Introducción a la Programación Orientada a Objetos

Tema 1

TACC II
Curso 2008/09

Indice

- **Ciclo de Vida y Calidad del Software.**
- Paradigmas de Programación.
- Programación Orientada a Objetos.
- Conceptos de Programación Orientada a Objetos.
  - Clases y Objetos
  - Encapsulamiento.
  - Herencia.
  - Polimorfismo.
  - Frameworks y Notaciones.

Ciclo de Vida del Software

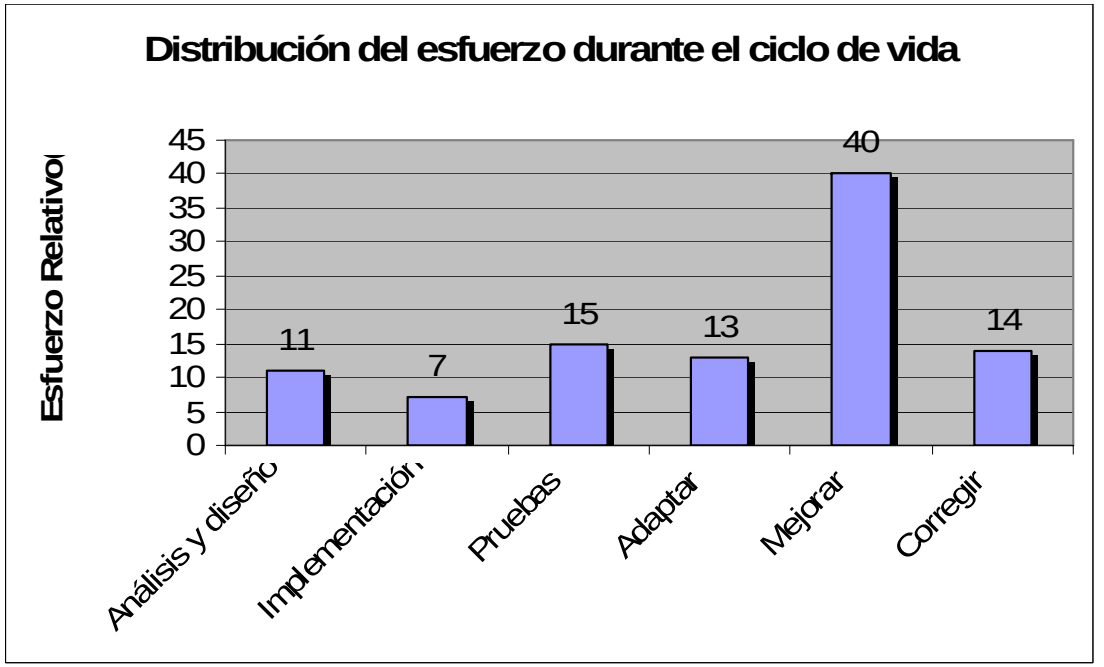
- Conjunto de fases por las que pasa el sistema que se está desarrollando desde que nace la idea inicial hasta que el software es retirado o reemplazado:
  - Análisis.
  - Diseño.
  - Codificación.
  - Pruebas.
  - Mantenimiento.

- Construir Software no es sólo programar.

Ciclo de Vida del Software

Distribución del Esfuerzo

- Análisis ≈ 6%.
- Diseño ≈ 5%.
- Codificación ≈ 7%.
- Pruebas ≈ 15%.
- Mantenimiento ≈ 67%.



Calidad del Software

Factores de Calidad

- Eficiencia
- Portabilidad
- Facilidad de prueba
- Integridad (protección contra procesos sin derecho de acceso)
- Facilidad de uso
- Corrección
- Fiabilidad (situaciones anómalas)
- Extensibilidad
- Reutilización
- Compatibilidad
- ...

Indice

- Ciclo de Vida y Calidad del Software.
- **Paradigmas de Programación.**
- Programación Orientada a Objetos.
- Conceptos de Programación Orientada a Objetos.
  - Clases y Objetos
  - Encapsulamiento.
  - Herencia.
  - Polimorfismo.
  - Frameworks y Notaciones.

Paradigmas de Programación

Abstracción

- Capacidad para encapsular y aislar la información del diseño y ejecución.
- Mecanismos en programación:
  - Procedimientos y funciones
  - Tipos abstractos de datos (TAD)
  - Clases: son TAD a los que se añaden mecanismos como herencia, métodos, etc.

Paradigmas de Programación

- Estructurada.
  - Fortran.
  - Basic.
  - Pascal.
  - C.
  - …
- Funcional.
  - Lisp.
  - …
- Lógica.
  - Prolog.
  - …
- POO.
  - Smalltalk.
  - C++.
  - Java.
  - …

- Desde principios de los 70.
- Dificultad para el trabajo en grupo.
- No hay correspondencia estrecha entre datos reales y programas.

Indice

- Ciclo de Vida y Calidad del Software.
- Paradigmas de Programación.
- **Programación Orientada a Objetos.**
- Conceptos de Programación Orientada a Objetos.
  - Clases y Objetos
  - Encapsulamiento.
  - Herencia.
  - Polimorfismo.
  - Frameworks y Notaciones.

Programación Orientada a Objetos

- Ventajas de uso:
  - Reusabilidad (mecanismos de abstracción y herencia)
    - En programación convencional: uso de funciones y procedimientos
  - Adecuación a entornos de bases de datos.
  - Idónea para tratamiento de Interfaces de Usuario.
  - Adecuada en prototipos y simulación.

Programación Orientada a Objetos

Características Generales

- Construcción de sistemas complejos a partir de componentes.
- Modelado más fiel del mundo real.
- Estimación de reducción de 40% con respecto a la programación convencional.

Programación Orientada a Objetos

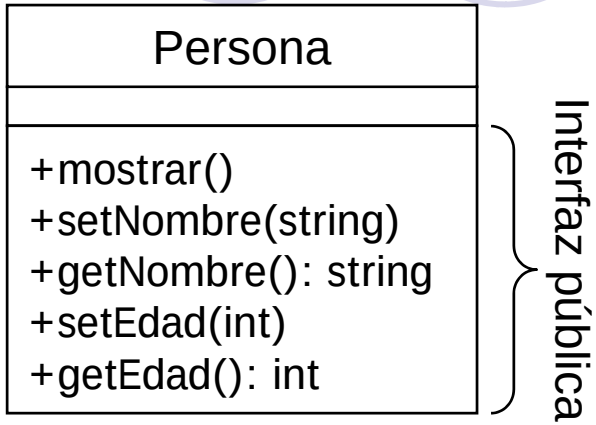
Características

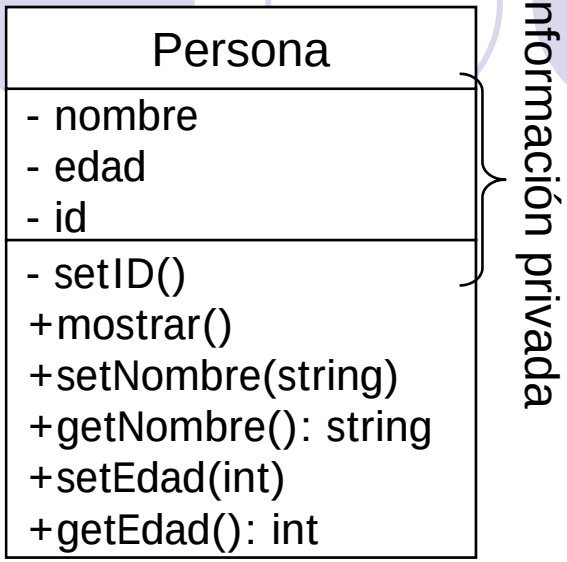
- El modelo objeto (Booch, 1994):
  - Abstracción
    - Las características esenciales del objeto:
      - Documento: insertar, borrar, ...
      - Una grapadora: rellenar, grapar, ...
  - Encapsulamiento (ocultación de información)
    - Una clase contiene:

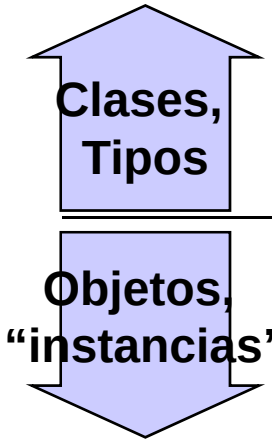
      una interfaz pública.

      una implementación.

Ejemplo

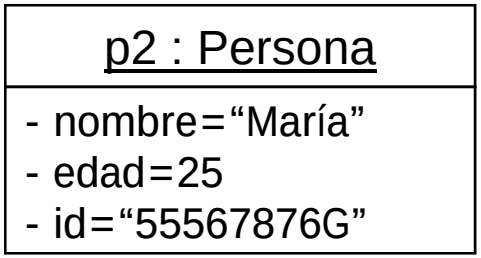


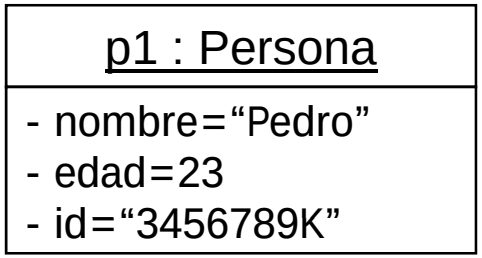




especificación

ejecución



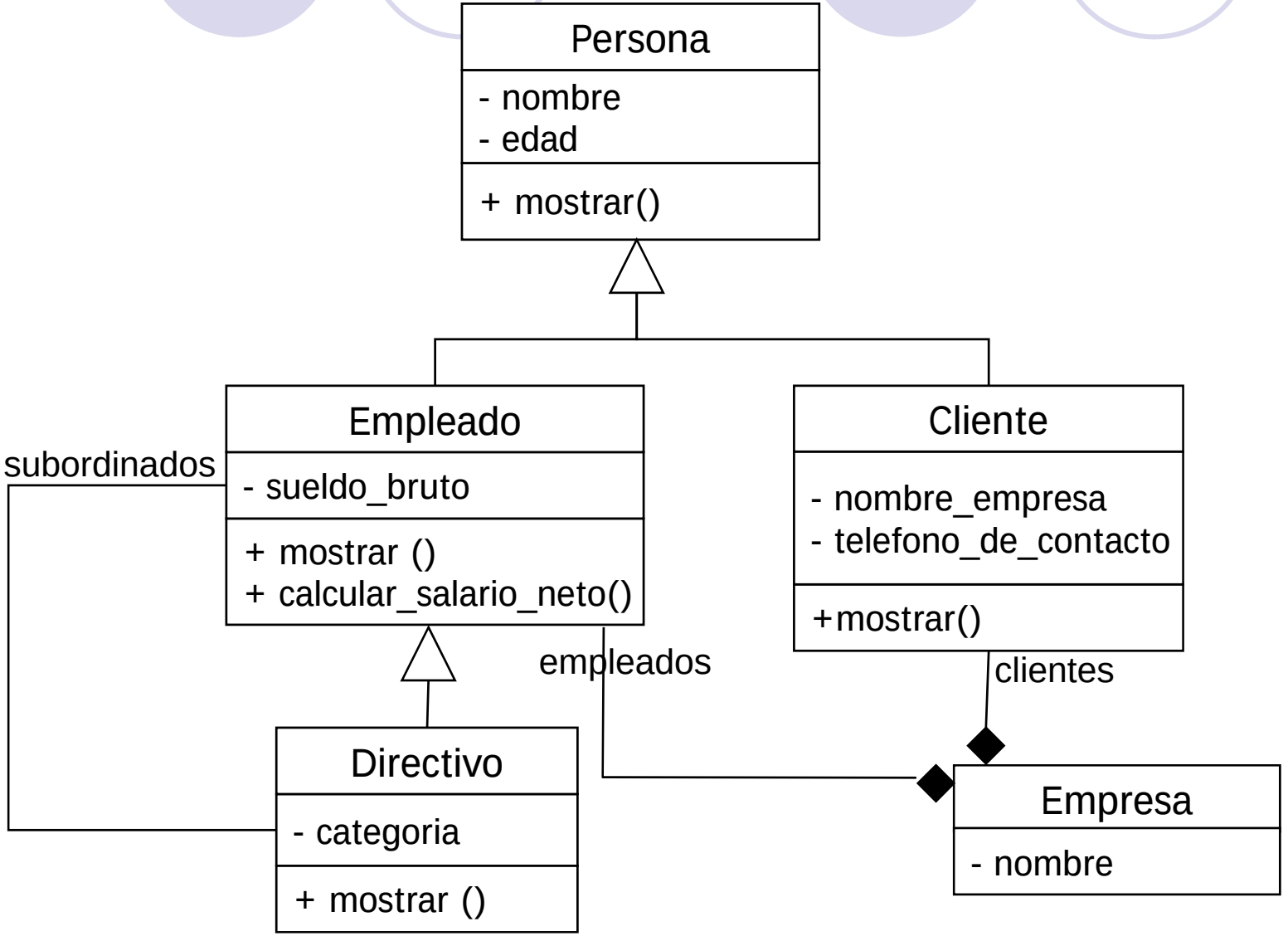


Programación Orientada a Objetos

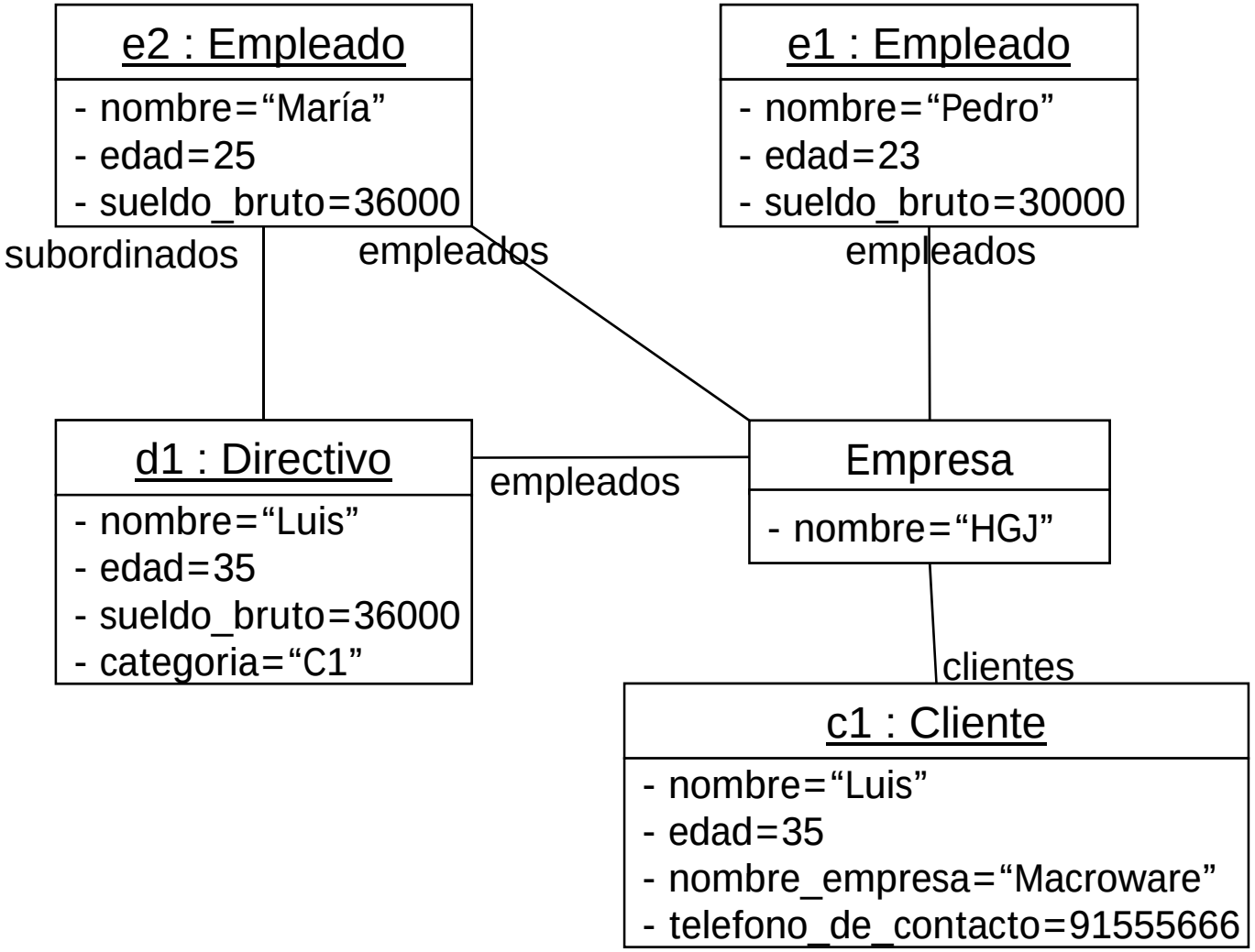
Características

- Modularidad:
  - Subdivisión de una aplicación en otras más pequeñas (módulos).
  - Un módulo es un conjunto de clases.
- Jerarquía
  - Ordenación de las abstracciones
  - Tipos:
    - Herencia (“es-un”); generalización/especialización
      - Herencia simple o múltiple
    - Agregación (“parte-de”)
- Polimorfismo
  - Una misma operación (método) realizada de diferente modo
    - comer (vaca, persona, león); clase mamífero
    - dibujar (triángulo, cuadrado); clase figura
- Otras propiedades
  - concurrencia (multitarea), persistencia, uso de excepciones

Ejemplo


Persona
- nombre
- edad
+ mostrar()
Empleado
- sueldo_bruto
+ mostrar ()
+ calcular_salario_neto()
Cliente
- nombre_empresa
- telefono_de_contacto
+mostrar()
subordinados
empleados
clientes
Directivo
- categoria
+ mostrar ()
Empresa
- nombre

Ejemplo


e2 : Empleado
- nombre=“María”
- edad=25
- sueldo_bruto=36000
e1 : Empleado
- nombre=“Pedro”
- edad=23
- sueldo_bruto=30000
subordinados
empleados
empleados
d1 : Directivo
- nombre=“Luis”
- edad=35
- sueldo_bruto=36000
- categoria=“C1”
empleados
Empresa
- nombre=“HGJ”
clientes
c1 : Cliente
- nombre=“Luis”
- edad=35
- nombre_empresa=“Macroware”
- telefono_de_contacto=91555666

Programación Orientada a Objetos

Modelado del mundo real

- Nombres:
  - Objetos
  - Propiedades de objetos
- Adjetivos:
  - Valores de las propiedades
- Verbos:
  - Comportamiento de los objetos

"El coche tiene color rojo y se mueve"

"El documento tiene letra grande y se muestra"

Lenguajes de POO

- Cronología:
  - Fortran (1958), LISP (1959), BASIC (1964), Pascal (1969), Prolog (1971), C (1973), Smalltalk-80 (1980), C++ (1986), Object Pascal (1988), CLOS (1989), Java (1995).
  - *The jounal of object-oriented programming* (1988). *Journal of Object Technology (2002-), etc.*
  - Muchas conferencias científicas sobre el tema: OOPSLA, ECOOP, etc.
- Primeros lenguajes POO:
  - Simula-67
    - Objeto (datos+métodos). Clase. Herencia.
  - Smalltalk-80
    - Verdadero primer lenguaje de POO
    - Concepto de paso de mensajes (activación de métodos)

Lenguajes de POO

Clasificaciones

- Orientación:
  - puros (Smalltalk)
  - híbridos (C++)
- Tipificación:
  - estática (en tiempo de compilación), Object Pascal
  - dinámica (en tiempo de ejecución), Python
- Ligadura:
  - estática (C++)
  - dinámica (Java, C++)

Lenguajes de POO

Paradigmas

- Clase-Elemento
  - Ninguna clase es objeto
    - C++
  - Toda clase es un objeto
    - Smalltalk, Java
- Prototipo-Elemento
  - Todo objeto puede ser prototipo de otros
    - Amulet

Otras Ventajas de POO

- Mejor mantenimiento.
- Estructuras más reales de la información.
- Escalabilidad.
- Adaptabilidad.
- Más apropriada para aplicaciones dirigidas por eventos.

Inconvenientes de POO

- Necesidades de estandarización:
  - Notación de Modelado (OMG, Object Management Group).
  - Lenguajes de Programación.

- Coste de conversión de software legado

Indice

- Ciclo de Vida y Calidad del Software.
- Paradigmas de Programación.
- Programación Orientada a Objetos.
- **Conceptos de Programación Orientada a Objetos.**
  - **Clases y Objetos**
  - **Encapsulamiento.**
  - **Herencia.**
  - **Polimorfismo.**
  - **Frameworks y Notaciones.**

Conceptos de POO

- Conceptos principales:
  - Clase.
    - Similar al concepto de estructura en Pascal.
    - Además de datos (atributos), se añaden funciones (métodos) que operan sobre esos datos.
    - Estructuración (herencia).
  - Objeto (una instancia de una clase)
  - Jerarquía de herencia entre clases.
    - Herencia de atributos y métodos.

Conceptos de POO

- Objetos:
  - objeto=datos+métodos
  - miembros de un objeto (o clase):
    - datos (atributos)
    - métodos
  - identificador del objeto
    - nombre de variable

Conceptos de POO

Ejemplo

- Clase: Robot
  - Datos: x (entero), y (entero)
  - Métodos:
    - void avanzar (entero, entero)
    - entero posicionX ()
    - entero posicionY ()
    - void avanzar (entero)
    - Constructores:
      Robot (entero, entero)
      Robot (entero)
  - Instanciación:
    - objeto “robot1” (Robot)
    - robot1 = Robot(1,2)
  - Datos de clase:
    - numeroRobots (entero)

Creación y Destrucción de Objetos

- Equivalencia de conceptos con programación clásica:
  - Tipo ←→ Clase
  - Dato ←→ Objeto
  - Variable: existe en ambos tipos de programación
- Creación (uso de constructor):
  - Ejemplo:

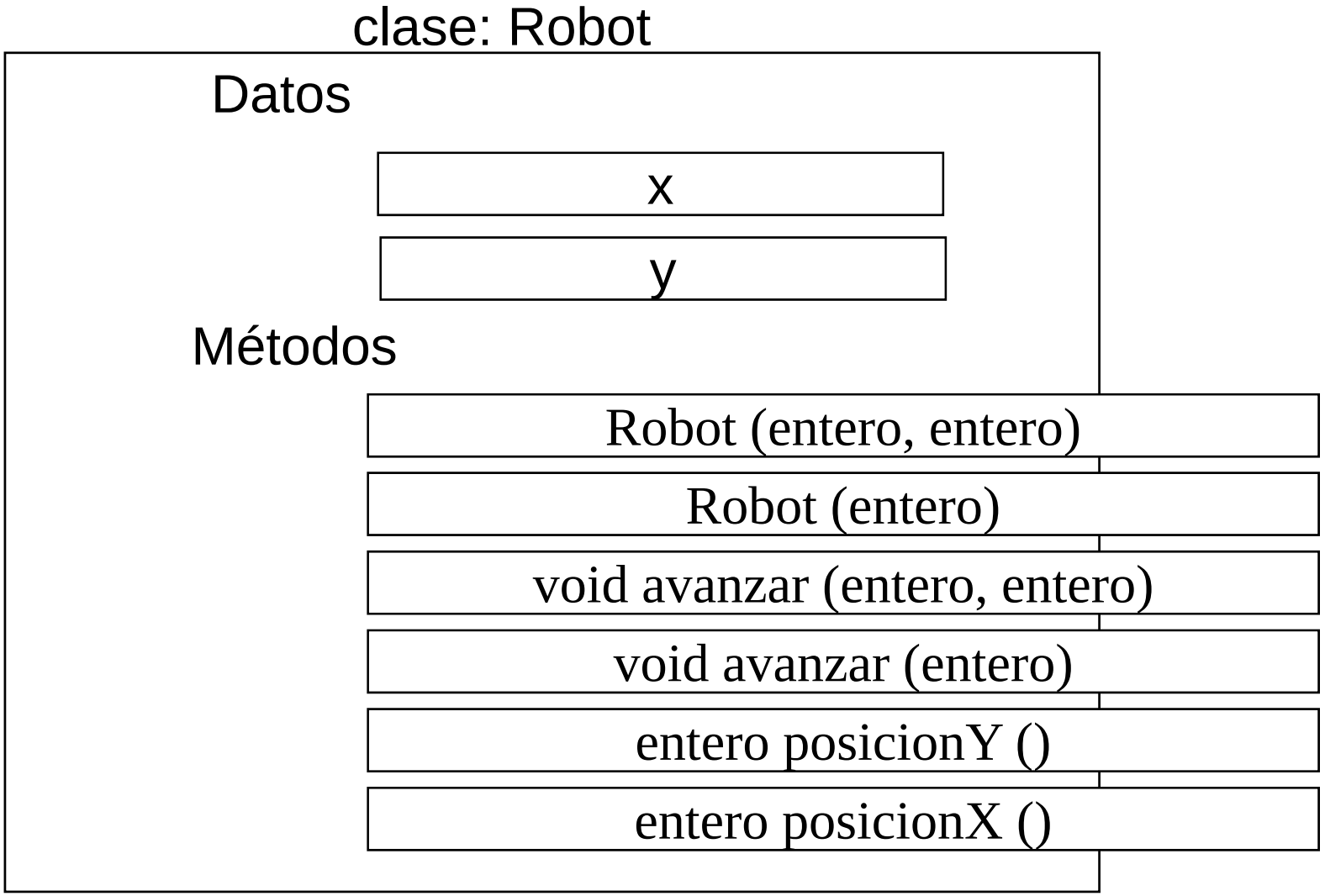
    robot1 = Robot(1,2) robot2 = Robot(3)
- Destrucción
  - Automática (Java, *Garbage collection*)
  - Explícita (montón) o automática (pila) (C++). Destructores.

Encapsulamiento

- Miembros privados y públicos
- Interfaz pública de una clase (miembros públicos, datos y métodos)
    - Se pueden invocar desde fuera de la clase
- Ejemplo (clase Robot)
    - Datos:
        - privado x (entero)
        - privado y (entero)
    - Métodos:
        - público void avanzar (entero, entero)
        - público entero posicionX ()
        - público entero posicionY ()
        - público void avanzar (entero)
    - Constructores:
        - público Robot (entero, entero)
        - público Robot (entero)

Encapsulamiento

Representación



Ejecución de Métodos

- Un método es una definición de una función.
  - Se dice que un método se ejecuta cuando el objeto recibe un mensaje de ejecución del método.
  - Puede acceder a otros miembros de la clase.
- Ejemplo:
  - Objeto robot1 (Robot)
  - robot1=Robot(3,2)
  - robot1.avanzar(1,2)

Herencia

- Representa el concepto de “ser un tipo especial de” o “ser un/a”.
- Se establece mediante la definición de subclases, que dan lugar a una jerarquía de clases.
- Las subclases hijas heredan los datos y métodos de las clases padre.

Herencia

Ejemplo

- Clase: RobotConFrontera
  - Clase padre: Robot
  - Datos:
    - privado limX (entero)
    - privado limY (entero)
  - Métodos:
    - void avanzar (entero, entero)
    - void avanzar (entero)
    - Constructores:
      - RobotConFrontera (entero, entero, entero, entero)
      - RobotConFrontera (entero, entero, entero)
  - Instanciación:
    - objeto “robot4” (RobotConFrontera)
    - robot4 = RobotConFrontera(1,2,0,0)

Herencia

- La definición en RobotConFrontera de:
  - void avanzar (entero, entero)
  - void avanzar (entero)
  - es un “*overriding*” (especialización, redefinición, etc) de los métodos ya definidos en la clase Robot

Herencia

Tipos

- Herencia simple:
  - Figura, Círculo, Rectángulo, Cuadrado, Triángulo
- Herencia múltiple:
  - Persona, Profesor, Investigador, ProfesorUniversitario
  - Problemas de ambigüedad

Ejemplo

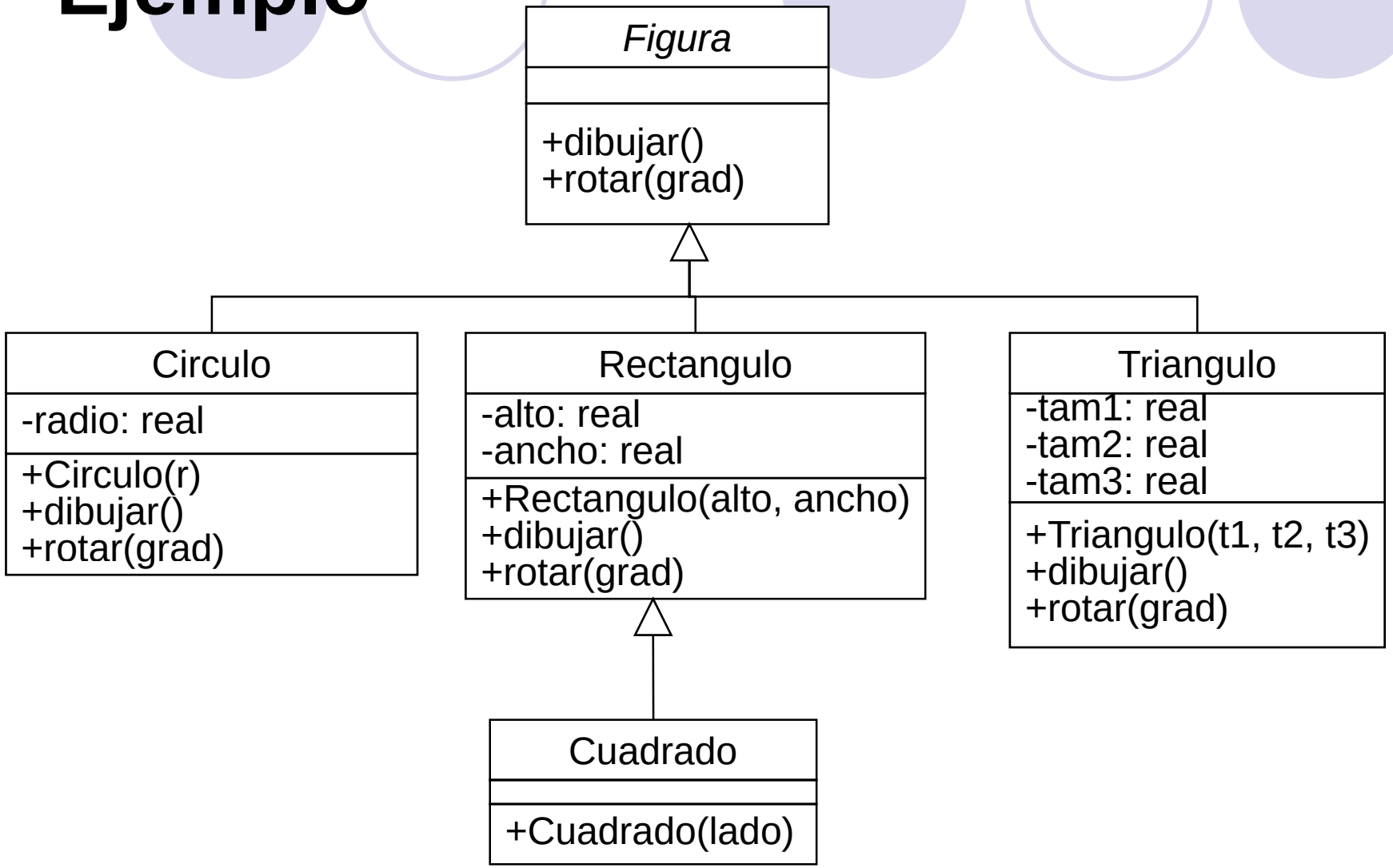
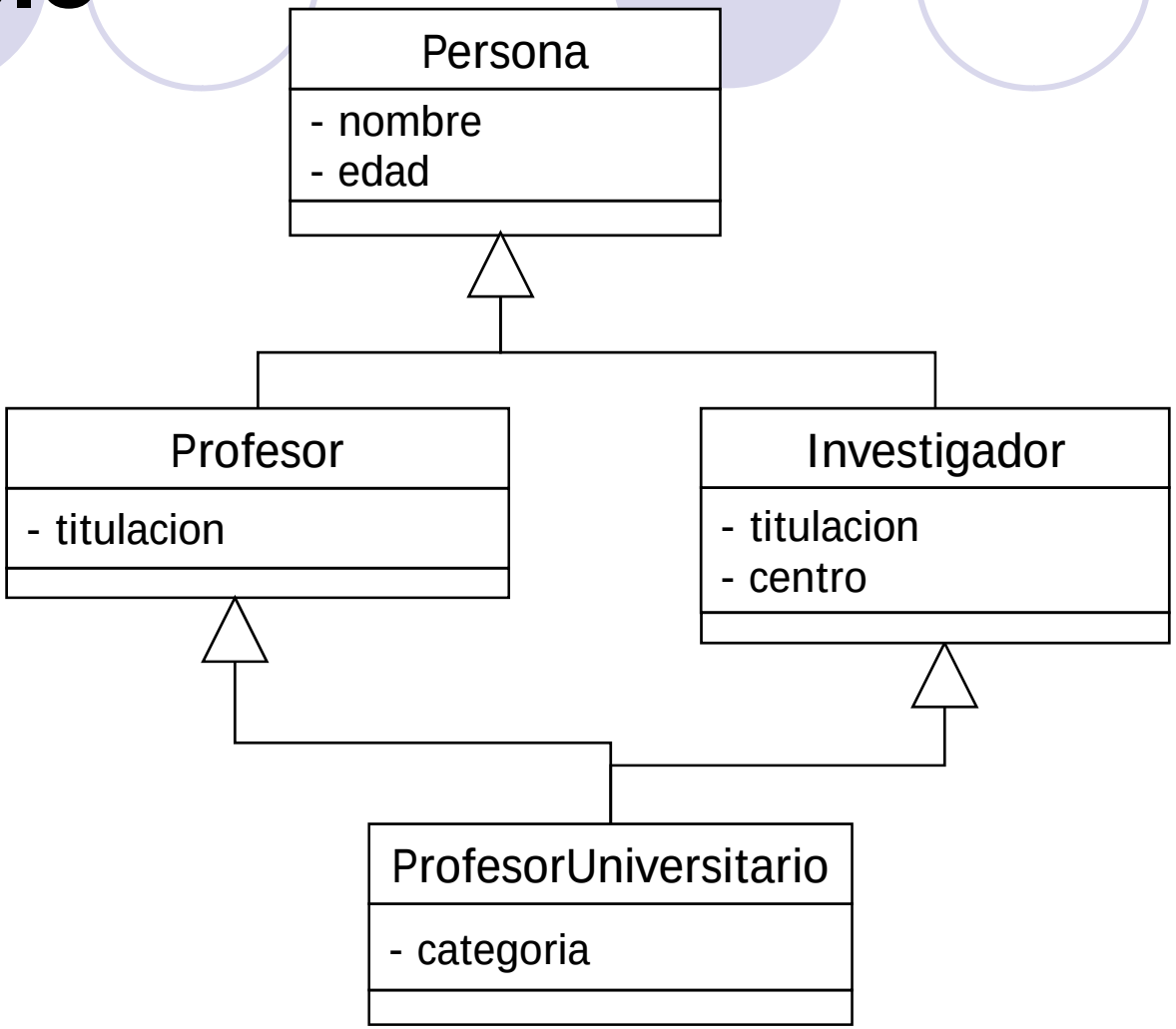


Figura
+dibujar()
+rotar(grad)
Circulo
-radio: real
+Circulo(r)
+dibujar()
+rotar(grad)
Rectangulo
-alto: real
-ancho: real
+Rectangulo(alto, ancho)
+dibujar()
+rotar(grad)
Triangulo
-tam1: real
-tam2: real
-tam3: real
+Triangulo(t1, t2, t3)
+dibujar()
+rotar(grad)
Cuadrado
+Cuadrado(lado)

Ejemplo


Persona
- nombre
- edad
Profesor
- titulacion
Investigador
- titulacion
- centro
ProfesorUniversitario
- categoria

Herencia

Clases Abstractas

- Clase abstracta:
  - No admite una instancia directa. Ejemplo: clase Figura. Sería inválido
    - objeto f (Figura)
    - f=Figura(....)

    aunque sí sería válido
    - objeto f (Figura)
    - f=Cuadrado(3)

Polimorfismo

- Sobrecarga (overloading) de un método:
  - En una clase, el mismo nombre de método definido de modos distintos
    - Ejemplo: el método “avanzar” está sobrecargado en la clase Robot

Polimorfismo

- En la programación convencional, también existe “overloading”:
  - Ejemplo: Pascal
    - println(“aldfkja”)
    - println(34)
- “Overloading” y “Overriding” son casos de polimorfismo:
  - El mismo método definido de modos distintos
- Ligadura dinámica:
  - objeto r (Robot)
  - r=RobotConFrontera(2,3,0,0)
  - Al ejecutar r.avanzar(3), ¿qué definición se aplica?
    - la de Robot (*ligadura estática*)
    - la de RobotConFrontera (*ligad. dinámica*)

Frameworks

- Conjunto de clases que se coordinan para realizar una función
- Para construir una aplicación hay que subclasificarlas.
- Ejemplo:
  - MFC (C++).
  - AWT, Swing (Java).
- El “main” usualmente incluido en el Framework
- Diagramas de clases
  - Necesidad de un estándar de representación gráfica

UML

- Diagramas de Estructura
  - Diagramas de clase (condensación gráfica de estructuras de clases y relaciones entre objetos y clases)
- Diagramas de Comportamiento.