fiuba

algo3

Diseño por contrato y Test-First

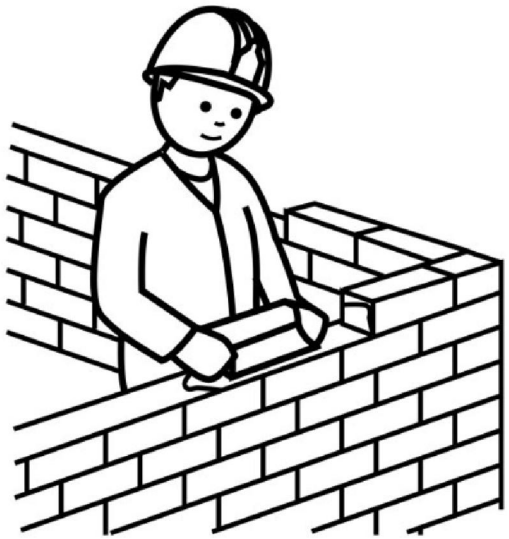
Carlos Fontela
cfontela@fi.uba.ar

Contexto

Vimos la estrategia de resolución de problemas usando objetos

Pero no cómo implementar los propios objetos

=> Construcción de objetos



Diseño por contrato y Test-First

Temario

- Diseño por contrato
  - Conceptos
  - Procedimiento recomendado
- Ejemplo de aplicación del procedimiento
- Pruebas unitarias automatizadas y sus subproductos
- Paradigma OO y sus objetivos

fiuba algo3

Implementación de objetos y lenguajes: ejemplos

Lenguaje	Implementación del comportamiento	Creación de objetos	Comprobación de tipos
Smalltalk	Clases	Dinámica	T de ejecución
Java	Clases	Dinámica	T de compilación
JavaScript	Prototipos	Dinámica	No hay
Python	Clases	Dinámica	T de ejecución
C++	Clases	Estática	T de compilación
C#	Clases	Dinámica	T de compilación
Ruby	Clases	Dinámica	T de ejecución
Self	Prototipos	Dinámica	No hay

Clase: noción

- Clase: conjunto de objetos que, por lo menos, tienen el mismo comportamiento
  - celda23 y celda57 son instancias de la clase Celda
  - Todos los objetos de una clase entienden los mismos mensajes
  - Todos los objetos de una clase responden a los mensajes de la misma manera
- Clase: tipo de un objeto
  - Definible por el programador

Lenguajes basados en clases: implementación de comportamiento

En la clase Celda (Smalltalk):

```
estaLibre
  ^ ( contenido = 0 )
```

En la clase Celda (Java):

```
public boolean estaLibre () {
  return contenido == 0;
}
```

Todos los objetos de la clase usan el mismo método

Métodos y atributos se definen a nivel de clases

Lenguajes basados en prototipos: sólo hay objetos

```
var celda23 = {
    fila: undefined,
    columna: undefined,
    numero: undefined,
    estaLibre: function() {
        return this.numero === undefined;
    }
};
```

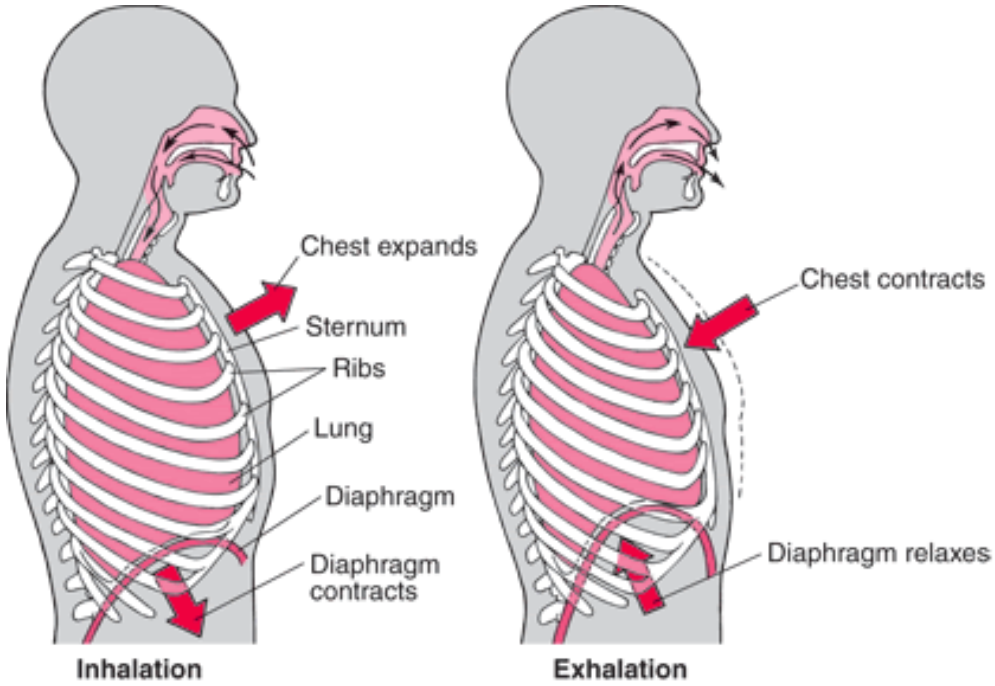
¿Queremos otro objeto similar?

```
celda57 = object.create(celda23);
```

Atributos

Una variable interna del objeto que sirve para almacenar parte del estado del mismo

Ejemplo: numero



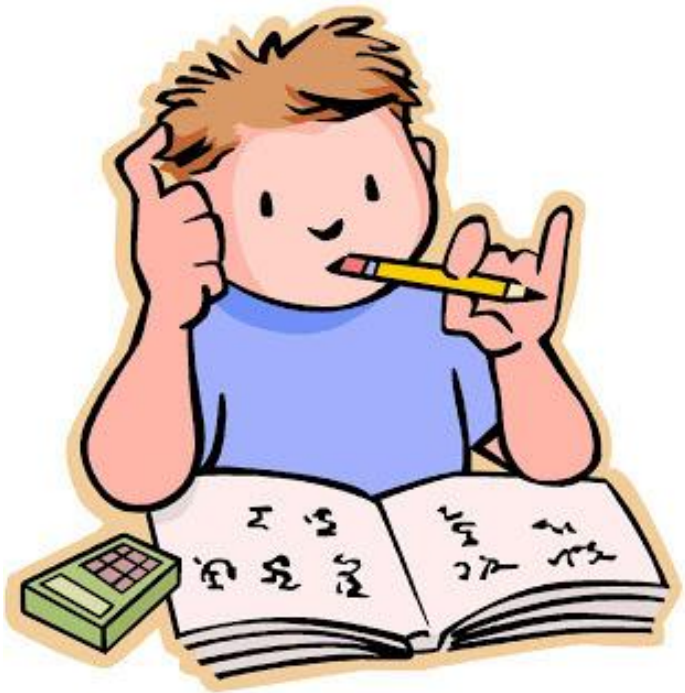
Recapitulación

fiuba algo3

Recapitulación: preguntas

¿Qué es una clase?

¿Qué es un atributo?



fiuba algo3

Lenguajes basados en clases: creación por instanciación

Creación de una celda (Smalltalk):

```
celda23 := Celda new.
```

Creación de una celda (Java):

```
Celda celda23 = new Celda ( );
```

La variable celda23 queda con una referencia a la nueva instancia de Celda

Referencias

Las variables son referencias a objetos:

```
lista1 := OrderedCollection new.
lista1 add: 'hola'.
lista2 := lista1.
```

"lista1" y "lista2" referencian al mismo objeto

(hay una sola llamada a "new")

Si hago:

```
lista2 := OrderedCollection new.
```

Ahora "lista2" referencia a otro objeto

Una variable que no referencia un objeto tiene el valor "nil" (Smalltalk), "null" (Java), "undefined" (JavaScript)

Puedo hacer: `lista := nil.`

Recolección de basura

Si hago:

```
lista1 := OrderedCollection new.
lista1 add: 'hola'.
lista1 := OrderedCollection new.
```

El objeto inicial quedó como basura

Smalltalk, JavaScript y Java tienen mecanismos de recolección automática de basura

fiuba algo3

Recolección de basura

No determinística

Asegura que

- No va a faltar memoria mientras haya objetos sin referenciar
- No se va a liberar ningún objeto que esté siendo referenciado desde un objeto referenciado

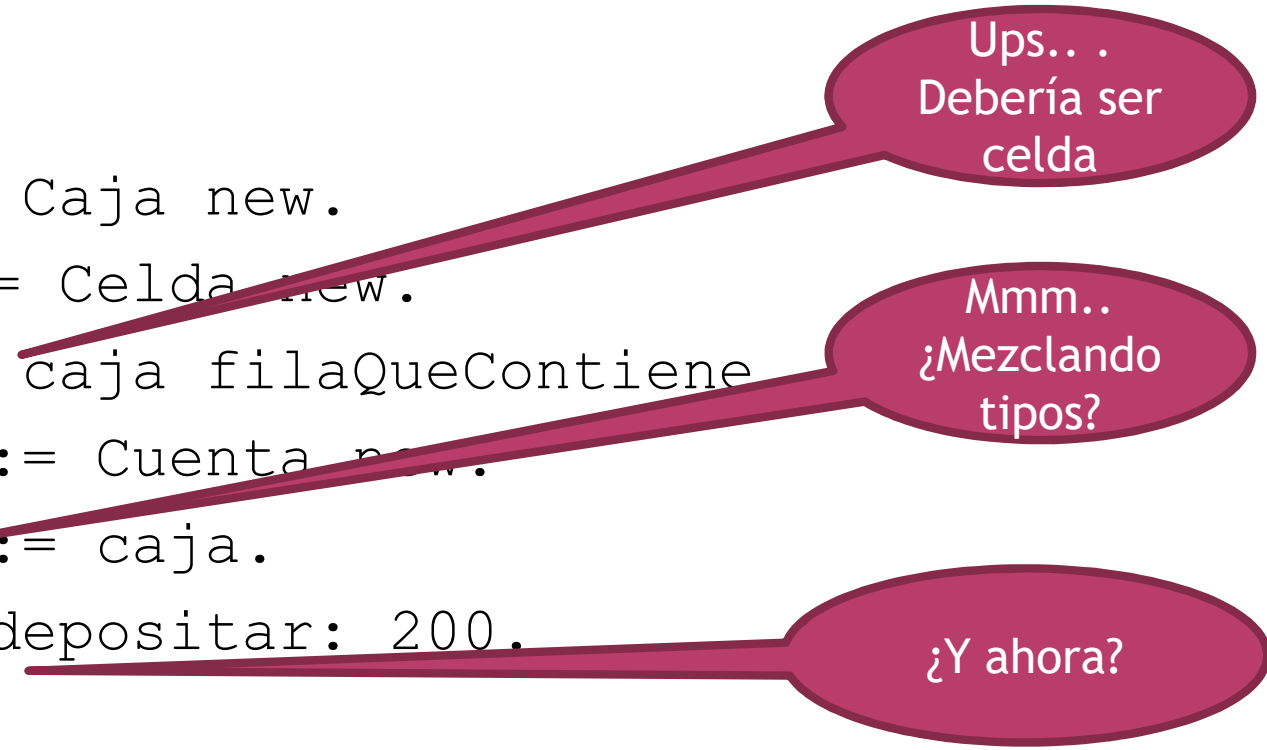
Extremadamente cómoda

Y evita errores muy difíciles de encontrar y reparar

Verificación de tipos: Smalltalk

Analizar

```
caja := Caja new.
celda := Celda new.
fila := caja filaQueContiene
cuenta := Cuenta new.
cuenta := caja.
cuenta depositar: 200.
```



El compilador no encuentra errores

- Saltan en tiempo de ejecución
  - Ejemplo: MessageNotUnderstood: Caja>>filaQueContiene

fiuba algo3

Verificación de tipos: Java

Analizar

```
Caja caja = new Caja();
Celda celda = new Celda();
Cuenta cuenta = new Cuenta();
Fila fila = caja.filaQueContiene();
cuenta = caja;
cuenta.depositar(200);
```

Ups.. . Debería ser celda

Esto no va a andar

¿Y de esto no se da cuenta?

El compilador encuentra errores y no compila el programa

fiuba algo3

Verificación de tipos: ¿flexibilidad o robustez?

fiuba algo3

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

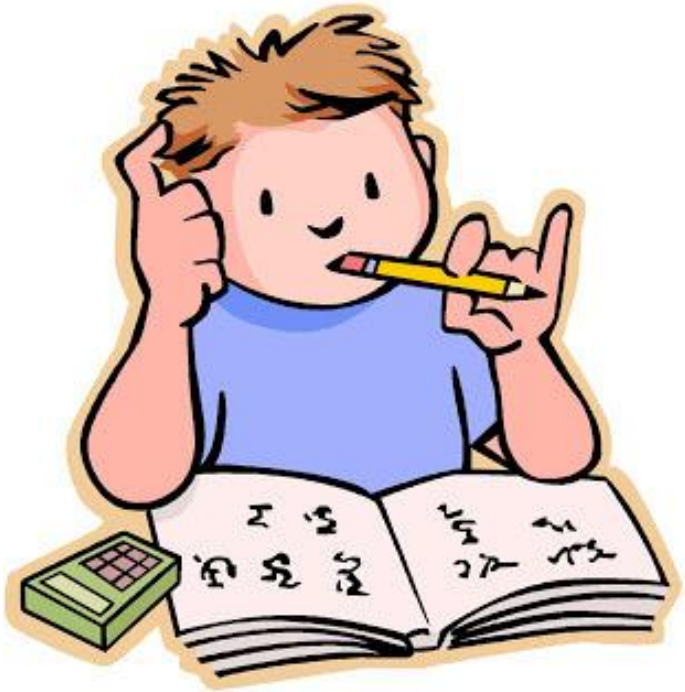
Recapitulación

fiuba algo3

Recapitulación: preguntas

¿Qué diferencia a los lenguajes de comprobación estática y dinámica?

¿Qué ventajas acarrea la recolección de basura?



fiuba algo3

Diseño por contrato

Es una forma de especificar el comportamiento de un objeto

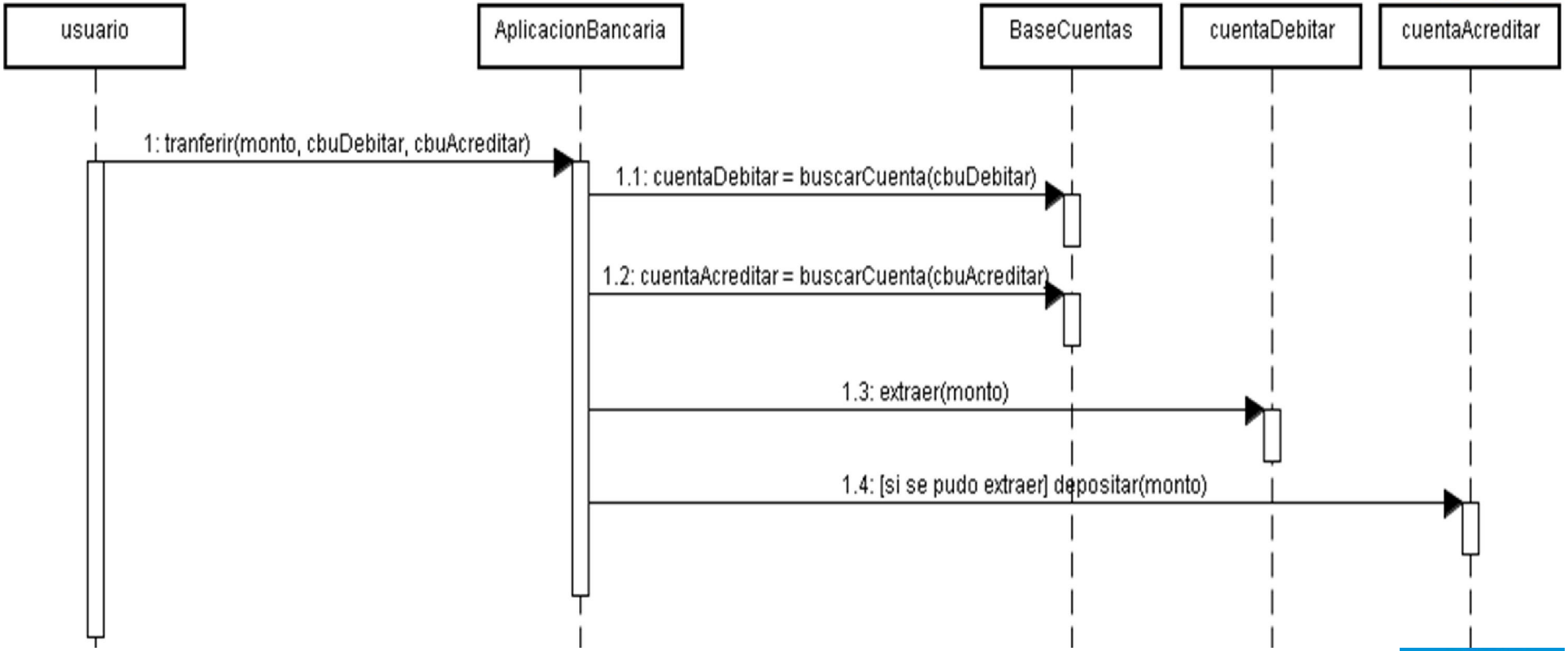
Betrand Meyer: un objeto brinda servicios a sus clientes cumpliendo un contrato

4 elementos

- Firmas de métodos
- Precondiciones
- Postcondiciones
- Invariantes

fiuba algo3

Escenario: transferencia entre cuentas bancarias



Firmas de métodos

Determinan cómo hacer para pedirles servicios a los objetos

Ejemplos

```
AplicacionBancaria >> transferir
  (monto, cbuDebitar, cbuAcreditar)
cuenta >> extraer (monto)
cuenta >> depositar (monto)
BaseCuentas >> buscarCuenta (cbu)
tablero >> fila (filaCelda)
tablero >> caja (filaCelda,
  columnaCelda)
fila >> contiene (numero)
```

Precondiciones

Expresan en qué estado debe estar el medio ambiente antes de que un objeto cliente le envíe un mensaje a un receptor

Ejemplos:

- Para que se pueda depositar en una cuenta, el monto a depositar debe ser un número positivo
- Para que se pueda extraer de una cuenta, la misma debe tener un saldo mayor al monto a extraer

¿Y si una precondición no se cumple?: excepciones

Una excepción es un objeto que el receptor de un mensaje envía a su cliente como aviso de que el propio cliente no está cumpliendo con alguna precondición de ese mensaje

Debería definirse una por cada precondición

Ejemplo:

Si el monto pasado para depositar es cero o negativo, lanzaremos una excepción *MontoInvalido*

Postcondiciones

El conjunto de postcondiciones expresa el estado en que debe quedar el medio como consecuencia de la ejecución de un método

- En términos operativos, es la respuesta ante la recepción del mensaje

Ejemplo:

- Si el monto no es un número mayor que 0, lanzaremos una excepción *MontoInvalido*
- Si las precondiciones se cumplen (camino feliz), el saldo de la cuenta debe verse incrementado en el monto pasado como argumento

¿Y si una postcondición no se cumple?: algo hicimos mal

Precondición: responsabilidad del cliente

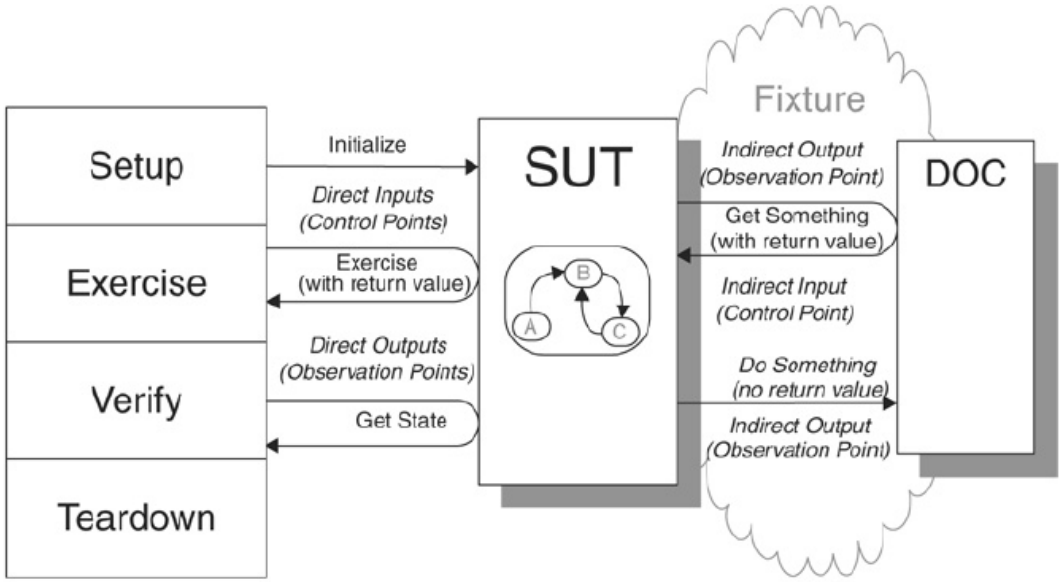
Postcondición: responsabilidad del programador del comportamiento del objeto receptor

=> se debe chequear con alguna prueba que verifique los resultados esperados del método

Pruebas unitarias

Una prueba unitaria es aquélla prueba que comprueba la corrección de una única responsabilidad de un método

Corolario: deberíamos tener al menos una prueba unitaria por cada postcondición



fiuba algo3

Invariantes

Son condiciones que debe cumplir un objeto durante toda su existencia

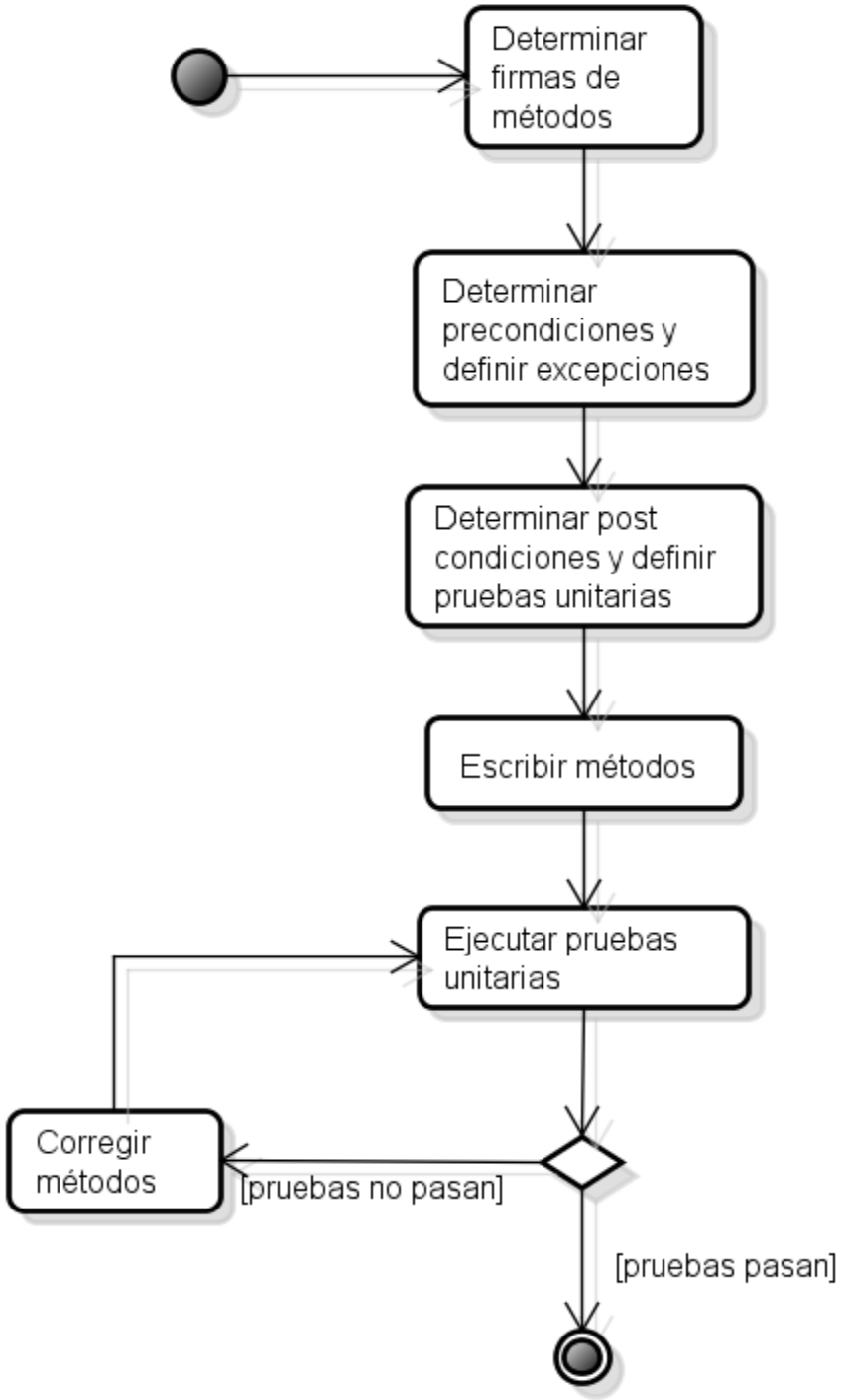
Ejemplo:

Cualquier objeto *cuenta*, o bien no tiene saldo, o su saldo es mayor que 0

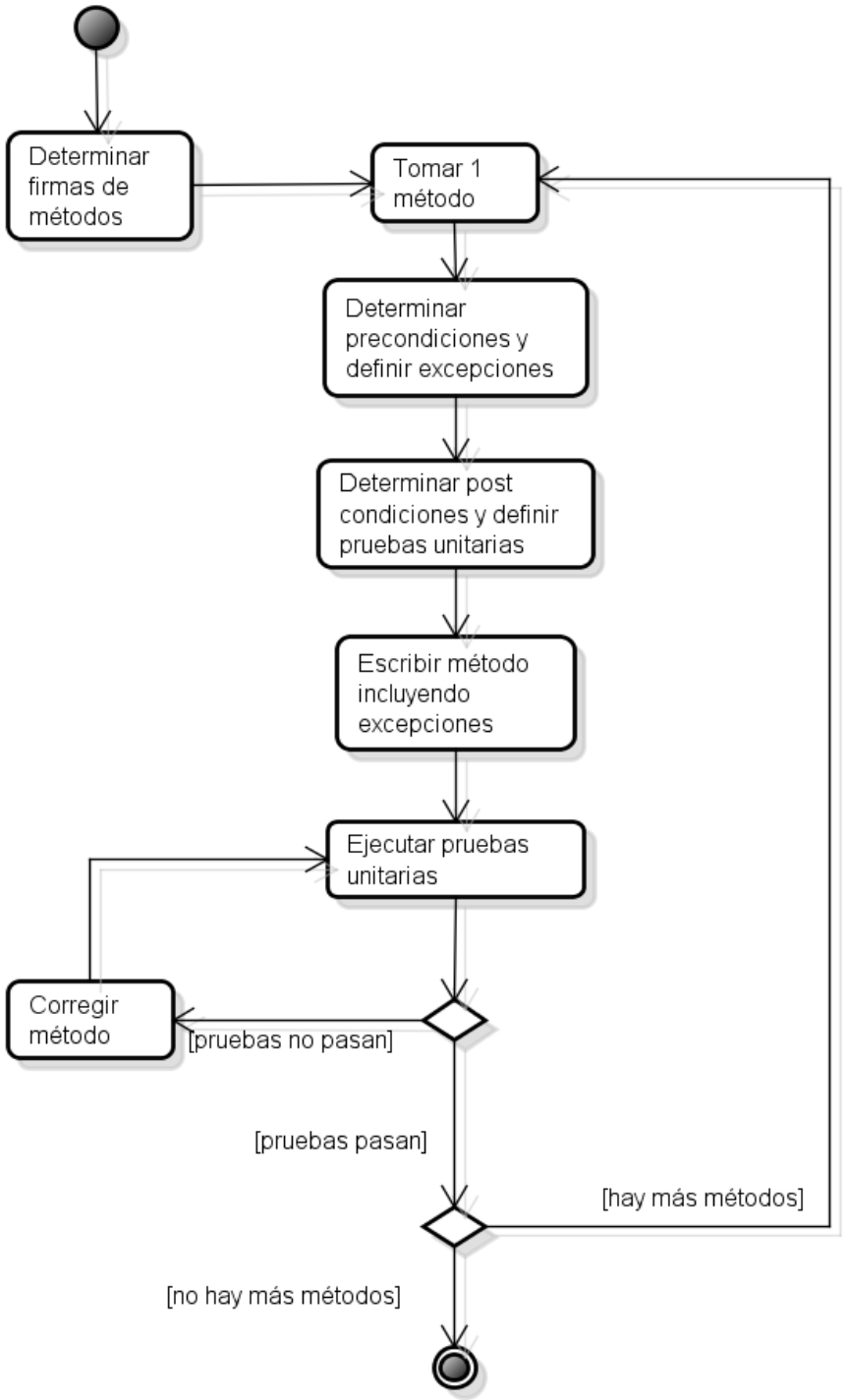
Responsabilidad de todos los métodos de un objeto

Suelen expresarse en forma de precondiciones o postcondiciones

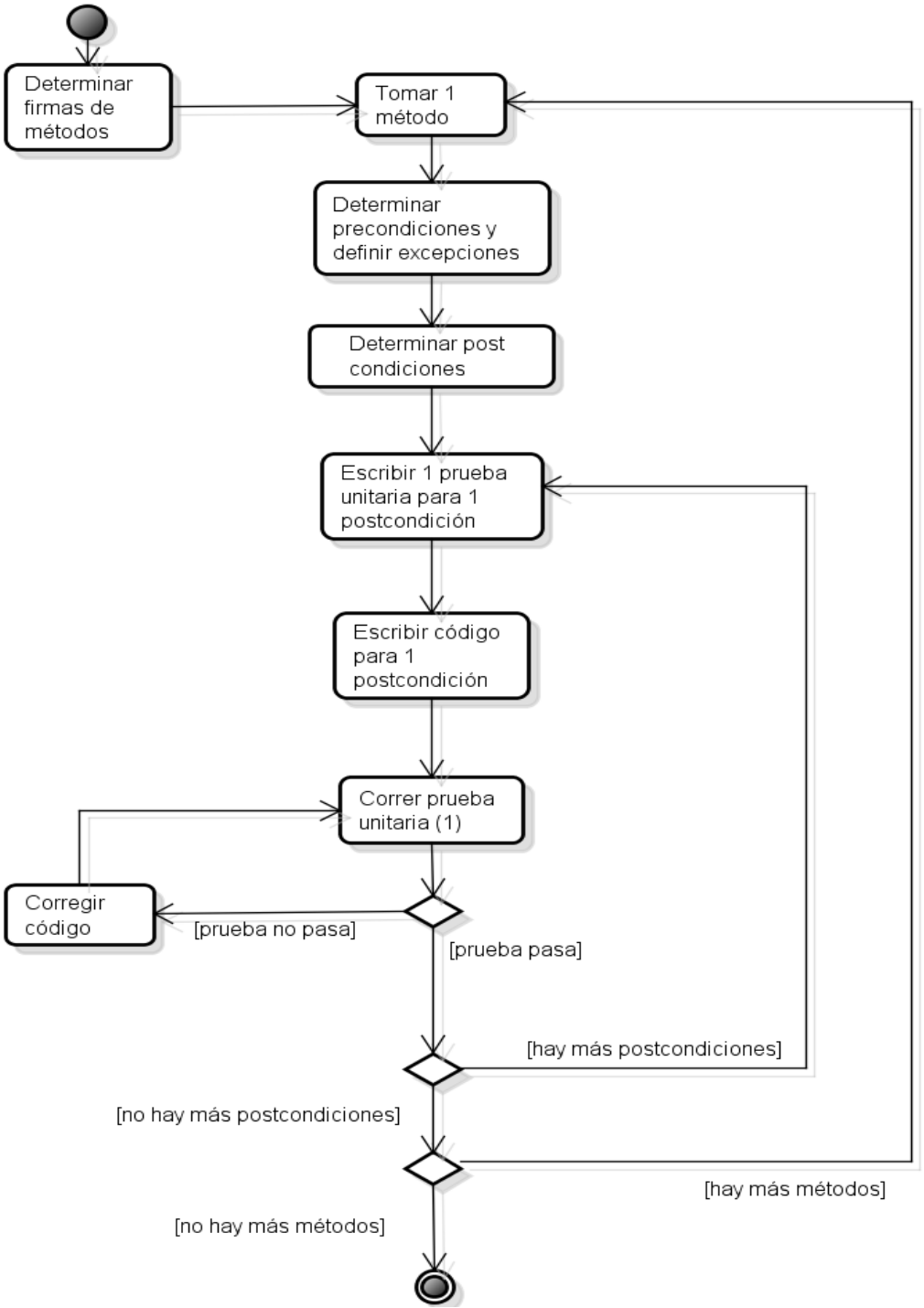
Procedimiento básico



Procedimiento incremental



Procedimiento con incrementos pequeños



fiuba algo3

Recapitulación

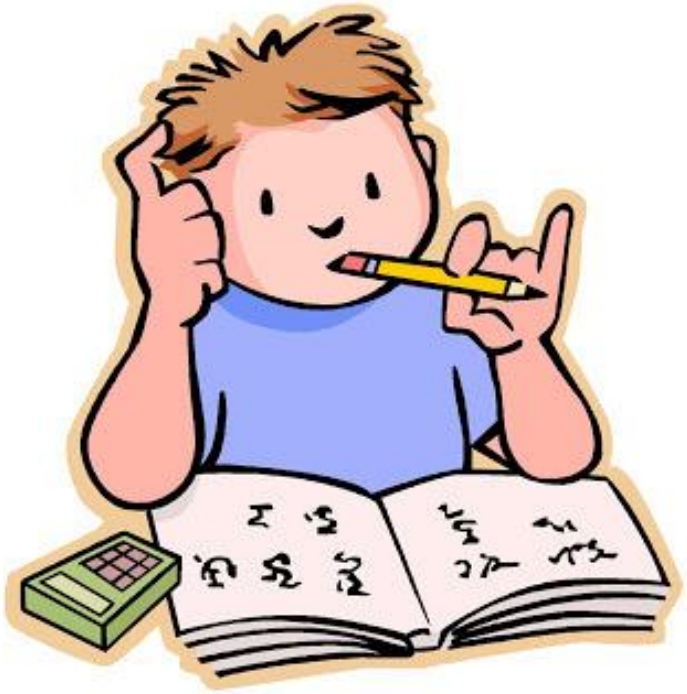
fiuba algo3

Recapitulación: preguntas

¿Para qué me sirve definir las precondiciones?

¿Y las postcondiciones?

¿Qué es una excepción?



¿Hacemos un ejemplo?

Implementemos un método *depositar* para un objeto cuenta bancaria

Pasos, en forma iterativa:

- Firmas de métodos
- Descubrir precondiciones
- Definir excepciones
- Definir postcondiciones
- Escribir pruebas unitarias
- Escribir el método

fiuba algo3

¡A trabajar!

fiuba algo3

Recomendaciones post-ejemplo

- Volver a ejecutar todas las pruebas del sistema a ciertos intervalos de tiempo
  - Para asegurarnos de que los últimos cambios no introdujeron nuevos errores
- Prueba de regresión: se ejecuta luego de un cambio para comprobar que no haya afectado partes del programa que debieran seguir funcionando bien
  - Conviene ejecutarlas luego de cada cambio
- Definir constructores para inicializar objetos
  - En Smalltalk, método initialize

Invariantes e inicializadores

- Invariantes deben cumplirse durante toda la vida del objeto
  - También a la salida de un constructor
- En Smalltalk es un método más
  - `Clase >> initialize`
  - Llamado automáticamente por new
  - ¡Sin parámetros!
- En Java hay constructores
  - `Clase()`
  - Pueden tener parámetros

Atributos y propiedades: ojo con las apariencias

≠

No todos los atributos tienen “accesors” / “getters” y “setters”

- Sólo los necesarios

Hay propiedades que no corresponden a atributos

- `unString size` => ¿tiene que haber un atributo?
- `numeroComplejo getModulo` => propiedad calculable

Noción: (propiedad = “atributo conceptual”)

=> Los atributos conceptuales deberían estar implementados al menos con “getters”

Desarrollo que empieza por las pruebas: ventajas

- Minimiza el condicionamiento del autor
  - No se prueba solamente el camino feliz
- Las pruebas se convierten en especificaciones de lo que se espera como respuesta del objeto ante el envío de un mensaje
  - => No se debería codificar nada que no tenga su prueba (no implementar lo que no se necesita)
- Permite especificar el comportamiento sin restringirse a una implementación
  - => Se refuerza el encapsulamiento
- Deja un conjunto de pruebas de regresión
  - Que puede ir creciendo junto con el programa

Pruebas en código: ventajas

- Permite independizarse del factor humano
  - Menor subjetividad y variabilidad en el tiempo
    - Mismo resultado un viernes que un miércoles
    - A las 6 de la tarde o a las 11 de la mañana
    - Al principio de un proyecto o minutos antes de una entrega
- Facilita la repetición de las mismas pruebas, con un costo ínfimo
  - El programador las ejecuta con mayor frecuencia y regularidad
    - => trabaja con más confianza, porque el código que escribe siempre funciona

¿TDD?

Test-Driven Development

- Práctica iterativa-incremental de desarrollo de software

Incluye:

- Test-First
  - Listo

- Automatización
  - Bastante bien, pero mejorable
- Refactorización
  - No lo vimos

fiuba algo3

Subproducto: pruebas como ejemplos

Documentan

- Firma del método
- Posibles excepciones
- Resultados que se esperan de su ejecución

Mejor que cualquier documento escrito en prosa

Se suelen mantener junto con el código

=> resulta más fácil mantenerlas actualizadas

Subproducto: control de calidad

No es el objetivo con el que las hacemos

Pero está bien: sirven también como herramientas para los controles de regresión

Ojo: las pruebas unitarias nunca pueden ser suficientes para controlar la calidad de un producto

Necesitaremos pruebas más abarcativas

- Pruebas de integración (en el sentido de que prueben escenarios con varios objetos)
- Pruebas de aceptación de usuarios
- Pruebas exploratorias

Herramientas xUnit

Para facilitar la construcción y ejecución de pruebas automáticas

- SUnit, JUnit, NUnit, etc.
- Integradas en los IDE

Trabajo en la práctica

Veamos…

Recapitulación

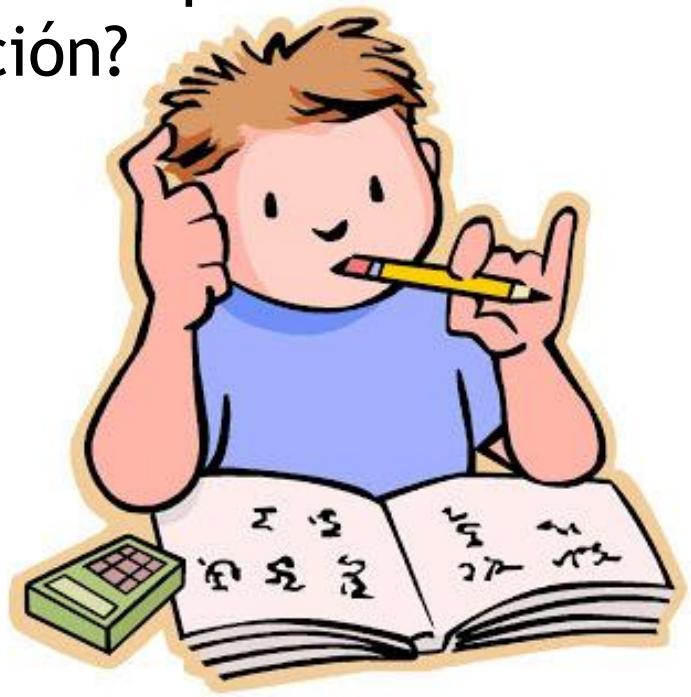
fiuba algo3

Recapitulación: preguntas

¿Por qué escribir las pruebas antes?

¿Sirven las pruebas unitarias como elemento de control de calidad?

¿En qué sentido las pruebas son documentación?

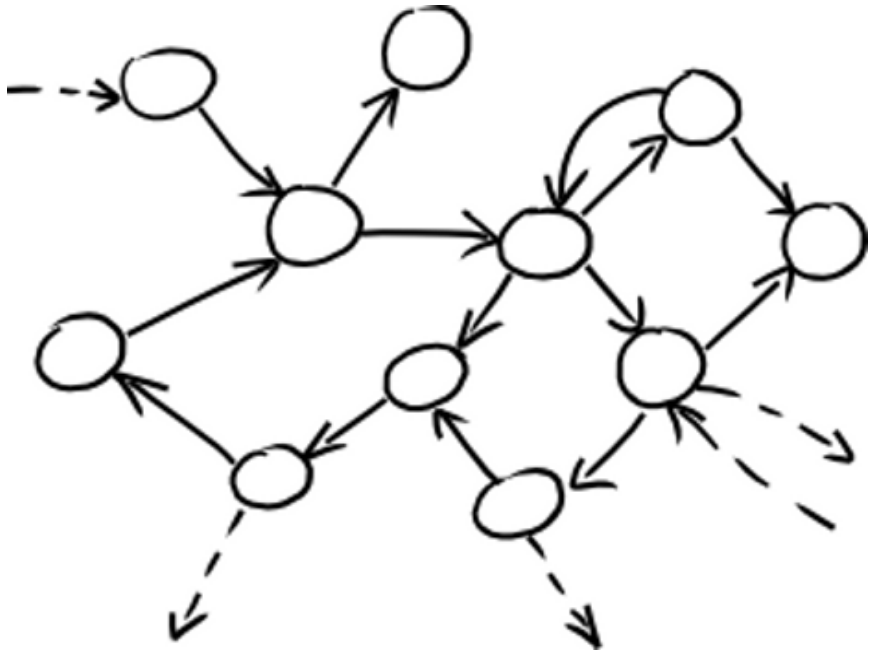


fiuba algo3

Paradigma OO y complejidad

OO permite el manejo de la complejidad a un costo razonable

=> Construir en base a componentes



fiuba algo3

Paradigma OO y componentes

- Componentes en POO = Objetos
- Condición 1: encapsulamiento
  - Cada componente debe tener el comportamiento esperado
  - Sin que nuestro desarrollo dependa de la manera en que está implementado
- Condición 2: contratos
  - Qué nos ofrece cada componente
  - Y cómo conectarnos con él
- Encapsulamiento + Contratos = Abstracción

Recapitulación

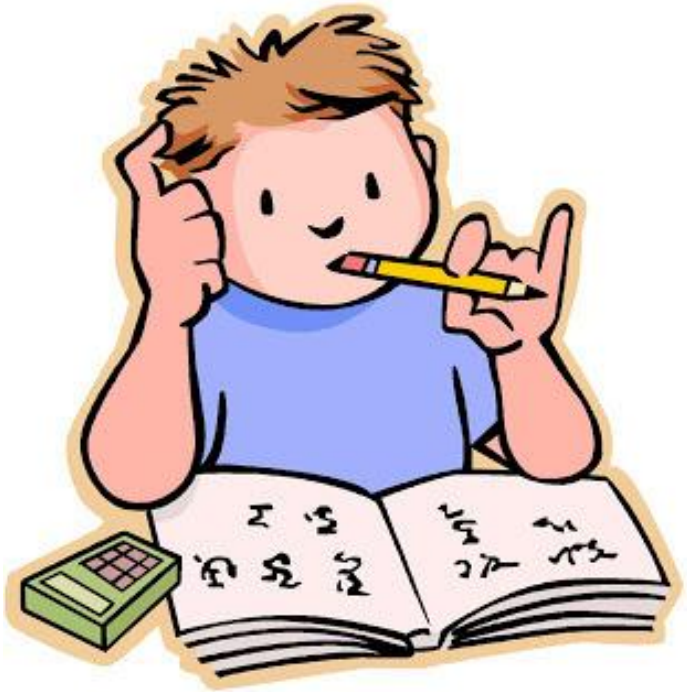
fiuba algo3

Recapitulación: preguntas

¿Para qué sirve la OO?

¿A qué llamamos abstracción?

¿Por qué le parecen importantes el encapsulamiento y los contratos?



fiuba algo3

Claves

Los objetos se implementan en base a un modelo cliente-proveedor

De las precondiciones deducimos las excepciones

Con las postcondiciones escribimos pruebas unitarias

Las pruebas unitarias guían la implementación de a incrementos pequeños

Hoy OO sirve para manejar la complejidad

Y su auxiliar es la abstracción

Lecturas obligatorias

Apunte de Pruebas (campus)

"Unit Testing Guidelines",

http://geosoft.no/development/unittesting.html

"8 Principles of Better Unit Testing",

http://esj.com/Articles/2012/09/24/Better-Unit-Testing.aspx?p=1

algo3

Lecturas opcionales

- Object-Oriented Software Construction, Bertrand Meyer
  - Está en la biblioteca
  - Especialmente capítulos 7, 8, 11 y 12
- Test Driven Development: By Example, Kent Beck
  - No está en la Web ni en biblioteca
- Code Complete, Steve McConnell, Capítulo 6: “Working Classes”
  - No está en la Web ni en biblioteca
- Implementation Patterns, Kent Beck, Capítulos 3 y 4: “A Theory of Programming” y “Motivation”
  - No está en la Web ni en biblioteca
- Orientación a objetos, diseño y programación, Carlos Fontela 2008, capítulo 3 “Programación basada en objetos” y capítulo 4 “Construcción de clases”

Qué sigue

- Colaboraciones de objetos
  - Más herencia y delegación
- Polimorfismo
- Refactorización
- Temas varios de POO