Herencia

→ permite a clases con similitudes

superclase
↓
subclase → hereda atributos y variables de superclase.
→ también puede reemplazar o modificar comportamiento de superclase. (→ sobreescritura)
→ Reusabilidad del código.

- En java hay herencia simple, no puedo heredar de múltiples superclases.
- NO puedo acceder a var privadas de superclase desde subclase → solo desde misma clase.
  sí a una protected.

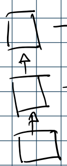
extends (public class alumno extends persona)

sobreescritura de métodos

Cuando subclase redefine lo q' tiene superclase.

Tiene q' tener = nombre, tipo de retorno y lista de args q-e un método de superclase.

public
↳ protected
↳ package
↳ private



→ public int detalles() no compila
→ int detalles() → tiene tiene q-e tener = o + visibilidad q' el de arriba

```java
public String detalles() {
  return "Vehiculo marca: "+ getMarca()+
  "\n" + "Precio: "+ getPrecio();
}

  . . .
}
```

```java
public class Camion extends Vehiculo {
  private boolean tieneDobleCaja;
  private int cargaMaxima;

  public String detalles() {
    return super.detalles()+ "\n"
    + "carga máxima:" + getCargaMaxima();

  }
}
```

} Mejor usar super

upcasting → convertir clase derivada en una superior. } Camión Ca = new Camión();
→ Es seguro porque ya es de ese tipo. } Vehiculo V = Ca

downcasting → peligroso → Tengo q' declararle explícitamente
Camion Ca2 = (Camion) v ;

Vehiculo vc = new Camión();

```java
Vehiculo vc = new Camion(); ✓
vc.setMarca("Mercedes Benz"); ✓
vc.setPrecio(35120.4); ✓
vc.setCargaMaxima(3000); ✗ No está visible para Vehiculo
System.out.println(vc.detalles());
```

} se ejecuta como camión
lo veo como vehículo.

¿Qué método se ejecuta?
El asociado con el objeto al que hace referencia la variable en ejecución, es decir, Camion. Esta característica se llama binding dinámico y es propio de los lenguajes OO

} Padre o hijo?

Clase object

- Clase x defecto de java.
- Todas las clases heredan de object. → por eso podemos sobreescribir equals() y toString()

equals(object)
f1.equals(f2) → se aplica upcasting a object a f2

- Compara como el ==
- Se fija si referencian (apuntan) a lo mismo.

Para sobreescribir equals hay que pasarle param object.

```java
public boolean equals(Object o){
  boolean result=false;
  if ((o!=null) && (o instanceof Fecha)){
   Fecha f=(Fecha)o;
    if ((f.getDia()==this.getDia())
       &&(f.getMes()==this.getMes())&&
       (f.getAño()==this.getAño())) result=true;
  }
  return result; }
```

instanceof para no hacer lío al hacer downcasting. Como le mando un objeto hay que tener cuidado de que no se le mande algo q' nada que ver y yo querer ponerlo en un ob. de tipo Fecha.

Ej. si a equals le mando un vehículo me va a explotar al intentar hacer Fecha f = (Fecha) o.

Clases abstractas

- Finalidad: No es crear instancias de algo de ese tipo.
- Encapsula el estado o comportamiento de varias

Métodos abstractos → Definidos en clases abstractas. Cada subclase define el comportamiento.

public abstract class. → Tiene que estar en todas las subclases.

Listas

- Puedo sacar y poner de cualquier pos
- Puede implementarse a través de → estructura estática → array
  → " dinámica → Nodos