Unidad 7

Programación Orientada a Objetos

Objetivos

- Conocer un poco la historia sobre el paradigma orientado a objetos.
- Entender la necesidad de usar y definir TAD.
- Entender el concepto de encapsulación y ocultación y sus ventajas en la POO.
- Conocer los principios y características básicas de la POO.
- Comprender y usar el concepto de interfaz como mecanismo para el diseño de tipos.
- Comprender los conceptos de objeto, clase, atributo, método e instancia
- Saber diseñar tipos de datos a partir de contratos e interfaces.
- Iniciarse en el diseño de clases para atender las necesidades de un problema completo.
- Interpretar el código fuente de una aplicación Java donde aparecen implementados los conceptos anteriores
- Construir una aplicación Java sencilla, convenientemente especificada, que emplee los conceptos anteriores.
- Aplicar un planteamiento orientado a objetos para la resolución de un problema.
- Valorar la facilidad de la POO para la reutilización de código.

Contenidos

1. INTRODUCCIÓN
2. EVOLUCIÓN HISTÓRICA
   - 2.1. Orígenes del paradigma OO
   - 2.2. Evolución histórica del diseño de software
   - 2.3. ¿Qué significa orientado a objetos?
3. LA FAMILIA ORIENTADA A OBJETOS
4. OO. CONCEPTOS BÁSICOS
   - 4.1.Tipos abstractos de datos
   - 4.2. Objetos
   - 4.3. Interfaz
   - 4.4. Clases
5. PASOS EN EL DISEÑO OO

Introducción

- **La técnica estructurada:**
  - **Fue usada hasta los 90 en el desarrollo de aplicaciones de todo tipo.**
  - **Hoy cuando el software a desarrollar no es complejo.**
  - **y sobre todo en la implementación de módulos específicos.**
- **La técnica de AOO, DOO y POO:**
  - **Es muy eficaz a la hora de enfocar y enfrentarse a aplicaciones informáticas complejas.**
  - **Permite la construcción de software de más calidad y más barato.**

En este capítulo, se presenta la evolución histórica de las técnicas de programación y los conceptos básicos que soporta el paradigma OO.

Orígenes

- **Ideas centrales de POO: Simula (acrónimo de "Simulation language", Oslo 1967)**
  - **creado para simular procesos paralelos.**
  - **los módulos no se basan en procedimientos como en la programación estructurada, sino en los *objetos* que se modelan en la simulación.**
- **Smalltalk (72-76) para Rank Xerox.**
  - **Reutiliza las ideas de simula.**
  - **Genera un entorno gráfico basado en ellas (sistema multiventana que luego le copió Windows).**
  - **El lenguaje se populariza en universidades.**
- **Con máquinas más potentes aparecen nuevos lenguajes:**
  - **C++, Eifell, ObjetiveC, Object Pascal, Modula Orientado a Objetos (Oberon), JAVA.**

Evolución Histórica del diseño

- **Sin método (antes de los 60): *Crisis del software.***
- **Metodologías (60).**
  - **Programación estructurada.**
    - **Estructuras-Abstracción de datos.**
    - **Estructuras-Abstracción de control.**
  - **Programación modular.**
    - **Construcción de Software con módulos.**
    - **Reutilización de módulos.**
    - **Tipos abstractos de datos, TADS.**

Evolución Histórica del diseño

A más complejidad en los proyectos, equipo de desarrollo más numeroso, gran esfuerzo en coordinación, menor rendimiento individual.

En este contexto nace la,

- **POO (70).**
  - **Dividir el trabajo de forma más coherente y en unidades más independientes.**
  - **Brand J. Cox (Objetive-C) acuñó el término *software-IC* →CIS (Componentes Integrados Software, al estilo de Componentes Integrados Hardware).**
  - **Productores de componentes.**
  - **Consumidores de componentes.**

Evolución de la metodología OO

- **Programación por contrato (Eiffel).**
- **Programación orientada a eventos (Visual Basic).**
- **Programación orientada a componentes (.Net, C#).**
- **Programación orientada a aspectos (AspectJ).**
- **Programación reflexiva (Ruby)**

Qué significa OO

- Entidades básicas: **Objetos** - abstracciones del mundo real, representadas mediante cajas negras, cada uno responsable de si mismo, que encapsulan funcionalidades y se comunica con los demás mediante mensajes.
  - **Fuerte relación entre:**
    - ***Estructura de datos y***
    - ***Comportamientos de esos datos en respuesta a mensajes,***
  - **Dejando visible al exterior su interfaz de comunicación.**

Desarrollo OO

- **Definición**
  - La OO es un método de desarrollo de software que basa la arquitectura de la aplicación en módulos derivados de los tipos de objetos que se manipulan, en lugar de basarse en la función o funciones que la aplicación debe realizar.

*No preguntes primero **qué hace el sistema**, pregunta*
¡¡QUIÉN LO HACE!!

- La idea principal consiste en pensar en la identificación de entidades de una aplicación y no en la representación final en un lenguaje de programación concreto.
- **La OO es aplicable a:**
  - **Análisis.**
  - **Diseño.**
  - **Programación.**
  - **Bases de datos.**

La calidad y sus factores en OO

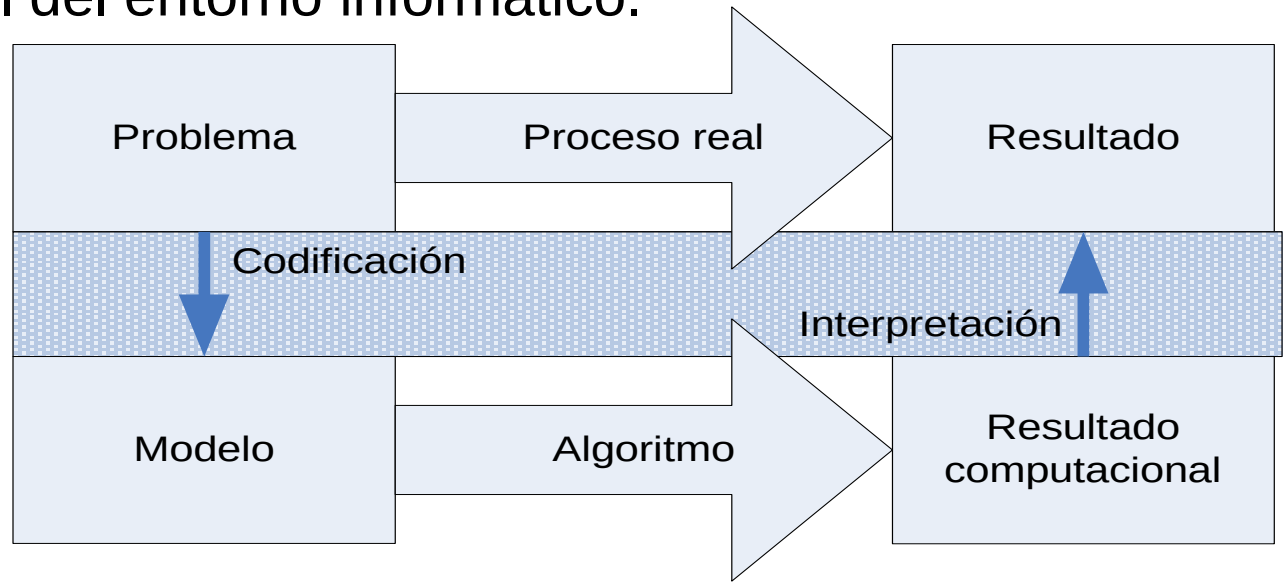
- Externos: son perceptibles por los usuarios.
  - Corrección
  - Robustez
  - Compatibilidad
  - Eficiencia
  - Portabilidad
  - Facilidad de uso
  - Funcionalidad
- Internos: son los perceptibles por los profesionales en programación.
  - Modularidad
  - Legibilidad
  - Modificabilidad
  - Reusabilidad
  - Extensibilidad

Familia OO

- **Programación:**
  - ***Lenguajes.***
    - Puros: Smalltalk, **Java**.
    - Extensiones: C++, Object Pascal.
- **Análisis y diseño:**
  - **OMT (*Object Modeling Technique*) (RUMBAUGH, 1991)**
  - **UML (*Lenguaje Unificado de Modelado*), metodología establecida como estándar en 1997.**
  - **RUP (*Proceso Unificado de Rational*) (KRUCHTEN, 2000)**
  - **Sistemas distribuidos multinivel: *CORBA*, *J2EE***
- **Bases de datos:**
  - **Extensiones de relacionales: Oracle, SQL Server** (mal).
  - **Nuevos sistemas: GESTONE/OPAL de ServioLogic, ONTOS de Ontologic, Objectivity de Objectivity Inc., etc.**
  - **Hibernate:** herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java
- **Sistemas operativos OO.**
  - **Next (Escrito en Objetive C).**
  - **Interfaces: X/Windows.**

OO, Conceptos básicos

- Los objetos del mundo real tienen sus propias características y comportamientos ante estímulos externos.Pueden, además, tener una variedad infinita de efectos sobre otros, creando, destruyendo, comprando, enviando, etc.
- Esto debe modelarse en el mundo informático.
- La idea de la OO es estrechar la franja que separa el mundo real del entorno informático.



Primer acercamiento

Una aplicación diseñada con metodología OO es un **conjunto de objetos** (surtidor, coche, persona) que **intercambian mensajes** (MoverCoche, ActivarSurtidor, ColocarManguera) que **activan procesos** que pueden cambiar sus estados internos (Cantidad de gasolina en el surtidor, Cantidad gasolina depósito coche, Cantidad de dinero de la persona) y pueden devolver parámetros (Tanque coche lleno, ticketCompra).

El usuario también es visto como un objeto que interactúa con el resto de objetos del sistema, generalmente por medio del ratón, iconos, menús desplegables y cuadros de diálogos.

Tipos Abstractos de Datos (TAD)

- **Definición**: Un TAD describe una estructura de datos junto a las operaciones que pueden realizarse sobre ellos, integrando las propiedades formales (F. acceso, axiomas) de esas operaciones.

Realizar la descripción del TAD PILA (pag 8).

- Un TAD permite:
  - La encapsulación: facilidad para guardar conjuntamente datos y operaciones que actúan sobre ellos.
  - Ocultación: capacidad de ocultar cara al exterior la representación interna de los objetos y el código de los algoritmos utilizados para implementar su comportamiento.

Tipos Abstractos de Datos (TAD)

- Ventajas:
  - Facilidad para modelar el mundo real.
  - Asegura la consistencia de los datos.
  - Separa la implementación de la interfaz.

    Facilita la modificación de la primera sin que afecte

    a la segunda, ni a los programas que la usen.
  - Los tipos se generan y pueden depurarse independientemente.
  - Aumenta la reutilización de código, facilitando la creación de librerías de componentes (fáciles de usar, mantener y ampliar).

Objetos

- **El concepto general depende del nivel de abstracción elegido para definirlo:**
  - **Representan los datos de un problema (definición incompleta).**
  - **Unidad tangible con comportamiento bien definido (Booch).**
  - **Encapsulación abstracta de información junto con los métodos o procedimientos para manipularla (Devis).**
  - **Es estado un elemento individual (abstracción de datos), identificable por sus propiedades identidad ,con un comportamiento definido:**

Objeto = estado + identidad+ comportamiento

Objetos: Propiedades

- Las propiedades son las características “observables” de un objeto desde el exterior del mismo.
- Pueden ser una o varias
- Pueden ser de diversos tipos, enteros, reales, cadenas, booleanos o cualquier otro tipo no básico definido con anterioridad.
- Pueden ser
  - **Básicas**: independientes entre sí
  - **derivadas**: si puede ser calculada a partir de otras propiedades del mismo objeto.
- Por otro lado, las propiedades pueden ser:
  - **Consultables**: Solo se puede consultar desde el exterior.
  - **Modificables**: se puede modificar su valor desde el exterior.

Objeto. Identidad

- **La identidad es una característica que distingue a un objeto del resto de los objetos.**
  - **Ejemplo: Una persona puede cambiar cualquiera de sus atributos (nombre, estado,..) y lo que sabe hacer (métodos), pero nunca cambia su identidad mientras viva.**
- **Dos objetos pueden ser iguales pero no idénticos si:**
  - **Tienen los mismos atributos y**
  - **Los mismos valores en sus atributos.**
    - **Ejemplos: gemelos al nacer, coches recién salidos de fábrica.**
- **Se hablará de esto más adelante.**

Objeto. Estado

- **El estado de un objeto está determinado por el conjunto de propiedades (*estructura estática*. Persona: lo que soy) con sus valores.**
  - ***Cada propiedad tiene un valor en un dominio.***
  - ***Los valores de las propiedades pueden variar mientras viva el objeto.***
  - ***Los valores dependen del dominio del problema.***
  - ***Los atributos deben ser privados.***

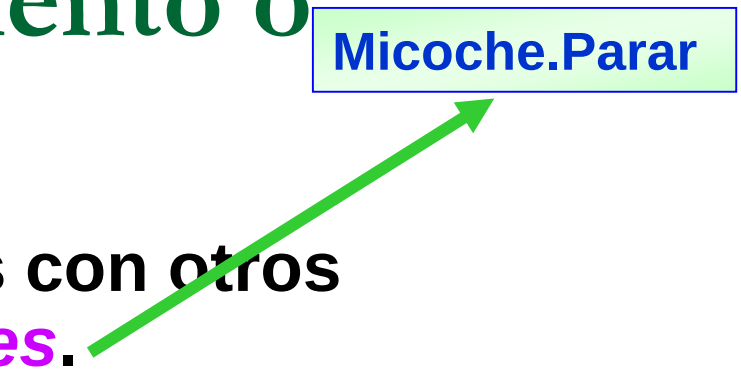
color

Comprar un coche:
precio,
color,
potencia del motor,
accesorios,
número de puertas.

Participar en un rally:
velocidad,
aceleración,
anchura de las ruedas
capacidad de frenado.

Tener un estado supone ocupar un espacio en memoria, existe durante un tiempo, puede cambiar el valor de sus propiedades, ser referido, creado y destruido

Objeto. Comportamiento o funcionalidad



- **Los objetos interactúan unos con otros mediante el envío de *mensajes*.**
- **El mensaje activa un comportamiento, *método* o servicio.**
- **El conjunto de métodos se llama *interf* *protocolo*.**
- **La interfaz debe ser pública.**

Coche:
Atributos
velocidad,
aceleración...
Metodos:
Parar
Frenar
Encender

Los términos **llamada a método**, **llamada a función** o **envío de mensaje** se utilizan para referirse a que se ejecuta una operación.

Objeto. Comportamiento

- **Pueden generarse operaciones para:**
  - ***Consultar aspectos internos del objeto (Métodos Accesores).***
  - ***Modificar atributos (Métodos Modificadores o Mutadores).***
  - ***Enviar mensajes a otros objetos.***
  - ***Construir el objeto (Métodos Constructores).***
  - ***Destruir el objeto (Métodos Destructores).***

Envío de mensaje

- En un mensaje intervienen:
  - El ***receptor*****,** que es el objeto que recibe el mensaje.
  - El ***selector***, que es el comportamiento o método implicado en el mensaje.
- Un mensaje se realiza con el nombre del objeto receptor seguido del nombre del método selector.
- Si un método requiere información adicional para realizar su tarea, el mensaje incluye la información como ***parámetro***.
- Formato:

NombreObjeto.NombreMétodo(argumentos)

Ejemplo: Siendo ***O1*** el objeto receptor del mensaje, ***met1*** el selector o comportamiento implicado en dicho mensaje y ***Var*** el parámetro enviado, el envío lo escribiremos:

O1. Met1 (Var)

Envío de mensaje

- El resultado tras el envío de mensaje depende:
  - Del estado en que se halle el objeto al recibir el mensaje.
  - Del método activado con el mensaje.
  - De la información que el mensaje transporte.
- Cualquier método de un objeto tiene **total visibilidad** sobre los atributos (privados) de ese objeto.
- La forma de acceder a los atributos de un objeto es por activación de un método que lo permita a través del envío de un mensaje.

Ejemplo: Objetos: lámpara, lupa, papel

```
Lampara.encender () //devuelve un haz de luz
Lupa.concentrahaz(luz) // devuelve calor
Papel.AumentaTemperatura(calor) //?????????
```

Referencia a si mismo

- Los servicios o métodos de un objeto pueden ser:
  - Públicos: si admiten el envío de mensajes desde el exterior.
  - Privados: si son métodos que actúan exclusivamente por envío de mensaje desde el propio objeto a sí mismo.
- Un objeto envía un mensaje a si mismo activando un método privado.
  Ejemplo: al recibir el papel el mensaje *AumentaTemperatura*, comprueba su estado y activa un método interno *quémate* que lo hace cambiar de estado si el papel no estuviera quemado ya.
- Este envío de mensaje se referenciará en pseudocódigo con la palabra **mimismo**. El envío al propio papel se escribe

  mimismo.quemate().
- Se referencia como *this* en Java y C++, en Smalltalk como *self.*
- En algunos lenguajes puede eliminarse la referencia a si mismo, Java y C++.

Interfaz o protocolo - contrato

- Es un elemento de la POO que establece:
  - Las propiedades de un objeto.
  - La funcionalidad de un objeto.
- Una interfaz propone un contrato sintáctico
  - Dice ***qué*** *se puede hacer con un objeto, pero* ***no cómo****.*
  - Contiene signaturas de métodos.
  - Contiene restricciones.

Interfaz o protocolo - contrato

- Es un elemento de la POO que establece:
  - Las propiedades de un objeto.
  - La funcionalidad de un objeto.
- Una interfaz propone un **contrato** sintáctico
  - Dice ***qué*** *se puede hacer con un objeto, pero* ***no cómo****.*
  - Contiene signaturas de métodos.
  - Contiene restricciones.

Interfaz (II)

- Metodología de diseño de interfaz de objetos:
  1. Establecer las propiedades relevantes. Las propiedades pueden depender de parámetros.
  2. Para cada propiedad indicar su tipo, si es básica, si es consultable y/o modificable y parámetros de los que depende.
  3. Indicar las relaciones entre propiedades, si es que existen. Es decir, especificar las propiedades derivadas.
  4. Indicar si una propiedad es compartida por todos los objetos del tipo (propiedad de clase *static*).
  5. Por cada propiedad:
     1. Si es consultable: Escribir método observador (**get**)
     2. Si es modificable: Escribir método modificador (**set**).
     3. Si es derivada: Escribir método observador (**get**)
  6. Escribir las restricciones si existen (pre, post,…).
  7. Escribir otros métodos que puedan ser necesarios.

Interfaz (III)

- Ejemplo: Modelar el tipo Punto aplicando la metodología anterior
  1. Establecer las propiedades relevantes.
  2. Para cada propiedad indi… es básica, si es consultable y/o modifi… etros de los que depende.
     - Coordenada X
     - Coordenada Y
  3. Especificar las propiedades derivadas.
  4. Indicar si una propiedad es compartida.
  5. Por cada propiedad:
     1. Si es consultable: Escribir método observador (**get**)
     2. Si es modificable: Escribir método modificador (**set**).
     3. Si es derivada: Escribir método observador (**get**)
  6. Escribir las restricciones si existen (pre, post,…).
  7. Escribir otros métodos que puedan ser necesarios.

Interfaz (III)

- Ejemplo: Modelar el tipo Punto aplicando la metodología anterior
  1. Establecer las propiedades relevantes.
  2. Para cada propiedad indicar su tipo, si es consultable y/o modificable.
  3. Especificar las prop
  4. Indicar si una propi
  5. Por cada propiedad
     1. Si es consultable: E
     2. Si es modificable: E
     3. Si es derivada: Escr
  6. Escribir las restricci
  7. Escribir otros méto os.

- **Coordenada X**
  - **Tipo: Real ==> Double**
  - **básica**
  - **¿Consultable?: Sí**
  - **¿Modificable?: Sí**
- **Coordenada Y**
  - **Tipo: Real ==> Double**
  - **básica**
  - **¿Consultable?: Sí**
  - **¿Modificable?: Sí**

Interfaz (III)

- Ejemplo: Modelar el tipo Punto aplicando la metodología anterior
  1. Establecer las propiedades relevantes.
  2. Para cada propiedad indicar su tipo, si es básica, si es consultable y/o modificable.
  3. Especificar las propiedades derivadas.
  4. Indicar si una prop
  5. Por cada propieda
     1. Si es consultable: Escribir método observador (**get**)
     2. Si es modificable: Escribir método modificador (**set**).
     3. Si es derivada: Escribir método observador (**get**)
  6. Escribir las restricciones si existen (pre, post,…).
  7. Escribir otros métodos que puedan ser necesarios.

No hay propiedades derivadas

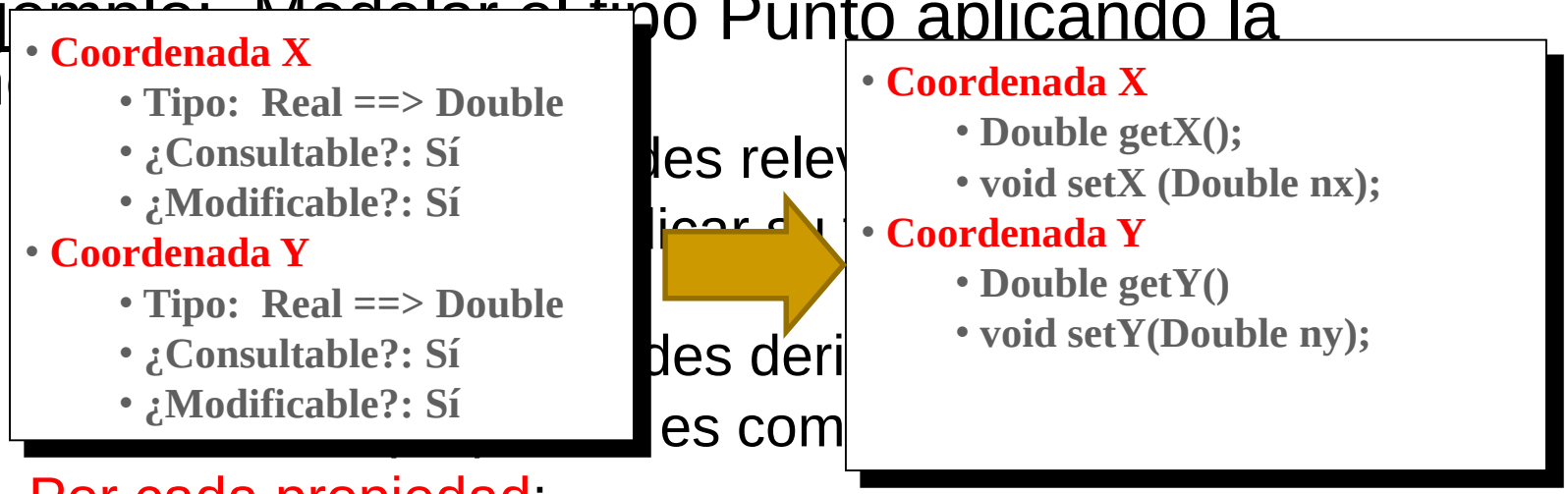
Interfaz (III)

- Ejemplo: Modelar el tipo Punto aplicando la metodología anterior
  1. Establecer las propiedades relevantes.
  2. Para cada propiedad indicar su tipo, si es básica, si es consultable y/o modificable.
  3. Especificar las propiedades derivadas.
  4. Indicar si una propiedad es compartida.
  5. Por cada propiedad:
     1. Si es consultable: Esc
     2. Si es modificable: Escribir método modificador (**set**).
     3. Si es derivada: Escribir método observador (**get**)
  6. Escribir las restricciones si existen (pre, post,…).
  7. Escribir otros métodos que puedan ser necesarios.

No hay propiedades compartidas

Interfaz (III)

- Ejemplo: Modelar el tipo Punto aplicando la m



1. des relev
2. icar s
3. des deri
4. es com
5. Por cada propiedad:
   1. Si es consultable: Escribir método observador (**get**)
   2. Si es modificable: Escribir método modificador (**set**).
   3. Si es derivada: Escribir método observador (**get**)
6. Escribir las restricciones si existen (pre, post,…).
7. Escribir otros métodos que puedan ser necesarios.

Interfaz (III)

- Ejemplo: Modelar el tipo Punto aplicando la metodología anterior
  1. Establecer las propiedades relevantes.
  2. Para cada propiedad indicar su tipo, si es básica, si es consultable y/o modificable.
  3. Especifica
  4. Indicar si u
  5. Por cada p
     1. Si es co
     2. Si es mo
     3. Si es de
  6. Escribir las restricciones si existen (pre, post,…).
  7. Escribir otros métodos que puedan ser necesarios.

- **Coordenada X**
  - **No puede ser menor que 0**
  - **No puede ser mayor que ?**
- **Coordenada Y**
  - **No puede ser menor que 0**
  - **No puede ser mayor que ?**

Interfaz (III)

- Ejemplo: Modelar el tipo Punto aplicando la metodología anterior
  1. Establecer las propiedades relevantes.
  2. Para cada propiedad indicar su tipo, si es básica, si es consultable y/o modificable.
  3. Especificar las propiedades derivadas.
  4. Indicar si una propiedad es compartida.
  5. Por cada propiedad:
     1. Si es consultable: Escribir método observador (**get**)
     2. Si es modificable: Escribir método modificador (**set**).
     3. Si es derivada:
  6. Escribir las restr…).
  7. Escribir otros métodos que puedan ser necesarios.

No hay otros métodos

Interfaz (III)

Mejorar el tipo punto (pag 15).

- Ejemplo: Modelar el tipo Punto aplicando la metodología anterior.
  - Resultado final

```
Interfaz
 Punto
      Double getX();
      void setX (Double x1);
      Double getY();
      void setY (Double y2);
```

Interfaz. Ejercicios de refuerzo

- Especificar el tipo Circulo
  - Del círculo nos interesa su radio, área, centro y longitud.
- Especificar el tipo Persona
  - De la persona nos interesa su DNI, nombre, apellidos, sueldo base, IRPF, sueldo.

Añade para ambos propiedades y funcionalidades que consideres interesantes.

Ejercicio

Tenemos una urna que contiene inicialmente un determinado número de bolas blancas y bolas negras.

Diseñar un programa que realice:

Extraer dos bolas de la urna mientras quede más de una bola en ella.

Si las dos bolas sacadas son del mismo color, se introducirá una bola negra en la urna.

Si ambas bolas son de distinto color, deberá meterse en la urna una bola blanca.

Cuando sólo quede una bola en la urna, se extraerá y determinará su color.

Clase

- Es un elemento de la POO que permite definir:
  - El estado interno de un objeto (atributos).
  - Implementar las funcionalidades ofrecidas por los objetos (métodos).
  - Establecer la forma de crear objetos (constructores).
- Proporciona el **cómo** para un objeto

Clase

- **Definición**: Es un tipo de dato abstracto (TAD), caracterizada por atributos y métodos comunes para todas sus instancias (sus objetos).

  Clase = atributos + métodos + instanciación

  - Ejemplo: **Urna u** → **Urna** es la clase, **u** su instancia

```
URNA u = new Urna(60, 55), u2;  //declaración y construcción
u.MeteBola (NEGRA);      //método. Ahora u tiene 61 bolas negras
u2.MeteBola (BLANCA);    //Ahora u2 tiene 56 bolas blancas
```

Clase. Atributos

- Los atributos se llaman también variables de instancia.
- Un atributo puede ser un tipo de clase, en cuyo caso refiere un objeto de otra clase.
- La declaración de atributo tiene ámbito de clase, incluido el cuerpo de los métodos.
- Tienen asociados modificadores de visibilidad (los veremos con detalle más adelante). Normalmente **private**
- Formato de declaración:

[Modificadores] tipo Identificador [= valor inicial];

Clase. Atributos

- Qué propiedades se convierten atributos y cuáles no:
  - La clase tendrá un atributo por cada propiedad básica.
  - Las propiedades derivadas no tienen un atributo asociado.
  - Las **propiedades compartidas:**
    - son atributos de clase.
    - llevan el modificador **static** además del tipo y la visibilidad.

Clase. Atributos

- Ejemplo:
  - Definir atributos para una clase que implemente el tipo Punto

- **Coordenada X**
  - **Tipo: Real ==> Double**
  - **¿Consultable?: Sí**
  - **¿Modificable?: Sí**
- **Coordenada Y**
  - **Tipo: Real ==> Double**
  - **¿Consultable?: Sí**
  - **¿Modificable?: Sí**

interfaz
Punto

```
Double getX();
void setX (Double nx);
Double getY();
void setY (Double ny);
```

Atributos:

```
private Double x;
private Double y;
```

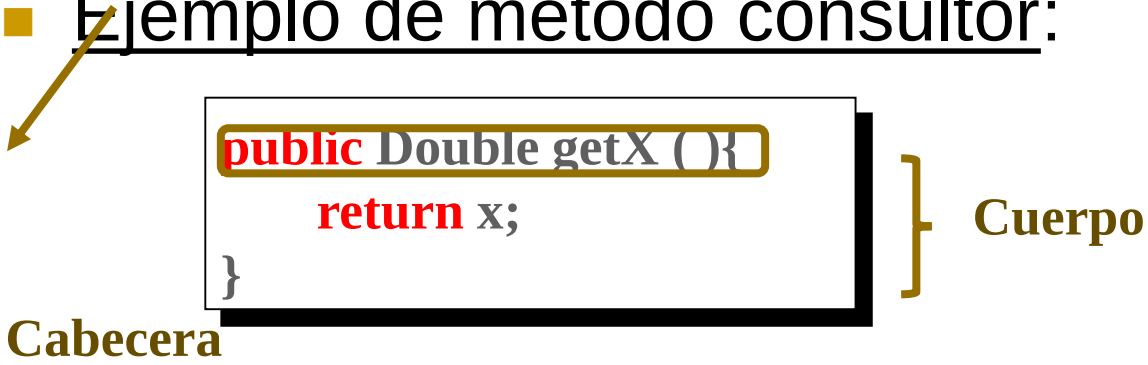
Tipo

Modificador Visibilidad

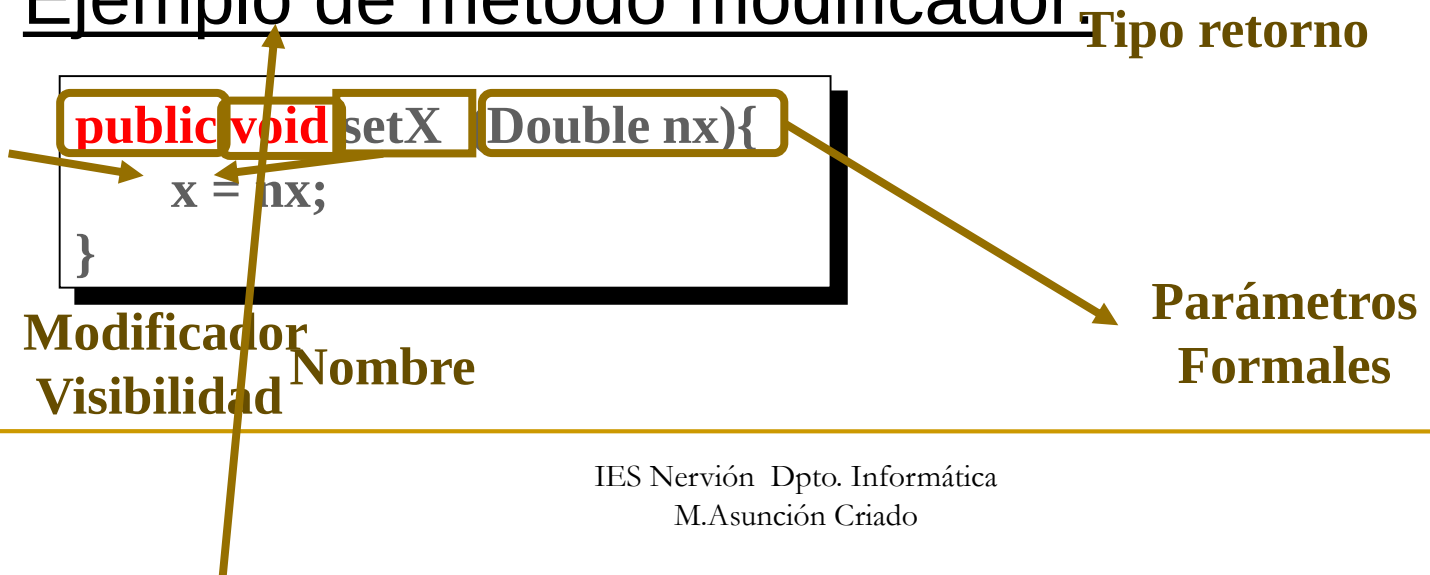
Nombre

Clase. Métodos

- Mecanismo para consultar o modificar las propiedades de un objeto.
- Ejemplo de método consultor:



- Ejemplo de método modificador:



Creación de un objeto

- Crear un objeto es similar a crear una variable cualquiera, hay que declararlo y definirlo para que exista.
  - Requiere:
    - **Declaración y asignación de zona de memoria.**
    - **Inicialización de su estado (dar valores a sus atributos).**
- En general, la creación de objetos se realiza a través de la activación de un ***método constructor***.
- Esto depende del lenguaje utilizado.

Clase. Métodos Constructores

- Métodos especiales que sirven para inicializar el estado de un objeto.
  - El nombre de un método constructor es el mismo que el de la clase que lo contiene.
  - **No devuelven nada, ni siquiera void**.
  - Siempre existe uno, incluso si no lo escribimos (constructor por defecto).
- Ejemplos de métodos constructores:

```
public  Punto (){
        x = 0.0;
        y = 0.0;
}
```

```
public  Punto (Double nx,  Double ny){
        x = nx;
        y = ny;
}
```

Destrucción de un objeto

- La destrucción de un objeto supone devolver la memoria que ocupa. Puede ser:
  - **Automática: *Recolector de basura* (Java y Smalltalk)**
  - **Manual: *método destructor* (C++)**

Ejemplo de creación de clase

```
public class Punto {
        private Double x;
        private Double y;
        public Punto(Double x1, Double y1) {
                x=x1;
                y=y1;
        }
        public Double getX() {  return x; }
        public Double getY() {  return y;  }
        public void  setX(Double x1)  {  x=x1; }
        public void  setY(Double y1)  {  y=y1; }
        public String toString() {
                String s="("+getX()+","+ getY()+")";
                return s; }
        public void mostrar(String s, Object p) {
                System.out.println(s + p);}
}
```

Cabecera de la Clase

Atributos

Métodos

Clase

- Algunas consideraciones a la hora de crear nuestras clases
  - Implementan una interfaz. A veces se añade alguna funcionalidad más.
  - El nombre de la clase será el de la interfaz asociada,.
  - El modificador de visibilidad de los atributos será private.
  - El modificador de visibilidad de los métodos será public.
  - Para invocar a un método constructor hay que anteponerle la palabra reservada new.
- Ejemplo de creación de objetos mediante constructores:

```
Punto p1 = new Punto(2.1, 3.7);
Punto p2 = new Punto( );
```

Declarar objetos

- Formato:

tipo NombreObjeto [= new tipo (arg)];

Ejemplos:

TipoA obj1 = new TipoA(0,0,0) **//crea *Obj1* de tipo *TipoA* con tres //atributos cuyos valores iniciales son cero.**

Tcoche coche = new Tcoche **(13000,"Rojo",1500,5) //crea *coche* de //tipo *Tcoche* con cuatro atributos cuyos // valores iniciales son ....**

Urna U = new (30, 44) **//crea el objeto U de tipo Urna, con dos //atributos cuyos valores son 30 y 44.**

Conductor

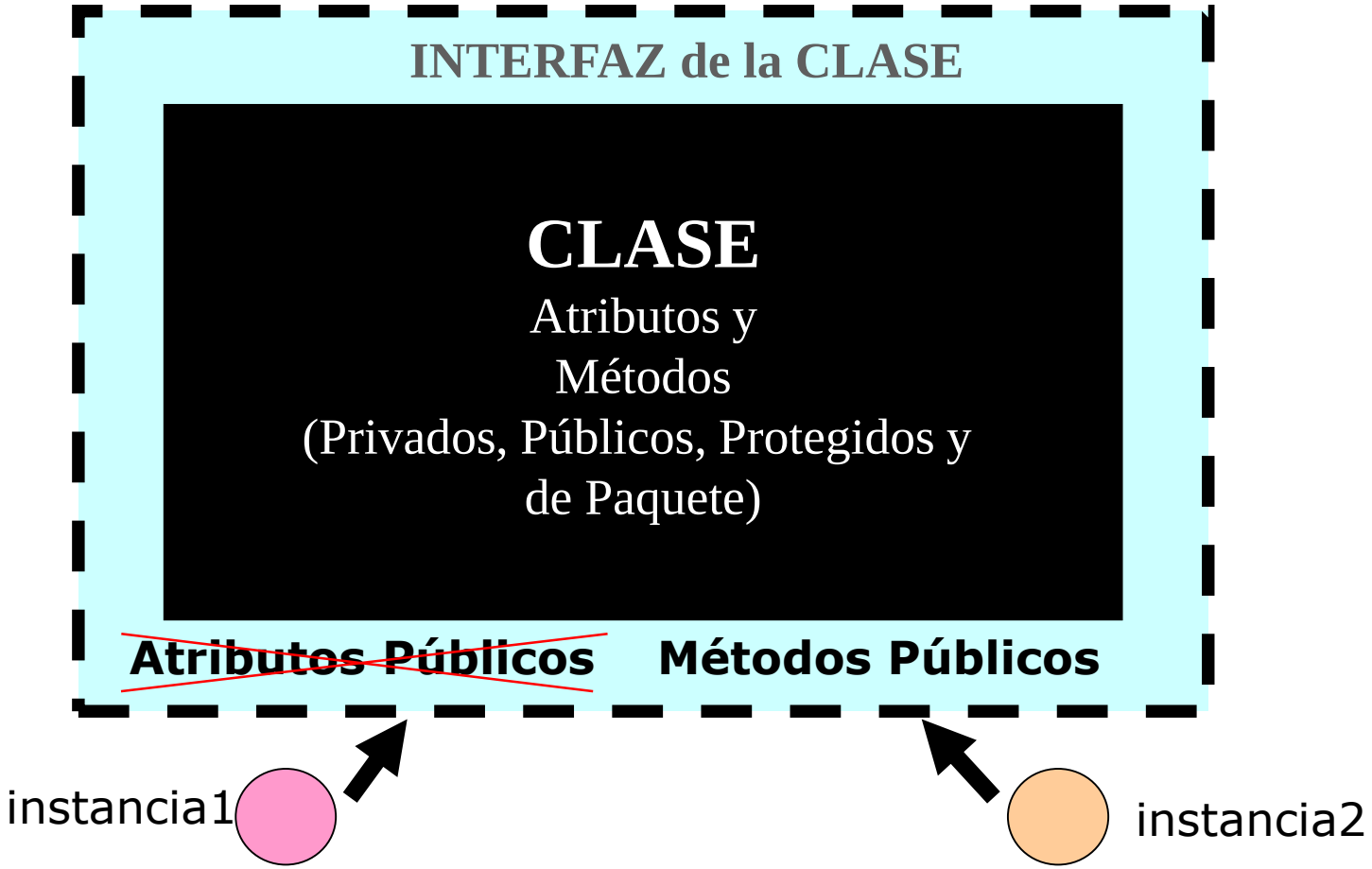
```
public  class  TestPunto {

  public static void main(String[ ] args) {
    Punto p= new Punto(2.0,3.0);
    mostrar("Punto:", p);
    p.setX(3.0);
    mostrar("Punto:", p);
    p.setY(2.0);
    mostrar("Punto:", p);
  }
}
```

Implementar un conductor que compruebe todas las funcionalidades de la clase Punto

Recuerda

Buenas Prácticas: los atributos deben ser privados (o Protegidos) y se debe acceder a ellos a través de los métodos públicos.



Clase. Ejercicios de refuerzo

- Escriba una clase Persona que implemente el tipo Persona especificado anteriormente.

- Escriba una clase Circulo que implemente el tipo Circulo especificado anteriormente.

Convenciones para idenficadores Java

- Clases
  - La primera letra de cada palabra en mayúscula.
- Atributos:
  - La primera letra del nombre en minúscula, para el resto, la primera letra de cada palabra en mayúscula.
- Métodos
  - Igual que los atributos.
- Paquetes
  - Igual que los atributos.
- Constantes
  - Todas las letras en mayúsculas. Para separar palabras se usa el carácter subrayado.

Grafos de clase

- **Una clase tiene distintas perspectivas:**
  - **es un tipo abstracto de datos**
  - **es un generador de instancias**
  - **puede tener como atributos objetos de otras clases**
  - **sus instancias mantienen relaciones con otros objetos de otras clases.**

  Con el objetivo de diferenciar gráficamente estos aspectos se usan distintos tipos de grafos:

- **Representación de una clase**
- **Instanciación**
- **Agregación**
- **Generalización (Árbol de jerarquía de clases)**

Representación de una clase

Aunque depende de la metodología usada, suele representarse mediante un rectángulo dividido en tres partes con:

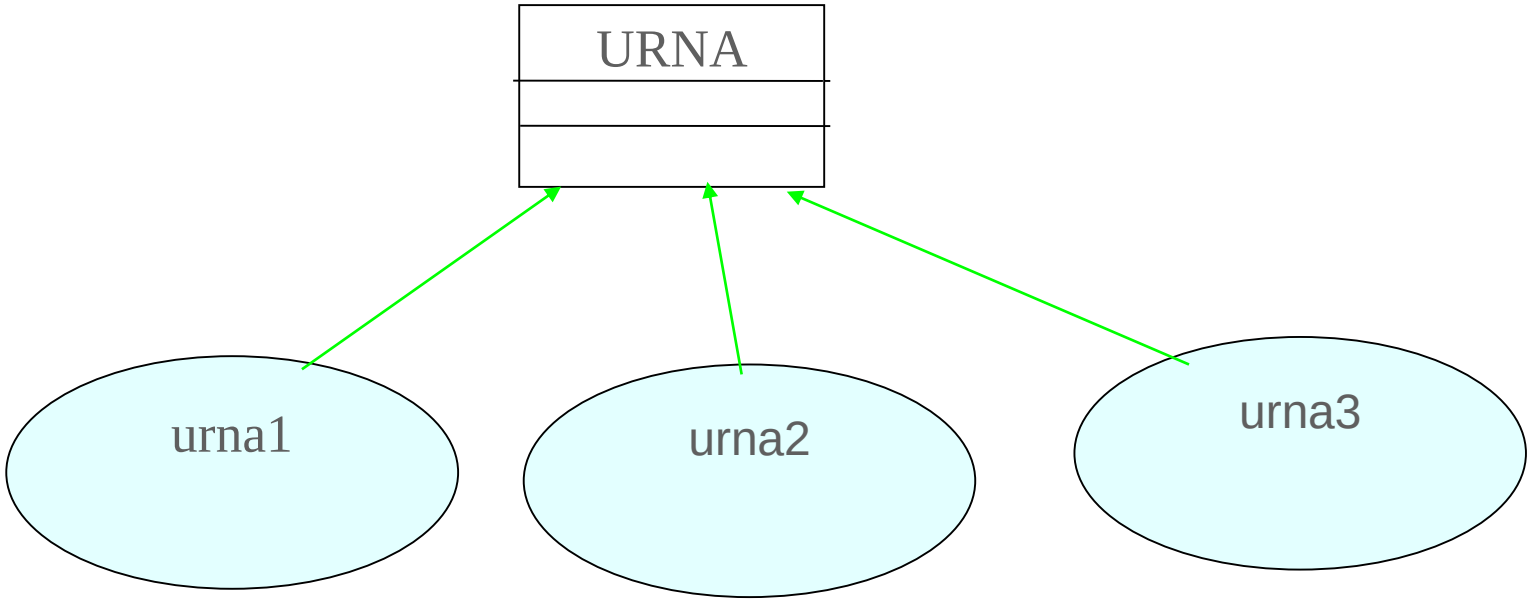
Primera el nombre de la clase

segunda los atributos

y tercera los métodos.

Grafo de instanciación

Relaciona las instancias de clase con su clase generatriz.

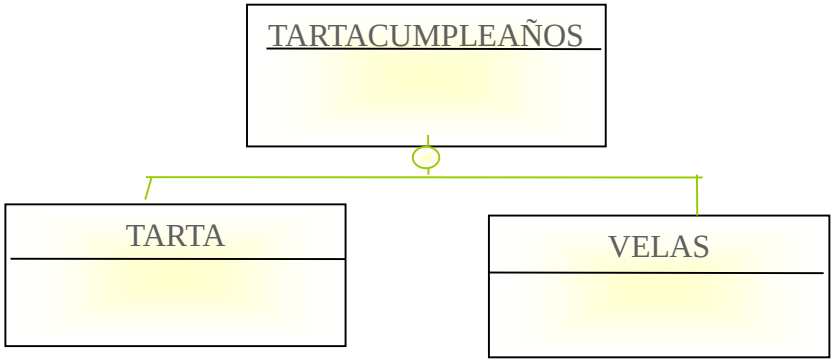


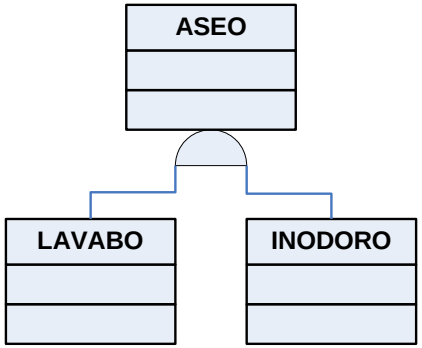
Grafo de agregación

- Un objeto puede tener como atributo otro objeto.
- La **agregación** es una relación entre dos clases que especifica los objetos de una clase que son componentes de otra.
- Esta relación es equivalente a la asociación (1-1 ó 1-N) entre entidades.
- Se representan con “Y lógico”.
- No confundir con **asociación:** objetos de clases diferentes se asocian circunstancialmente (Colaboración).
  - Ejemplo:

    Objetos independientes: Tarta, velas

    Asociados para: TartaCumpleaños





Polimorfismo

- “Facultad de un método u operador de poder aplicarse a objetos de clases diferentes ”.
- Hay dos tipos de polimorfismo:
  - **Por sobrecarga**:
    - ***De funciones***.
    - ***De operadores***. En Java no existe (aunque internamente sobrecarga el operador **+** para la concatenación de cadenas).
  - **Por vinculación dinámica**.

Sobrecarga de funciones

- La **sobrecarga de funciones** supone la redefinición de un método con un código distinto.
- Este tipo de polimorfismo se resuelve en tiempo de compilación.
- Permite un mismo nombre para métodos que realizan la misma actividad.
- Por tanto:
  a) Dos métodos externos a cualquier clase con el mismo nombre y distintos argumentos son métodos diferentes

  Ej. int main (void)

  int main (int argc)

Sobrecarga de funciones

a) ***Objetos de una misma clase pueden recibir mensajes con el mismo nombre y distintos argumentos***.

Dos métodos de una misma clase con el mismo nombre y distintos argumentos son métodos distintos.

Ej. **Urnita.InicializarUrna (int NumBB)**

Urnita.InicializarUrna (int NumBB, int NumBN)

a) ***Objetos de distintas clases, pueden recibir idénticos mensajes.***

Cada clase tendrá un método que responde dicho mensaje, por lo que, dos métodos con igual nombre y los mismos argumentos definidos en distintas clases también son métodos diferentes.

Ejemplo:

URNA ***urnita***.TotalBolas ( ) // bolas blancas + negras

URNA3 ***urnita3***.TotalBolas ( ) //bolas blancas+ negras + rojas

Sobrecarga de funciones: Ejemplo

- Con la sobrecarga de funciones podemos escribir:

  Complejo **Suma** (Complejo, Complejo)

  Suma (Racional, Racional, Racional)

Pasos en el diseño OO

- **Un objeto siempre debe responder a tres preguntas: ¿qué soy? (atributos), ¿qué sé hacer? (métodos), ¿cómo me relaciono con otros objetos? (relaciones)**

- **Localizar los objetos.**
- **Diseñar su interfaz-contrato**
  - **Establecer las propiedades.**
  - **Describir su conducta.**
  - **Describir las relaciones con otros objetos.**
  - **Describir las restricciones posibles.**
- **Usarlos en programas de aplicación.**
- **Implementar los tipos**

Localizar los objetos

- **El nombre de la aplicación a veces indica el nombre del objeto principal.**
- **Un objeto es un ente que pertenece al dominio del problema.**
- **Los objetos suelen ser sustantivos en el texto que describe la aplicación.**
- **Los objetos simulan el mundo real y no siempre han de corresponderse con objetos físicos.**

Ejemplos: Un número, un array, una urna, un bombo de bingo.

Establecer las propiedades

- **Las propiedades suelen ser sustantivos en el texto.**
- **Una propiedad guardará datos.**
  - **Por ejemplo, color, edad, número de bolas.**
- **Una propiedad tomará valores en el dominio del problema.**
  - **Por ejemplo, edad tomará valores entre 16 y 65 para los trabajadores de una empresa. Estos valores supondrán una restricción por lo que habrá que considerar las excepciones derivadas.**

Describir su conducta

- **Las operaciones suelen aparecer en el texto como verbos.**
- **Pensar en el objeto en primera persona es un truco que puede ayudar a identificar operaciones asociadas y atributos.**
  - **Por ejemplo, <<soy un círculo y puedo moverme, agrandar, reducir, cambiar de color y me definen un punto central, un radio, un área y un color de fondo>>.**
- **Sólo las operaciones comunes a los objetos de una clase deben definirse en ellas.**
- **Recuerde que a las operaciones típicas del objeto deben añadirse métodos para crear objetos (o destruir si no se dispone de colector de basura).**

Describir las relaciones con objetos de otras clases

- Responden a los predicados:
  - ***Es un***: conjunción que genera esquema jerárquico (**Generalización - Herencia**).
  - ***Tiene un***: atributo simple, posible agregado (**Agregación**) o una relación cliente servidor.
  - ***Es Como un***: Puede reflejar o no esquema jerárquico.
  - ***Usa un***: relación de uso que indica que una función miembro de una clase acepta un objeto de alguna otra clase como parámetro.

  Ej. Un perro ***es un*** mamífero, ***tiene un*** rabito, ***tiene una*** dentadura fuerte y ***es como un*** juguete para un niño. Por ejemplo, **Laika** (**instanciación**), primer ser vivo que fue al espacio.

  Ej. Un coche (**Cliente**) es un vehículo y tiene un conductor (**servidor**).

Paquete (I)

- Agrupación de clases e interfaces que, por la función que desempeñan, conviene mantener juntos.
- Similar a una carpeta que contiene ficheros y otras carpetas.
- Dentro de un paquete puede haber otros paquetes.
- El nombre de un paquete puede ser cualquier identificador.
- El acceso a los paquetes situados en el interior de otros paquetes se hace con el operador punto (.). Ejemplo: java.lang.
- El nombre cualificado de una clase está compuesto por todos los paquetes a los que hay que acceder para llegar a la clase. Ejemplo: java.lang.Object.
- El paquete en el que está una clase o una interfaz se indica anteponiéndole a la definición de la misma la sentencia package.

Paquete (II)

- Ejemplo de declaración de paquetes:

```
package nombreDelPaquete;
interface …{
 …
 …

}
```

```
package nombreDelPaquete;
public class … {
 …
 …

 }
```

```
package puntoPlano;
public interface Punto{
 …
 …

}
```

```
package puntoPlano;
public class PuntoImpl …{
 …
 …

 }
```

“No hay cosa,
por fácil que sea,
que no la haga difícil la desgana”

Jaume Vicens Vives