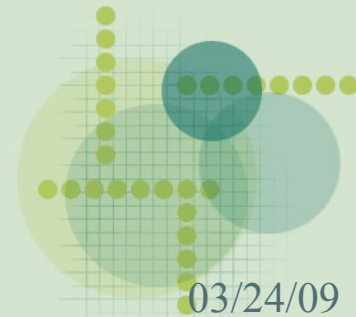


# Programación orientada a objetos

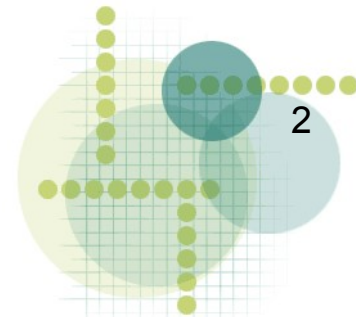
David Pinelo  
Marzo – Abril de 2009



03/24/09

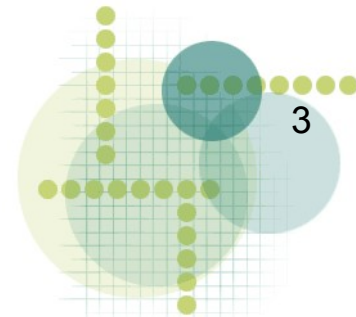
# Índice

- Introducción. Motivación de la POO
- Definiciones. Tipos de datos abstractos
  - Clases y objetos
- Características de la POO
  - Abstracción
  - Encapsulamiento
  - Principio de ocultación
  - Polimorfismo
  - Herencia
  - Recolección de basura
- Lenguajes orientados a objetos
- Fases en el diseño de software de calidad



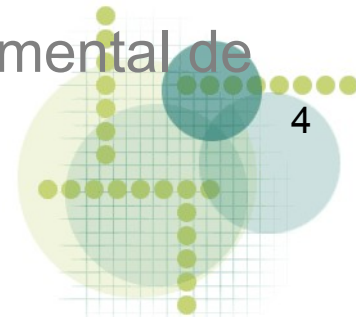
# POO. Calidad del software

- Ingeniería del software: Disciplina o área de la informática que ofrece métodos y técnicas para desarrollar y mantener software de calidad.
- Calidad implica dos tipos de factores
  - Externos (detectados por los usuarios)
    - Velocidad
    - Facilidad de uso
  - Internos (detectables en el código fuente). Destacan
    - Legibilidad
    - Modularidad
    - Reutilización
    - Robustez



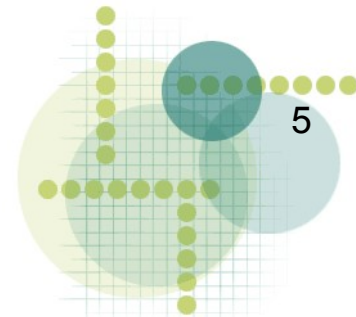
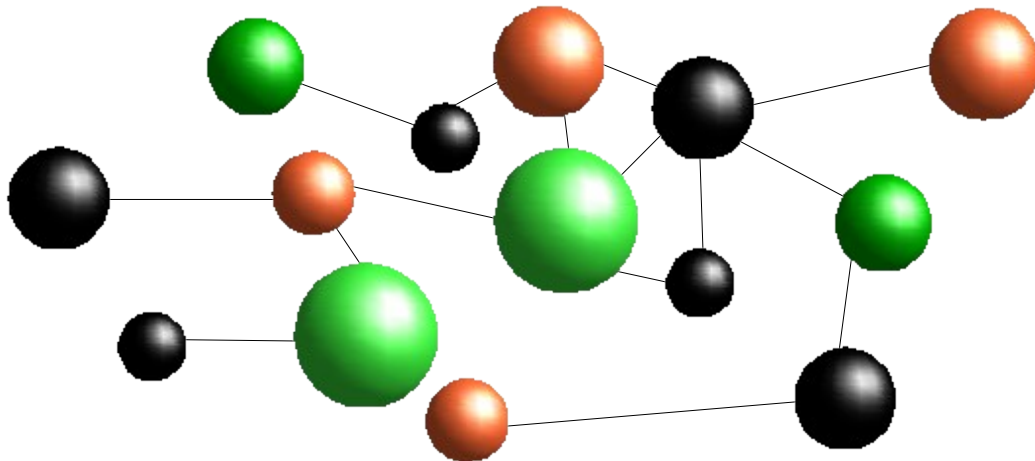
# POO. Definición

- Debería hablarse con propiedad de *Diseño orientado a objetos (DOO)*
- *La construcción de software orientado a objetos es el método de desarrollo de software que basa la arquitectura de cualquier sistema en módulos deducidos a partir de los tipos de objetos que manipula (en lugar de basarse en la función o funciones que el sistema está destinado a asegurar)*
- A diferencia de otros métodos, el DOO da como resultado un diseño que interconexiona los objetos de datos (elementos de datos) y las operaciones de procesamiento, de forma tal que encapsula la información y el procesamiento.
- Este encapsulamiento es el paradigma fundamental de la orientación por objetos.



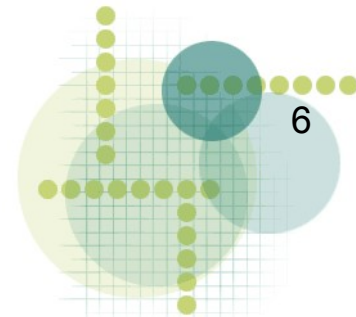
## POO. Definición (II)

- DOO provee objetos como el principal medio para abstraer y estructurar un sistema.
- Un objeto puede modelar entidades del mundo real, puede capturar abstracciones de fenómenos complejos, puede representar artefactos de software (pilas, gráficos, etc).
- El contexto del problema se ve como objetos que interactúan entre ellos



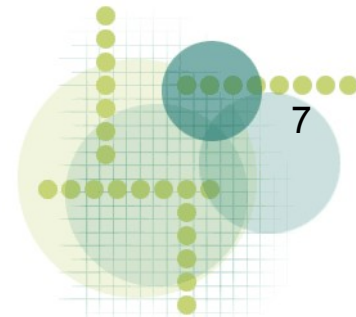
# POO. Tipos de Datos Abstractos (TDA)

- Tipos de Datos Abstractos:
  - Es una colección de *valores + operaciones*
  - Se definen mediante una especificación, que es independiente de cualquier representación (abstracción). Dará lugar a las API de programación.
  - El acceso a los valores del tipo está limitado al uso de las operaciones (interfaz con el usuario limitada)
  - Establecida la interfaz, el programador elige la representación adecuada (implementación)
  - Los usuarios del TDA sólo conocen su nombre y la especificación de las operaciones
  - Cambios en la representación no afectarán al resto de programas



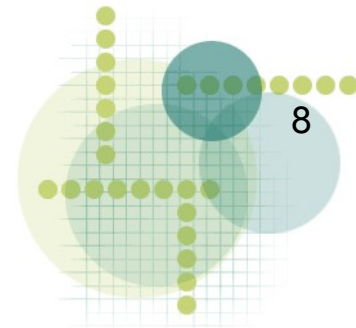
# POO. Diseño orientado a objetos

- En POO un programa es *un grupo de objetos que interactúan entre ellos*
- Cada objeto tiene un rol en la solución del problema
- Cada objeto provee un conjunto de servicios (o métodos)
- Los servicios de un objeto son usados por otros objetos
- Los objetos tienen un estado. Parte de ese estado será conocido por los demás. Otra parte no.
- La naturaleza única del diseño orientado por objetos se debe a su habilidad para construir basándose en tres conceptos importantes del diseño del software:
  - **Abstracción**
  - **Ocultamiento de la información**
  - **Modularidad**



# POO. Objetos

- El término "orientación por objeto" significa que organizaremos el software como una colección de objetos discretos que incorporan tanto estructuras de datos como procedimientos. (Un objeto es TDA)
- Esto contrasta con la programación convencional, en la cual las estructura de datos y el comportamiento están solo aproximadamente conectados
- El desarrollo orientado por objetos es una *nueva forma de pensar acerca del software basado sobre abstracciones que existen en el mundo real*
- En este contexto, el desarrollo es referido a la primera parte del ciclo de vida del software: *análisis, diseño e implantación*



# POO. Objetos (II)

- **¿ Qué es un Objeto ?**

- *Un objeto es una entidad física o abstracta que tiene un comportamiento antes ciertos estímulos, tanto externos como de otros objetos específicos que se encuentran dentro del sistema*

- La forma en que los objetos se comunican entre ellos es a través de mensajes:

- Un mensaje encapsula la petición de un servicio

- El receptor del mensaje presta el servicio y devuelve una respuesta al que solicitó el servicio

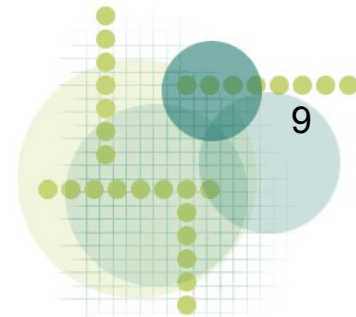
- **¿Cómo encontrar los objetos?**

- Una descripción adecuada de los objetos debe:

- Ser precisa y no ambigua

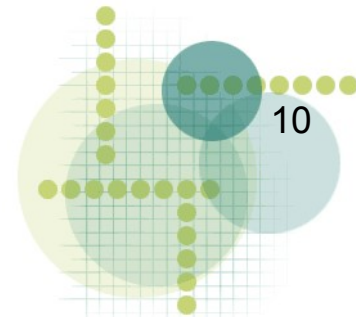
- Ser completa

- No debe especificar más de lo necesario



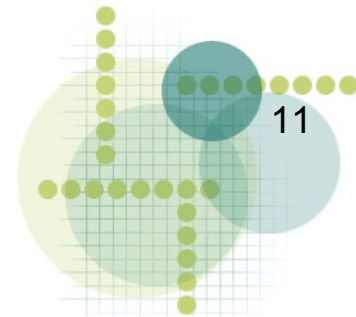
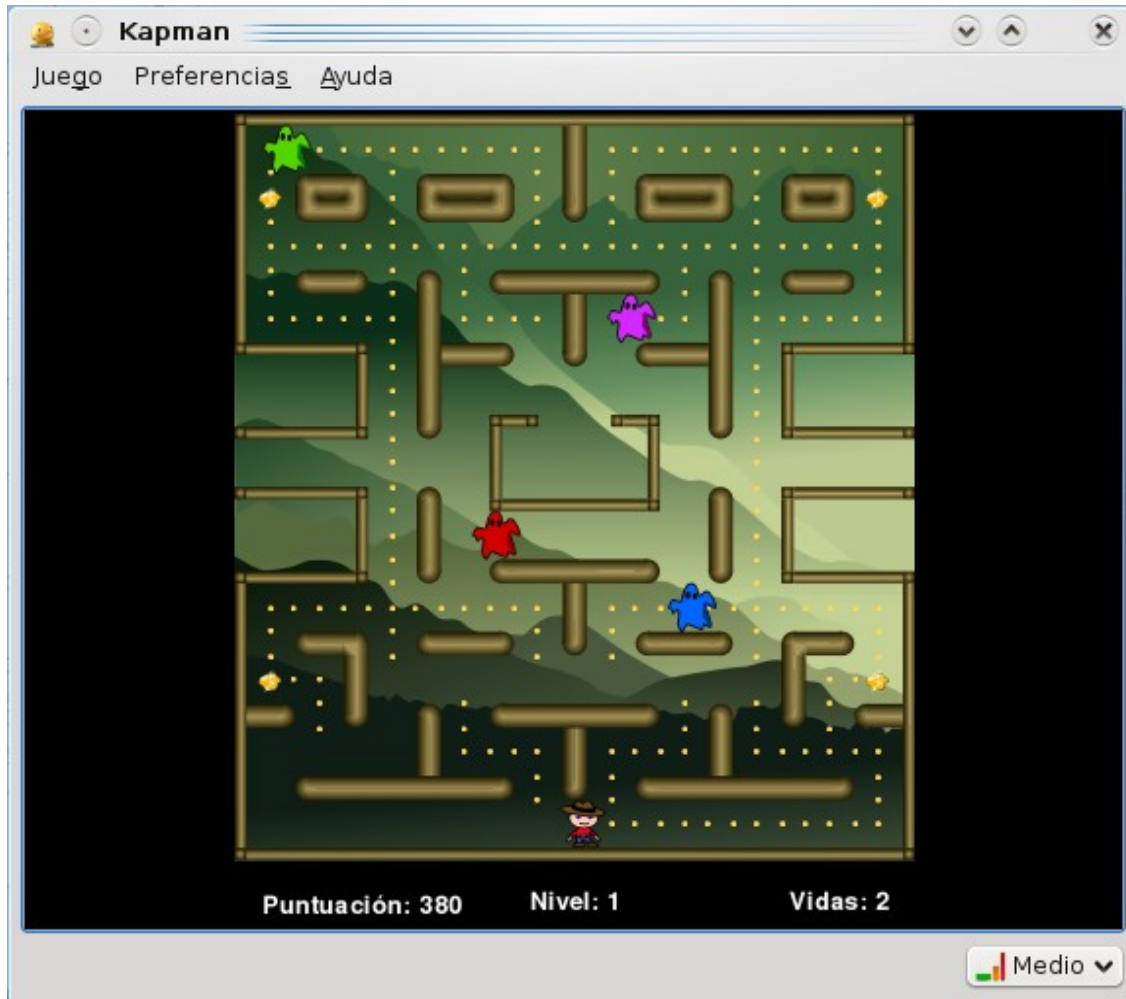
# POO. Objetos (III)

- ¿ Qué se puede considerar como objeto ?
  - Persona
  - Equipo Hardware
  - Materiales
  - Información
  - Software
  - Procesos
  - Procedimientos
  - Relaciones o asociaciones entre objetos
  - INCLUSO Entes matemáticos o lógicos
- En POO un objeto es una abstracción de la realidad para modelar un problema con solución vía software.



# POO. Objetos (IV)

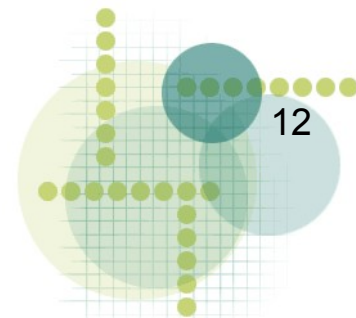
- Discusión. Objetos aquí presentes.



# POO. Objetos (V)

- Un objeto empaqueta datos (una representación concreta) y procedimientos que operan sobre los datos.
- El envío de mensajes es la única forma para que un objeto realice una operación.
- Las operaciones son la única forma para cambiar el estado de los datos.
- Cuando esto se cumple, se dice que el estado interno del objeto (el valor de sus datos) está **encapsulado**; éste no puede ser manipulado directamente desde el exterior y su representación concreta es invisible.

Libro
+titulo +autores +isbn
+prestar( ) +devolver( ) +destruir( ) +reservar( )

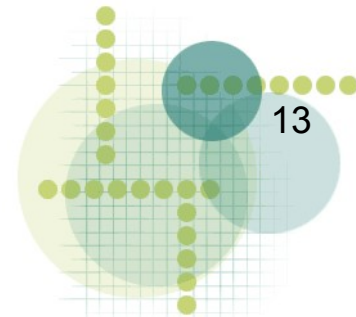


# POO. Clases

- **Clasificación:**

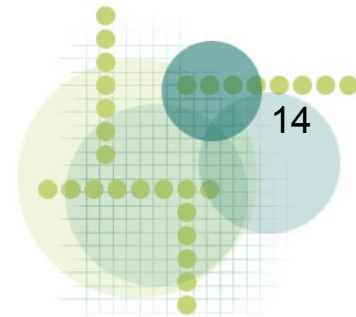
- El proceso de clasificación es el enfoque central de la orientación por objeto y concierne a la agrupación de objetos con propiedades (estructuras de datos o atributos) y comportamiento (operaciones) similares dentro de una *clase*
- *Una clase es la abstracción que describe propiedades importantes para una aplicación*
- Los objetos que pertenecen a una clase se describen colectivamente por la definición de una clase. En lugar de describir los objetos individuales *la orientación por objeto concentra en un patrón tanto el estado como el comportamiento que es común a todos los objetos de la clase*

Esta clase de estructura que abarca tanto propiedades como comportamiento es la unidad natural de la abstracción en los sistemas de orientación por objeto y puede ser utilizada para modelar tanto entidades u objetos como relaciones entre los objetos.



# POO. Objetos y clases

- La distinción entre una clase y sus instancias (objetos) es similar a la distinción entre una definición de tipo y la declaración de una variable en un lenguaje de programación convencional. Sin embargo, la mayoría de los sistemas orientados por objeto crean dinámicamente instancias (objetos) por envío de mensajes "Nuevo" y "Crear" una clase.
- **Clase:** Tipo de datos definido por el usuario
- **Objeto:** Es un dato del tipo definido por su clase, es decir, es una instancia específica de su clase



# POO. Objetos y clases

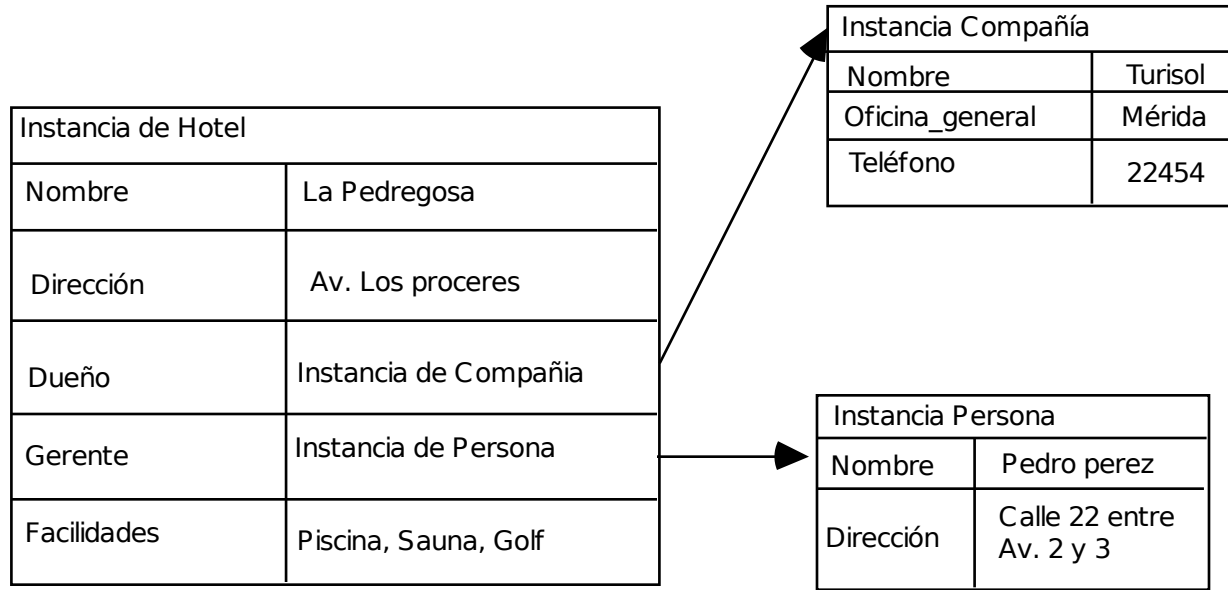
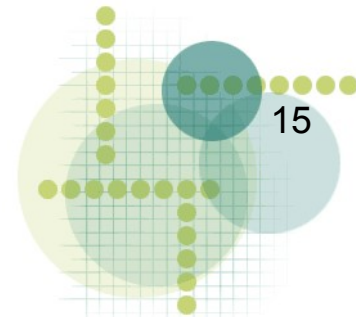


Fig. 5 Ejemplo de la clase Hotel

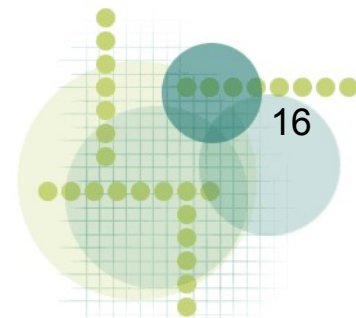


# POO. Notación

- Una clase se representa por una caja la cual puede tener 3 regiones.
  - Primera: nombre de la clase.
  - Segunda: contiene la lista de atributos. cada nombre de atributo puede estar seguido por detalles opcionales tales como tipo de atributo y valores por defectos.
  - Tercera: contiene los nombres de las operaciones. Cada nombre de operación puede ser seguido por detalles opcionales tales como listas de argumentos y tipos de resultados.

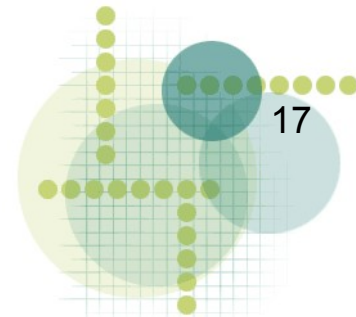
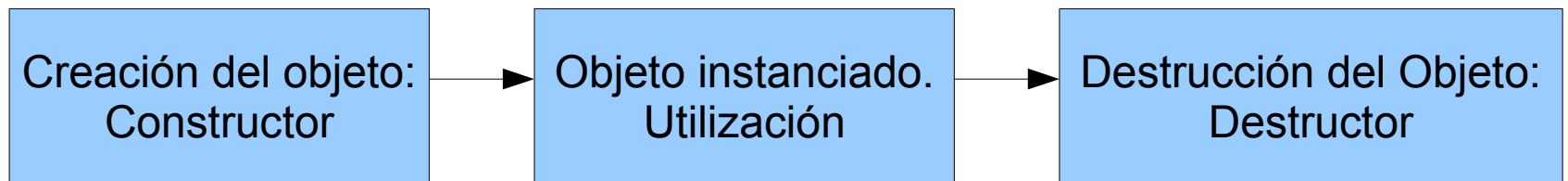
## Nombre de la clase

```
- Atributo 1 : Tipo de dato = Es parte del estado del objeto. Este atributo es privado  
- num_serie : int = 456423  
+ Atributo 2 : Tipo de dato = Es parte del estado del objeto. Este atributo es público  
+ nombre : string = Nombre del objeto  
+ Operacion_1(Parametros de la operación : Tipo de dato) : Tipo de dato  
+ getNumSerie() : int
```



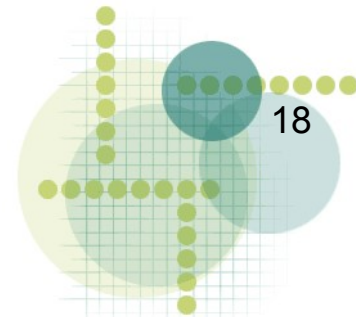
# POO. Trabajando con clases y objetos

- Fases de la vida de un objeto
  - Creación de un objeto
    - Primero debe declararse
    - Luego debe ser instanciado
  - Utilización del objeto instanciado
  - Destrucción del objeto instanciado



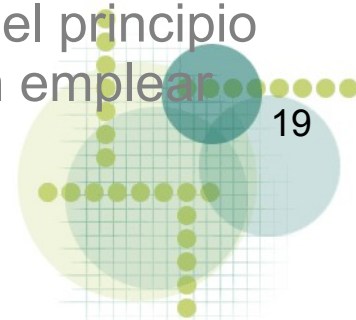
# POO. Dentro de una clase

- Campos de datos dentro de una clase: *variables miembros* o *propiedades de la clase*.
- Operaciones con los datos de una clase: *funciones miembros* o *métodos de la clase*.
- Los miembros de una clase (propiedades y métodos) pueden ser:
  - *Públicos*: se puede acceder a ellos desde fuera de la clase
  - *Privados*: sólo se puede acceder a ellos desde dentro de la clase
- Para acceder a propiedades privadas de la clase se definen métodos de acceso.
- Un miembro puede ser estático, si pertenece a la clase en sí y no a los objetos de la clase.



# POO. Abstracción y encapsulamiento

- Tres características fundamentales de la POO
  - **Abstracción:** Denota las características esenciales de un objeto, donde se capturan sus comportamientos. Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos y cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción.
  - **Encapsulamiento:** Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.



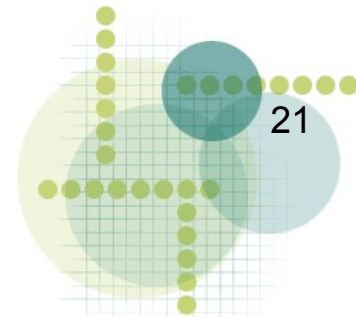
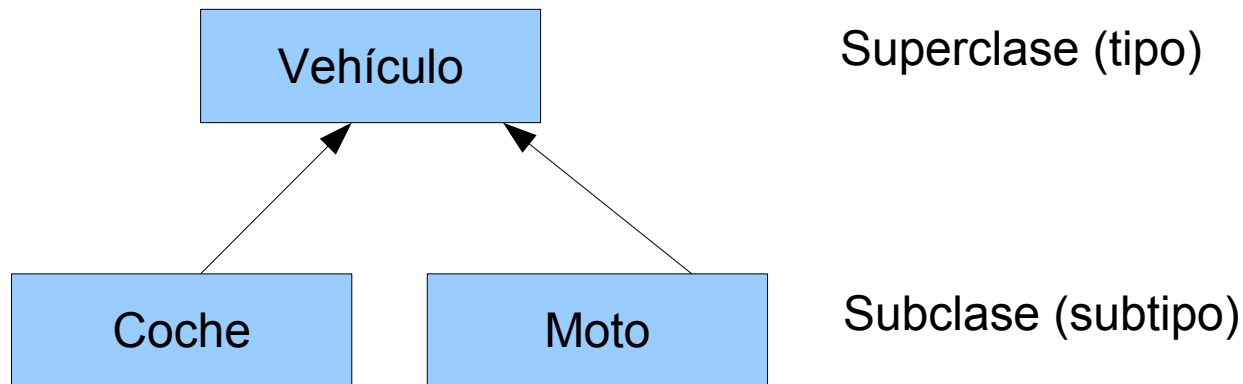
# POO. Principio de ocultación

- **Principio de ocultación:** Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que especifica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas, eliminando efectos secundarios e interacciones inesperadas.
- Algunos lenguajes (C++ entre ellos) relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos.



# POO. Herencia

- La herencia consiste en el compartir atributos y métodos entre clases basándose en una relación jerárquica.
- Una clase puede definirse ampliamente y redefinirse sucesivamente en subclases más refinadas.
- Cada subclase que se incorpora, hereda todas las propiedades de su superclase y adiciona sus propias y únicas propiedades.
- La herencia permite *reutilizar código*



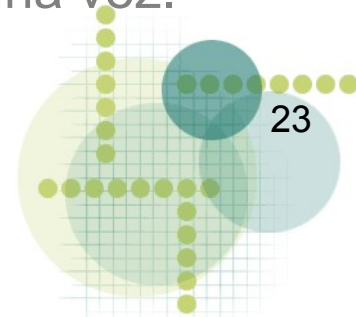
## POO. Herencia (II)

- Las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen.
- La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementarlo. Esto suele hacerse habitualmente agrupando los objetos en clases y estas en árboles o enrejados que reflejan un comportamiento común.
- **Relación de subtipaje:** *Si B es subtipo de A, entonces cualquier objeto de tipo B puede ser usado en cualquier parte donde un objeto tipo A puede ser usado*

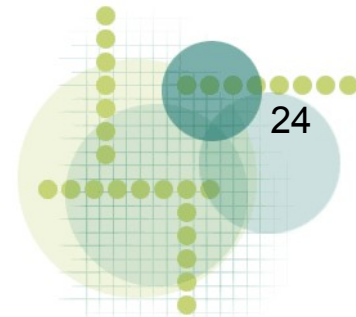
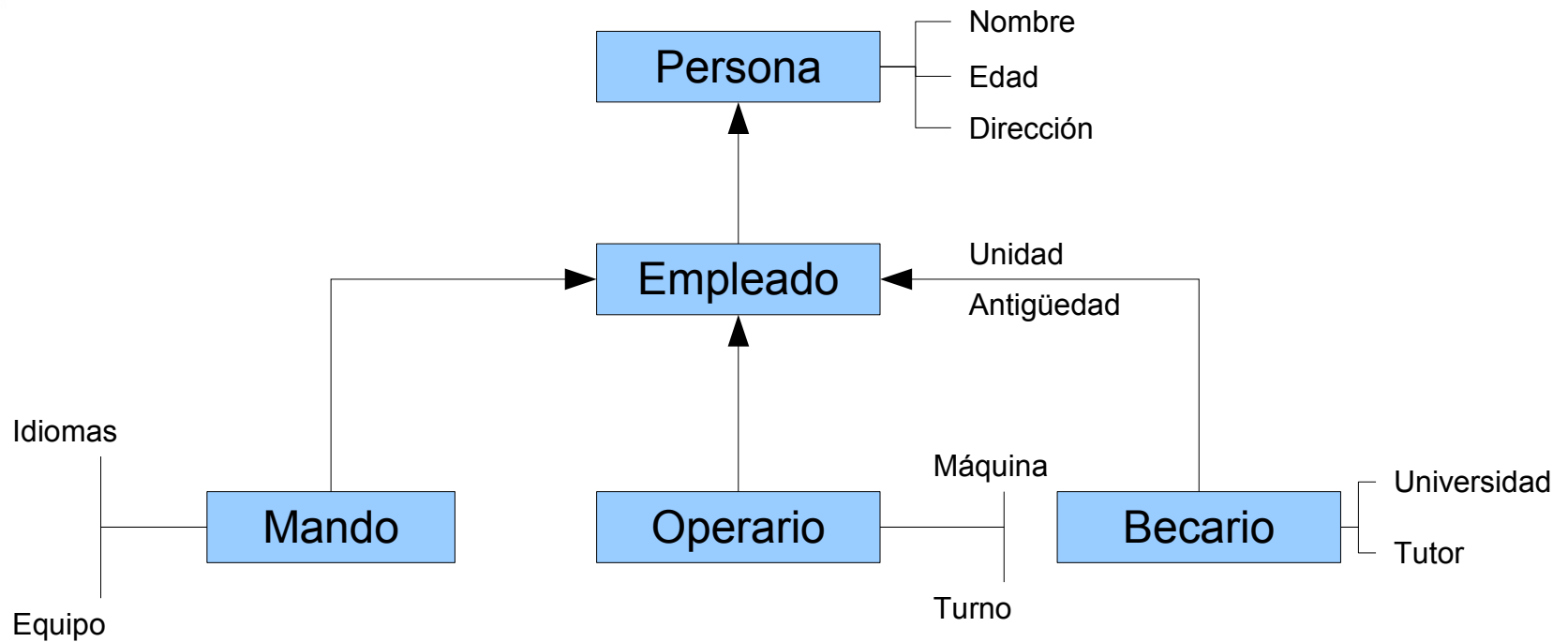


# POO. Herencia (III)

- Elementos capaces de ser heredados
  - Herencia Estructural
  - Herencia de Comportamiento ( herencia de métodos)
- Tipos de Herencia
  - Simple
  - Múltiple
- Definición de Herencia Múltiple
  - Una clase puede heredar rasgos de más de una superclase.
  - Una clase con más de una superclase es llamada clase junta.
  - Un rasgo de una clase ancestro que se encuentra más de una vez a lo largo de una ruta solo se hereda una vez.



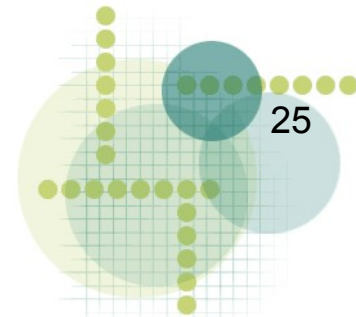
# POO. Herencia (IV)



# POO. Herencia (IV)

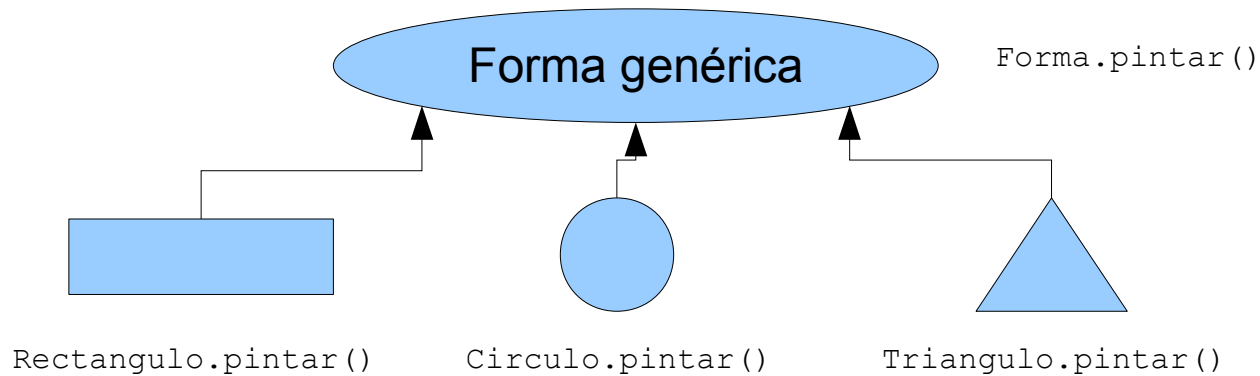
## ● Ventajas

- Ayuda a los programadores a ahorrar código y tiempo, al ser posible reutilizar el mismo a lo largo de diferentes clases/objetos con comportamientos similares para un determinado aspecto.
- Los objetos pueden ser construidos a partir de otros similares. Para ello es necesario que exista una clase base y una jerarquía de clases.
- La clase derivada puede heredar código y datos de la clase base, añadiendo código o modificando lo heredado.
- Las clases que heredan propiedades de otra clase pueden servir como clase base de otras.

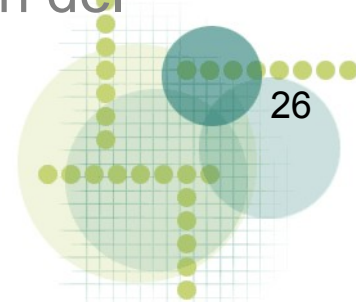


# POO. Polimorfismo

- Es la posibilidad de definir de forma distinta un método, dependiendo del objeto.

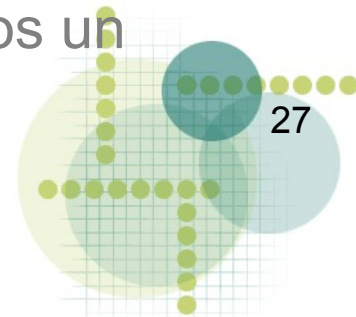


- Capacidad que tienen los objetos de una clase de responder al mismo mensaje o evento en función de los parámetros utilizados durante su invocación.
- Un objeto polimórfico es una entidad que puede contener valores de diferentes tipos durante la ejecución del programa.
- El polimorfismo existe gracias a la herencia



## POO. Polimorfismo (II)

- En la práctica: un puntero a un tipo puede contener varios tipos diferentes, no solo el creado. De esta forma podemos tener un puntero a un objeto de la clase Trabajador, pero este puntero puede estar apuntando a un objeto subclase de la anterior como podría ser Marketing, Ventas o Recepcionistas (todas ellas deberían ser subclase de Trabajador).
- El concepto de polimorfismo se puede aplicar tanto a funciones como a tipos de datos. Así nacen los conceptos de *funciones polimórficas* y *tipos polimórficos*. Las primeras son aquellas funciones que pueden evaluarse o ser aplicadas a diferentes tipos de datos de forma indistinta; los tipos polimórficos, por su parte, son aquellos tipos de datos que contienen al menos un elemento cuyo tipo no está especificado.



# POO. Polimorfismo (III). Clasificación.

- Se puede clasificar en dos grandes clases:
  - **Polimorfismo dinámico** (o polimorfismo paramétrico): Aquél en el que el código no incluye ningún tipo de especificación sobre el tipo de datos sobre el que se trabaja. Así, puede ser utilizado a todo tipo de datos compatible. (En C++ tenemos los templates para realizar esto). —▶ (junto con Herencia) Programación genérica
  - **Polimorfismo estático** (o polimorfismo ad hoc): Aquél en el que los tipos a los que se aplica el polimorfismo deben ser explicitados y declarados uno por uno antes de poder ser utilizados.
- Según la herencia puede clasificarse en
  - **Sobreescritura**: reemplazar los métodos de la superclase en las subclases.
  - **Sobrecarga**: varios métodos se llaman igual, pero difieren en el número, tipo u orden de sus argumentos.



# POO. Herencia y polimorfismo. Ejemplo

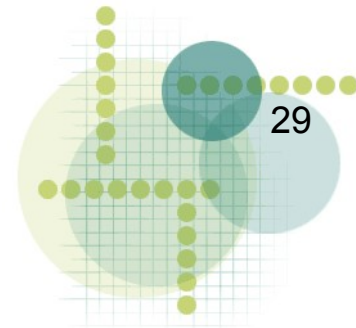
## ● Clase base ( en C++)

```
class Empleado {  
protected:  
    static const unsigned int SUELDO_BASE = 700; // Supuesto sueldo base para todos  
  
public:  
    /* OTROS MÉTODOS */  
    virtual unsigned int sueldo() = 0;  
};
```

## ● Clase derivada

```
class Director : public Empleado {  
public:  
    /* OTROS MÉTODOS */  
    unsigned int sueldo() { return SUELDO_BASE*100; }  
};
```

```
class Ventas : public Empleado {  
private:  
    unsigned int ventas_realizadas; // Contador de ventas realizadas por el vendedor  
  
public:  
    /* OTROS MÉTODOS */  
    unsigned int sueldo() { return SUELDO_BASE + ventas_realizadas*60; }  
};
```



# POO. Herencia y polimorfismo. Ejemplo (II)

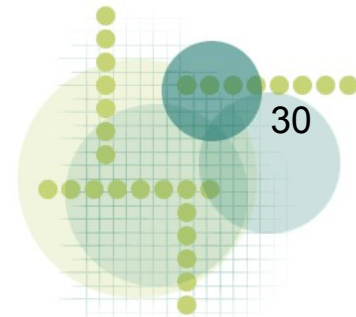
- ¿Cómo se usan estas clases?

```
int main() {
    Empleado* e[2]; // Punteros a Empleado
    Director d; // Declaración como objeto normal en la pila
    Ventas v; // Estas dos las declararemos como objetos normales en la pila

    e[0] = &d; // Asignamos a un puntero a Empleado la dirección de un objeto del tipo Director
    e[1] = new Ventas(); // Creamos dinámicamente un nuevo objeto

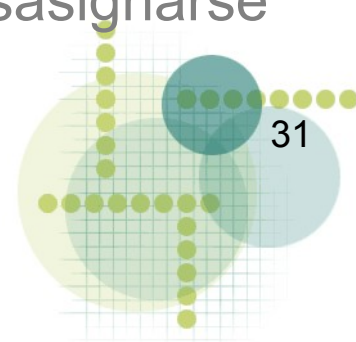
    unsigned int sueldo = 0;
    for (int i = 0; i < 4; ++i)
        sueldo += e[i]->sueldo();

    cout << "Este mes vamos a gastar " << sueldo << " dinero en sueldos" << endl;
}
```



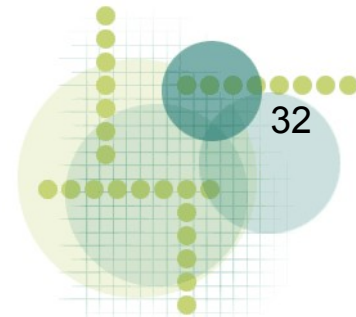
# POO. Recolección de basura

- Garbage Collection
- Técnica por la cual el ambiente de Objetos se encarga de destruir automáticamente, y por tanto de desasignar de la memoria, los Objetos que hayan quedado sin ninguna referencia a ellos.
- Esto significa que el programador no debe preocuparse por la asignación o liberación de memoria, ya que el entorno la asignará al crear un nuevo Objeto y la liberará cuando nadie lo esté usando.
- En la mayoría de los lenguajes híbridos que se extendieron para soportar el Paradigma de Programación Orientada a Objetos como C++ u Object Pascal, esta característica no existe y la memoria debe desasignarse manualmente.



# POO. Resumen de principios básicos

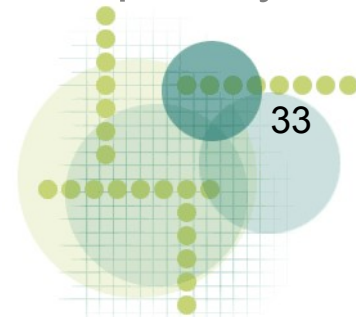
- Todo es un objeto
- Los objetos realizan operaciones solicitando servicios entre ellos a través del paso de mensajes (llamándose entre ellos a través de sus diferentes operaciones)
- Todo objeto tiene su propia memoria o estado y consiste de datos básicos u otros objetos
- Todo objeto es una instancia de una clase. Una clase agrupa objetos similares.
- En la clase se describe el comportamiento de los objetos.
- Las clases están organizadas en una jerarquía con una única raíz, llamada la jerarquía de herencia.
- Los objetos pueden redefinir métodos de clases superiores, adaptándolos a sus necesidades.



# POO. Pautas básicas para un diseño OO

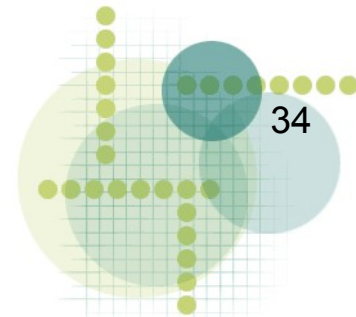
## ● Pautas

- Hacer una lista de todas las propiedades y métodos que requiera el programa.
  - Clasificar dichas propiedades y métodos dentro de clases.
  - Examinar las clases para ver las posibles relaciones de herencia.
  - Establecer los métodos necesarios para realizar la interface entre las diversas clases.
  - Comprobar que dentro de la estructura de clases todo encaja.
- Realizar un buen diseño supone:
- Ahorro e tiempo y esfuerzo en la fase de programar
  - Código resultante de mayor calidad: más fácil de depurar y mantener



# POO. Lenguajes orientados a objetos

- No todos son puros, existen muchos lenguajes híbridos (como C++)
  - ABAP
  - ActionScript
  - Ada
  - C++
  - C#
  - Clarion
  - Lenguaje de programación D
  - Object Pascal (Delphi)
  - Harbour
  - Eiffel
  - Java
  - JavaScript (la herencia se realiza por medio de la programación basada en prototipos)
  - Objective-C
  - Ocaml
  - Lenguaje de programación R
  - Perl
  - PHP (en su versión 5)
  - Python
  - Ruby
  - Smalltalk
  - VB.NET
  - Visual FoxPro (en su versión 6)
  - Visual Basic



# POO. Fases en el diseño de software

## ● Análisis de requisitos

- Se requiere de habilidad y experiencia en la ingeniería de software para reconocer requisitos incompletos, ambiguos o contradictorios.
- El resultado del análisis de requisitos con el cliente se plasma en el documento ERS, *Especificación de Requerimientos del Sistema*, cuya estructura puede venir definida por varios estándares, tales como CMM-I. Asimismo, se define un *diagrama de Entidad/Relación*, en el que se plasman las principales entidades que participarán en el desarrollo del software.

## ● Especificación

## ● Diseño y arquitectura

- Casos de uso

## ● Programación

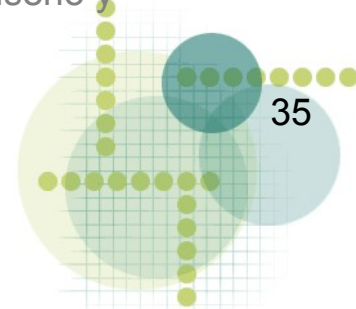
- No necesariamente es la que demanda mayor trabajo ni la más complicada

## ● Pruebas

- Fundamental. **Pruebas unitarias.** Se diseñan e implementan incluso desde el análisis de requisitos (en algunos casos, lo normal es a partir del diseño y arquitectura).

## ● Documentación

- Fundamental. Se realiza desde la primera fase.



# POO. Evolución

- Programación dirigida a eventos
  - Paradigma de programación en el que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema o que ellos mismos provoquen. Muy utilizada en la implantación de aplicaciones de escritorio (con interfaces de usuario).
- Programación orientada a aspectos
  - Paradigma de programación relativamente reciente cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos. Gracias a la POA se pueden encapsular los diferentes conceptos que componen una aplicación en entidades bien definidas, eliminando las dependencias entre cada uno de los módulos. De esta forma se consigue razonar mejor sobre los conceptos, se elimina la dispersión del código y las implementaciones resultan más comprensibles, adaptables y reusables.

