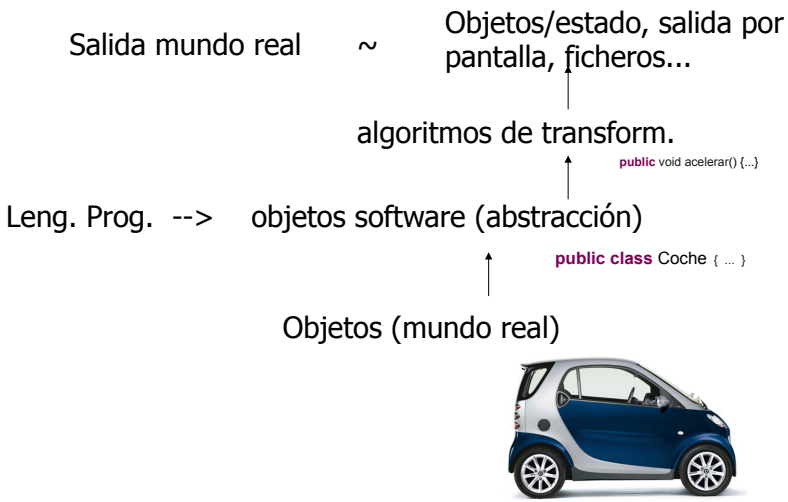
Contenidos

1. Introducción
2. Metodología de diseño
   a. Identificación de clases de objetos
      i. Guía de Booch
      ii. Guía de Ellis
   b. Identificación de las operaciones
   c. Establecimiento de la visibilidad
   d. Implementación

1. Introducción

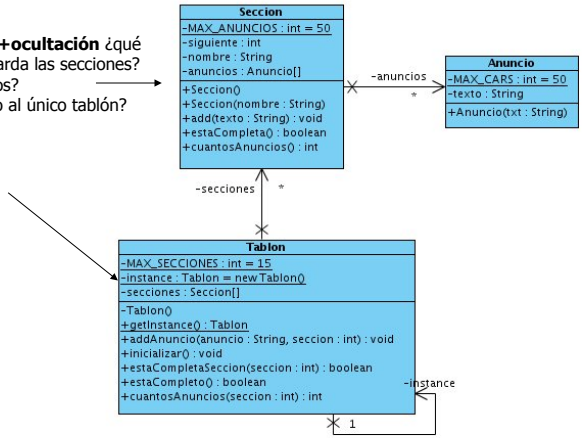
- Los LPs actuales proporcionan mecanismos para representar los **objetos** del mundo real: el programador abstrae los objetos del problema y los implementa en software.
- Se aplican **algoritmos** que transforman estos objetos software (*abstracción lógica de los algoritmos del mundo real)*
  - Los algoritmos producen alguna salida que corresponde físicamente a alguna acción del mundo real

Abstracción



Ejemplo Diagrama de Clases

Abstracción+ocultación ¿qué tipo datos guarda las secciones?
¿y los anuncios?
¿cómo accedo al único tablón?

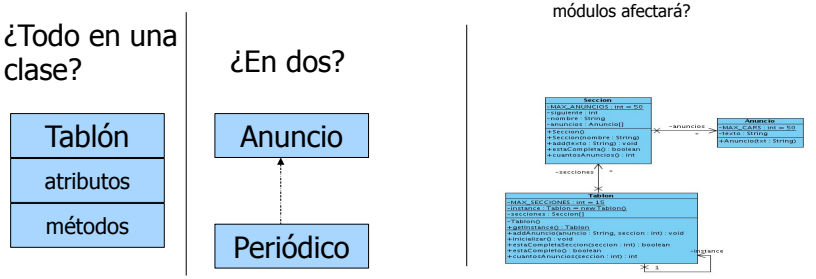


"*cada módulo en el sistema denota un objeto o una clase de objetos del problema*"

1. Introducción

¿Qué diseño te parece más: *modificable, inteligible, reutilizable?*

Pista: ¿una modificación, a cuántos módulos afectará?



PISTA: "*cada módulo en el sistema denota un objeto o una clase de objetos del problema*"

1. Introducción

Cuanto más cerca esté la **abstracción** que hemos definido del **problema real,** más sencillo será lograr los objetivos de *modificabilidad*, *eficiencia*, *fiabilidad* e *inteligibilidad*

1. Introducción

- Importancia de los objetos como actores, cada uno de ellos con su propio conjunto de operaciones
- Criterio principal para la descomposición:
  "*cada módulo en el sistema denota un objeto o una clase de objetos del problema*"
- Los principios de **Abstracción** y **Ocultación de Información** constituyen el fundamento del desarrollo orientado a objetos

1. Introducción

- **Objeto** = "*Entidad que tiene un estado, es decir, tiene un valor*"
- La **conducta del objeto** está definida por las acciones que realiza
- Cada objeto es una instancia de una **clase** de objetos
- Un **programa** debe verse como un conjunto de objetos que interactúan entre sí

2. Método de diseño OO

- Identificación de **clases** y **atributos**
- Identificación de las **operaciones**
- Establecimiento de la **visibilidad** de cada clase en relación con las otras clases
- Establecimiento de la **visibilidad** de los miembros
- **Implementación** de cada **clase**

2.1. Identificar las clases de objetos

- Identificar los **objetos**
- Identificar las características de los objetos (**atributos**)
- Guías para la identificación
  - Booch
  - Ellis

Los dos enfoques pueden utilizarse conjuntamente

Ejemplo: Gestión de Supermercado

Se desea automatizar el proceso de actualización de existencias de un supermercado. El supermercado cuenta con 40 terminales (cajas registradoras). Cada una de ellas tiene un identificador y registra durante la jornada toda la información de las ventas efectuadas desde la misma; por cada venta (artículo que pasa por caja) se registra la información:

- código de artículo, número de unidades vendidas

Al final del día se utiliza la información registrada en las 40 cajas para actualizar las existencias globales del supermercado en lo que respecta a los artículos vendidos. El inventario de existencias guarda por cada artículo a la venta la siguiente información:

- código de artículo, existencias, precio por unidad, stock mínimo

Identifica las clases del problema

2.1.1. Guía de Booch

- Nombres que aparecen en la descripción del problema
  - Nombres **comunes**: mesa, terminal, sensor (*clases*)
  - Nombres **propios** y nombres de **referencia directa** , para una instancia específica, ej: sensor de calor, la mesa, el conjunto de todos los empleados de una empresa, … (*instancias* o Clases con instancia unica - Singleton)

2.1.1. Ejemplo (guía de Booch)

Se desea automatizar el proceso de actualización de existencias de un supermercado. El supermercado cuenta con 40 terminales (cajas registradoras) . Cada una de ellas tiene un identificador y registra durante la jornada toda la información de las ventas efectuadas desde la misma; por cada venta (artículo que pasa por caja) se registra la información:

- código de artículo, número de unidades vendidas

Al final del día se utiliza la información registrada en las 40 cajas para actualizar las existencias globales del supermercado en lo que respecta a los artículos vendidos. El inventario de existencias guarda por cada artículo a la venta la siguiente información:

- código de artículo, existencias, precio por unidad, stock mínimo

2.1.1. Ejemplo (guía de Booch)

- Nombres que aparecen en la descripción del problema:
  - Clases:

    Terminal, Venta, Artículo, ListaVentas
  - Clases con instancia única (Singleton):

    Supermercado, Inventario

2.1.1. Ejemplo (guía de Booch)

Clases y atributos

Supermercado (Singleton)
listerminales: Lista

Inventario (Singleton)
lisArticulos: Lista

Terminal
identificador: String
lisVentas: ListaVentas

Articulo
codArticulo: int
existencias: int
precioUnidad: float
stockMinimo: int

ListaVentas
lisVentas: Lista

Venta
codArticulo: int
unidades: int

2.1.2. Guía de Ellis

- **Esbozar el algoritmo principal** antes de que se hayan especificado todas las abstracciones de datos
- Las **operaciones** se identifican al considerar el proceso que debe realizar la aplicación
- Los **objetos** se identifican como los agentes que realizan dicho proceso.

2.1.2. Ejemplo (guía de Ellis)

Se desea automatizar el proceso de actualización de existencias de un supermercado. El supermercado cuenta con 40 terminales (cajas registradoras) . Cada una de ellas tiene un identificador y registra durante la jornada toda la información de las ventas efectuadas desde la misma; por cada venta (artículo que pasa por caja) se registra la información:

- código de artículo, número de unidades vendidas

Al final del día se utiliza la información registrada en las 40 cajas para actualizar las existencias globales del supermercado en lo que respecta a los artículos vendidos. El inventario de existencias guarda por cada artículo a la venta la siguiente información:

- código de artículo, existencias, precio por unidad, stock mínimo

2.1.2. Ejemplo (guía de Ellis)

Método inicial en la clase Supermercado

- Para cada `Terminal` de la `Lista`:
  - ➢ Terminal
    - Acceder a la `ListaVentas` para actualizar el `Inventario`
      - ➢ ListaVentas
        - Para cada `Venta` de la `Lista`, actualizar el `Inventario` tras la `Venta`
          - ➢ Venta
            - Acceder a la `Venta` y actualizar el `Inventario`
              - ➢ Inventario
                - Actualizar las unidades del `Articulo`
                - ➢ Articulo

2.2. Identificar las operaciones

- Caracterizar la conducta de cada objeto o clase de objetos
- Determinar las operaciones que pueden realizarse sobre el objeto o por el objeto
- Especificar parámetros (número y tipo)
- Identificar restricciones de espacio o tiempo (ej: orden temporal para las operaciones)

2.1.2. Ejemplo (guía de Ellis)

Supermercado
```
actualizarInventario(): void
```
//1. Para cada Terr

Terminal
```
actualizarInventario(): void
```
//2. Acceder a la listaVentas

ListaVentas
```
actualizarInventa
```
//3. Para cada Venta de l

Venta
```
actualizarInventario(): void
```
//3.1. `obtCodArticulo(): int`
//3.2. `obtUnidades(): int`
//4. Acceder al Inventario

Inventario
```
actualizarExis
```
//4.1. `buscarArticul`
//5. Actualizar las uni

Articulo
```
asigUnidades(pUnid: int): void
```
// 5.1. Actualizar las Unidades del Articulo

Diagrama de secuencia

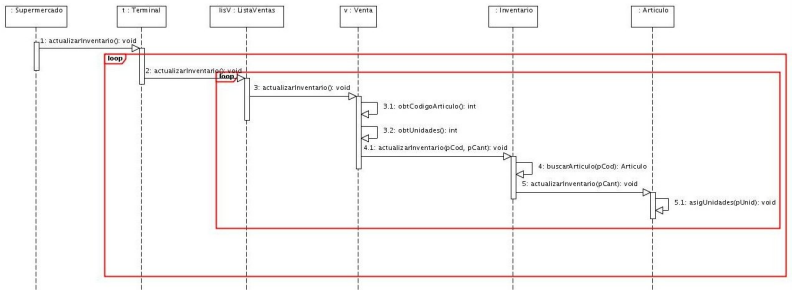
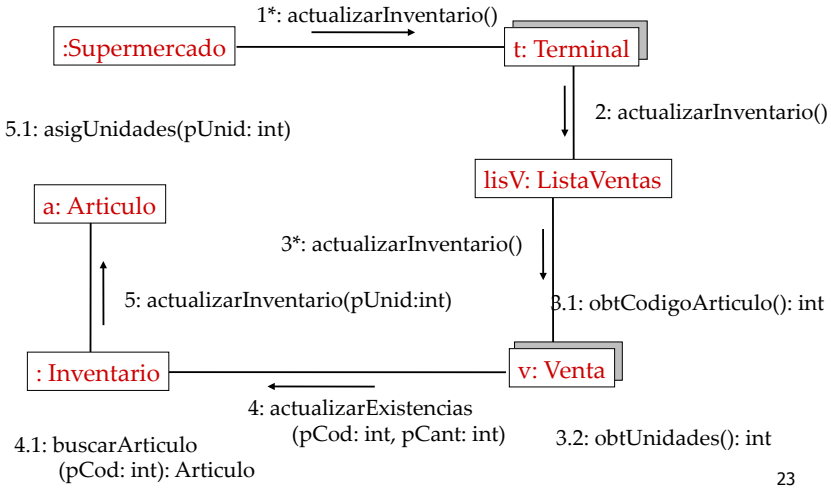


Diagrama de comunicación



2.2. Identificar las operaciones - Guía I

- Tipos de métodos que puede contener una clase:
  - **Métodos constructores**: generan instancias de la clase
    - *Clase*: constructor público
    - *Singleton*: constructor privado y método de acceso público
  - **Métodos de acceso**: obtienen el valor de los atributos
  - **Métodos modificadores**: cambian el valor de los atributos
    Se definen para cada atributo modificable de la clase
  - **Métodos productores**: generan nuevas instancias a partir de otras iniciales
    Definen operaciones que generan nuevos objetos

2.2. Identificar las operaciones - Guía II

- El ***Iterador*** es un objeto que nos proporciona un acceso homogéneo a una colección de objetos.

- hayMasElementos(): booleano ← hasNext()
- obtSiguiente(): Elemento ← next()
- inicializar(): void ← Constructor
- borrar(): void ← remove()

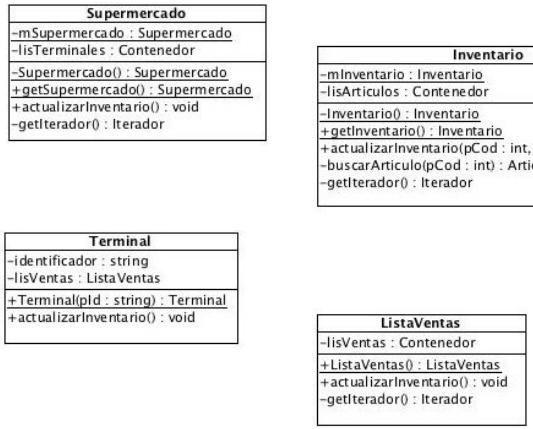
2.3. Establecer la visibilidad

- Identificar las dependencias entre clases.
- Identificar la visibilidad de atributos y métodos
- Utilizaremos una representación gráfica para representar estas características:
  - UML

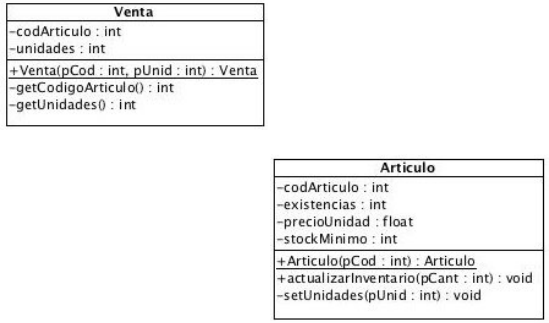
2.3. Establecer la visibilidad. Guía

- Los *atributos* siempre **privados**
- Los *métodos*, inicialmente, **privados**
- Si se detecta que algún método privado definido en una clase es **indispensable** para la implementación de otra, su visibilidad se cambia, pasando a ser **pública**

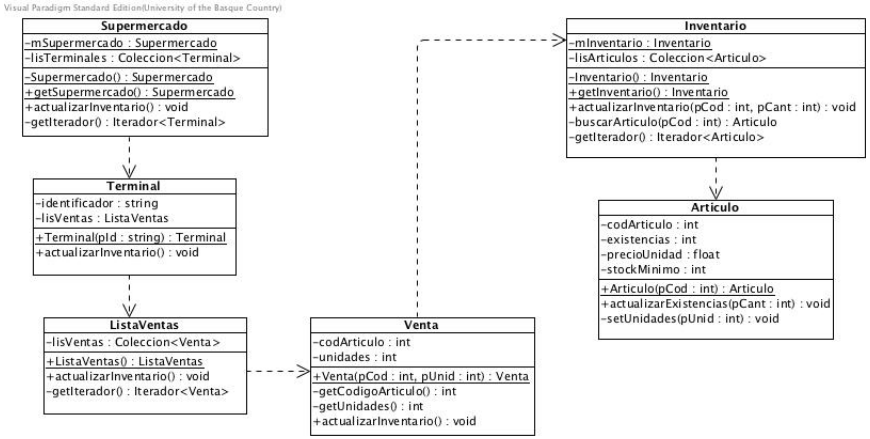
Ejemplo: Gestión de Supermercado



Ejemplo: Gestión de Supermercado



Ejemplo: Gestión de Supermercado



2.5. Implementación de los objetos

- Elegir la representación adecuada para cada objeto o clase de objetos e implementarlos
- Este paso puede incluir a su vez nueva descomposición o composición:
  - Repetiremos el método de forma iterativa
- Suele ser más frecuente componer a partir de objetos más simples

2.5. Implementación de los objetos

- Suele relegarse la implementación de los detalles, utilizando únicamente la especificación de los objetos para experimentar con la arquitectura y conducta del sistema
- Pueden probarse también varias representaciones de los objetos e implementaciones

Implementación de los objetos. Ejemplo

CLASE ListaVentas

```
public class ListaVentas {
   //Atributos
     private List<Venta> lisVentas;

   //Constructor
     public ListaVentas() {
      // Inicializar las ventas
      lisVentas = new ArrayList<>();
}
```


Implementación de los objetos. Ejemplo

```
//Otros métodos
…
   private Iterator<Venta> getIterador() {
         return lisVentas.iterator();
   }
   public void actualizarInventario() {
       for (Venta ventaActual: listaVentas) {
              ventaActual.actualizarInventario();
       }
   }
} //Fin de la clase
```


Implementación de los objetos. Ejemplo

CLASE Inventario

```
public class Inventario {
   //Atributos
     private static Inventario mInventario
                              = new Inventario();
    private Map<Integer, Articulo> lisArticulos;

   //Constructor
    private Inventario() {
      lisArticulos = new HashMap<>();
   }
   public static Inventario getInventario(){
      return mInventario;
}
```

Implementación de los objetos. Ejemplo

```
//Otros métodos
…

   private Articulo buscarArticulo(int pCod) {
       return lisArticulos.get(pCod);
   }
} //Fin de la clase
```

Implementación de los objetos. Ejemplo (II)

CLASE ListaVentas

```
public class ListaVentas {
   //Atributos
     private List<Venta> lisVentas;

   //Constructor
     public ListaVentas() {
      // Inicializar las ventas
      lisVentas = new Vector<Venta>();
}
```


Implementación de los objetos. Ejemplo

```
//Otros métodos
…
   private Iterator<Venta> getIterador() {
         return lisVentas.iterator();
   }
   public void actualizarInventario() {
       Iterator<Venta> iterador = getIterador();
       Venta ventaActual;
       while (iterador.hasNext()) {
               ventaActual = iterador.next();
               ventaActual.actualizarInventario();
       }
   }
} //Fin de la clase
```


Implementación de los objetos. Ejemplo

CLASE Inventario

```
public class Inventario {
   //Atributos
     private static Inventario mInventario
                              = new Inventario();
    private Map<Integer,Articulo> lisArticulos;

   //Constructor
    private Inventario() {
      lisArticulos = new HashMap<Integer,Articulo>();
   }
   public static Inventario getInventario(){
      return mInventario;
}
```

Implementación de los objetos. Ejemplo

```
//Otros métodos
…

   private Articulo buscarArticulo(int pCod) {
       return lisArticulos.get(pCod);
   }
} //Fin de la clase
```