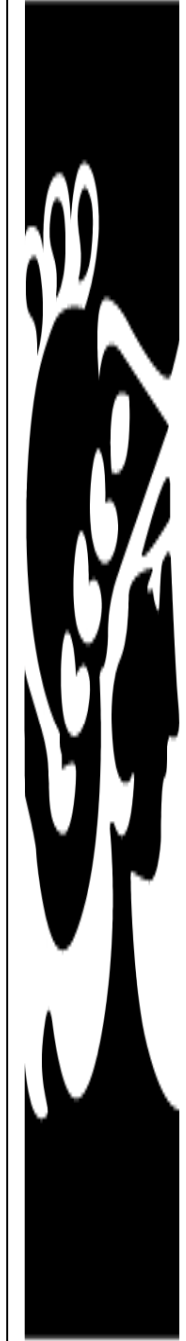




Objetos (uso)

Carlos Fontela
cfontela@fi.uba.ar



Temario

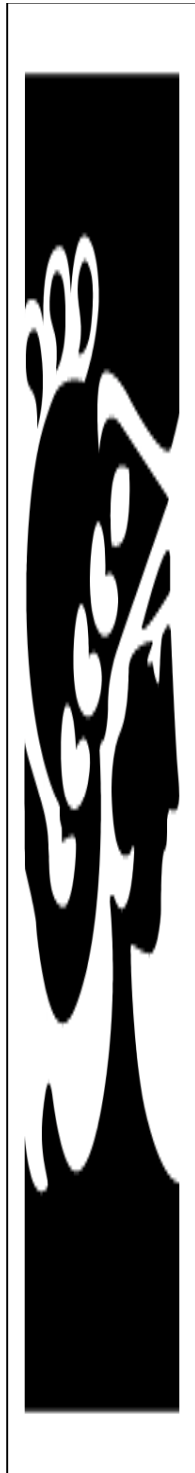
Objetos y mensajes

Objetos y clases

Estado, comportamiento, identidad

Historia hacia la POO

POO vs. procedural



Operando sobre objetos (Smalltalk)

```
lista := OrderedCollection new.
```

```
lista add: -7.
```

```
x := lista size.
```

```
Transcript show: x printString.
```

```
Transcript show: [ lista get:1 printString ].
```

```
Transcript show: [ 4 factorial squared printString ].
```

```
Transcript show:
```

```
    [ (8 < 9) ifTrue: ['verdad'] ifFalse: ['mentira'] ].
```

```
count := 0. total := 1.
```

```
[count > 10] whileFalse:
```

```
    [count := count + 2. total := total * count].
```

```
Transcript show: total printString.
```



Operando sobre objetos (Java)

```
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 conocimiento 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

```
x := lista size.
```

```
int x = lista.size();
```

Alterando su estado:

```
lista add: 44.
```

```
lista.add (fecha2);
```

Enviando mensajes a otros objetos:

```
Transcript show: [ 4 factorial squared printString ].
```

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



Objetos y clases

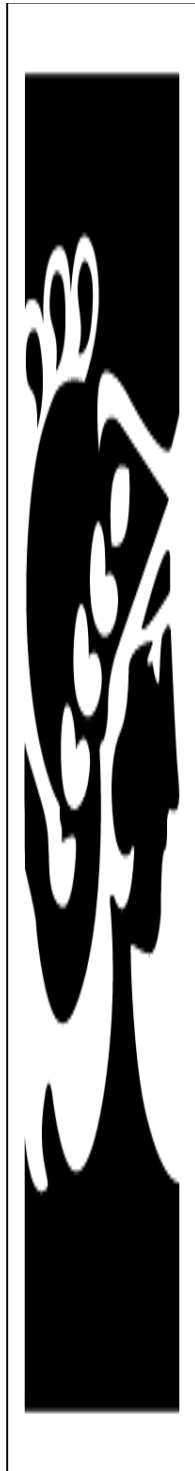
Los objetos son de determinados tipos

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 (Smalltalk)

La creación se hace mediante un mensaje a la clase:

```
x := OrderedCollection new.
```

Significado

OrderedCollection es la clase del objeto al que se refiere x

El objeto se crea recién cuando invoco el mensaje “new”

En x queda una referencia a un objeto de tipo OrderedCollection

Smalltalk es un lenguaje de tipos dinámicos

Luego:

```
x := Dictionary new.
```



Creación de objetos (Java y C#)

Declaración y creación:

```
ArrayList x;  
x = new ArrayList( );
```

Significado

ArrayList es el tipo de x (que es una variable tipada)

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

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

En x queda una referencia a un objeto de clase ArrayList

Java y .NET tienen tipos estáticos

```
// x = new Date ( );    => error de compilación
```



Referencias (Smalltalk)

Las variables son referencias a objetos:

```
lista1 := OrderedCollection new.
```

```
lista1 add: -7.
```

```
lista2 := lista1.
```

“lista1” y “lista2” referencian al mismo objeto

(hay una sola llamada a “new”)

Si hago:

```
lista2 := SortedCollection new.
```

Ahora “lista2” referencia a otro objeto

Una variable que no referencia un objeto tiene el valor “nil”

Puedo hacer: lista := nil.



Referencias (Java y C#)

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:

```
lista1 := OrderedCollection new.
```

```
lista1 := Point new.
```

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

```
x = new Date (2010, 8, 17);
```

El objeto inicial quedó como
basura

Java, C# y Smalltalk tienen
recolección automática de
basura



Errores y excepciones

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:

```
vacio := OrderedCollection new.
```

```
x := vacio at: 4.
```



Voy a obtener un error en tiempo de ejecución

Excepciones capturadas (Smalltalk)

Pero también puedo capturar la excepción:

```
vacio := OrderedCollection new.
```

```
[x := vacio at: 4]
```

```
on: Error
```

```
do: [:e | Transcript show: e messageText]
```

En este caso no se interrumpe el programa



Excepciones capturadas (Java)

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



Excepciones como objetos

Se acceden mediante variables que las referencian

Tienen estado, comportamiento e identidad

Por ejemplo, en Smalltalk:

```
Transcript show: e messageText. e retry.
```

retry y messageText son mensajes enviados al objeto referenciado por e

Y en Java:

```
e.printStackTrace();
```

printStackTrace es un mensaje enviado al objeto referenciado por e

Hay muchos tipos de excepciones => luego...



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



Concepto de Objeto

Definiciones

Una instancia de una clase

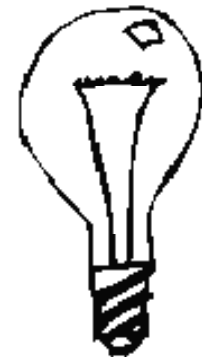
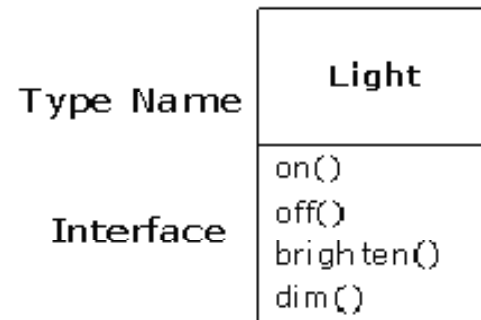
Cualquier cosa, real o abstracta, de la cual almacenamos datos y la forma de manipular esos datos

Características

Estado (visible e interno)

Comportamiento

Identidad



Características (1)

Estado visible

Se accede mediante propiedades (C#, Object Pascal)

O mensajes de consulta y asignación (“getters” y “setters”), en Smalltalk llamados “accessors”

Propios de la clase a la que pertenece

Comportamiento

Se obtiene mediante mensajes

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

Principio “Tell, don’t ask”

En vez de:

```
punto.setX: [ punto.getX + 1 ] .  
punto.setX ( punto.getX() + 1 );
```

Deberíamos hacer:

```
punto moverEnX: 1.  
punto.moverEnX ( 1 );
```



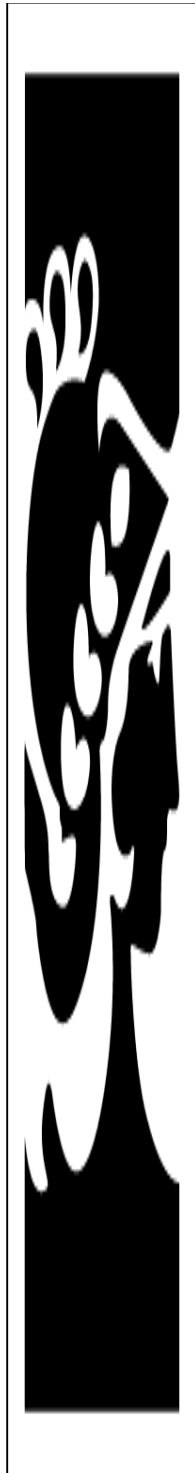
Mensajes enviados al objeto clase

En Smalltalk, las clases son objetos

¿Qué hicimos cuando escribimos ... ?

```
lista := OrderedCollection new.
```

OrderedCollection es una clase



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



Mecanismos de abstracción

Clasificación (individuo-especie)

Agrupación (entre individuos)

Generalización (entre especies)



Clasificación

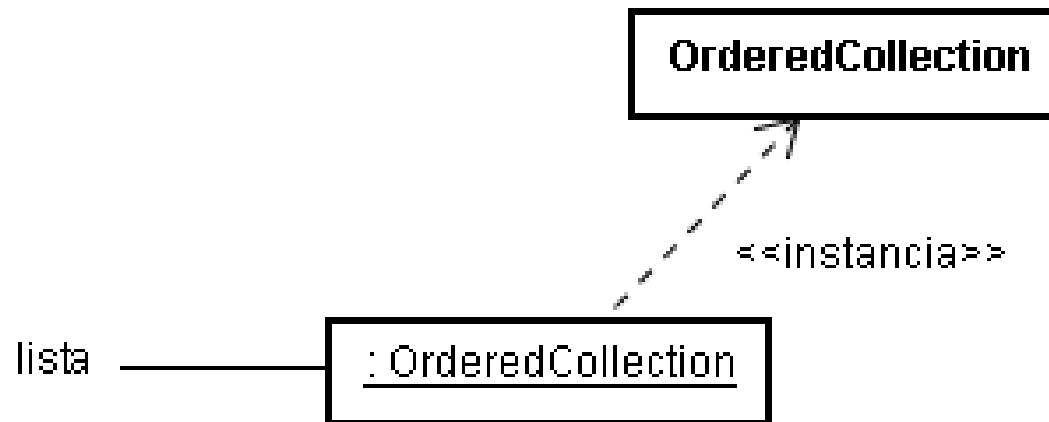
Relación individuo - especie

Lassie – perro / Juan Pérez – ser humano



Clasificación en OO: instanciación

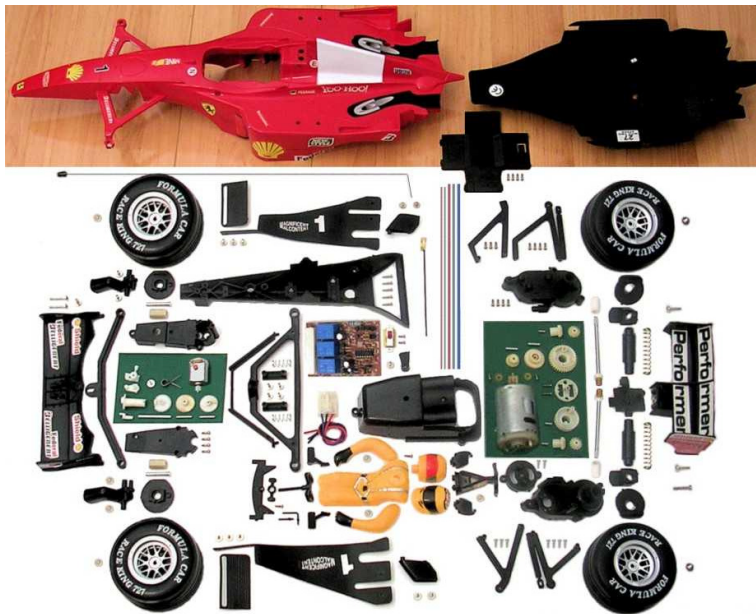
Relación objeto - clase



Agrupación

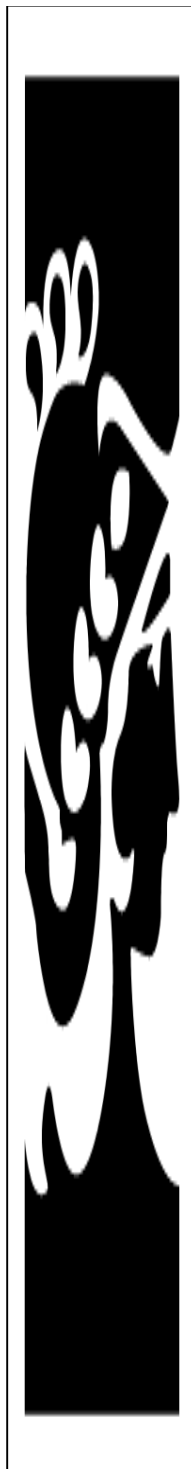
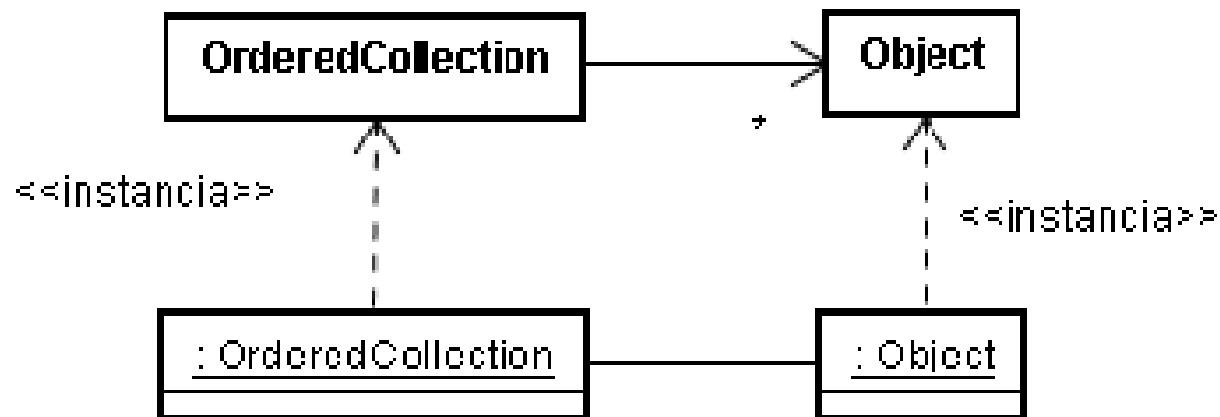
Relación entre individuos

Auto - rueda



Agrupación en OO: agregación

Relación entre objetos



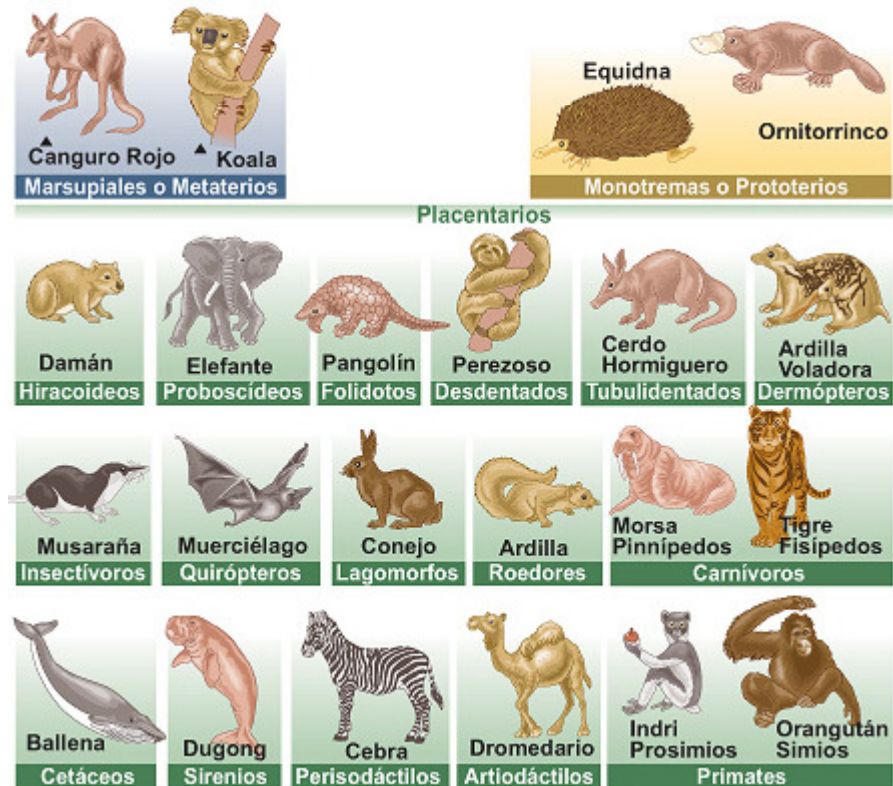
Generalización

Relación entre especies

Lápiz – herramienta de escritura

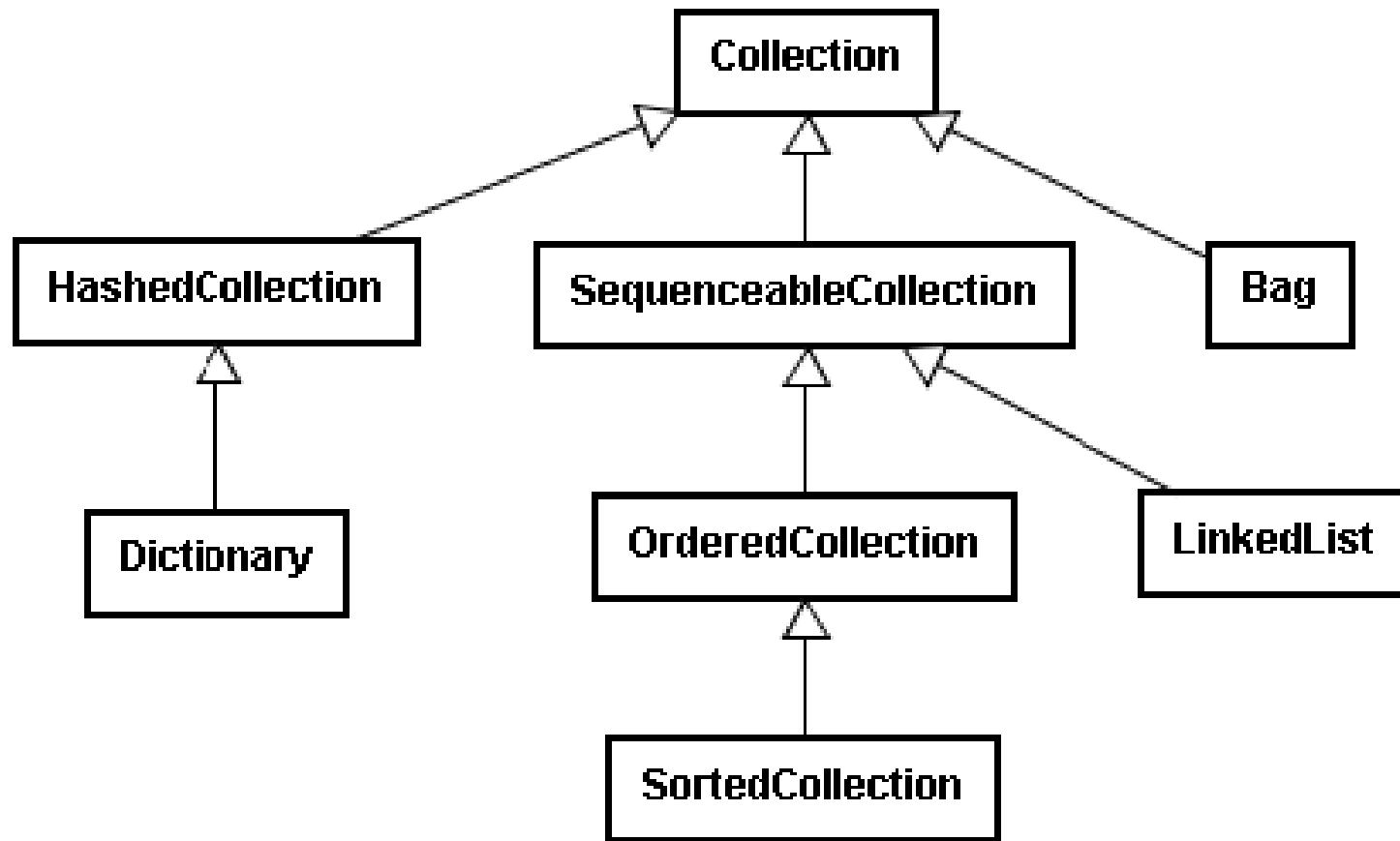
Animal – ser vivo

Al revés: especialización



Generalización en OO: herencia

Relación entre clases



Reglas de POO en Smalltalk

Todo es un objeto

Incluso las clases y las constantes

Todo objeto es instancia de una clase

Toda clase tiene una clase madre

La madre por defecto es Object (en Pharo, ProtoObject)

Todo ocurre debido al envío de un mensaje

El mensaje a invocar se busca siguiendo la jerarquía de herencia

Si no existe hasta Object, hay un “doesNotUnderstand”



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



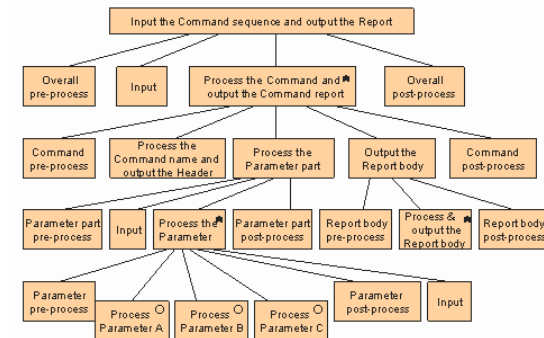
Paradigma “estructurado” o “procedural”

Centrado en lo que “hace” el software

Separación de funcionalidades

Sin separación de entidades o tipos

Lenguajes: Fortran 77, C, Pascal, ¿Ada?



Mini-Historia de paradigmas (2)

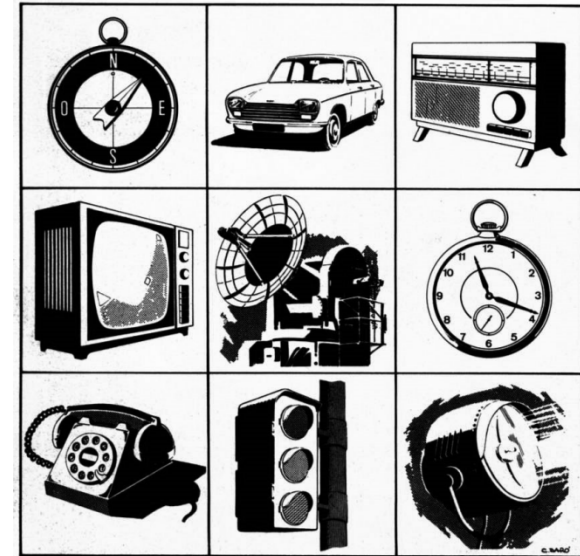
Paradigma orientado a objetos

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 funcional

Lenguajes: AspectJ, Spring, JBoss

Otros paradigmas

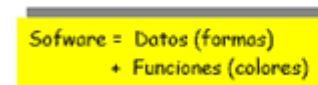
Relacional, funcional, lógico, ...



1ª Generación:
Código spaghetti



2ª y 3ª Generación:
Descomposición
funcional



4ª Generación:
Descomposición en
objetos

POO vs 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 no hay funciones?

Son los “métodos” dentro de las clases

Pero especifican más bien “comportamiento” de entidades

Son secundarias



POO: cómo lograrla

Entidades se convierten en clases

= tipos con comportamiento

Próxima semana



Claves

Se trabaja con objetos y mensajes

Las clases son tipos y los objetos sus instancias

Las clases son conjuntos de objetos

Las clases tienen una jerarquía

Java / C# / Smalltalk:

- Objetos se crean en tiempo de ejecución

- Las variables son referencias a objetos

- 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

Son libros antiguos

No existía Java ni C#, sí Smalltalk: ejemplos en C++

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



Qué sigue

Clases (construcción)

Delegación, herencia, polimorfismo

Construcción de excepciones

