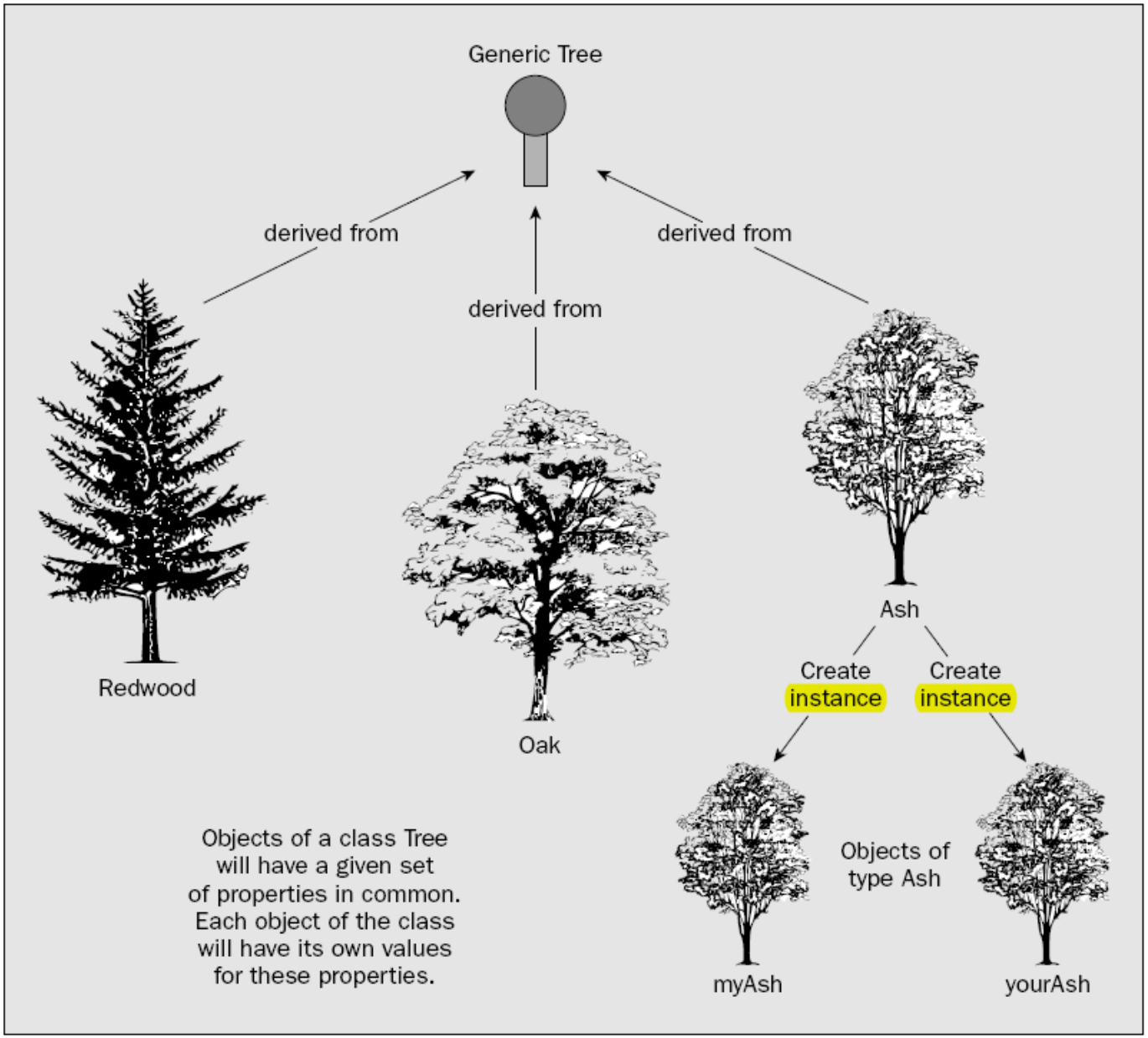
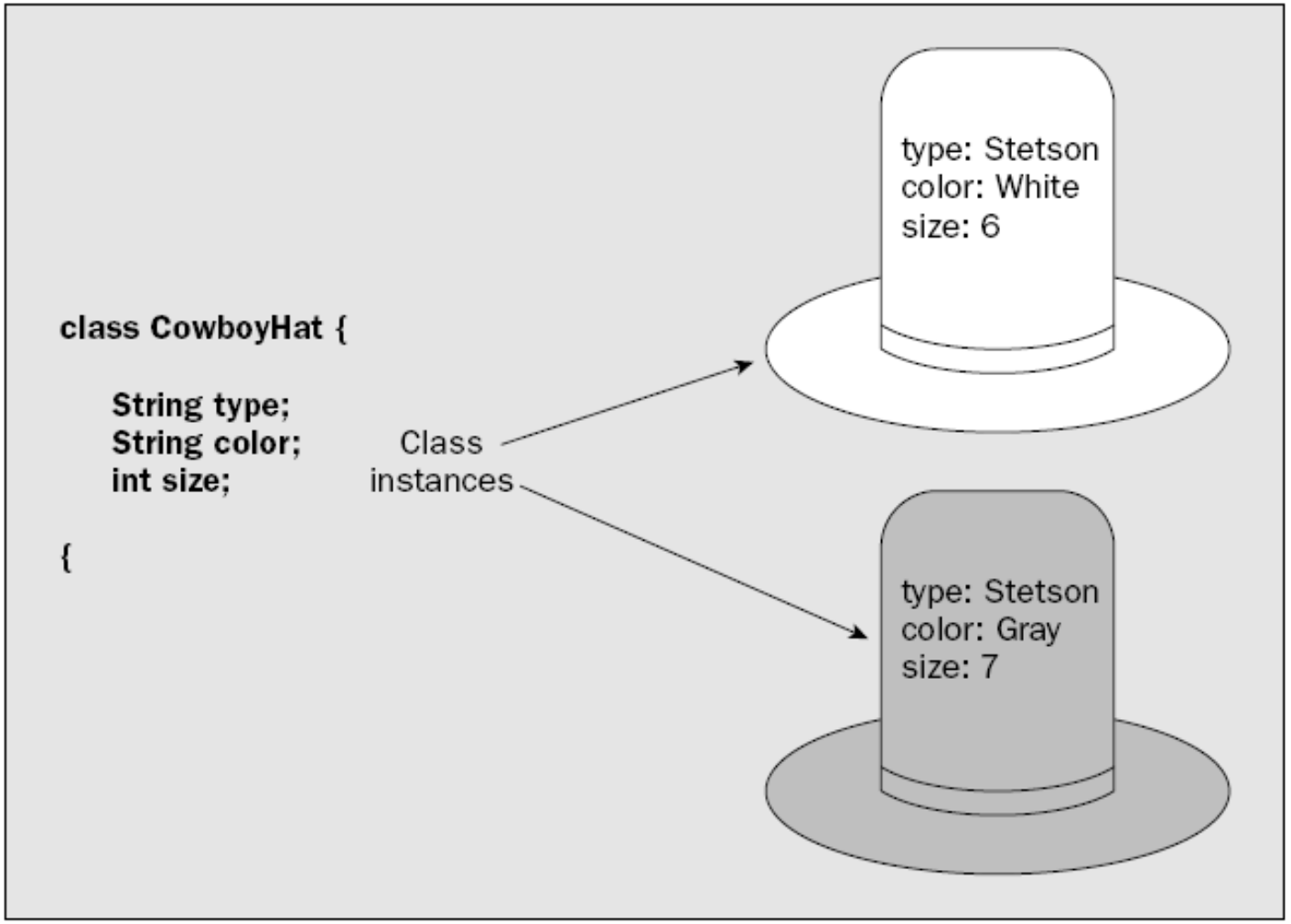
# Introducción a Programación Orientada a Objetos (OOP): Clases y Objetos

# Definición

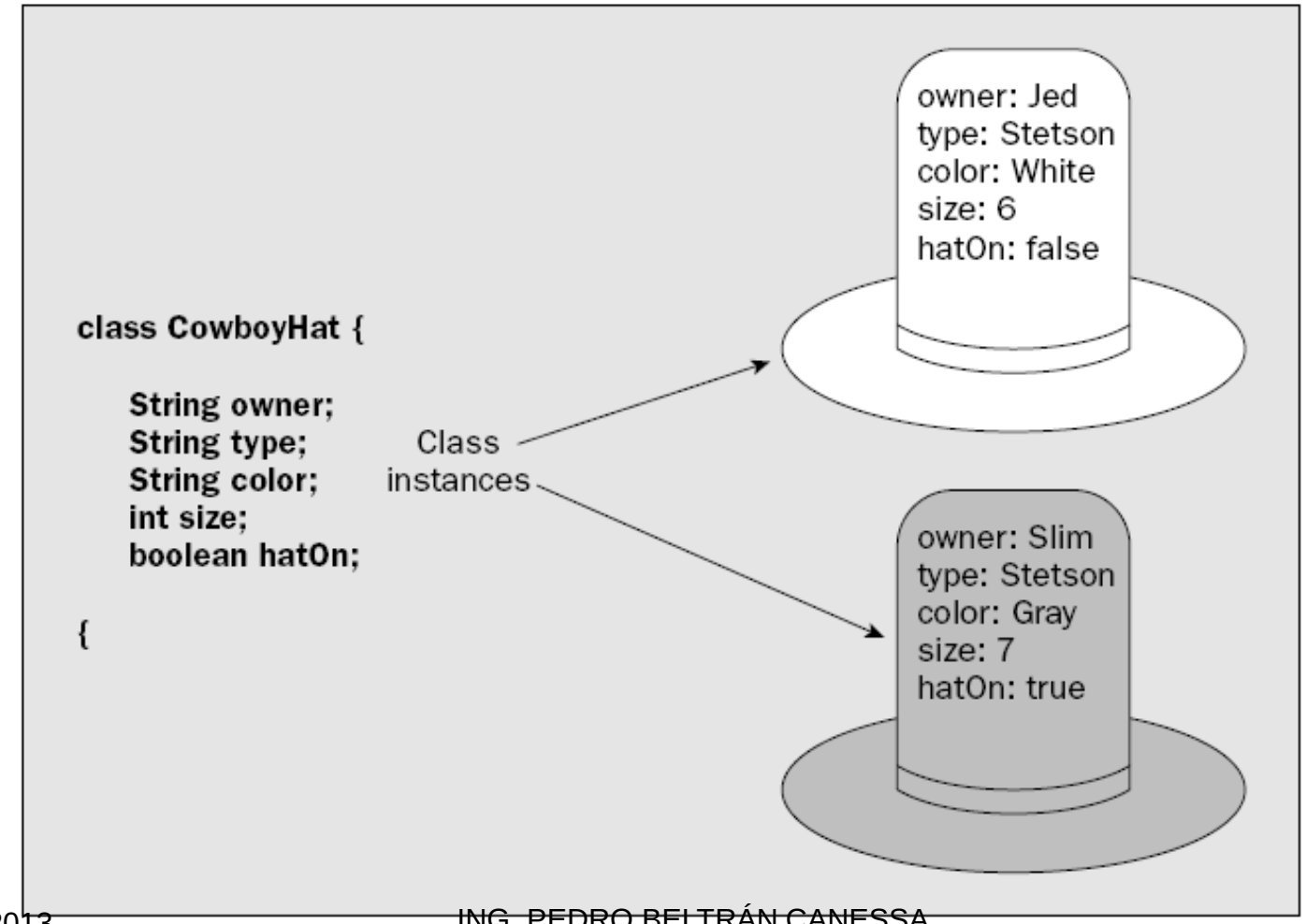
- Clase es la representación abstracta (modelo) de una entidad del mundo
- La clase especifica los atributos (propiedades) y métodos (comportamientos)
- El objeto es una Instancia de un  Clase (representación real de la clase)
- A los atributos del objeto se les asigna un valor.


Generic Tree
derived from
derived from
derived from
Redwood
Oak
Ash
Create instance
Create instance
myAsh
yourAsh
Objects of type Ash
Objects of a class Tree will have a given set of properties in common. Each object of the class will have its own values for these properties.

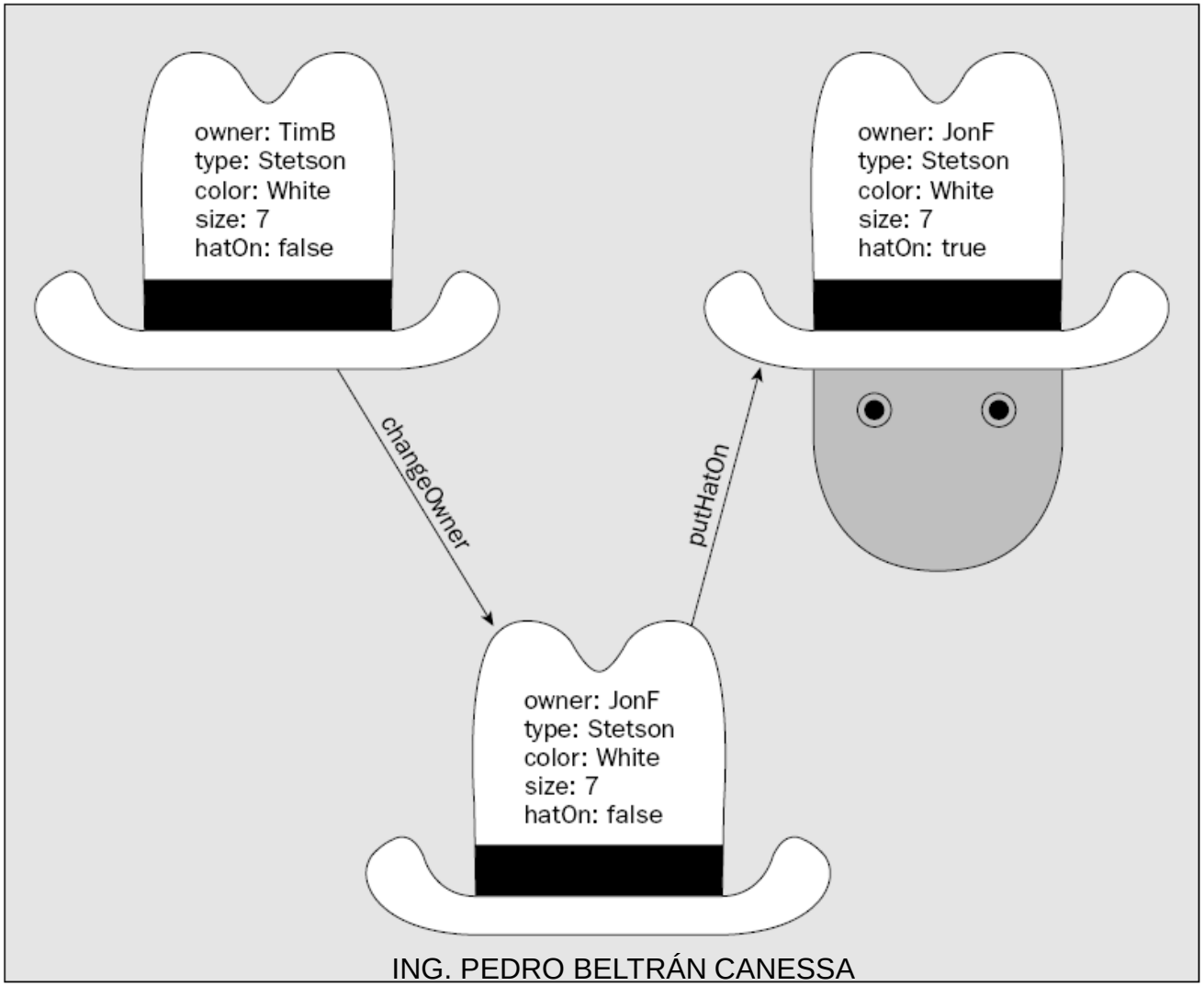
# Atributos


class CowboyHat {
String type;
String color;
int size;
{
Class
instances
type: Stetson
color: White
size: 6
type: Stetson
color: Gray
size: 7

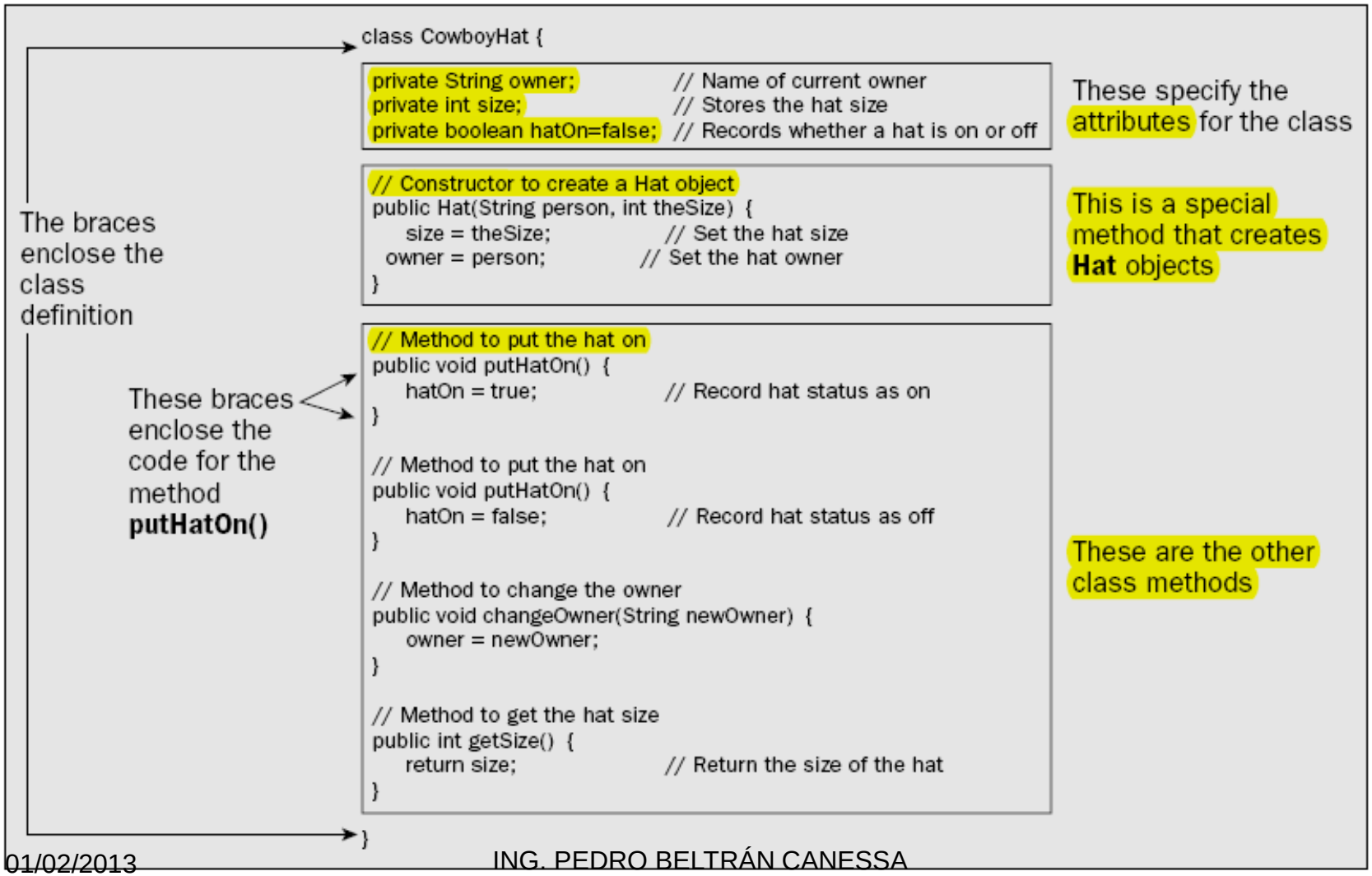
# Atributos


class CowboyHat {
String owner;
String type;
String color;
int size;
boolean hatOn;
{
Class
instances
owner: Jed
type: Stetson
color: White
size: 6
hatOn: false
owner: Slim
type: Stetson
color: Gray
size: 7
hatOn: true

# Comportamiento



# Ejemplo de Clase (Java)



# Características Fundamentales

- Un objeto no es un dato simple, sino que puede contener en su interior cierto número de atributos bien estructurados.
- Cada objeto no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo

# Estructura

- Un objeto puede considerarse como una especie de cápsula dividida en tres partes:

- PROPIEDADES
- METODOS
- RELACIONES

# Propiedades

- Las **propiedades** distinguen un objeto determinado de los restantes que forman parte de la misma organización y tiene valores que dependen de la propiedad de que se trate. Las propiedades de un objeto pueden ser heredadas a sus descendientes en la organización (a veces llamados **atributos**)

# Propiedades (Cont.)

- La diferencia con las "variables" de la programación estructurada es que las propiedades se pueden **heredar** de unos objetos a otros. En consecuencia, un objeto puede tener una propiedad de maneras diferentes:
- **Propiedades propias**. Asociadas directamente al objeto.
- **Propiedades heredadas**. Están definidas en un objeto diferente, antepasado de éste (padre,"abuelo", etc.). A veces estas propiedades se llaman propiedades miembro porque el objeto las posee por el simple hecho de ser miembro de una clase.

# Métodos

- Los **métodos** son las operaciones que pueden realizarse sobre el objeto, que normalmente estarán incorporados en forma de programas (código) que el objeto es capaz de ejecutar y que también pone a disposición de sus descendientes a través de la herencia. Los objetos se comunican con el exterior por medio de mensajes (Métodos) estos establecen su ‘interfaz’ para el mundo

# Métodos (Tipos)

- **Métodos propios**. Están incluidos dentro de la cápsula del objeto.
- **Métodos heredados**. Están definidos en un objeto diferente, antepasado de éste (padre,"abuelo", etc.). A veces estos métodos se llaman métodos miembro porque el objeto los posee por el simple hecho de ser miembro de una clase.

# Relaciones

- Las relaciones permiten que el objeto se inserte en la organización y están formadas esencialmente por punteros a otros objetos.
  - Es un
  - Parte de
  - Tiene un

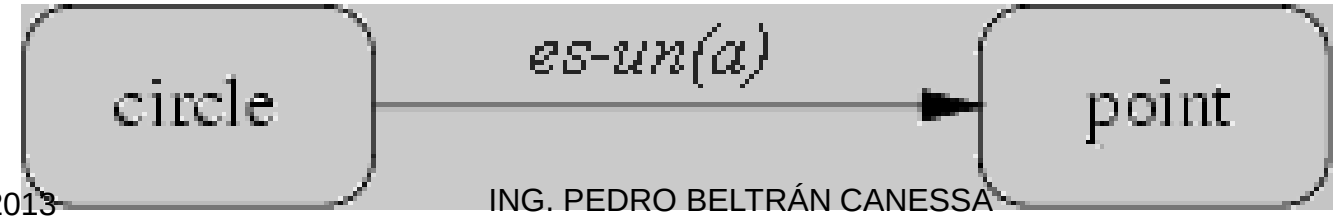
# Relación Es un..

- Relación de especialización
- “Un objeto contiene la definición de otro y añade comportamientos adicionales”

# Ejemplo

```
class Point {
  attributes:
    int x, y
  methods:
    setX(int newX)
    getX()
    setY(int newY)
    getY()
  }
```

```
class Circle {
 attributes:
   int x, y,
   radius
 methods:
   setX(int newX)
   getX()
   setY(int newY)
   getY()
   setRadius(newRadius)
   getRadius() }
```



# Relación Parte de

- Objetos compuestos por otros, relación de contenencia
- un objeto es parte de otro cuando conserva su independencia.

# Ejemplo

class Logo {

attributes:

**Circle circle**

**Triangle triangle**

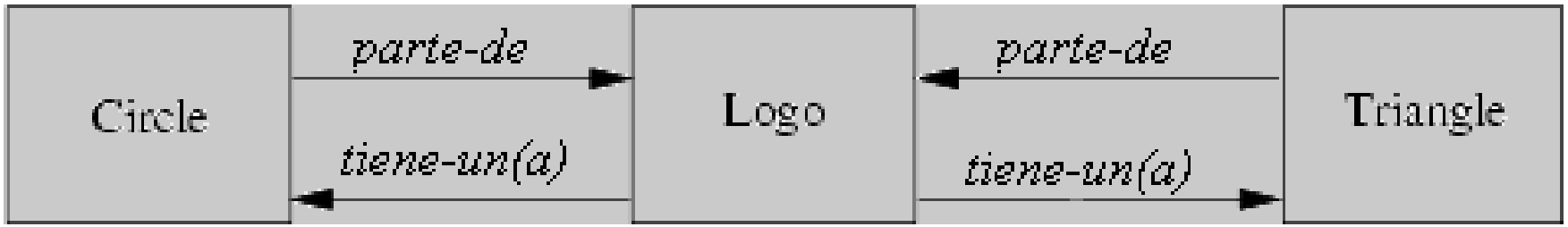
methods:

set(Point where)

}

# Relación Tiene un

- Esta relación es justamente la inversa de la relación parte-de



# Encapsulamiento

- Los objetos son inaccesibles, e impiden que otros objetos, los usuarios, o incluso los programadores conozcan cómo está distribuida la información o qué información hay disponible. Esta propiedad de los objetos se denomina **ocultación de la información**.

# Encapsulamiento (cont.)

- La idea del encapsulamiento es garantizar independencia entre la forma de hacer las cosas y los servicios que se ofrecen al exterior, de esta forma se  garantiza el intercambio de componentes sin afectar el comportamiento de un sistema

# Herencia

- Es un tipo especial de relación (es un) donde un objeto comparte  la definición de otro y especializa su comportamiento
- *Es el mecanismo que permite que un clase A herede propiedades de una clase B. Decimos "A hereda de B". Objetos de la clase A tienen así acceso a los atributos y métodos de la clase B sin necesidad de redefinirlos*

# Herencia

```
class Circle extends Point {
 // atrributes:
   int radius
 // methods:
   setRadius(int newRadius)
   getRadius()
 }
```

# Superclase/Subclase

- Si la clase A hereda de la clase B, entonces B es la **superclase** de A. A es **subclase** de B.

# Polimorfismo

- El polimorfismo no es otra cosa que la posibilidad de construir varios métodos con el mismo nombre, pero con relación a la clase a la que pertenece cada uno, con comportamientos diferentes. Esto conlleva la habilidad de enviar un mismo mensaje a objetos de clases diferentes. Estos objetos recibirían el mismo mensaje global pero responderían a él de formas diferentes