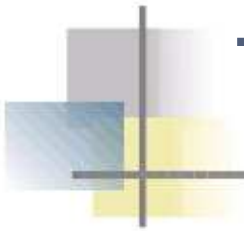
Programación en JAVA

Unidad 4
Herencia y Polimorfismo

Universidad de Chile
Departamento de Ciencias de la Computación
Profesor: Felipe Aguilera V.
faguiler@dcc.uchile.cl, felipe@aguilera.cl

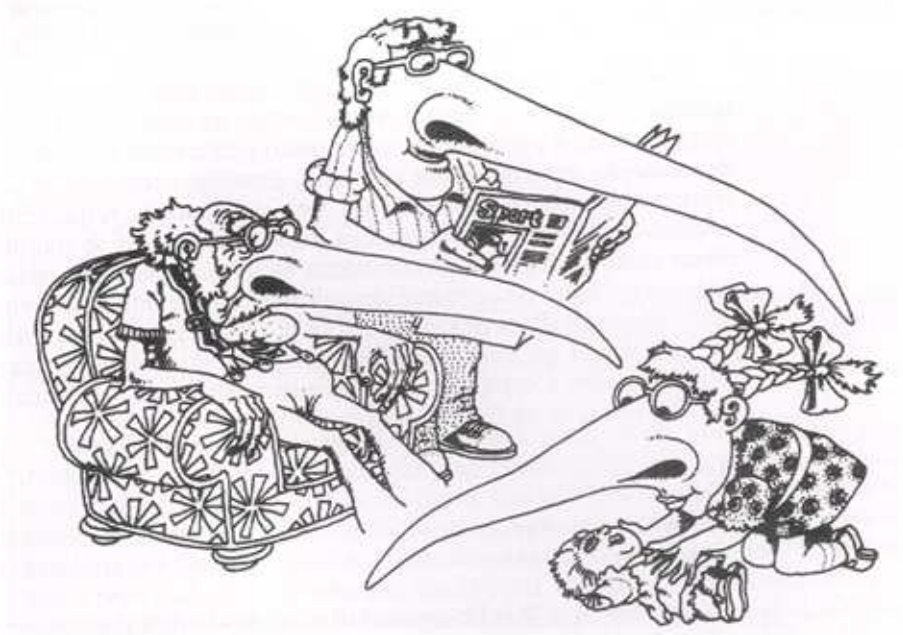


Temario

- Herencia
- Reescritura de métodos
- Polimorfismo
- Clases abstractas
- Interfaces

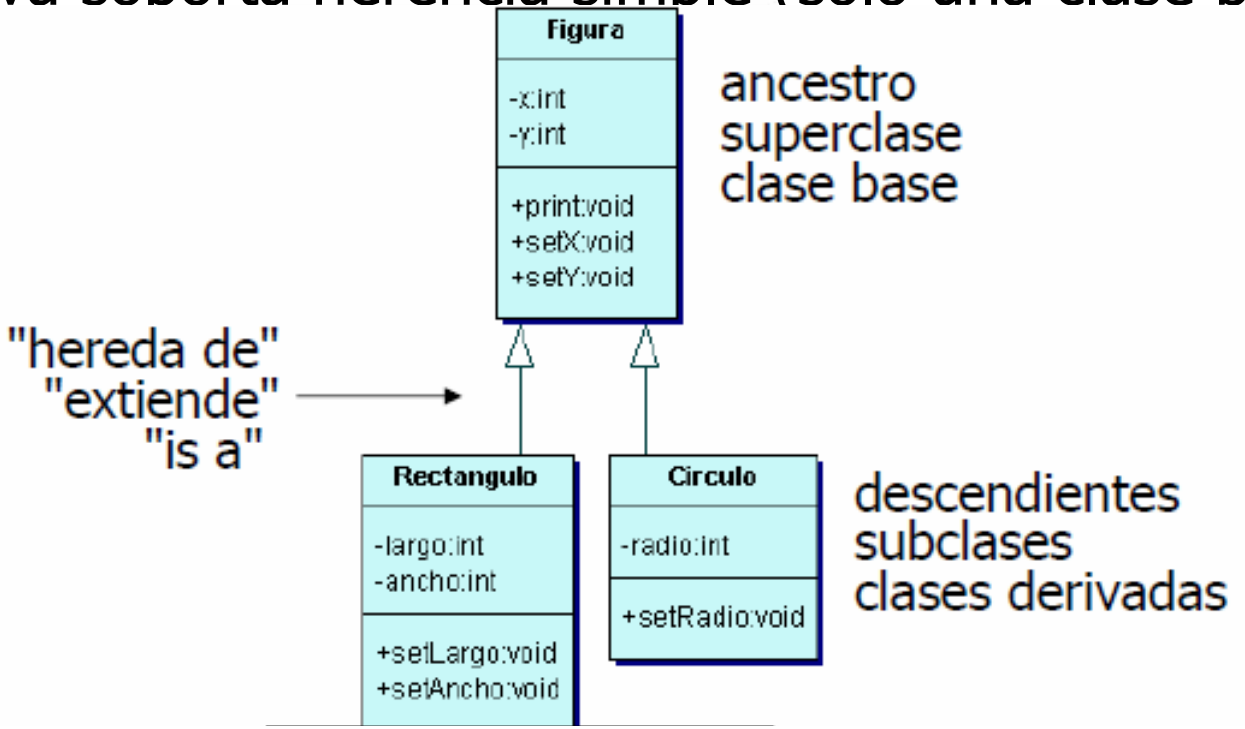
Herencia

- Concepto de OOP que permite que una clase herede los campos y métodos de otra
- Relación "Is a" entre clases
  - El descendiente "is a" tipo especial del ancestro
- Generalización/Especialización
  - Elementos comunes son generalizados en un ancestro
  - Particularidades son especializadas en un descendiente



Herencia en Java

- Todas las clases son descendientes de la clase Object
- La cláusula extends especifica el ancestro inmediato de la clase
- Una subclase o clase derivada hereda todos los campos y métodos de la superclase o clase base
- Java soporta herencia simple (sólo una clase base)



Ejemplo

```
class Figura {
    int x, y;
    public void print() { ... }
    public void setX(int x) { this.x = x; }
    public void setY(int y) { this.y = y; }
}

class Rectangulo extends Figura {
    int largo, ancho;
    public void setLargo(int largo) { this.largo = largo; }
    public void setAncho(int ancho) { this.ancho = ancho; }
}

class App {
    void f() {
        Rectangulo r = new Rectangulo();
        r.setX(10); r.setY(20);
        r.setAncho(100); r.setLargo(300);
    }
}
```

Encapsulación

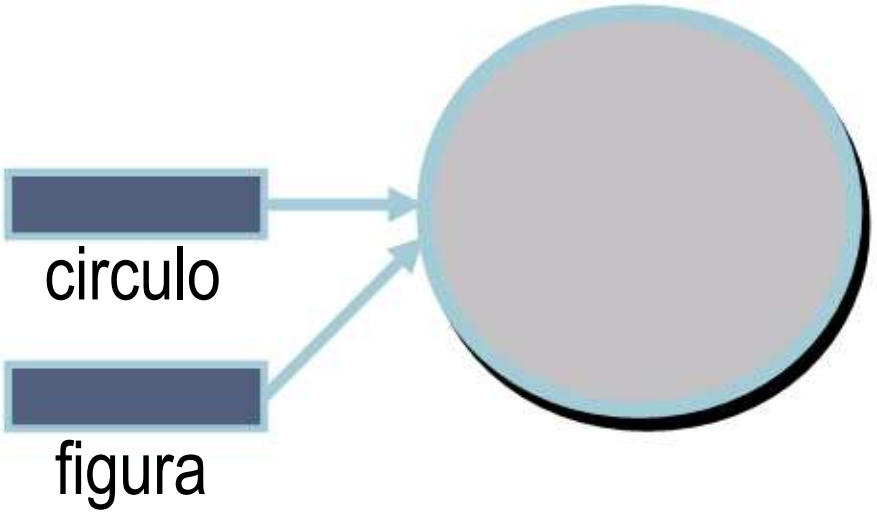
- Una clase derivada tiene acceso a los miembros public y protected de una clase base, aunque pertenezcan a paquetes diferentes

- Una clase derivada tiene acceso a los miembros package de una clase base si ambas clases pertenecen al mismo paquete

- Una clase derivada no tiene acceso a los miembros private de una clase base

Polimorfismo

```
Circulo circulo;
circulo = new Circulo();
Figura figura;
figura = circulo;
```



Compila y ejecuta bien (un círculo es una figura)
Restricción: no se puede usar figura para acceder a métodos especializados de Circulo

```
figura.getRadio(); // no compila
```

Polimorfismo

- Java permite asignar un objeto a una variable declarada con un tipo de datos ancestro

```
void metodo1(Figura f) {
f.print();

}

void metodo2() {
   metodo1(new Circulo());
}
```

Reescritura de Métodos

- Reescritura: overriding

- Reemplazar la implementación de un método existente en la clase base
  - La firma y el tipo de retorno deben ser idénticos a los de la clase base (de lo contrario se produce sobrecarga)
  - El nivel de acceso puede ser el mismo o mayor (un método protegido no puede ser redefinido como privado)

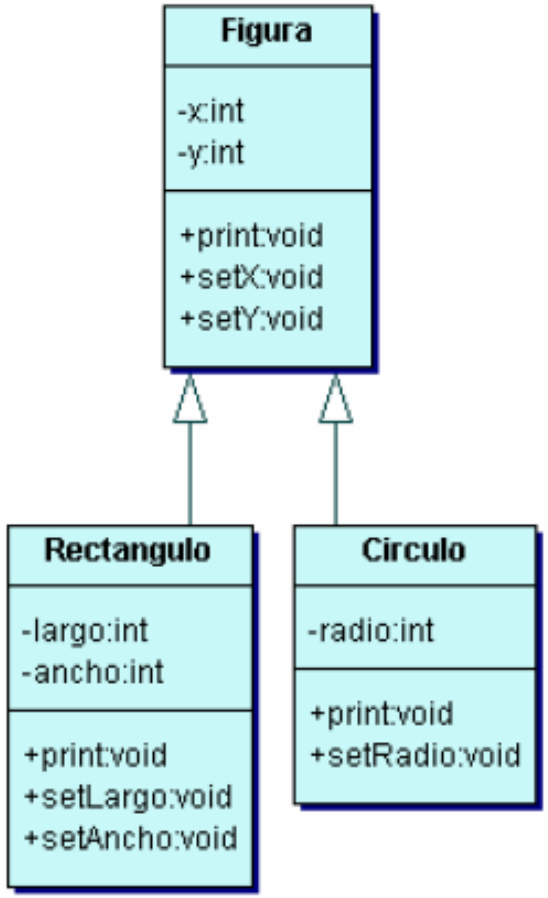
Ejemplo

```
public class Figura {
public void print() {

  }
}

public class Circulo extends Figura {
public void print() {

  }
}
```



La Palabra Clave super

- La palabra clave super puede ser utilizada para invocar explícitamente la implementación de un método de la clase base
- Hace referencia al ancestro inmediato
- Disponible en métodos de instancia

```
public class Circulo extends Figura {
  public void save() {
    super.save();

  }
}
```

Dynamic Binding

- Al invocar un método de instancia, el tipo real (dinámico) del objeto — no el tipo de la referencia (estático) — es utilizado para determinar qué versión del método invocar

```
void miMetodo(Figura f) {

  f.print(); // invoca a Circulo.print() si f es
             // una referencia a un Circulo

}
```

Compatibilidad de Tipos

- Java es fuertemente tipeado, exige compatibilidad de tipos en tiempo de compilación:
  - Permite asignar un objeto a una variable de un tipo ancestro
  - Requiere un cast explícito para asignar un objeto a una variable de un tipo descendiente

```
Circulo c = (Circulo) figura;
```

  - Si el cast falla en ejecución, la máquina virtual lanza un ClassCastException
  - El compilador no permite realizar una conversión de un objeto a un tipo que no es ancestro ni descendiente

Identificación de Tipo

- El método getClass de la clase Object retorna un objeto de tipo Class correspondiente a la clase real a la que pertenece el objeto

- El operador instanceof indica si un objeto es de una clase determinada o de alguna clase descendiente

```
if (figura instanceof Circulo) {
    Circulo circulo = (Circulo) figura;
    // uso de circulo
}
```

Constructor en Subclases

- El constructor de una subclase debe invocar algún constructor de la clase base:
  - Explícitamente: usando super() en la primera línea
  - Implícitamente: si no se invoca el constructor de la clase base explícitamente, se invoca el constructor default

Ejemplo

```
class Figura {
    int x, y;
    Figura(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

class Rectangulo extends Figura {
    int largo, ancho;
    Rectangulo(int x, int y, int l, int a) {
        super(x, y);
        largo = l;
        ancho = a;
    }
}
```

Clases Abstractas

- Una clase abstracta no puede ser instanciada, lo cual es asegurado por el compilador

- Puede contener métodos abstractos, a ser implementados en subclases

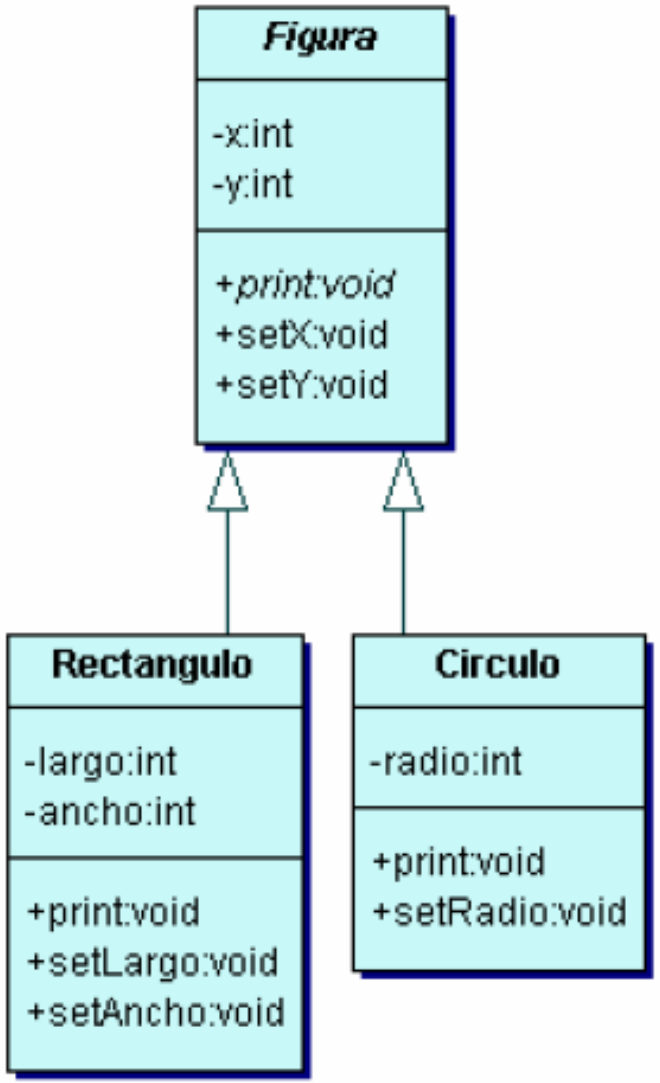
- Puede contener métodos concretos

- Una subclase de una clase abstracta debe:
  - implementar todos los métodos abstractos heredados, o bien
  - ser a su vez declarada abstracta

Ejemplo

```
public abstract class Figura {
    public abstract void print();
}

public class Rectangulo extends Figura {
    public void print() {
    // implementa print para Rectangulo
  }
}
```



Ejemplo: Diseño Tradicional

```
// En un lenguaje tradicional: C

void print()
{
  window* w = getCurrentWindow();
  switch (w->type) {
    case CARTA: printCarta(w); break;
    case MEMO:  printMemo(w);  break;

  }
}
```

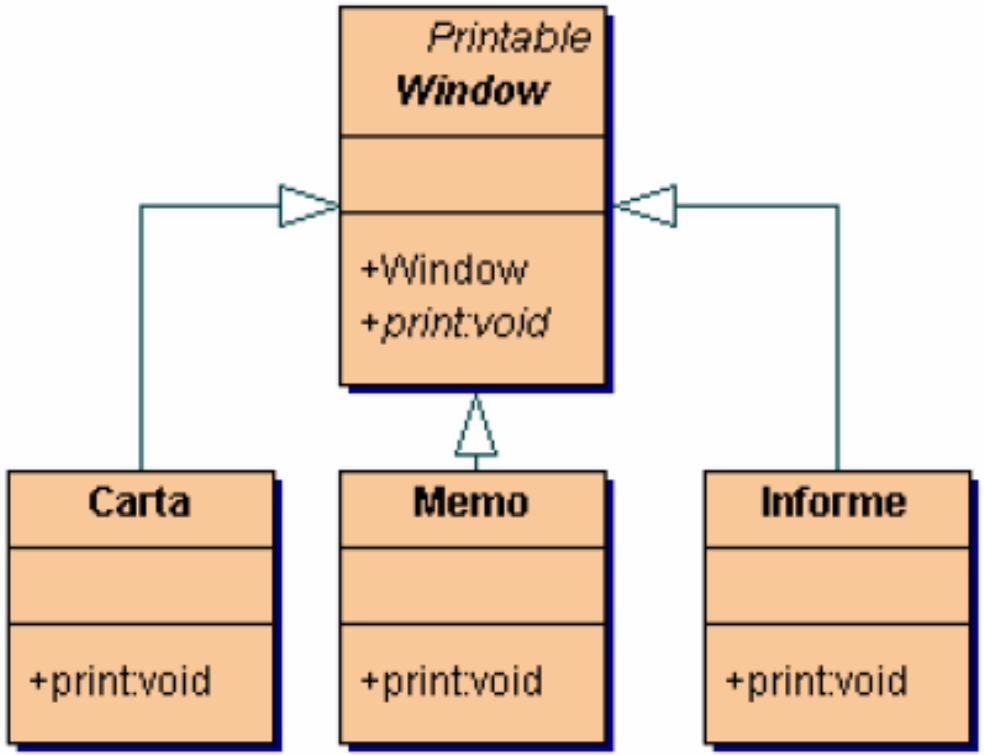
Ejemplo: Diseño con Polimorfismo

```
abstract class Window {
  public abstract void print();
}

class Carta extends Window {
public void print() {...}
}

class Memo extends Window {
public void print() {...}
}

class MiAplicacion {
void print() {
    Window w = GUI.getCurrentWindow();
    w.print();
  }
}
```



Interfaces

- En ocasiones es útil que una clase se pueda ver de varias formas, utilizando polimorfismo

- En C++, lo anterior se implementa con herencia múltiple, pero esto genera problemas cuando se hereda implementación

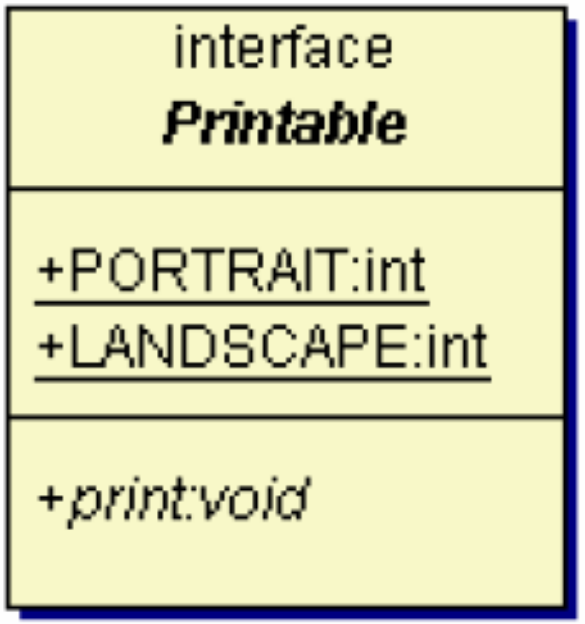
- Para evitar estos problemas, Java no soporta herencia múltiple, pero sí permite que una clase implemente múltiples interfaces

Interfaces

- Una interfaz (interface) es una colección de métodos abstractos y constantes
- No puede ser instanciada
- Define un tipo de datos que se puede utilizar en la declaración de variables
- Java soporta herencia múltiple de interfaces

Ejemplo

```
interface Printable {
    int PORTRAIT = 0;
    int LANDSCAPE = 1;
    void print(int orientacion);
}
```



- Campos son automáticamente public final static
- Métodos son automáticamente public abstract

Implementando una Interfaz

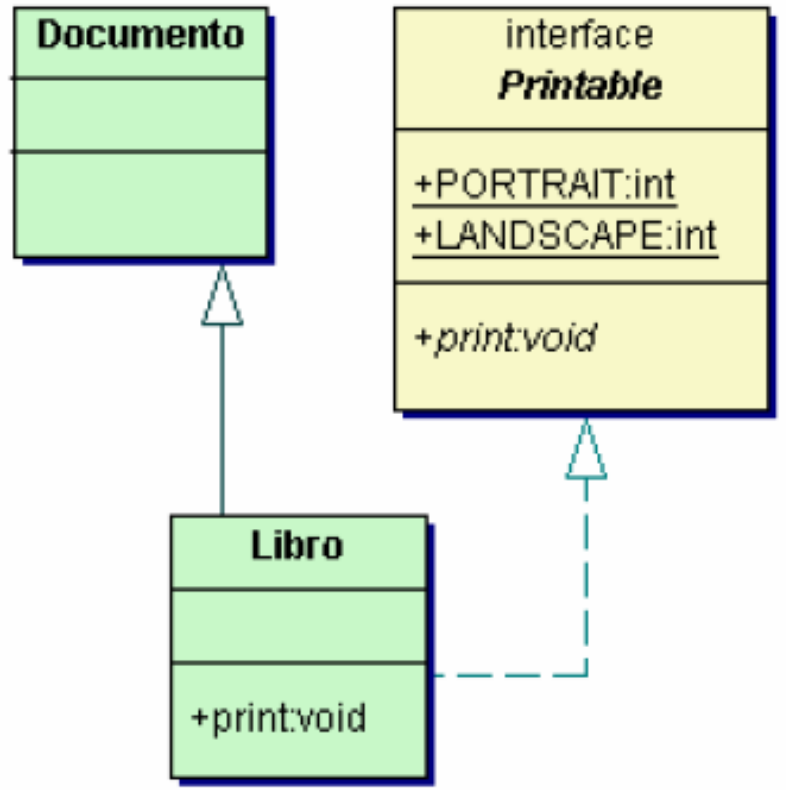
- Una clase puede implementar un número ilimitado de interfaces
- Una clase que implementa interfaces debe:
  - Implementar todos los métodos definidos en las interfaces, o bien
  - Ser declarada abstract

Implementando una Interfaz

```
class Libro extends Documento
            implements Printable
{
  public void print(int orientacion) {
    // implementación
  }
}

class Empleado implements Printable {
public void print(int orientacion) { ... }
}

class Rectangulo implements Printable {
public void print(int orientacion) { ... }
}
```



Uso de Interfaces

- Una definición de una interfaz crea un tipo de datos
- No es posible instanciar este tipo
- Sí es posible declarar variables de este tipo
- Es posible asignar a estas variables objetos de clases que implementen la interfaz
- Ejemplo

```
Printable printable = new Libro();
```

Uso de Interfaces

- Capa genérica

```
class ColaImpresion {
  static void creaJob(Printable p) {
  …
    p.print(Printable.PORTRAIT);
  }
}
```

- Capa cliente

```
ColaImpresion.creaJob(new Empleado(…));
ColaImpresion.creaJob(new Rectangulo(…));
Libro libro = new Libro();
…
ColaImpresion.creaJob(libro);
```

Resumen

- Herencia es un concepto de OOP mediante el cual una clase adquiere las propiedades y los métodos de otra
- Una variable de un tipo de datos ancestro puede ser utilizada para referenciar una instancia de una clase descendiente
- Al invocar sobre un objeto un método de instancia que ha sido redefinido en subclases, la máquina virtual invoca el método definido en la clase real del objeto, obtenida en tiempo de ejecución

Resumen

- Una clase abstracta (abstract) puede contener métodos abstractos
- Una interfaz (interface) es una colección de métodos abstractos y constantes
- Las clases abstractas y las interfaces no pueden ser instanciadas
- Una clase puede extender a una clase, e implementar un número ilimitado de interfaces
- Si una clase define o hereda métodos abstractos, debe ser declarada abstracta