FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Objetos (uso)

Carlos Fontela
cfontela@fi.uba.ar

A3F

Temario

Objetos y mensajes

Objetos y clases

Estado, comportamiento, identidad

Paquetes

Historia hacia la POO

POO vs. procedural

Operando sobre objetos (1)

```
String saludo = new String ("Hola");
String saludo1 = saludo.concat (", qué tal");
System.out.println (saludo1);        // Hola, qué tal
String saludo2 = saludo.replace ('a', 'u');
System.out.println (saludo2);        // Holu, qué tul
int pos = saludo.lastIndexOf('H');
System.out.println (pos);            // 0
int longitud = saludo1.length();
System.out.println (longitud);       // 13
```

Operando sobre objetos (2)

```
ArrayList lista = new ArrayList ( );
if (lista.size() == 0)
        lista.add(-4);
lista.add(1,-7);
Object valor = lista.get (0);
System.out.println (valor);          // -4
System.out.println (lista.size());   // 2
lista.clear();
```

Operando sobre objetos (3)

```
Date fecha1 = new Date (1983, 12, 10);
Date fecha2 = new Date (2009, 8, 1);
Date fecha3 = new Date (2002, 6, 10);
String nombre = new String ("Carlos Fontela");
ArrayList lista = new ArrayList ( );
lista.add (fecha1); lista.add (fecha2); lista.add (fecha3);
Collections.sort(lista);
lista.add (nombre);
for (int i = 0; i < lista.size(); i++)
    System.out.println(lista.get(i).toString());
```

Objetos y responsabilidades

- Los objetos tienen responsabilidades
  - Actuar ante la llegada de un mensaje => "comportamiento"
  - Guardar datos internos => "estado"
- Objetos = entidades con comportamiento
  - En principio, guardamos sólo el estado que nos permite realizar el comportamiento
  - Aunque en algunos casos hay objetos que nos interesan sólo por su estado
    - En POO, deberían ser muy pocos

Objetos y mensajes

Comportamiento => todo programa trabaja con objetos que reciben mensajes y actúan

Dándonos información sobre su estado

```
int longitud = saludo1.length();
```

Alterando su estado:

```
lista.add (fecha1);
```

Enviando mensajes a otros objetos:

```
System.out.println ( lista.get(i).toString( ) );
```

Los mensajes están implementados como "métodos"

Objetos y clases

Los objetos son de determinados tipos

- Idea de concepto e individuo

Los tipos se llaman clases

Pueden ser definidos por el programador

- Próximo capítulo

Clases y objetos

Clase

- Define estructura y comportamiento de los objetos
  - Los datos internos de un objeto
  - Los mensajes que un objeto entiende
- Molde de objetos

Objeto

- Una instancia de la clase
- Tiene existencia en tiempo de ejecución

Creación de objetos

Declaración:

ArrayList x;

Definición:

x = new ArrayList( );

Significado

ArrayList es la clase de x

ArrayList( ) es el “constructor” de la clase ArrayList

El objeto se crea recién cuando llamo al constructor con el operador “new”

En x queda una referencia a un objeto de tipo ArrayList

Referencias

Las variables son referencias a objetos:

```
Date x, y;
x = new Date (2009, 7, 25);
y = x;
```

"x" e "y" referencian al mismo objeto

(hay una sola llamada a constructor)

Si hago:

```
y = new Date (1950, 8, 17);
```

Ahora "y" referencia a otro objeto

Una variable que no referencia un objeto tiene el valor "null"

Puedo hacer: Date x = null;

Recolección de basura

Si hago:

```
Date x;
x = new Date (2009, 7, 25);
x = new Date (1950, 8, 17);
```

El objeto inicial quedó como basura

Java, C# y Smalltalk tienen recolección automática de basura

Excepciones (1)

Cuando un objeto no puede responder a un mensaje, reacciona enviándonos una excepción

Una excepción es

- Un objeto lanzado desde un método
- Que puede ser capturado

Por ejemplo, si hago:

```
ArrayList lista = new ArrayList (-5);
```

Voy a obtener un mensaje:

```
Exception in thread "main"
  java.lang.IllegalArgumentException
```

Excepciones (2)

Pero también puedo capturar la excepción:

```
try {
   ArrayList lista = new ArrayList (-5);
}
catch (IllegalArgumentException e) {
   System.out.println("error");
}
```

En este caso no se interrumpe el programa

Es un tema a desarrollar luego

Excepciones (3)

Recordemos que las excepciones son objetos

- Se acceden mediante variables que los referencian
- Tienen estado, comportamiento e identidad

Por ejemplo:

```
try {
   ArrayList lista = new ArrayList (-5);
}
catch (IllegalArgumentException e) {
   e.printStackTrace();
}
```

Programa OO

Conjunto de objetos enviando mensajes a otros objetos

Los objetos receptores reciben los mensajes y reaccionan

- Haciendo algo (comportamiento)
- Devolviendo un valor (que depende de su estado)

Los mensajes pueden implicar la creación de nuevos objetos

El comportamiento puede delegarse a su vez en otro objeto

Los objetos entienden los mensajes para los cuales están preparados => Modelo esencialmente concurrente

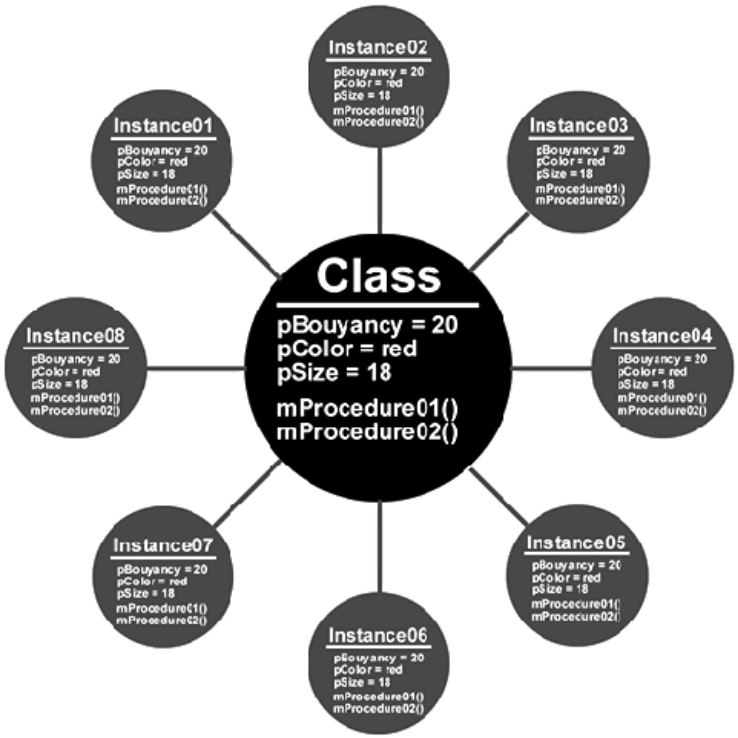
Concepto de Objeto

Definiciones

- Una instancia de una clase
- Cualquier cosa, real o abstracta, de la cual almacenamos datos y la forma de manipular esos datos

Características

- Estado (visible e interno)
- Comportamiento
- Identidad



Características (1)

Estado visible

- Se accede mediante propiedades (C#)
- O métodos de consulta y asignación (“getters” y “setters”)
- Propios de la clase a la que pertenece

Comportamiento

- Se obtiene mediante métodos
- Propios de la clase a la que pertenece

Características (2)

Estado interno

- Almacenado en atributos, no accesible de afuera
- Definidos en la clase del objeto

Identidad

- Única para cada objeto
- La mantiene el sistema (referencia), no accesible directamente en Java, C# y Smalltalk

Importancia del comportamiento

Diferencia más importante con programación estructurada

- No estamos solamente usando variables y tipos simples
- Tampoco datos estructurados
- Son “objetos” que saben cómo comportarse

Corolarios:

- Los objetos deben manejar su propio comportamiento
- No deberíamos manipular su estado desde afuera

En vez de:

```
punto.setX ( punto.getX ( ) + 1 );
```

Deberíamos hacer:

```
punto.avanzarEnX(1);
```

Paquetes

Agrupación de clases, anidables

Para manejar complejidad y resolver nombres

Ejemplos:

ArrayList es java.util.ArrayList (Java)

ArrayList es System.Collections.ArrayList (C#)

```
import java.util.*;                 // Java
import java.util.ArrayList;         // Java
using System.Collections;           // C#
```

En Java no es necesario importar las clases de java.lang

Implementaciones de mensajes en C#

C# distingue “métodos” de “propiedades”

Métodos: para comportamiento de los objetos

```
lista.Add(-4);

saludo2 = s.Replace (‘a’, ’u’);
```

Propiedades: para consultar el estado

```
Console.WriteLine (lista.Count);

int longitud = saludo.Length;
```

Mensajes enviados a la clase

¿Qué hicimos cuando escribimos … ?

Collections.sort(lista);

No hay ningún objeto para el que estemos llamando el método ("lista" es un argumento)

"sort" es un **método de clase** de la clase "Collections"

En C# hay propiedades de clase:

DateTime ahora = DateTime.Now;

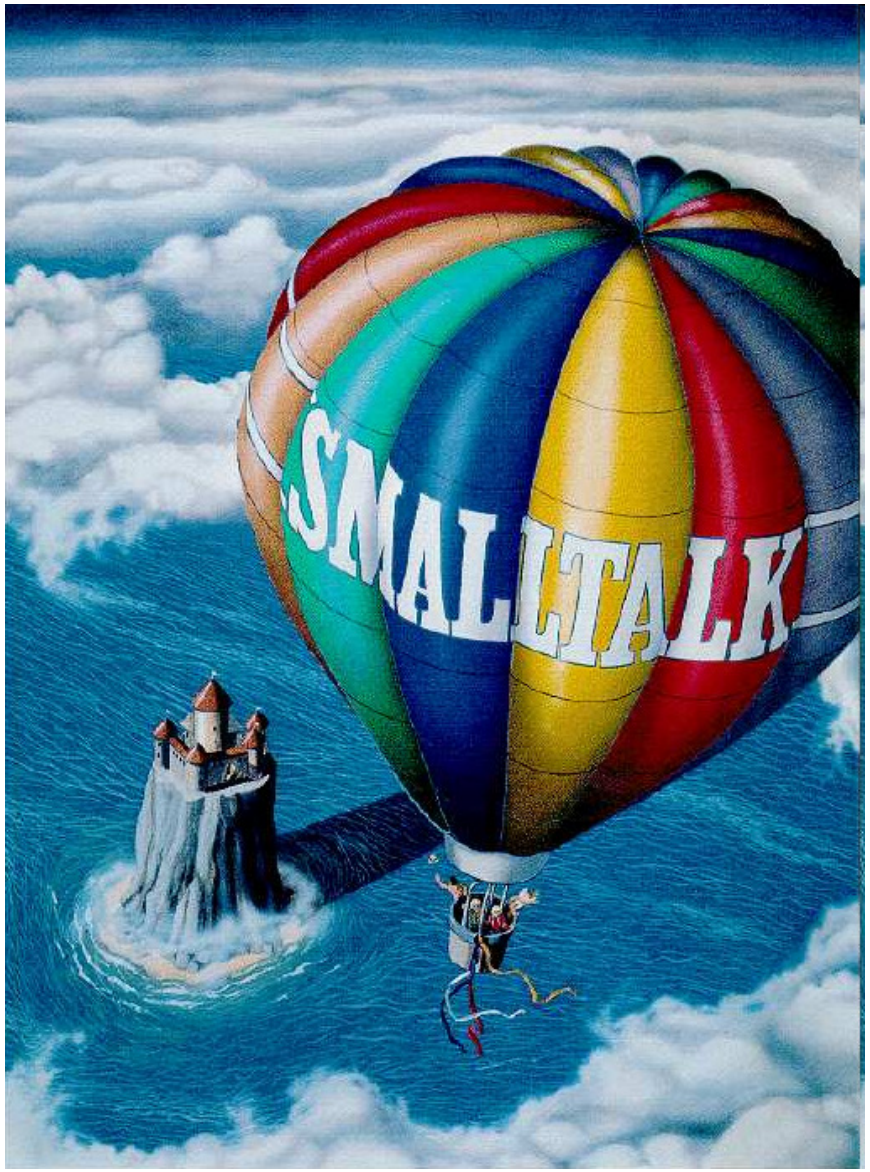
"Now" es una **propiedad de clase** de la clase "DateTime"

En Smalltalk el "new" es un método de clase:

fecha = Date new.

Smalltalk

- Notación diferente
- Todo son objetos y mensajes
- Tipos dinámicos



Lo veremos en un capítulo aparte

POO

Influencias previas

- Programación estructurada
- Programación modular
- Abstracción
- Tipos definidos por el programador
- Ocultamiento de implementación

Más otras cuestiones

- Énfasis en encapsulamiento
- Herencia
- Polimorfismo, con o sin herencia

OO: objetivo principal

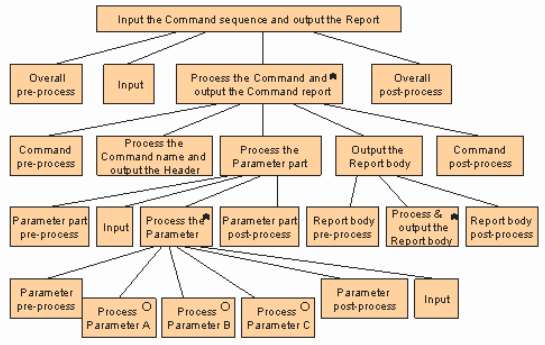
- Manejo de la complejidad
  - Abstracción: construir en base a componentes
  - Lo hacen todas las industrias
    - Economía
    - División del trabajo
    - Ya probado y optimizado
    - Se adquiere y se ensambla
    - Hay que definir interfaces: contrato
    - No fue cierto hasta fines de los 90

Mini-Historia de paradigmas (1)

Paradigma “lineal” o “espagueti”

- Código dirigido por orden de ejecución, con saltos y sin modularidad
- Todas las cuestiones mezcladas
- Lenguajes: Fortran IV, Cobol

Paradigma “estructurado” o “procedural”

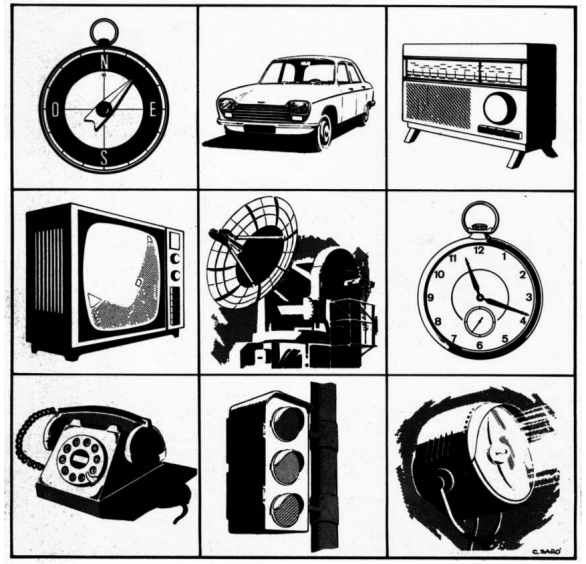


- Centrado en lo que “hace” el software
- Separación de funcionalidades en módulos
- Sin separación de entidades, tipos o clases
- Lenguajes: Fortran 77, C, Pascal, ¿Ada?

Mini-Historia de paradigmas (2)

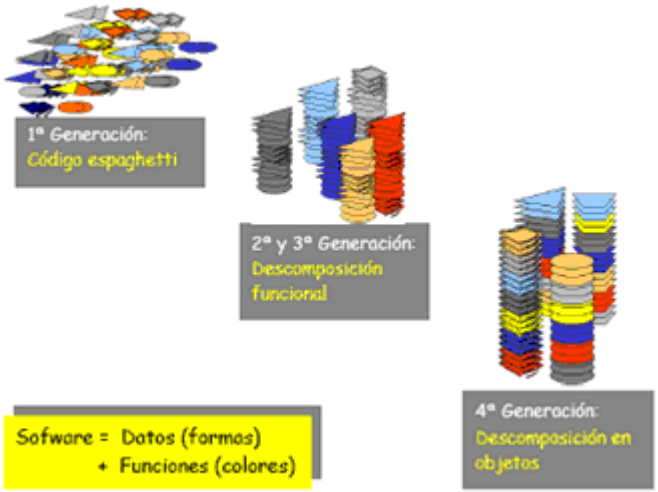
Paradigma orientado a objetos

- Centrado en las entidades del dominio
- Las entidades son clases
- Y sus instancias, objetos
- Lenguajes: Smalltalk, Eiffel, Ada95, C++, ObjectPascal, Java, C#, Python, Ruby



Paradigma(?) de aspectos

- Modulariza cuestiones no funcionales
- Lenguajes: AspectJ, Spring, JBoss, …



Otros paradigmas

- Relacional, funcional, lógico, …

POO vs Programación Procedural

- Ambas modularizan, separan en partes
- Procedural modulariza por funciones
  - (lo que hace el sistema = dominio de la solución)
  - ¿verbos?
- POO modulariza por entidades
  - (sobre qué trabaja el sistema = dominio del problema)
  - ¿sustantivos?
- En POO también hay funciones
  - Son los métodos dentro de las clases
  - Pero especifican más bien “comportamiento” de entidades

POO: cómo lograrla

Entidades se convierten en clases

= tipos con comportamiento

Próximo capítulo

Claves

Se trabaja con objetos y mensajes

Las clases son tipos y los objetos sus instancias

Las clases son conjuntos de objetos

Los paquetes agrupan clases

Java / C# / Smalltalk:

- Constructores crean objetos (son referencias)
- A los objetos sin uso los elimina el sistema

POO modulariza en base a las entidades del dominio del problema

Lecturas opcionales

Object-oriented analysis and design : with applications, Grady Booch

- Capítulo 4: “Classification”

Análisis y diseño orientado a objetos, James Martin y James Odell

- Capítulo 15: “Conceptos y objetos”
- Capítulo 17: “Concepto vs. Tipo de objeto”

Ambos libros están en biblioteca

- El de Booch tiene una versión en castellano, agotada

Son libros antiguos

- Más de 15 años
- No existía Java ni C#, sí Smalltalk

Orientación a objetos, diseño y programación, Carlos Fontela 2008, capítulo 3 “Programación basada en objetos”

Qué sigue

Clases (construcción)

Delegación y herencia

Polimorfismo