Ciencias de la Computación
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS UNIVERSIDAD DE CHILE

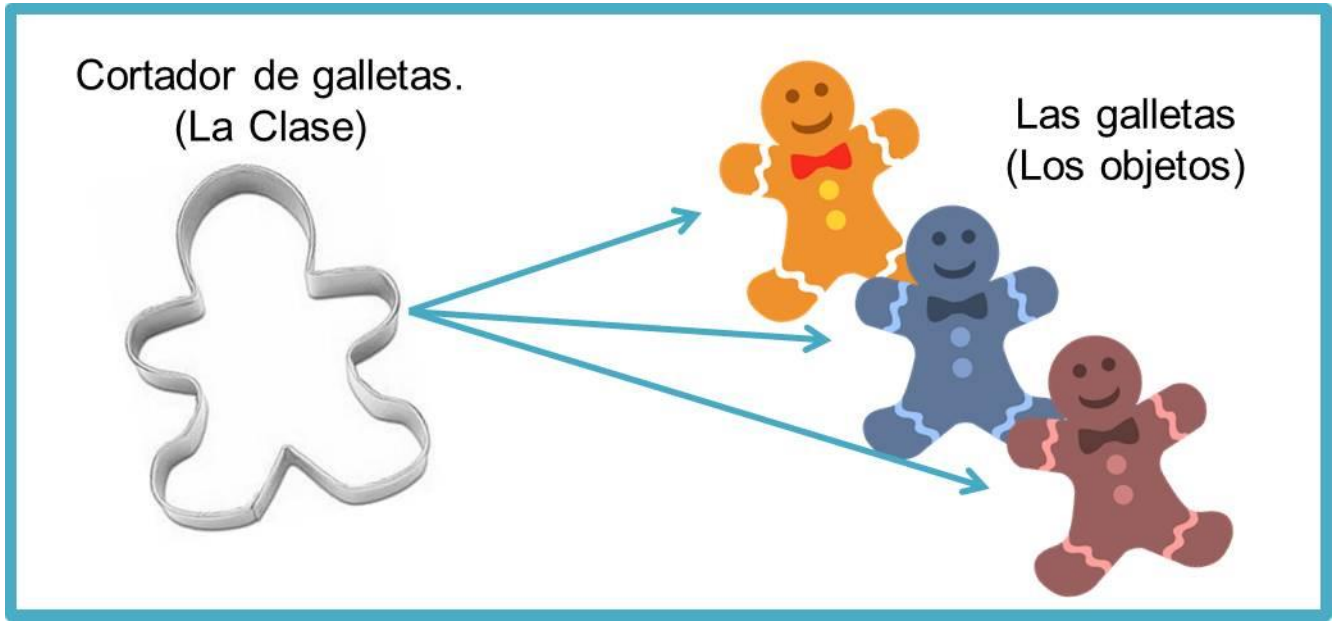
CC3002 – Metodologías de Diseño y Programación

# Auxiliar 1: Objetos y herencia

Juan-Pablo Silva
jpsilva@dcc.uchile.cl

# Clases y Objetos

- **Clase**: definición de los datos o procedimientos disponibles para un cierto tipo o “clase de objeto”
- **Objeto:** instancia de una clase



# Objetos

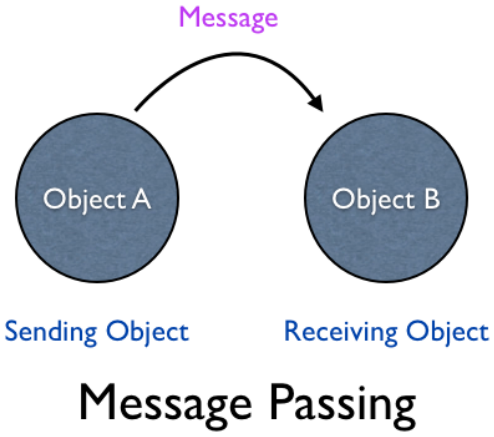
- Se comunican utilizando ***mensajes***
- *En java, los mensajes* ***siempre*** son entendidos por el objeto
- No se accede directamente a variables internas de otro objeto
  - Se pide al objeto con mensajes
- En java **casi** todo es un objeto. **¿Qué cosas no son?**

# Objetos

- Se comunican utilizando ***mensajes***
- *En java, los mensajes* ***siempre*** son entendidos por el objeto
- No se accede directamente a variables internas de otro objeto
  - Se pide al objeto con mensajes
- En java **casi** todo es un objeto. ¿Qué cosas no son?
  - Tipos primitivos (int, float, char, etc)

# Mensajes

- Objetos se pasan mensajes entre si
- Al recibir un mensaje, el objeto comienza a buscar el *método que invocar* para responder al mensaje (**Method Lookup**)
  - Pero es el objeto que recibe el mensaje quien ve cómo responder (el objeto que envió el mensaje no tiene acceso directo al objeto que recibe)
- Pseudo-variables que participan en el Method Lookup:
  - ***this***
  - ***super***
- Métodos estáticos **NO** participan del Method Lookup
  - Pertenecen a la *clase* no al *objeto*



# this y super (***IMPORTANTE***)

- this y super referencian al **mismo** objeto
  - ¿Cuál objeto?

# this y super (***IMPORTANTE***)

- this y super referencian al **mismo** objeto
  - ¿Cuál objeto? El que recibe el mensaje (llamada)

# this y super (***IMPORTANTE***)

- this y super referencian al **mismo** objeto
  - ¿Cuál objeto? El que recibe el mensaje (llamada)

**¿En qué se diferencian entonces?**

# this y super (***IMPORTANTE***)

- this y super referencian al **mismo** objeto
  - ¿Cuál objeto? El que recibe el mensaje (llamada)
- Se diferencian en la forma que comienza el *Method Lookup*

# this y super (***IMPORTANTE***)

- this y super referencian al **mismo** objeto
  - ¿Cuál objeto? El que recibe el mensaje (llamada)
- Se diferencian en la forma que comienza el *Method Lookup*
- Todo método que no tenga antepuesto *super*, tiene un *this* implícito
- La búsqueda en ambos casos sube por la jerarquía de clases en caso de no encontrar el método deseado, hasta llegar a Object

# this y super (***IMPORTANTE***)

- this y super referencian al **mismo** objeto
  - ¿Cuál objeto? El que recibe el mensaje (llamada)
- Se diferencian en la forma que comienza el *Method Lookup*
- Todo método que no tenga antepuesto *super*, tiene un *this* implícito
- La búsqueda en ambos casos sube por la jerarquía de clases en caso de no encontrar el método deseado, hasta llegar a Object
- *super* comienza el Method Lookup en… ¿Dónde?

# this y super (***IMPORTANTE***)

- this y super referencian al **mismo** objeto
  - ¿Cuál objeto? El que recibe el mensaje (llamada)
- Se diferencian en la forma que comienza el *Method Lookup*
- Todo método que no tenga antepuesto *super*, tiene un *this* implícito
- La búsqueda en ambos casos sube por la jerarquía de clases en caso de no encontrar el método deseado, hasta llegar a Object
- *super* comienza el Method Lookup en la superclase ¿De quién?

# this y super (***IMPORTANTE***)

- this y super referencian al **mismo** objeto
  - ¿Cuál objeto? El que recibe el mensaje (llamada)
- Se diferencian en la forma que comienza el *Method Lookup*
- Todo método que no tenga antepuesto *super*, tiene un *this* implícito
- La búsqueda en ambos casos sube por la jerarquía de clases en caso de no encontrar el método deseado, hasta llegar a Object
- *super* comienza el Method Lookup en la superclase de la clase que contiene la llamada a super

# this y super (***IMPORTANTE***)

- this y super referencian al **mismo** objeto
  - ¿Cuál objeto? El que recibe el mensaje (llamada)
- Se diferencian en la forma que comienza el *Method Lookup*
- Todo método que no tenga antepuesto *super*, tiene un *this* implícito
- La búsqueda en ambos casos sube por la jerarquía de clases en caso de no encontrar el método deseado, hasta llegar a Object
- *super* comienza el Method Lookup en la superclase de la clase que contiene la llamada a super

*no olvidar

# this y super (2)

- Reutilizar constructores

```
public class A {
    private final int a;
    private final int b;

    public A(int a, int b) {
        this.a = a;
        this.b = b;
    }

    public A() {
        this(0, 0);
    }
}
```

```
public class B extends A {
    public B(int a, int b) {
        super(a, b);
    }

    public B() {
        //super();
    }
}
```

- Llamadas implicitas a super()

# Object Oriented Programming (OOP)

- Encapsulation:
  - ocultar valores o estado de un objeto dentro de una clase.
  - partes no autorizadas no entran en contacto con estas partes
- Composition:
  - objetos contienen otros objetos.
  - relaciones "has-a" (bases de datos)
- Distribution of responsability:
  - patrones web en general, arquitecturas de software.
- Message passing (delegation):
  - cada método hace solo las cosas que le corresponde
  - cosas que no le corresponden las delega a quienes le correspondan.
- Inheritance:
  - relaciones "is-a-type-of“ (bases de datos)
  - Organizacion jerarquica.
  - Polimorfismo

# Polimorfismo (***IMPORTANTE***)

**La capacidad/habilidad de un tipo A, de verse y poder usarse como otro tipo B**

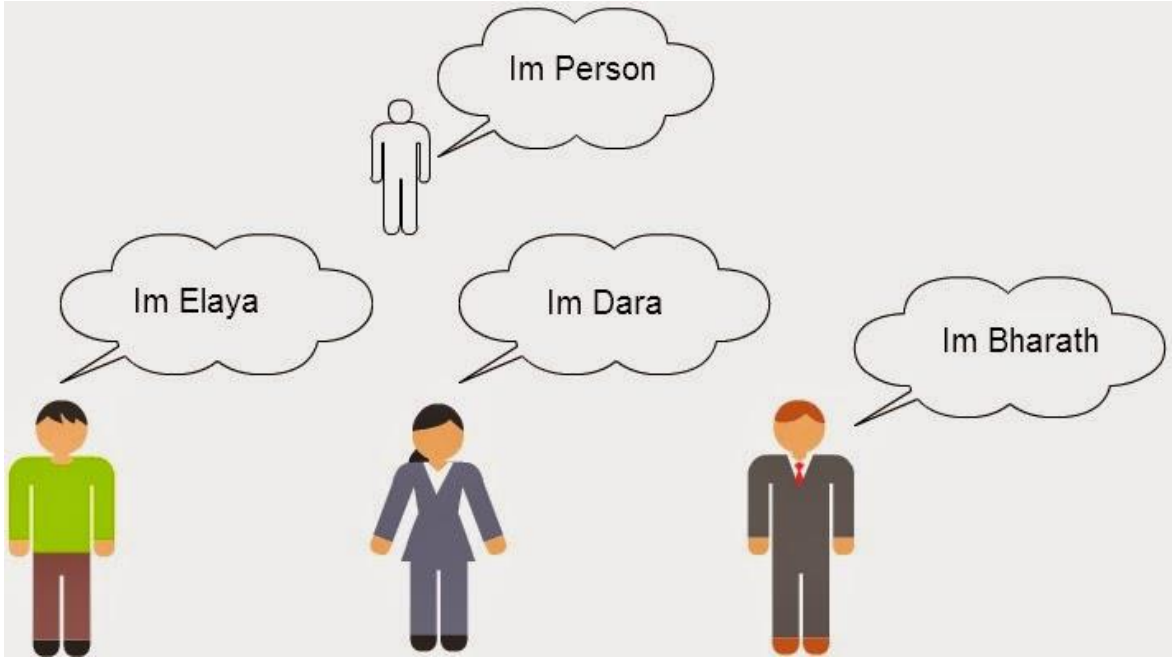
*No olvidar

# Herencia e interfaces

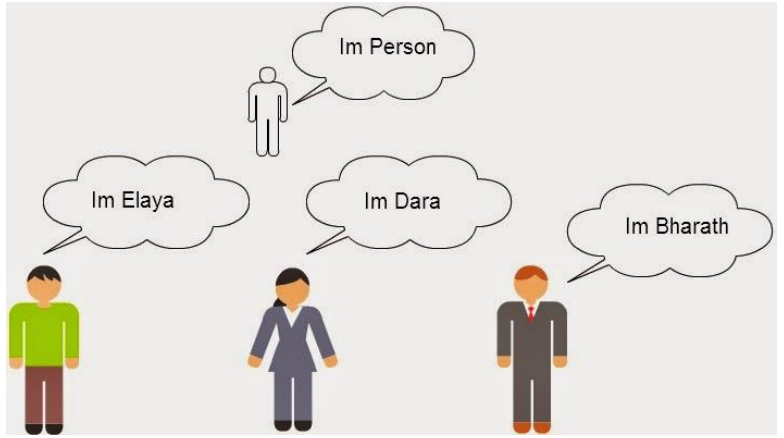
- En java solo se puede extender una clase (directamente), pero se pueden implementar múltiples interfaces
- La reutilización de código es una consecuencia de la herencia, **nunca** el fin de esta. NUNCA usar herencia ***solo*** para reutilizar código
- Los constructores NO se heredan

# Clases abstractas vs Interfaces

- Qué son:
  - CA) es una clase incompleta (no se puede instanciar)
  - I) es un contrato o garantía que doy a quien llame



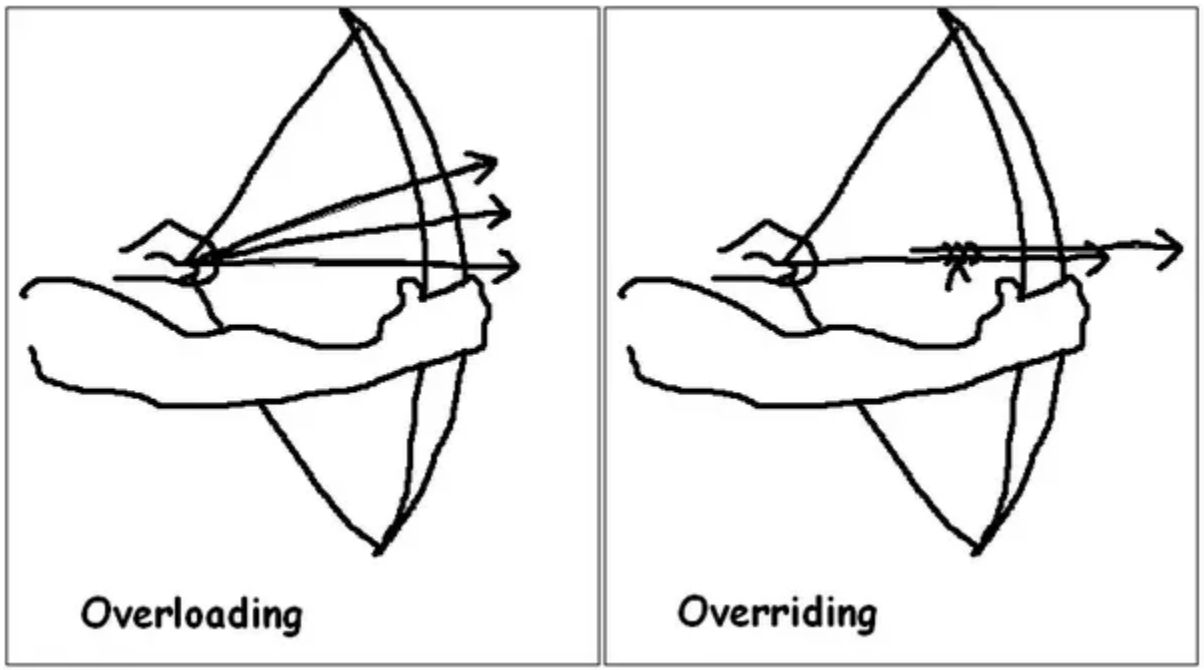
# Clases abstractas vs Interfaces




- Qué son:
  - CA) es una clase incompleta (no se puede instanciar)
  - I) es un contrato o garantía que doy a quien llame
- Cuándo se usan:
  - CA) abstraer el funcionamiento genérico de una o más clases
  - I) nuevos tipos (polimorfismo)
- ¿Debe implementar todos los métodos? (en caso de ser una clase)
  - CA) sí, si no debe ser declarada clase abstracta
  - I) sí, si no debe ser declarada clase abstracta

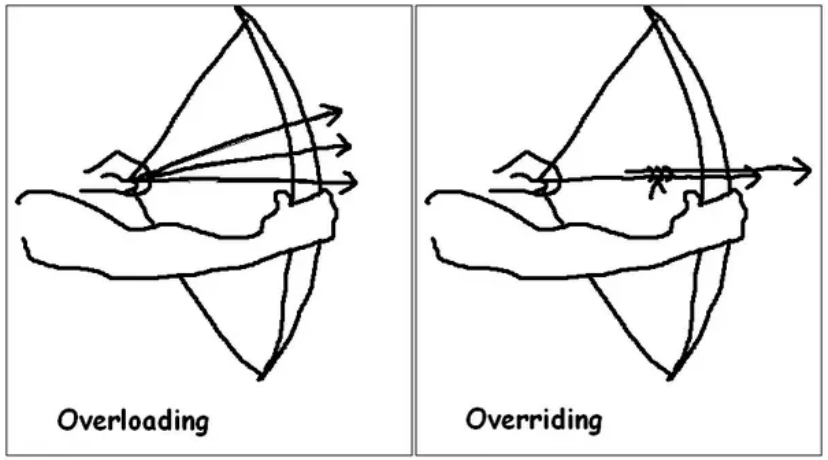
# Overloading y Overriding

# Overloading y Overriding



# Overloading y Overriding

- Overriding:
  - Métodos tienen la misma **firma**
  - Define un comportamiento especifico para la subclase
  - Sobrescribe la funcionalidad
- Overloading:
  - Permite tener más de un método con el mismo **nombre**
  - Común: overloading de constructores
  - Cuidado con los bugs



# Principio de Liskov


# Principio de Liskov

*Los subtipos deben poder ser sustituidos por sus clases padre*

# Problemas

Ciencias de la Computación
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS UNIVERSIDAD DE CHILE

CC3002 – Metodologías de Diseño y Programación

# Auxiliar 1: Objetos y herencia

Juan-Pablo Silva
jpsilva@dcc.uchile.cl