Clases, Objetos y Paquetes

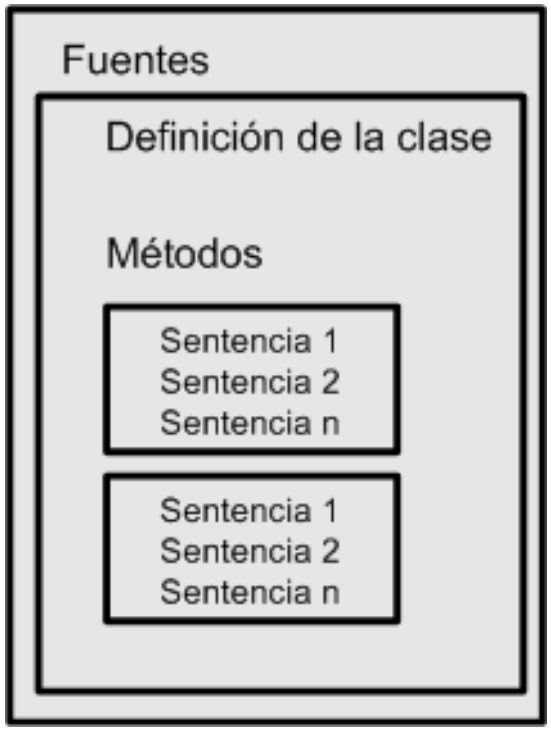
Objetos 2 - 2011

Contenido

- Definición de clases nuevas:
  - Declaración
  - Constructores
  - Variables
  - Métodos

Clases

- Cada clase, excepto la clase Object, es una extensión (subclase), de una sola clase ya existente (herencia simple).
- En Java, una clase se compone de:
  - Declaración
  - Cuerpo



Clases

- Forma general de declaración
  - Si una clase no declara explícitamente su superclase, entonces se asume que extiende a la clase Object.

```
package nombrePaquete;

// importaciones

[modificadores] class NombreClase
                [extends NombreSuperClase]
                [implements NombresInterfaces] {

// cuerpo

}
```

Clases - Paquete

- Un paquete organiza clases e interfaces detrás de un espacio de nombres. Por lo tanto, está formado por clases e interfaces.

- **`nombrePaquete`**:

  - Todo nombre de paquete debe, por convención, comenzar con una letra minúscula.

  - Java utiliza caracteres Unicode: **model, cliente.esquema**, **araña**, **vistaCliente2** son nombres de paquetes válidos.

  - El alcance de un identificador de paquete es todo el paquete en donde se declara, por lo tanto no puede haber dos paquetes con el mismo nombre dentro de un mismo paquete.

Clases – Declaración

[modificadores] class NombreClase [extends SuperClase]
  [implementes i1,i2] {}

Ejemplos

public abstract class Persona{}

class Auto implements Brakeable{}

public final class DoubleStack implements Stackable{}

Clases - Declaración

- **`Modificadores`**:
  - Existen dos tipos de modificadores:
    - De clase:
      - **`abstract`**
      - **`final`**
    - De accesibilidad:
      - **`public`**
  - Son opcionales
  - Se pueden combinar, salvo **abstract** y **final**.

Clases

- **`NombreClase`**:
  - El nombre debe comenzar con mayúsculas, por convención.
  - Java utiliza caracteres Unicode: Persona, Pequeño y AutoDobleTracción son válidos.
  - El alcance de un identificador de clase es todo el paquete en donde se declara la clase, por lo tanto no puede haber dos clases con el mismo nombre dentro de un mismo paquete.
  - Si la clase es pública, el nombre de la clase debe concordar con el nombre del archivo: Persona -> Persona.java.

Clases – Cuerpo

- El cuerpo de una clase esta delimitado por los signos { y }.
- En el cuerpo se declaran:
  - Atributos
  - Constructores
  - Métodos
  - Otras clases
- El orden de los elementos declarados no está pre-establecido. Por convención se mantiene el orden propuesto.

Clases - Ejemplos

```
package muebles;

abstract class Mueble {
...
}
```

```
package muebles;

public class Mesa
       extends Mueble {
...
}
```

```
package cliente;

public final class Conexión
{
...
}
```

```
package servidor;

public abstract class
Estado{
...
}
```

Clases - Constructores

- Necesarios para definir la instanciación de una clase. Contienen el comportamiento de inicialización de las instancias de una clase.
- Deben llamarse igual que la clase que los define.
- No heredan.
- No son métodos.

```
package nombrePaquete;

public class NombreClase {

   [accesibilidad] NombreClase([parámetros])
                                   [throws excepciones] {
     // cuerpo
   }
}
```

Clases - Constructores

- Constructores, accesibilidad:
  - Private
  - Package
  - Protected
  - Public

```
public class MesaAlgarrobo extends Mesa {
          public MesaAlgarrobo() {
          }
        }
abstract class Mesa {
          Mesa() {
          }
        }
```

Clases – Constructores por defecto

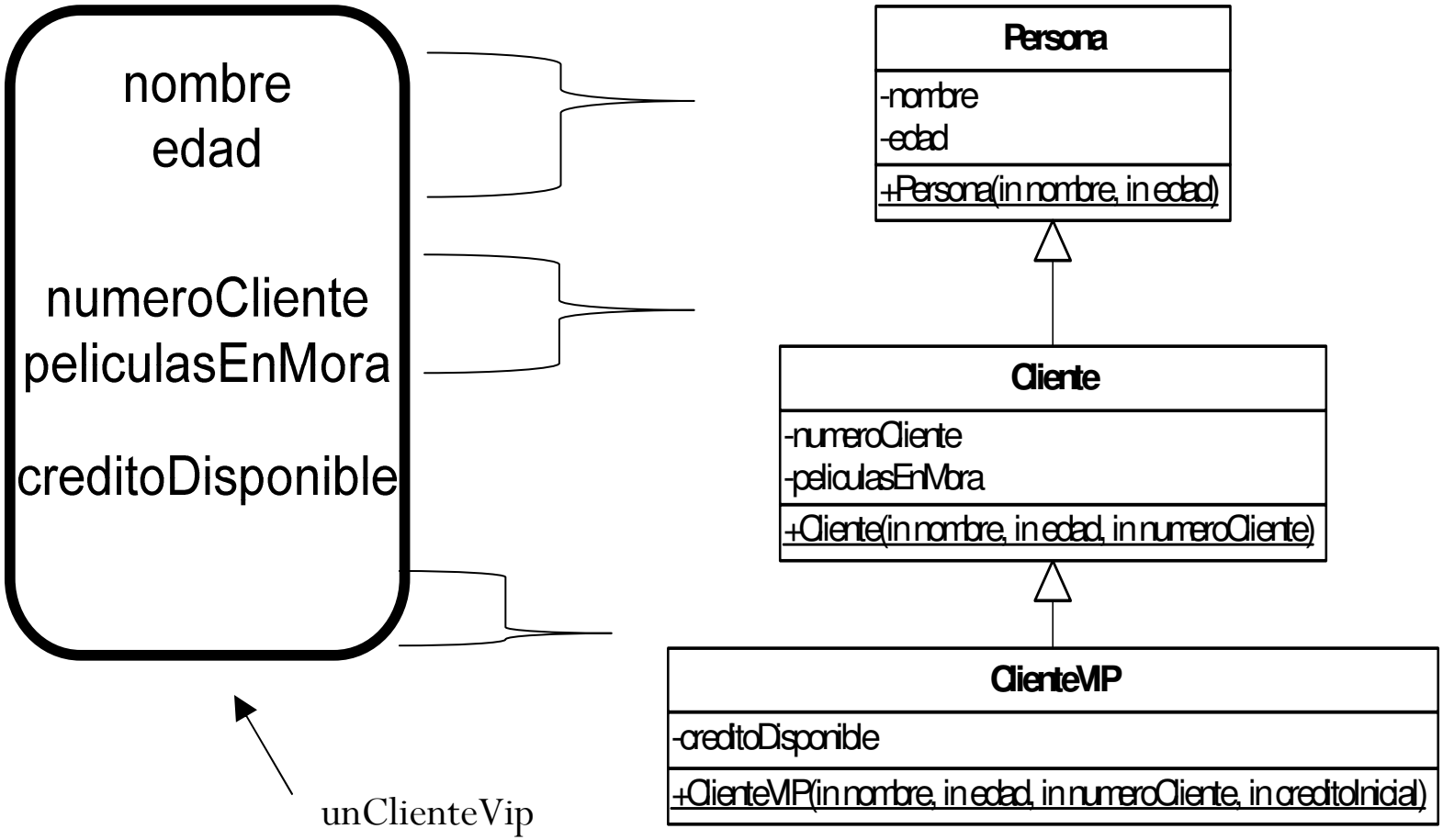
- Si una clase no declara ningún constructor, entonces tiene un constructor por defecto.

```
public class Persona{
}
```

=

```
public class Persona{
  public Persona() {
    super();
  }
}
```

Constructores



Constructores

- Cadena de llamado en jerarquías de clase

```
public class Figura{
  public Círculo(Punto origen) {
    // cuerpo
  }
}
```

```
public class Círculo extends Figura{
  public Círculo(Punto origen) {
    super(origen);
    // cuerpo
  }
}
```

Clases - Constructores

- Una clase puede tener varios constructores (overloading).
- Si no se indica, la primer línea de un constructor es "super();"
- Los diferentes constructores se diferencian por el número, tipo y orden de los parámetros.
- Los constructores tienen el mismo manejo de excepciones que los métodos.

```
public class Círculo {

  protected Círculo(Punto
punto) {
     ...
  }

  public Círculo(Punto punto,
                 Color color)
{
    this(punto);
     ...
  }
}
```

Clases - Variables

- Todas las variables tienen un tipo. El tipo puede ser:
  - Tipo primitivo
  - Clase
  - Interfaz

  - Ejemplos de declaraciones

```
[accesibilidad] [modificadores] tipo nombre [= valor];
```

```
[final] tipo nombre [= valor];
```

Clases - Variables

- Accesibilidad:
  - Private, package, protected, public
  - Importante pensar en POO

```
public class Empleado extends Persona {
  private int cantidadDeHijos;
}
```

Modificadores en las Variables

- **`static`**: variables de clase. Existen desde que el ClassLoader carga la clase. Tiene el mismo alcance que la clase y vive toda la ejecuciòn de la JVM.

- **`final`**: constantes.

- **`transient`**: no persistentes.

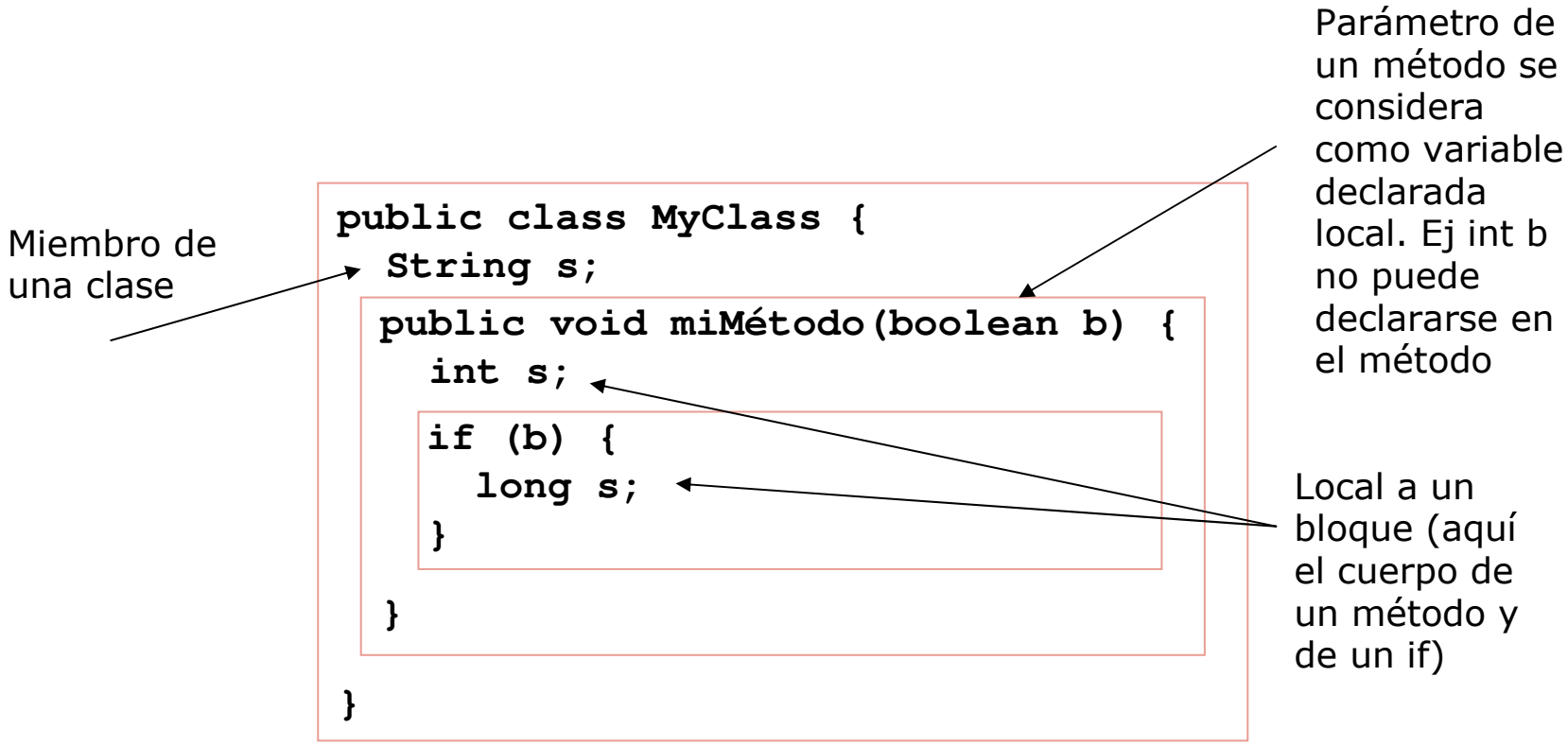
- **`volatile`**: indica a la JVM que la variable puede ser modificada en forma asincrónica por cualquier thread.

Nombres de las Variables

- Compuesto de caracteres Unicode.
- Por convención los nombres de las variables empiezan con minúscula.
- Como en las clases, si se juntan varias palabras, al principio de cada una se coloca mayúscula.
- Sensible a mayúsculas y minúsculas (Case Sensitive).

Variables - Visibilidad

- Visibilidad: bloque de código en donde es accesible la variable
- Hay 3 categorías:



Variables - Inicialización

- Las variables locales y las miembro de una clase pueden inicializarse con un valor en su declaración

```
int count = 0;

int count = 'a';
```

  - Las variables parámetro de un método **NO** pueden ser inicializadas.

Variables - Constantes

- Para reflexionar
  - ¿Qué particularidad tiene la siguiente inicialización?

```
public final String CONSTANTE = "Constante";
```

```
public static final String CONSTANTE = "Constante";
```

Clases

- Ejercicio:

  Cierto banco posee dos tipos de cuentas bancarias, Caja de Ahorro y Cuenta corriente. Ambas poseen un número de cuenta, un saldo y se les pueden realizar depósitos y extracciones. Sin embargo en la Caja de Ahorro no se puede tener saldo negativo, mientras que sobre las Cuentas Corrientes se pueden realizar hasta 30 extracciones sin importar el importe.

  Además, cada cuenta bancaria posee un titular del que se conoce nombre y DNI.

  Implementar las clases necesarias, con sus atributos y métodos.

Métodos

Declaración: donde se declaran modificadores, tipo de retorno, nombre, parámetros y posibles excepciones que dispara.

Cuerpo: variables locales y sentencias.

Métodos - Declaración

[accesibilidad] [modificadores] tipoDeRetorno nombreDelMétodo ([lista, de, parámetros] [throws excepciones]

Ejemplos:

- `public boolean isEmpty(){}`
- `void setSize(int x, int y) {}`

Métodos – Accesibilidad y Modificadores

- Acceso
  - Private
  - Package
  - Protected
  - Public
- Retorno
  - Void
  - Tipo definido

- Modificadores:
  - **`abstract`**:
    - Un método abstracto no tiene implementación.
    - Debe ser miembro de una clase abstracta.
  - **`static`**:
    - Declara al método como método de clase.
  - **`final`**:
    - El método no puede ser redefinido por las subclases.
  - **`native`**:
    - El método está implementado en otro lenguaje.
  - **`synchronized`**:
    - Permite que múltiples objetos invoquen el mismo método con exclusión mutua.

Métodos - Nombres

- Codificación UNICODE.
- Los nombres de los métodos empiezan con minúscula por convención.
- Pueden existir múltiples métodos con el mismo nombre, pero no con la misma signatura (retornos, nombre, parámetros).
- Se recomienda el uso de getters y setters:
  - getX()
  - setX()

Clases

- Métodos, lista de parámetros:
  - Es una lista delimitada por coma de la forma "**`tipo parámetro`**".
  - Los tipos primitivos se pasan por valor.
  - Los objetos usan pasaje de referencias por valor.
  - No se puede declarar una variable dentro de un método con el mismo nombre que un parámetro.
  - Ejemplo:
    - **`void unMétodo(int x, int y, String s)`**

Métodos - Cuerpo

- Se considera como cuerpo todo lo encerrado entre "**{**" y "**}**".

- Las variables locales enmascaran a las variables miembro de la clase.

- El tiempo de vida de una variable declarada dentro de un método, esta limitado a la vida de éste.
- this: se refiere al ojbeto receptor del mensaje
- super: se refiere a la super clase del objeto actual.
- return: finaliza la ejecución y retorna el objeto a su derecha.

Paquetes

- Un paquete es un conjunto de clases e interfaces relacionadas que proveen acceso protegido y administración de nombres.

- Las clases e interfaces que son parte del lenguaje están agrupadas en paquetes de acuerdo a su función:
  - **java.lang**: Clases del lenguaje. Se importa por defecto.
  - **java.io**: para manejo de Entrada/Salida

- El programador agrupa sus clases e interfaces en paquetes, anteponiendo la cláusula **`package`** `nombreDelPaquete;` a las declaraciones de todas las clases e interfaces agrupadas.

Paquetes

- Un paquete es creado simplemente incorporando una clase o una interfaz.
- Se requiere escribir la sentencia **`package`** como primer sentencia del archivo fuente en donde se está definiendo la clase o la interfaz.

```
package graphics;
class Circle extends Graphic
       implementes Draggable {
  ...
}
```

```
package graphics;
class Rectangle extends Graphic
          implementes Draggable {
  ...
}
```

Paquetes

- Beneficios de utilizar paquetes:
  - Reconocer y encontrar fácilmente las clases e interfaces relacionadas.
  - Evitar conflictos de nombres, ya que cada clase pertenece a un único paquete.
  - Controlar el acceso a las clases del paquete.
  - Si no se usa la sentencia **`package`**, las clases e interfaces se ubican en el paquete por defecto (default package), que es un paquete sin nombre.

Paquetes

- Las clases e interfaces miembros de un paquete declarados públicos pueden ser accedidos desde afuera del paquete, de alguna de las siguientes formas:
  - Refiriéndose a su nombre largo:

```
graphics.Rectangle miRect = new graphics.Rectangle();
```

  - Importándolo:

```
import graphics.Circle;
Circle miCir = new Circle();
```

  - Importando el paquete íntegro:

```
import graphics.*;
Circle miCir = new Circle();
Rectangle miRec = new Rectangle();
```

Ciclo de vida de un objeto

- Su creación
- Su Uso
- Su Destrucción.

Creación

Utilizando los constructores y la palabra clave new.

```
Rectangle rect = new Rectangle();
```

- Esta sentencia realiza:
  - Declaración
  - Instanciación
  - Inicialización

Instanciación

- se aloca memoria para el objeto mediante el operador new.
- Un constructor es llamado con el operador **new**.

```
new Rectangle();
Rectangle rectOne = new Rectangle(originOne, 100, 200);
```

Utilización

- Se puede usar un objeto de dos formas:
  - Manipulando directamente las variables
  - Invocando métodos

- Invocación de métodos, acceso a variables y constantes:
  - En clases:

```
Movible.ORIGEN;
Math.abs(-1);
```

  - En instancias,

```
rect.width();//es mejor utilizar getter y setter
new Point(10,10).x;
```

Finalización

- Finalización:
  - Conceptualmente existen dos variantes:
    - El usuario se encarga de liberar los recursos (Delphi).
    - El sistema es el encargado de liberar los recursos (Smalltalk, Java).

Finalización

- Finalización:
  - Las referencias mantenidas en variables locales son liberadas cuando salen del alcance o cuando se les asigna **`null`**.
  - Los objetos son recolectados cuando no existen más referencias a ellos.
  - Antes de destruir un objeto el Garbage Collector llama al método **`finalize()`** de dicho objeto. Se utiliza para liberar recursos.