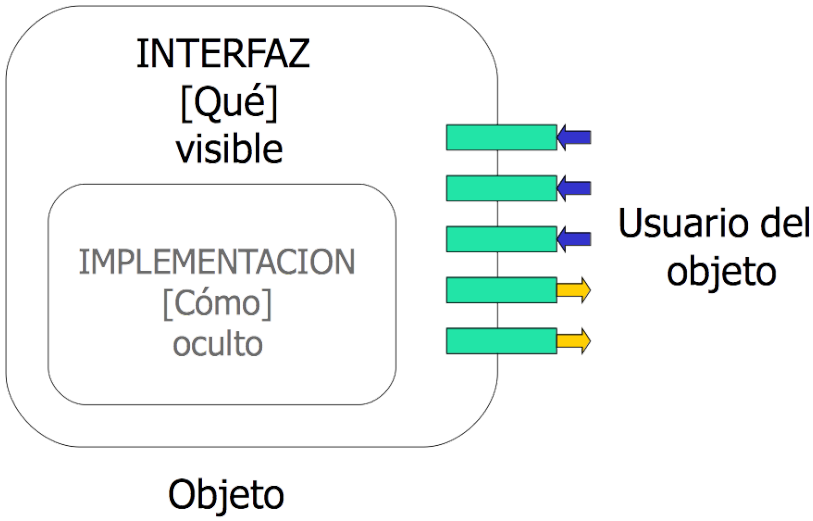
Programación 3 - Guía de Estudio

Esta guía está pensada como un resumen de todos los conceptos vistos en clase de teoría. Úsala como una referencia a la hora de estudiar la materia de Programación 3. Se incluyen referencias online donde se describen algunos conceptos o donde se puede ampliar información sobre ellos. Al final tienes una bibliografía que está disponible en la biblioteca de la universidad. Algunos libros es posible que estén disponibles también en formato digital desde la biblioteca.

U.D. 1 - Introducción al paradigma OO. Clases y objetos

Clases. Objetos (automáticos y en memo. dinámica). Referencias. Atributos de instancia. Visibilidad pública/privada. Getters/Setters. Constructores y destructores.



2.4.2 Tipos
2.4.3 Atributos de clase
2.5 Operaciones (métodos)
2.5.1 Signatura de un método
2.5.2 Métodos de instancia
2.5.3 Autoreferencia (this)
2.5.4 Métodos de clase (static)
2.5.5 Operaciones sobrecargadas
2.6 Constructores
2.6.1 Constructor por defecto
2.6.2 Constructor sobrecargado
2.6.3 Constructor de oficio
2.7. Copia de objetos
2.7.1 Constructor de copia / método clone()
2.7.2 Copia superficial
2.7.3 Copia profunda
2.7.4 Copia defensiva
2.8 Objetos inmutables (Java)
2.9 Destrucción de objetos
Destructores(C++); recolección de basura y método finalize() (Java)
2.10 Forma canónica de una clase

Enlaces de interés:

- Objetos inmutables: http://www.dlsi.ua.es/asignaturas/prog3/Objetos_inmutables.html
- Introducción a la programación orientada a objetos Introducción rápida y breve los conceptos OO básicos
- Introduction to Object Oriented Programming Concepts (OOP) and More - CodeProject (C#/.NET)

Diferenciar entre clases y objetos:

- http://blogs.ua.es/progoo/2009/10/14/clases-u-objetos/
- http://www.programmerinterview.com/index.php/java-questions/difference-between-object-and-class/
- https://es.wikibooks.org/wiki/Fundamentos_de_programaci%C3%B3n/Reconocimiento_de_Objetos_y_Clases
- https://es.wikibooks.org/wiki/Fundamentos_de_programaci%C3%B3n/Reconocimiento_de_Objetos_y_Clases_en_el_mundo_real

Holub Associates: UML Reference Card

Cursos OpencourseWare completos relacionados con la programación orientada a objetos:

- Elements of Software Construction | MIT OpenCourseWare
- Introduction to Programming in Java | MIT OpenCourseWare

The Java equivalent of 'const': Java code
Java Canonical Class Form

Copia de objetos:

- Difference between Shallow Copy and Deep Copy using clone()
- Java Practices -> Copy constructors
- Java, Copy Constructors, and clone() | xenoveritas.org

Destrucción de objetos

- Object finalization and cleanup - JavaWorld

Abstracción y encapsulamiento:
https://es.wikibooks.org/wiki/Fundamentos_de_programaci%C3%B3n/La_Abstracci%C3%B3n_y_el_Encapsulamiento

U.D. 2 - Relaciones entre objetos

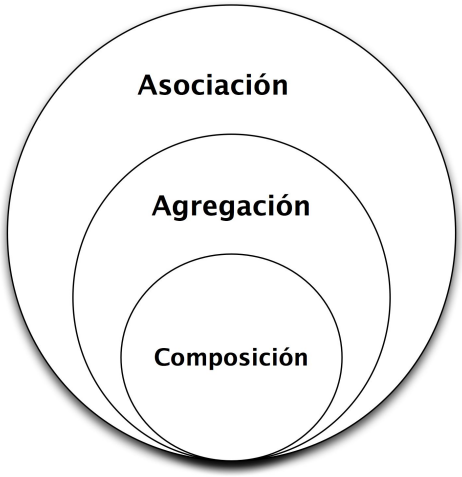
- Relaciones persistentes y no persistentes
- Asociación
  - representación UML
  - nombre de relación
  - cardinalidad (unaria/binaria)
  - direccionalidad (uni/bidireccional)
  - navegabilidad (quién conoce a quién)
  - multiplicidad (1:1, 1:N, M:N, otros…)
  - roles; visibilidad del rol (papel que juega el objeto en la relación; nombre del atributo usado para referirse a él)
  - tiempo de vida de los objetos independiente entre sí
  - implementación (1:1, 1:N, o 1:*)

- Relaciones Todo-parte
  - Asimetría y transitividad de las relaciones todo-parte
  - Agregación y composición (agregación ‘fuerte’)
  - Caracterización de la composición
    - ‘todo’ como propietario único de las ‘partes’
    - Tiempo de vida de las partes == tiempo vida del todo
  - Implementación
    - Atributos
    - Composición: copia defensiva (tiempo de vida de las partes)

- Resumen de relaciones persistentes
  - El diagrama pretende mostrar cómo la agregación es un caso particular de asociación y la composición un caso particular de agregación. Este diagrama aparece en varios lugares, como en

http://stackoverflow.com/questions/885937/difference-between-association-aggregation-and-composition

	Asociación	Agregación	Composición
Propietario	No hay	La clase 'todo'	La clase 'todo'
Tiempo de vida	independiente	independiente	Del propietario
Objetos subsidiarios	No hay	Pertenecen al 'todo'	Pertenecen al 'todo'



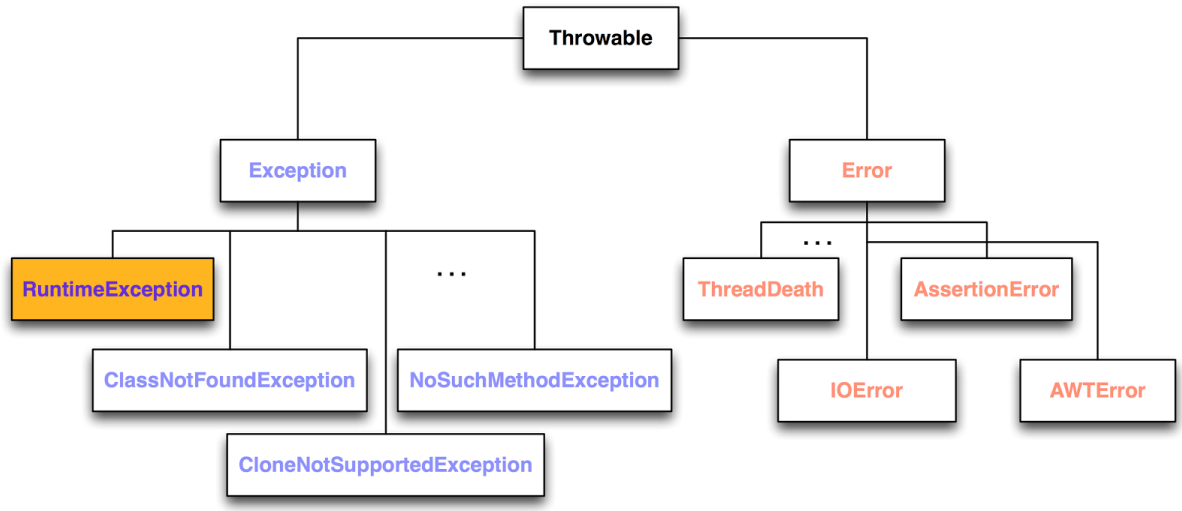
- Relaciones de uso/dependencia

Enlaces de interés

Association, Aggregation and composition implementation in java (Java in General forum at JavaRanch)
UML Relations & Java Implementation (OO, Patterns, UML and Refactoring forum at JavaRanch)
Understanding Association, Aggregation, and Composition - CodeProject

U.D. 3 - Gestión de errores

- Concepto
- Ventajas frente al tratamiento de errores tradicional (código espaguetti)
- Comportamiento
    - Lanzamiento (throw)
    - Captura (bloques try/catch/finally)
- Especificación de excepciones (cláusula ‘throws’)
- Excepciones estándar en Java



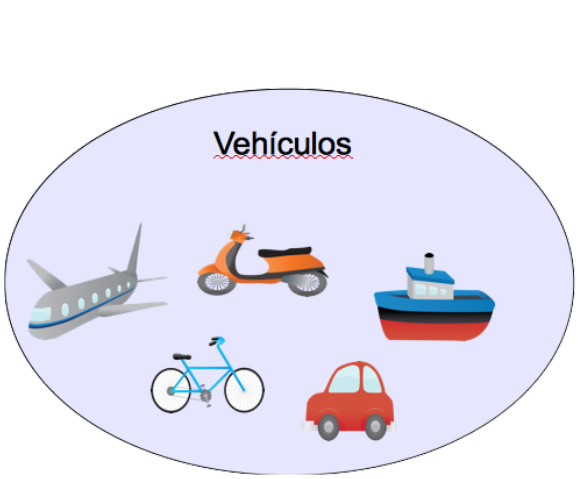
- Excepciones declaradas (*checked exceptions*): Exception
- Excepciones no declaradas (unchecked exceptions): RuntimeException
- Excepciones de usuario (extends Exception/RuntimeException)
- Particularidades de las excepciones
    - Orden de captura de excepciones
    - Bloque finally
    - Captura de varias excepciones (catch múltiple)
    - Excepciones en constructores
- Regeneración (relanzamiento) de excepciones

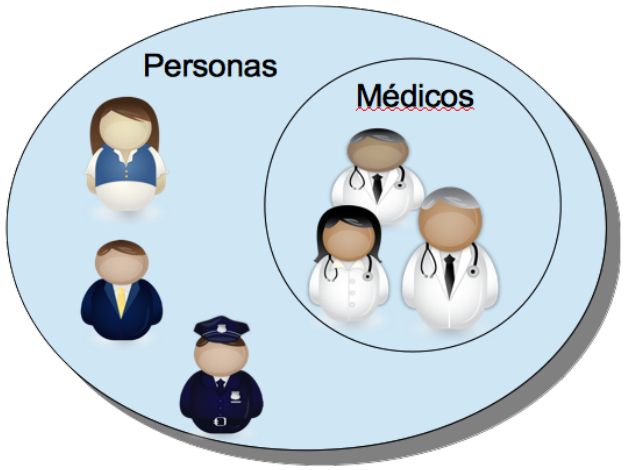
Enlaces de interés

[JavaSpecialists 120] - Exceptions From Constructors
c++ - Why should exceptions be used conservatively? - Stack Overflow
http://www.dlsi.ua.es/asignaturas/prog3/ejercicios/excepciones.html

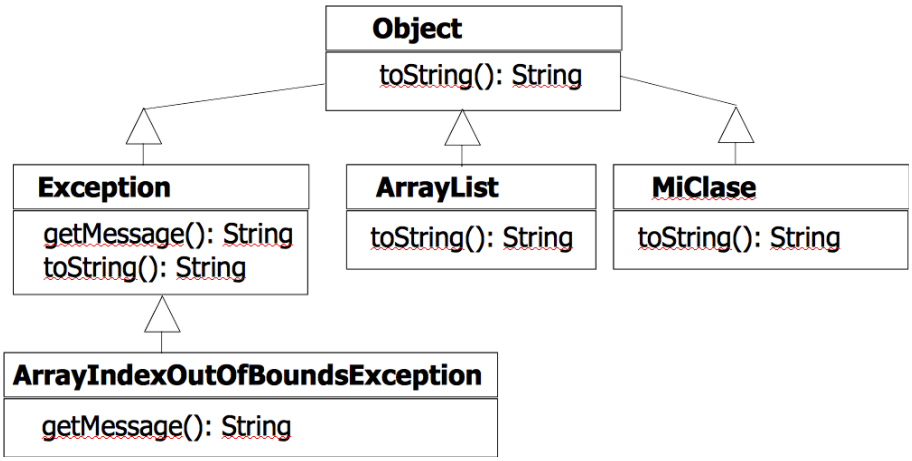
U.D. 4 - Herencia de implementación

- Generalización
    - superclase/clase base
- Especialización
    - subclase/clase dereivada
- Notación UML para herencia de implementación
- **Clase = Conjunto, Subclase = Subconjunto.**
- Subclases **extienden** comportamiento de la superclase.

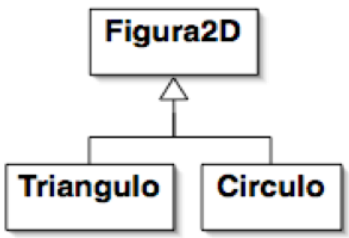


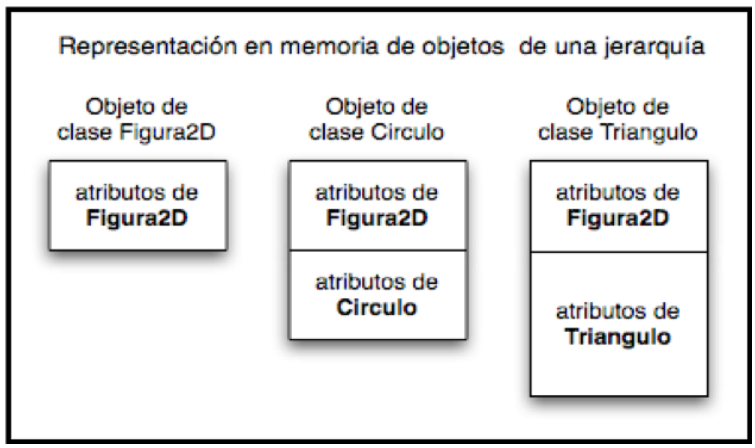


- Jerarquía de clases

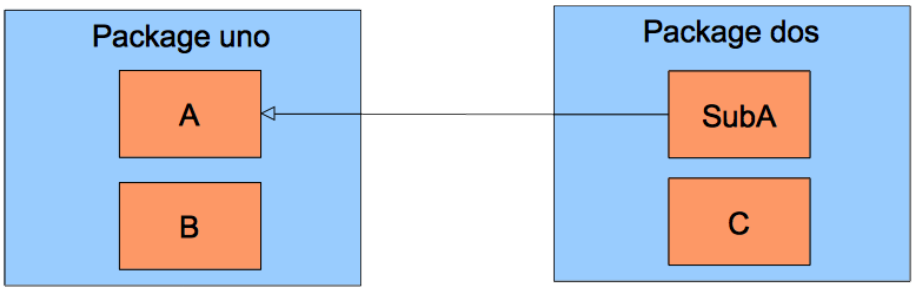


- La **herencia** como implementación de la generalización
- Tipos de herencia: herencia de implementación vs. herencia de interfaz.
- Semántica de la herencia
    - solapada/disjunta
    - completa/incompleta
    - Estática/dinámica
- **Herencia simple de implementación**



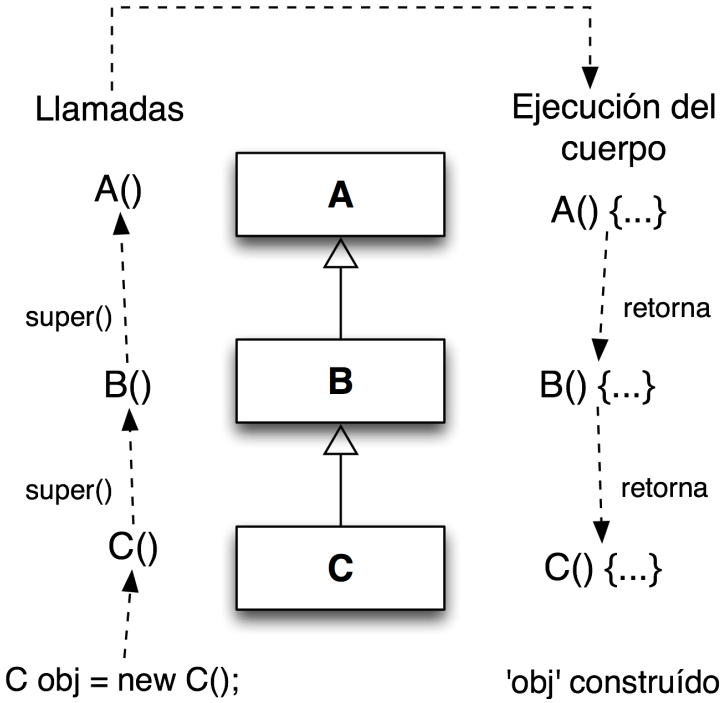


Visibilidad y herencia:



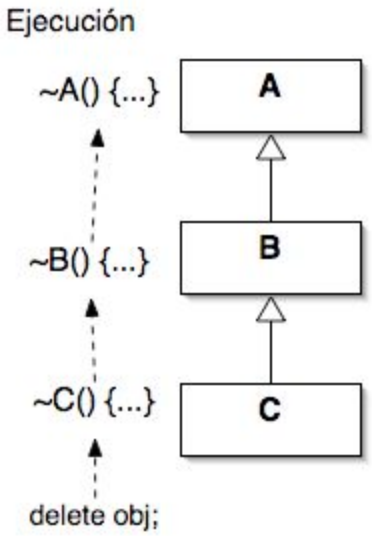
En A	A	B	SubA	C
public	ok	ok	ok	ok
protected	ok	ok	ok	-
package	ok	ok	-	-
private	ok	-	-	-

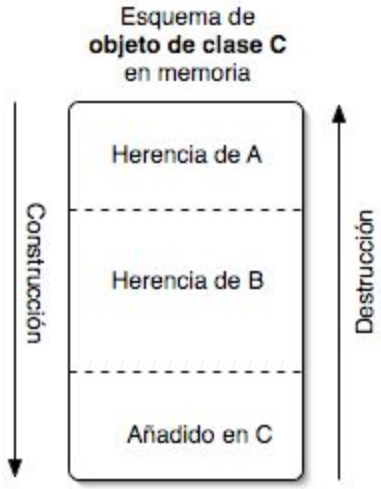
- Implementación de métodos en las subclases
  - Reemplazo
  - Refinamiento (uso de 'super')
- Constructores en herencia simple de implementación



  - Llamada a constructores de superclase (super())

- Destructores y herencia (C++)





- Destrucción de objetos en Java (finalize()) y herencia
- Upcasting
    - *Object slicing* en C++

- Herencia múltiple (de implementación)
    - Problemas en la herencia múltiple de implementación (en C++)
        - Colisión de nombres.
        - Duplicación de propiedades heredadas (‘herencia en diamante’)

- https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B
- http://javarevisited.blogspot.com.es/2012/03/finalize-method-in-java-tutorial.html
- Piensa en Java, 4ª ed., capítulo 5: Inicialización y limpieza, pag. 97 y ss.

U.D. 5 - Compilación, enlace y gestión de memoria en lenguajes OO

- Sistemas de tipos
    - Lenguajes fuertemente tipados
    - Lenguajes débilmente tipados
    - Tipado estático (tiempo de compilación)
    - Tipado dinámico (tiempo de ejecución)
- Compilador
    - Fases de un compilador
- Intérprete
- Tiempo de enlace
    - Estático

    - Dinámico
- Representación en memoria de clases y objetos en la máquina virtual de Java
    - La pila (stack)
    - El heap
    - El área de métodos

Enlaces de interés

Gestión de memoria en Java (en inglés)
Java (JVM) Memory Model – Memory Management in Java

U.D. 6 - Polimorfismo I. Sobrecarga y sobrescritura.

- Conceptos previos
    - Enlace de un identificador
    - Ámbitos
        - Espacio de nombres
            - Java: paquetes
            - C++: *namespaces*
- Polimorfismo
    - Definición
    - Tipos: sobrecarga, sobrescritura, variables polimórficas
- Sobrecarga
    - Basada en ámbito
    - Basada en signatura
    - Sobrecarga de operadores (C++)
    - Alternativas
        - Funciones poliádicas
        - Coerción
- Polimorfismo en herencia
    - Shadowing
    - Redefinición
    - Sobrescritura
        - covarianza
- Variables polimórficas
    - Variables receptoras
    - Downcasting
        - Estático (C++)
        - Dinámico (Java y C++)
        - Downcasting dinámico seguro
            - usando `Class`
            - usando `instanceof`

            - usando `Class.isInstance()`
        - Polimorfismo puro

U.D. 7 - Herencia de interfaz

- Herencia de implementación vs. herencia de interfaz
- Objetivos de la herencia de interfaz
- El principio de sustitución de Liskov
- Implementación
    - Clases abstractas
    - Interfaces
        - H. simple y múltiple
- Notación UML de la herencia de interfaz (clases y métodos abstractos, interfaces)
- Resumen Herencia
    - Herencia de implementación como reuso de código
    - Herencia de interfaz como reuso de conceptos

[The Liskov Substitution Principle - Robert C. Martin]

U.D. 8 - Polimorfismo II. Genericidad y reflexión

Genericidad
    Tipos genéricos
        Parámetros de tipo (Foo<T>) y argumentos de tipo (Foo<String>)
        Operador diamante (Foo<>)
        Múltiples parámetros de tipo
    Métodos genéricos
    Genericidad restringida
        Parámetros de tipo acotados
    Genericidad y herencia
        Subclases (genéricas o no) de clase genérica
        Box<Perro> != subtipo de Box<Animal>
    Comodines de tipo (Foo<?>)
    Borrado de tipos (Java)
    Genericidad en C++ (*templates*)
Reflexión
    Definición
    Usos
        Obtener la clase de un objeto
        Crear un objeto
        Invocar a un método de instancia o de clase

Enlaces de interés:

Lecciones de Oracle sobre genericidad
Java theory and practice: Generics gotchas

U.D. 9 - Frameworks y librerías

1. Definición y diferencia entre librería y framework
2. Concepto de clase anónima
3. Inversión de control
    a. Principio de Hollywood
4. Algunos frameworks Java
    a. Java Collections Framework (JCF)
    b. JUnit
    c. JDBC
    d. Hibernate

U.D. 10 - Técnicas de mantenimiento de código OO.

1. Ciclo de vida del software
    a. Situaciones de mantenimiento de código
        i. Corregir errores
        ii. Añadir nuevas funcionalidades
        iii. Errores de regresión
2. Refactorización
    a. Cambios funcionales y no funcionales
    b. Definición
    c. Procedimiento básico
        i. Introducir cambio
        ii. Probar sistema
    d. Objetivo de la refactorización
    e. Reglas de oro de la refactorización
        i. Pruebas
        ii. Metáfora de los dos sombreros
    f. Cuando no refactorizar
    g. Cuando refactorizar
        i. Código sospechoso (bad smell code)
            1. Código duplicado
            2. https://refactoring.guru/catalog
    h. Problemas al refactorizar
        i. Capa de persistencia (BBDD)
        ii. cambios de interfaz pública
            1. Interfaces ‘publicados’ y métodos @Deprecated
3. Pruebas

a. Pruebas funcionales (de caja negra)
b. Pruebas unitarias
    i. Características
    ii. Desarrollo basado en pruebas (test-driven development)

Enlaces de interés:

- Desarrollo guiado por pruebas - Wikipedia, la enciclopedia libre
- Refactoring guru

U.D. 11 - Principios fundamentales del paradigma OO.

- Acoplamiento y cohesión
- Patrón y antipatrón
- Principios STUPID
    - Singleton
    - Tight Coupling
    - Untestability
    - Premature Optimization
    - Indescriptive Naming
    - Duplication
- Principios de diseño SOLID
    - Single Responsibility Principle
    - Open/Closed Principle
    - Liskov Substitution Principle
    - Interface Segregation Principle
    - Dependency Inversion Principle

Enlaces de interés:
http://williamdurand.fr/2013/07/30/from-stupid-to-solid-code/
https://web.archive.org/web/20150905081111/http://www.objectmentor.com/resources/articles/lsp.pdf
http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

BIBLIOGRAFÍA GENERAL

Los libros que tratan sobre Java aportan también información sobre aspectos generales de la programación orientada a objetos.

La primera referencia es la que cubre en su mayor parte el temario de la asignatura.

BUDD, Timothy, **An introduction to object-oriented programming**, 3rd ed., Pearson Education, 2002.

CACHERO CASTRO, Cristina; PONCE DE LEÓN AMADOR, Pedro J.; SAQUETE BORÓ, Estela, **Introducción a la programación orientada a objetos**, Publicaciones de la Universidad de Alicante, 2006. (ejemplos de código en C++)

LETHBRIDGE, Timothy Christian; LAGANIERE, R., **Object-oriented software engineering : practical software development using UML and Java**, McGraw-Hill, 2005

ECKEL, Bruce, Piensa en Java, Pearson/Prentice Hall, 2007 (3ª edición en inglés online)
BLOCH, Joshua, **Effective Java**, Addison-Wesley,

McLAUGHLIN, Brett; POLLICE, Gary; WEST, David, **Head first object-oriented analysis and design**. O`Reilly, 2007.

FORMAN, Ira R. ; FORMAN, Nate, **Java reflection in action**, Manning Publication Co., 2005.

FOWLER, Martin, **Refactoring: improving the design of existing code**, Addison-Wesley, 2009.