Tema 1

Introducción al paradigma de programación orientado a objetos

Programación Orientada a Objetos

Curso 2014/2015

Contenido

- Paradigmas de programación vs. Lenguajes de programación.
- Características del paradigma de programación orientado a objetos
- Calidad del Software.

Evolución de los lenguajes de programación

- La evolución de los lenguajes de programación está ligada al crecimiento de la **complejidad de las aplicaciones**.
- Adaptación a las nuevas aplicaciones y a la mejora de la **capacidad de procesamiento** del hardware.
- La forma de programar se acerca más a los conceptos del dominio de la aplicación (**abstracción**).

Paradigmas de programación

- **Paradigma de programación**:
  - Colección de conceptos que guían el proceso de construcción de un programa, determinando su estructura.
  - Estos conceptos controlan la forma en que pensamos y formulamos los programas.
- Un **lenguaje de programación** refleja un paradigma.

Paradigma de programación	**Lenguaje de programación**
Imperativo	Pascal, C, …
Orientado a Objetos	**Java**, C++, C#, …

Orientación a Objetos

- Técnica de programación que organiza el software como una colección de **objetos que colaboran para realizar la funcionalidad de un sistema**.

Orientación a Objetos

- Nuevo enfoque de programación **centrado en los conceptos** (abstracciones) del dominio de la aplicación.

- **Estrategia de desarrollo**: el software se organiza en torno a los módulos que son deducidos de los tipos de objetos del dominio de la aplicación.

- Diferencia con el **paradigma imperativo**:
  - Centrado en las funciones, qué hace el sistema, en lugar de quién lo hace.
  - Estrategia de desarrollo: refinamiento por pasos sucesivos.

Desarrollo Orientado a Objetos

- Identificar los **objetos** relevantes al problema.
- Describir los **tipos de objetos** y su **propiedades**.
- Encontrar las **operaciones** para los tipos de objetos.
- Identificar **relaciones** entre objetos.
- Utilizar los tipos de objetos y relaciones para estructurar el software.

Ejemplo: Objeto Coche

Tiene las **propiedades**:

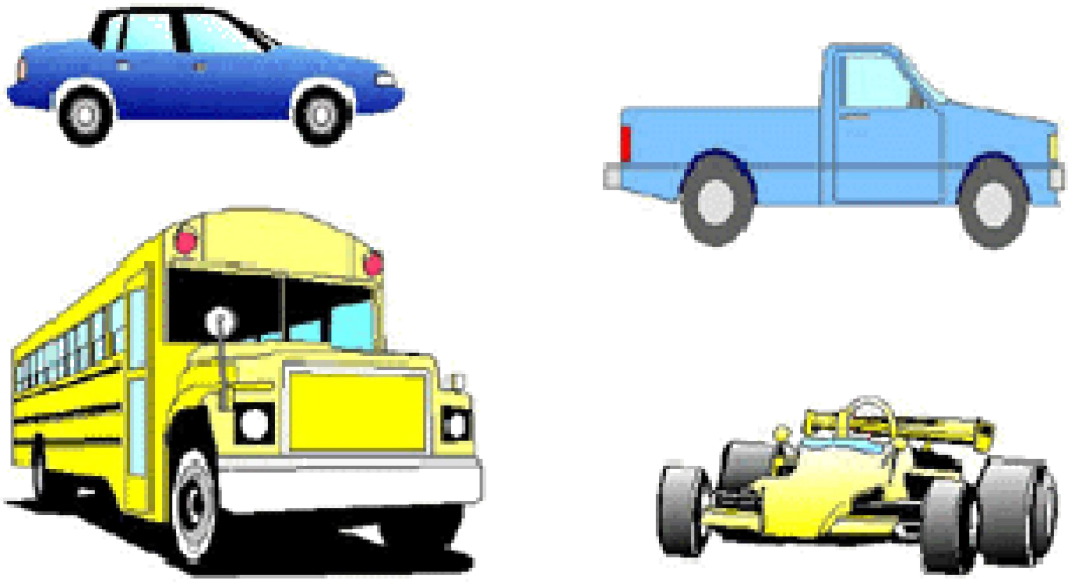
- Color
- Velocidad
- Carburante

Operaciones que puede realizar:

- Mover
- Parar
- Girar a la derecha
- Girar a la izquierda
- Arrancar

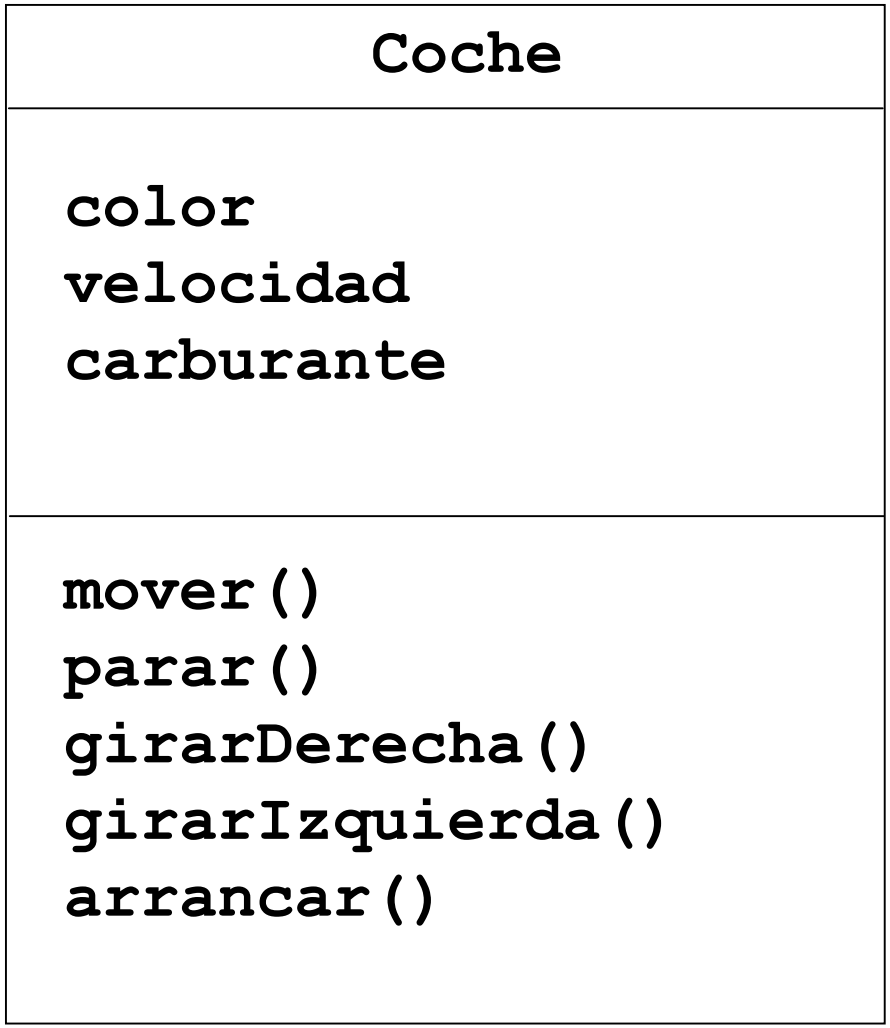
Clases de objetos

- Los objetos con propiedades similares y el mismo comportamiento se agrupan en **clases**.



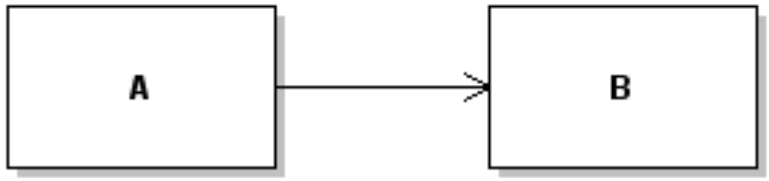
Objetos de la clase **Coche**

Clase Coche


Coche
color
velocidad
carburante
mover()
parar()
girarDerecha()
girarIzquierda()
arrancar()

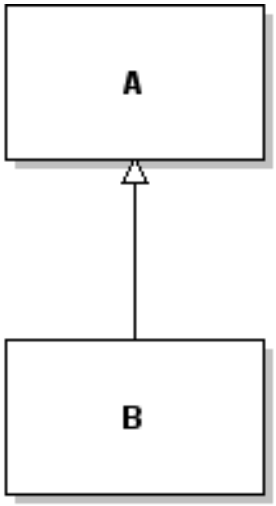
Relaciones entre objetos

- A es un **cliente** de B si todo objeto de A puede contener información sobre uno o más objetos de B.

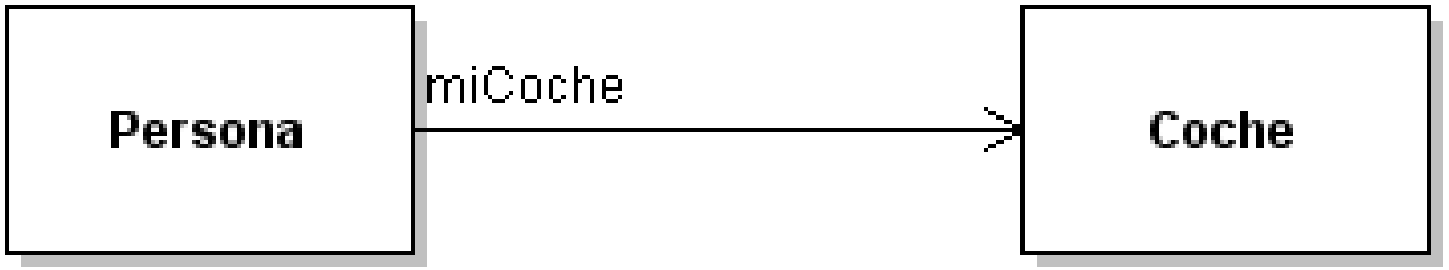


- B **hereda** de A si

  B representa una versión especializada de A

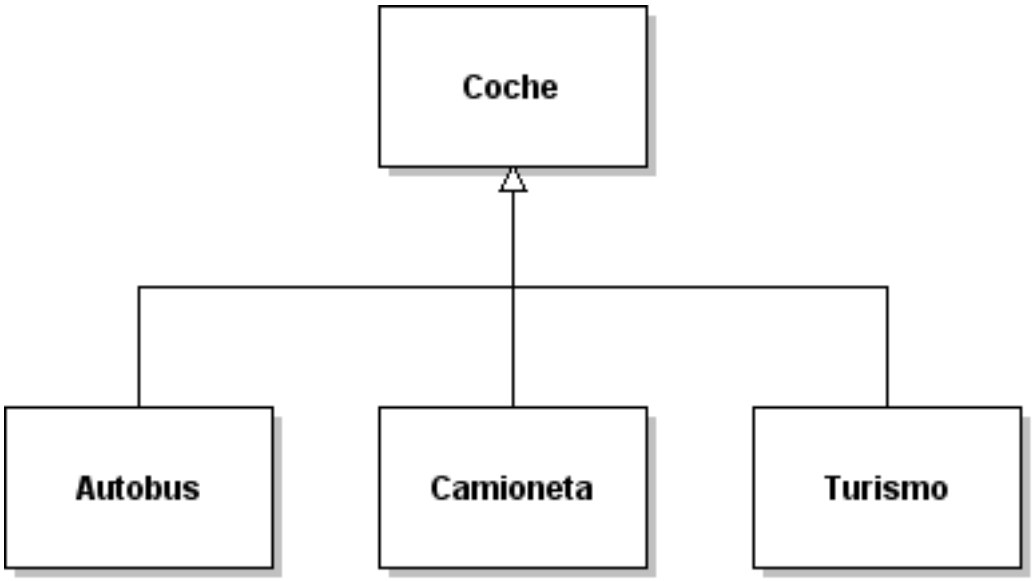


Relación de clientela



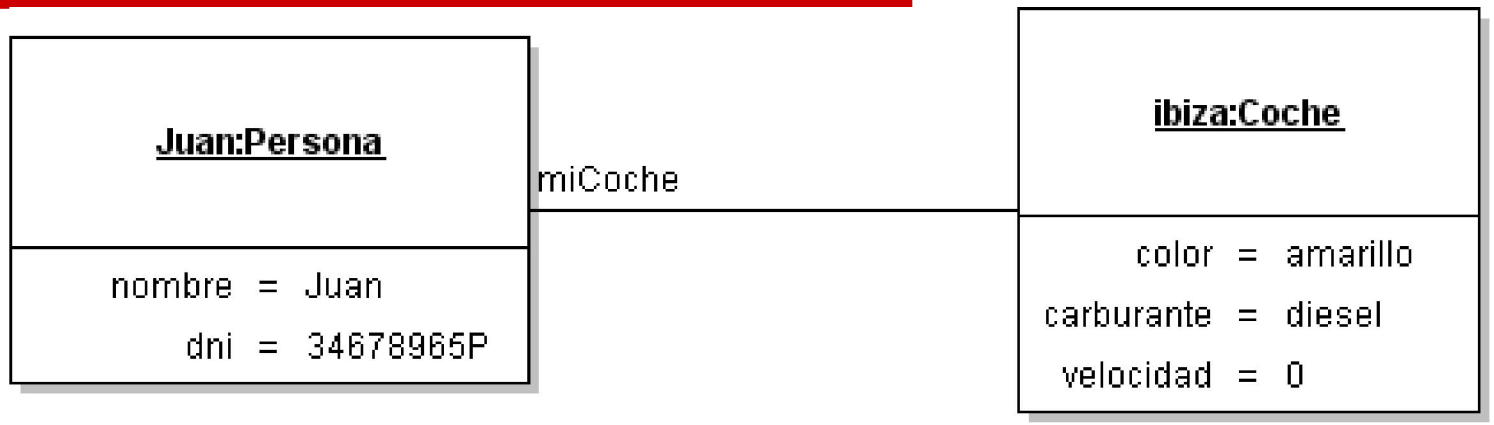
- Una persona tiene información sobre el coche que posee.

Relación de herencia

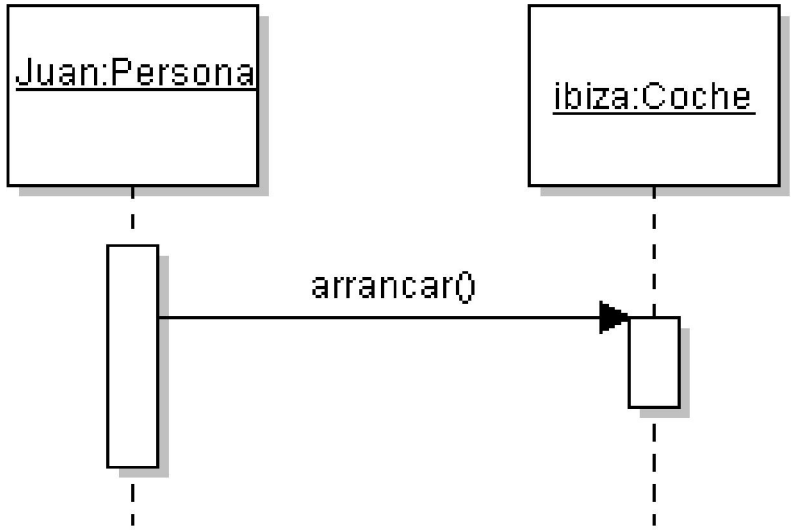


- Turismo ES UN Coche.
- Turismo es una especialización de Coche que tiene nuevas propiedades y funcionalidad.

Comunicación entre objetos



- Objetos se comunican mediante paso de **mensajes**



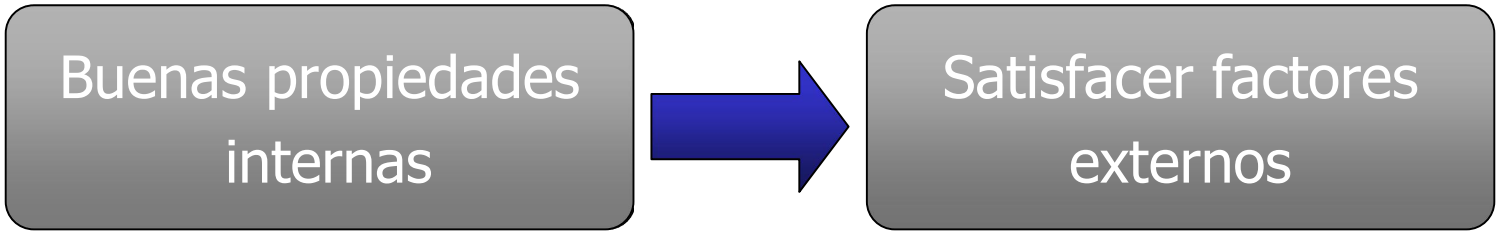
Calidad del software

- La ingeniería del software tiene como objetivo la producción de software de calidad.
- La OO se propone como una técnica para mejorar la calidad del software.
  - Los datos son más estables que las funciones.
- La calidad del software se describe como la combinación de varios factores internos y externos.

Calidad del software

- **Factores Externos**
  - Pueden ser detectados por los usuarios
  - Calidad externa es la que realmente preocupa
- **Factores Internos**
  - Sólo los perciben los diseñadores y programadores
  - **Medio para conseguir la calidad externa**

- **Objetivo**:



Factores Externos

- **Corrección**:
  - Es la capacidad de los productos software de realizar con exactitud su tarea (**cumplir su especificación**).
- **Robustez**:
  - Es la capacidad de los productos software de reaccionar adecuadamente ante **situaciones excepcionales**.
- **Extensibilidad**:
  - Es la facilidad de **adaptación** de los productos software a los **cambios en la especificación**.
  - La dificultad de adaptación es proporcional al tamaño de la aplicación.

Factores Externos

- **Reutilización**:
  - Es la capacidad de un producto software de ser utilizado en la construcción de diferentes aplicaciones.
  - Mejora la productividad.

- **Otros factores**:
  - **Eficiencia**.
  - **Portabilidad**.
  - **Facilidad de uso**.
  - **Facilidad reparación de errores**.

Modularidad

- **Factor interno** que contribuye a la calidad del software.
- **Definición**: Propiedad que tiene un sistema que ha sido descompuesto en un conjunto de **módulos cohesivos y débilmente acoplados**.
- **Alta cohesión**:
  - Módulo con responsabilidades altamente relacionadas y que no hace gran cantidad de trabajo (módulo especializado).
- **Bajo acoplamiento**:
  - Un módulo que depende de pocos módulos.
  - Facilita la comprensión del código.
  - Limita el impacto de los errores.

Principios de Diseño Modular

- **Ocultación de la información**:
  - Consiste en **ocultar los detalles de la implementación** al código cliente, reduciendo así el acoplamiento.
  - Se seleccionan un conjunto de propiedades que se hacen públicas a los clientes (**interfaz**) y se oculta el resto (**implementación**).
- **Principio de Elección Única**:
  - **Evitar manejar listas de variantes**.
  - A veces no es evitable. Minimizar los módulos que conocen la lista de variantes.

Principios de Diseño Modular

- **Principio Abierto-Cerrado**:
  - Un **módulo** está **abierto** si está disponible para ser adaptado: añadir o modificar funcionalidad.
  - Un **módulo** está **cerrado** si está disponible para ser utilizado.
  - La **extensibilidad** del código se ve favorecida por módulos que estén abiertos y cerrados al mismo tiempo.
  - **Objetivo**: extender un módulo sin afectar al código que ya hacía uso de ese módulo.

Mantenimiento de software

- Fase del ciclo de vida del software que sucede después de que haya sido entregado.
- Corresponde al mayor periodo en la vida de una aplicación y se estima un coste del 70%.

- **Tipos de mantenimiento**:
  - **Adaptación** a cambios en la especificación.
  - **Reparación de errores**.

- El mantenimiento se ve favorecido por:
  - **Extensibilidad**.
  - **Facilidad de reparación de errores**.
  - **Reutilización**.

OO y calidad del software

- **Mantenimiento:**
  - Los objetos (conceptos) de un problema son entidades más estables que las funciones.

- **Extensibilidad**:
  - La estructura del software “mimetiza” los conceptos el dominio de la aplicación.

- **Reutilización**:
  - La reutilización de código forma parte de los conceptos básicos de la OO (herencia).