

Intro POO (V)

- Concepto de Herencia
 - Package

Recordando

- Definición de clases
- Instanciación de objetos
- Constructores
- Destrucción
- Operador This
- Ocultamiento de información

Ocultamiento de Información

- Cuando se crea una nueva clase en Java, se puede especificar el nivel de acceso que se quiere para:
 - las variables de instancia y
 - los métodos definidos en la clase
- Lo anterior, se puede hacer mediante los denominados modificadores de acceso.
- En el caso de Java y otro LDP estos se definen a través de los operadores
 - ➔ **public, protected, private.**

Ocultamiento de información

- **Public:** Cualquier clase puede acceder a las propiedades y métodos públicos.
- **Protected:** Sólo las clases heredadas y aquellas situadas en el mismo paquete pueden acceder a las propiedades y métodos protegidos.
- **Private:** Las variables y métodos privados sólo pueden ser accedidos desde dentro de la clase.

Herencia

¿Qué es Herencia en POO?

La herencia es una relación entre clases, en la que una clase comparte la estructura y/o comportamiento definidos en una (**herencia simple**) o más clases (**herencia múltiple**).

Grady Booch

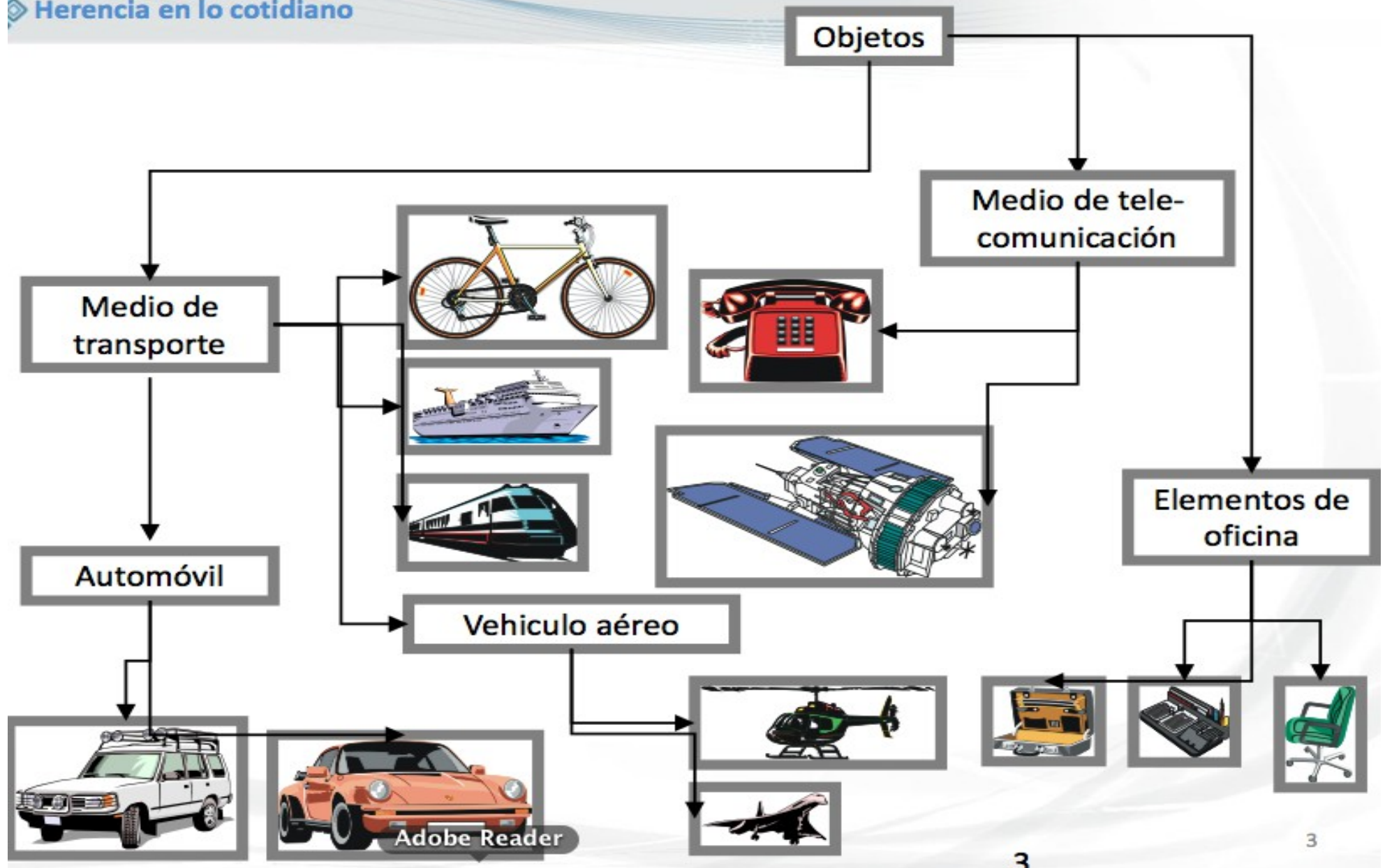
¿Quién es Grady Booch?

Herencia

- ☒ Cuando se crea una clase, la idea es NO volver a crearla más.
- ☒ Es posible reaprovechar el código haciendo uso de la herencia.
- ☒ Lo anterior puede realizarse:
 - ➔ Utilizando nuestra clase para crear nuevas clases a partir de ella en su definición.
 - ➔ Usarla para especializar una clase existente.

Herencia

Herencia en lo cotidiano



Herencia – características generales.

- Cuando heredamos de una clase existente, estamos **re-usando** código (métodos y atributos).
- Podemos agregar métodos y atributos para adaptar la clase a la nueva situación.
- La herencia la identificamos cuando encontramos la relación **es-un/es-de-tipo** entre la nueva clase y la ya existente. P. ej. un estudiante es una persona/estudiante es de tipo persona.
- La clase ya existente se es llamada **superclase, clase base o clase padre**.
- A la nueva clase se le llama **subclase, clase derivada, o clase hija**.

Herencia - composición

☠ Para definir una nueva clase

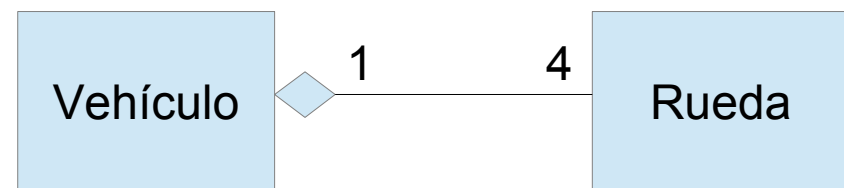
→ Sea una clase Rueda y una clase Vehículo

* Se sabe que un vehículo posee 4 instancias de tipo Rueda.

→ class Vehículo {
 Rueda r1,r2, r3, r4; ... }
}

☠ A esto se le llama una **relación de composición**, un Vehículo está compuesto por 4 ruedas.

Representación de la Composición en UML



Herencia - especialización

☠ Se tiene una clase y se desea crear algo de características similares pero con algunas extensiones o modificaciones que la hagan más particular.

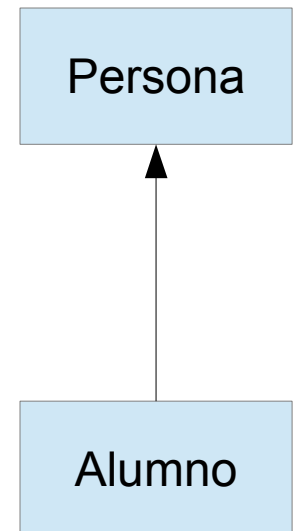
→ Se tiene la clase Persona

- Se sabe que todo Alumno es una Persona

→ class Alumno **extends** Persona { ... }

☠ A esto se le llama una **relación de especialización**

Representación de la Especialización en UML



Herencia - observaciones

- Conviene indicar que Java es un lenguaje de POO en el que todas las clases son heredadas, aún cuando no se indique explícitamente.
- Hay una jerarquía de objetos única, lo que significa que existe una clase de la cual son hijas todas las demás.
- Esta clase se llama **Object**, cuando no se indica que las clases hereden de nadie, heredan de ella.
- Esto permite que todas las clases tengan algunas cosas en común, lo que permite que funcione, entre otras cosas, el recolector de basura

Ver la clase **Object** en JavaDoc Oracle

<http://docs.oracle.com/javase/7/docs/api/java/util/Objects.html>

Herencia – implementación Java

- ❌ `class cHija extends cPadre { . . . }`
- ❌ Java no soporta la herencia múltiple.
- ❌ **Regla 1:** Una subclase hereda todos los atributos de su superclase que puedan ser accesibles desde la subclase.
- ❌ **Regla 2:** Una subclase hereda todos los métodos de sus superclase que son accesibles para la subclase.

Herencia – implementación Java

☒ **Las subclases y los atributos heredados:**

- **heredan** aquellos atributos declarados como **public** o **protected**.
- **heredan** aquellos atributos declarados sin especificador de acceso, siempre que la subclase esté en el mismo paquete que la clase.
- **no hereda** atributos de la superclase si la subclase declara un atributo que utiliza el mismo nombre. El atributo de la subclase se dice que oculta a la variable miembro de la superclase.
- **no hereda** los atributos de tipo **private**.

Herencia – implementación Java

☒ **Las subclases y los métodos heredados:**

- **heredan** aquellos métodos declarados como **public** o **protected**.
- **heredan** aquellos atributos declarados sin indicador de acceso, siempre que la subclase esté en el mismo paquete que la clase.
- **no heredan** métodos de la superclase si la subclase declara uno con el mismo nombre. El método de la subclase se dice que sobrescribe al método de la superclase.
- **no hereda** los métodos de tipo **private**.

Herencia – implementación Java

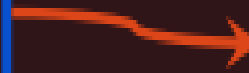
- ✖ Si el método oculta una de las variables miembro de la superclase, se puede referir a la variable oculta utilizando el operador **super**
- ✖ De igual forma, si el método sobrescribe uno de los métodos de la superclase, se puede llamar al método sobrescrito a través de **super**.

Herencia – implementación Java

- ☠ También puede ser usado para invocar desde una clase hija al constructor de la clase padre.
- ☠ Debe ser la primera instrucción usada en el constructor de la clase hija.

¿Qué sucede aquí?


```
class cPadre {  
    int numero;  
    cPadre() {  
        numero = 1; } }  
}
```



```
class cHija {  
    int otro;  
    cHija() {  
        super( );  
        otro=5;} }  
}
```


Herencia – implementación Java

```
class cPadre {  
    boolean unaVariable;  
    void unMetodo()  
    { unaVariable = true; } }
```



```
class cHija extends cPadre {  
    boolean unaVariable;  
    void unMetodo() {  
        unaVariable = false;  
        super.unMetodo( );  
        System.out.println(unaVariable);  
        System.out.println(super.unaVariable);  
    } }
```

¿Qué sucede en este código?
¿Qué se podría observar por pantalla?

Los Package

- ☒ El elemento fundamental de JAVA son las clases.
- ☒ Sin embargo, existe un elemento de mayor rango llamado **package**, que es un contenedor de clases, similar al concepto de bibliotecas de C / C++.
- ☒ Existen más que nada por comodidad y diseño, permiten agrupar varios archivos fuente *.java en un sólo paquete incluyendo una línea al comienzo de los mismos: **package net.programacion.ejemplos;**
- ☒ El código compilado (archivos con extensión *.class) se situarían en el directorio net/programacion/ejemplos.
- ☒ Pertenerían al paquete net.programacion.ejemplos, para poder utilizarlos desde otro programa se debería utilizar la sentencia **import**, que también se coloca al principio del programa (aunque después de package): **import net.programacion.net.*;**

Con esto podemos empezar a “independizar” nuestra clase pública de otras clases

Los Package - observaciones

- ☠ Hay que indicar que **import** es una palabra clave que indica las clases que se desean cargar, no los paquetes.
- ☠ Sin embargo, al admitir el uso del comodín *, se puede utilizar para cargar paquetes enteros.
 - Por ejemplo, si se desea utilizar la clase Date, situada en el paquete java.util, se puede cargar de dos maneras:

```
import java.util.Date;
```

```
import java.util.*;
```

Con esto podemos empezar a “independizar” nuestra clase pública de otras clases

Los Package - implementación

The image shows a Java IDE with two code editors. The left editor shows the file structure of a package named 'mios' inside 'usoPaquetes'. The right editor shows the implementation of 'pruebaPaquete.java', which imports the 'mios' package and uses its 'utiles' class. The bottom editor shows the implementation of 'utiles.java' within the 'mios' package.

File Path: ~/Desktop/ejPackage/usoPaquetes/mios/utiles.java

```
1 // *****
2 * Ejemplo de la creación de un paquete personalizado
3 * clase con utilitarios que serán llamados después desde
4 * otros programas mediante un import.
5 * Este package debe ser guardado en un directorio con el
6 * mismo nombre del package.
7 */
8
9 // se define el nombre del paquete
10 package mios;
11
12 public class utiles {
13
14     public int largo(String s) {
15         return s.length();
16     }
17
18     public boolean vacia(String s) {
19         return (s.length() == 0);
20     }
21
22     public boolean esPar(int n) {
23         return (n%2 == 0);
24     }
25 }
26
27
```

File Path: ~/Desktop/ejPackage/usoPaquetes/pruebaPaquete.java

```
1 // hacer un import de nuestro propio paquete
2 import mios.*;
3
4 public class pruebaPaquete {
5
6     /**
7      * @param args
8      */
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         String st = new String();
12         // se crea un objeto de la clase utiles
13         // de nuestro paquete mios
14         utiles u = new utiles();
15         int x = 3;
16         st = "hola";
17         // se invocan a operaciones de la clase
18         // utiles del paquete creado.
19         System.out.println(u.largo(st));
20         System.out.println(u.vacia(st));
21         System.out.println(u.esPar(x));
22     }
23
24 }
```

Resumiendo

- Herencia
 - Concepto
 - Características generales
 - Relación con el encapsulamiento
 - Operador super
 - Uso en Java
- Package
 - Concepto
 - Características
 - Uso en Java

A continuación resuelva el caso planteado en Campus Virtual, que requiere el uso de los conceptos aquí mencionados.

Próximos temas

- El método **toString()**
- Métodos **get...()** y **set...()**
- Operadores ***abstract*** y ***final*** para clases