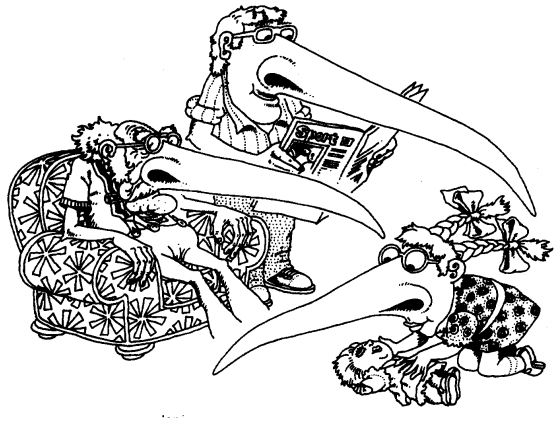
Herencia

Dr. Mario Marcelo Berón

1. Herencia
2. Beneficios
3. Costos
4. Herencia de Variables y Métodos
5. Búsqueda de Métodos
6. Herencia en Java
7. Inicialización de Objetos
8. Modificadores de Alcance
9. Anulación de Métodos y Variables
10. Ejercicio

Herencia: Conceptualización



Herencia

Es una relación entre las clases en la cual una clase comparte la estructura y el comportamiento de una (Herencia Simple) o más clases (Herencia Múltiple).

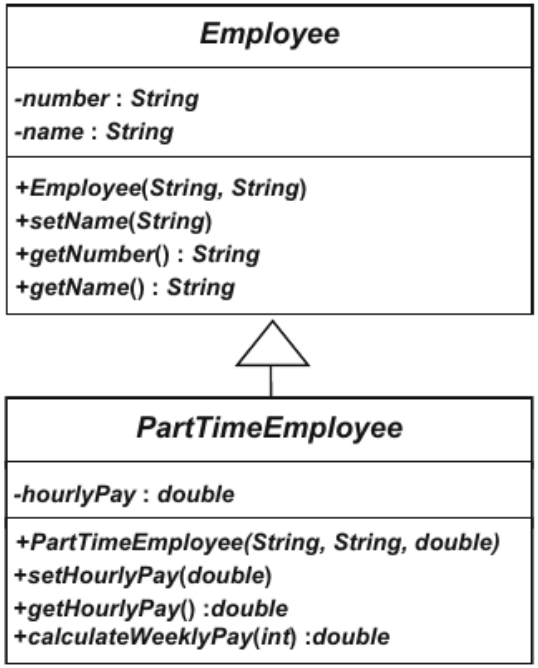
- La clase de la que las otras heredan se llama *Superclase*.
- Análogamente la clase que hereda de otra/s clase/s se llama *Subclase*.

Herencia: Conceptualización

Herencia

Con la *Herencia* una clase se puede definir en términos de otra. La nueva clase *Hereda* los atributos y métodos y además tiene atributos y métodos propios.

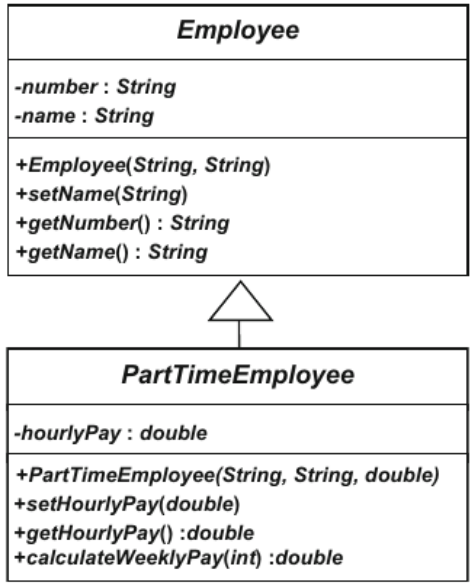
Herencia



Comentarios

- *Employee* es la clase base o la super clase o clase padre.
- *PartTimeEmployee* es la subclase, clase derivada o clase hijo.
- Los atributos y métodos de *Employee* son heredados por *PartTimeEmployee*.

Herencia



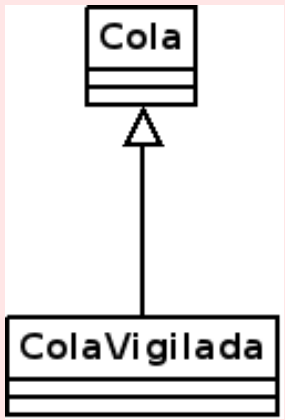
Comentarios

- La herencia es una relación jerárquica.
- La herencia también se referencia como una relación *es una clase de*. Un *PartTimeEmployee* es una clase de *Employee*.
- La clase *PartTimeEmployee* hereda los atributos y métodos de *Employee* y además tiene atributos y métodos propios.

Herencia

Herencia por Extensión

Una subclase debe tener todas las propiedades de la clase padre y otras propias de ella.

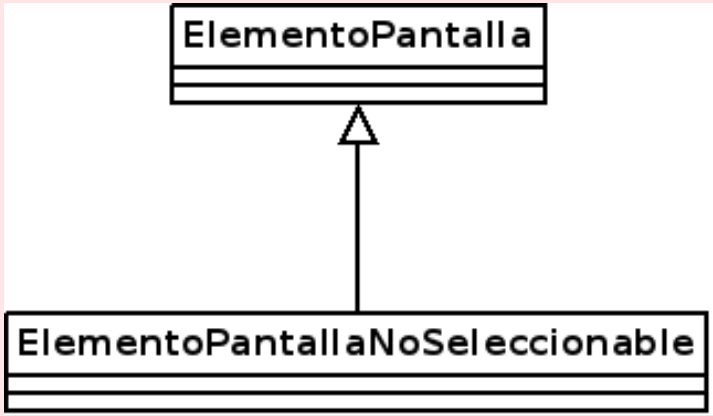


La clase *ColaVigilada* extiende el comportamiento de la clase *Cola* proporcionando operaciones que hacen que las instancias de ella estén protegidas.

Herencia

Herencia por Restricción

Como la subclase es más especializada que la clase padre es en cierto sentido una restricción de la clase padre.



La clase *ElementoDePantallaNoSeleccionable* podría restringir el comportamiento de la clase *ElementoDePantalla* prohibiendo a sus clientes la selección de sus instancias en una vista.

Herencia: Beneficios

Beneficios de la Herencia

Reusabilidad de Software: El código que provee el comportamiento heredado de otra clase no se implementa nuevamente.

Compartición de Código: Componentes de software (para Cox son: Software-ICs). También se da cuando dos o más clases heredan de una clase.

Consistencia de Interfaz: Cuando una o más clases heredan de una clase padre se está seguro que el comportamiento heredado será el mismo en todos los casos. En este contexto se asume la herencia sin sobre-escritura.

Componentes de Software: La herencia permite construir componentes reusables de software.

Herencia: Beneficios

Beneficios de la Herencia

Prototipado Rápido: Cuando un sistema se puede construir usando componentes reusables, el tiempo de desarrollo se concentra en entender la porción del sistema que es nueva e inusual.

Polimorfismo: Permite que el programador pueda definir componentes de alto nivel que pueden usarse en diferentes aplicaciones cambiando sus rutinas de bajo nivel.

Información Oculta: Cuando un programador reusa una componente de software solo necesita conocer la naturaleza de la componente y su interfaz.

Herencia: Costos

Costos

Velocidad de Ejecución: Los sistemas que usan herencia son generalmente más lentos que los que son hechos a mano. Esto se debe a que los métodos heredados deben estar preparados para ser usados por subclases arbitrarias.

Tamaño del Programa: El uso de bibiotecas de software impone una penalidad de tamaño sobre el sistema que se está construyendo.

Herencia: Costos

Costos

Overhead de Pasaje de Mensajes: Pasar un mensaje es una operación más costosa que una invocación a un procedimiento o función.

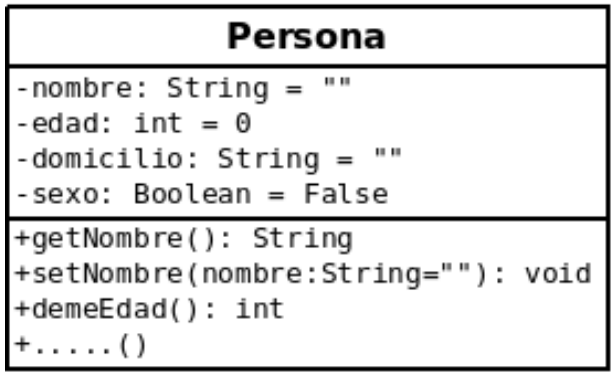
Complejidad del Programa: Algunas veces, cuando se hace abuso de la herencia, entender el flujo del programa requiere navegar hacia arriba y hacia abajo el grafo de la herencia. Este problemas se conoce con el nombre de: *El problema del yo-yo*.

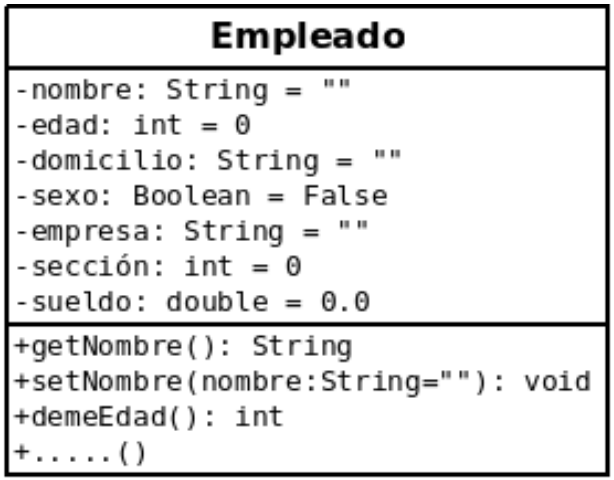
Herencia: Ejemplo

Personas

Empresario

Herencia: Ejemplo

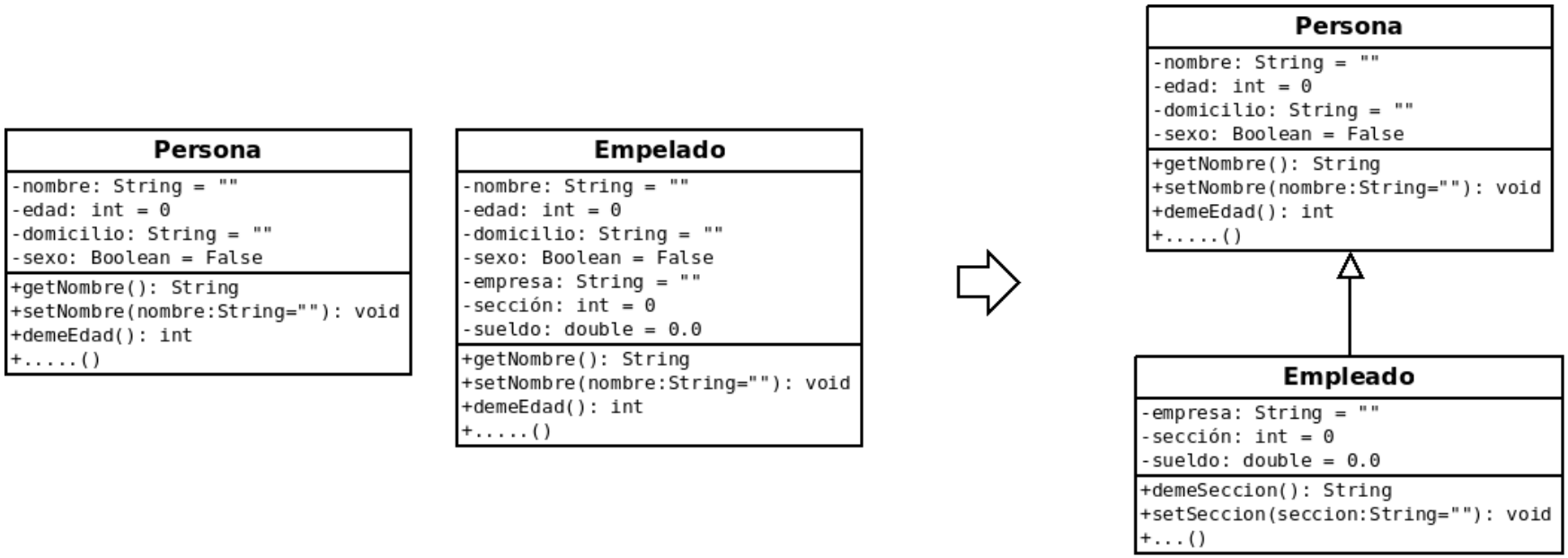




Observaciones

- Las dos clases son muy similares. Ambas clases tienen datos y muchas operaciones en común.
- Si se implementan las clases por separado sin observar las características en común entonces se duplica el código, es más difícil detectar errores, etc.

Herencia: Ejemplo



Observaciones

Si se aprovechan las similitudes entonces lo que se necesita implementar son solamente las nuevas características.

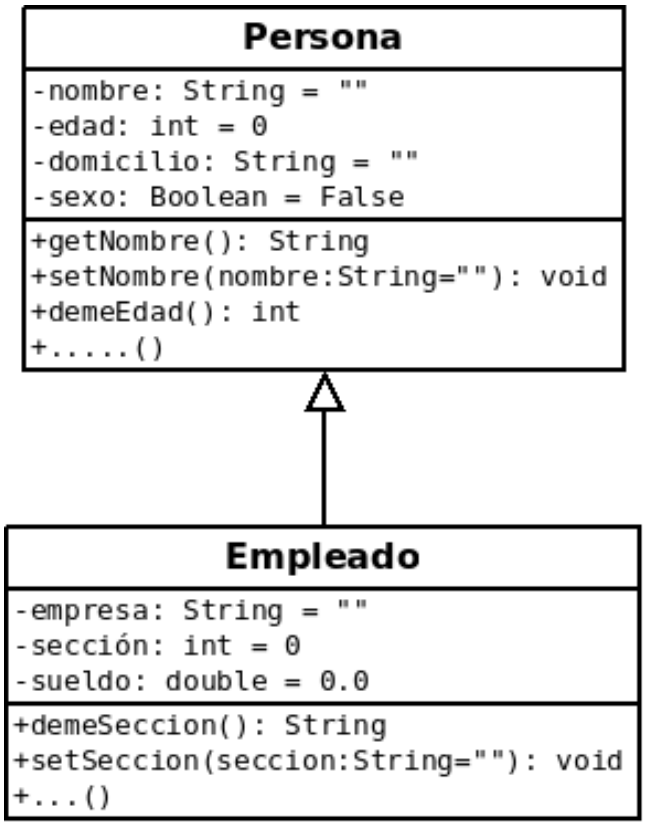
Herencia: Variables y Métodos

Variables y Métodos

Si B es una subclase de A entonces:

- B hereda todas las variables y métodos que no son privados.
- B puede definir nuevas variables y métodos de instancia propios.

Herencia: Variables y Métodos

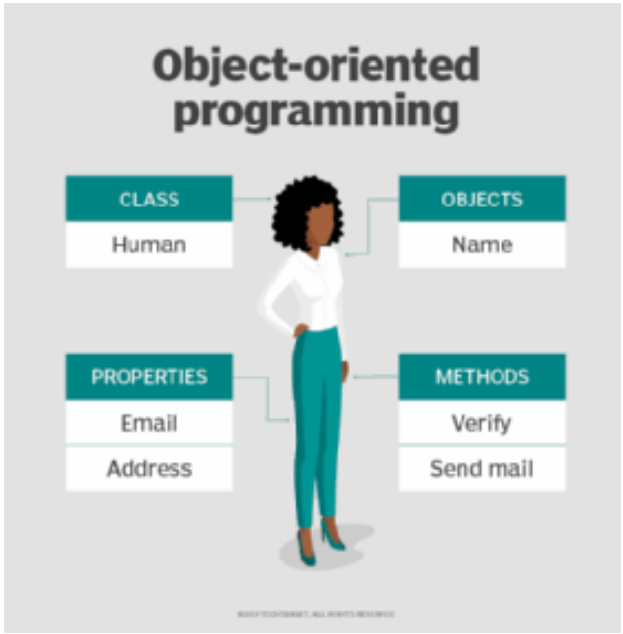


Variables y Métodos

Una instancia de la clase Empresario hereda:

- Las variables de instancia: nombre, edad, domicilio, sexo.
- Los métodos: demeNombre(), demeDomicilio(), demeSexo(), ... etc.

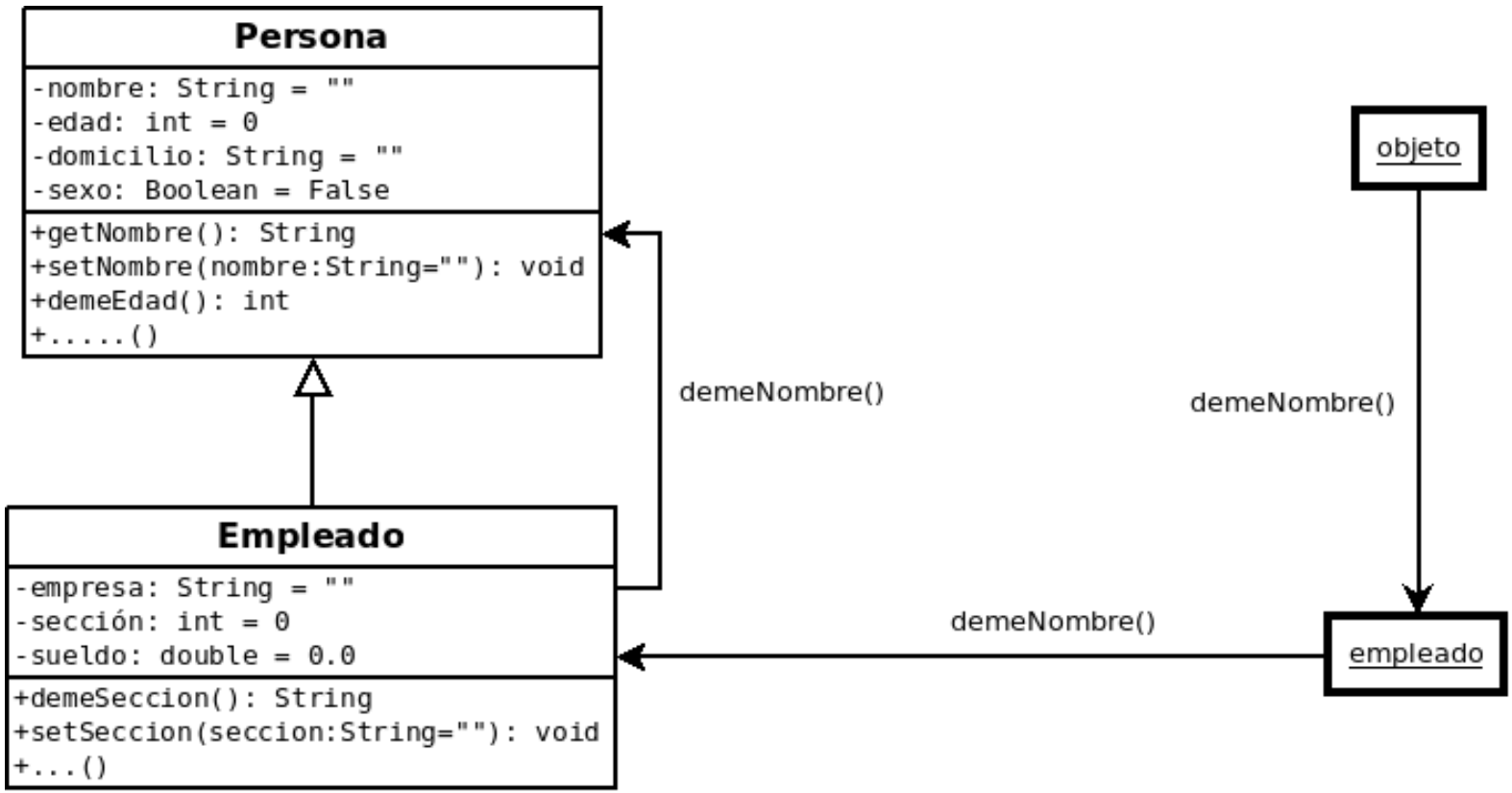
Búsqueda de Métodos



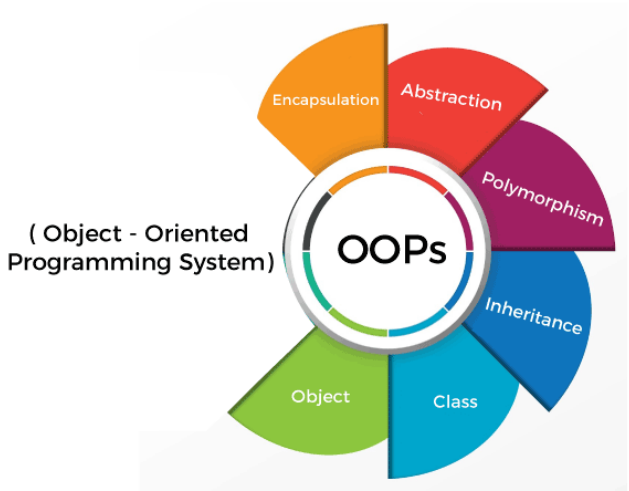
Búsqueda de Métodos

Cuando una instancia recibe un mensaje el método se busca en su clase si no está allí la búsqueda continua en su superclase y así sucesivamente hasta llegar al tope. En este caso si el método no se encuentra se produce un error.

Búsqueda de Métodos



Herencia en Java



1. Toda clase en Java es derivada de la clase Object.
2. Las clases en Java se organizan en jerarquías usando la palabra clave *extends*.
3. La relación de herencia establece una relación de superclase/subclase.
4. La superclase contiene los miembros comunes a sus subclases. Generalización.
5. La subclase contiene miembros específicos. Especialización.

Herencia

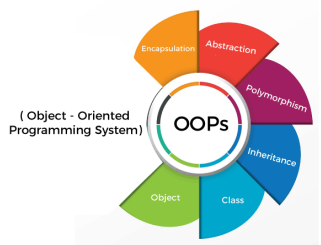
Herencia en Java

```
1  class A extends B {
2    .......
3  }
```

Herencia en Java

```
1  class Empresario extends Persona {
2    .......
3  }
```

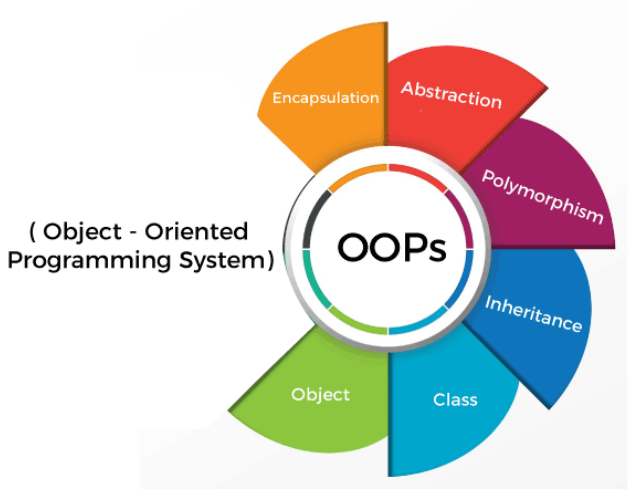
Herencia



Inicialización de Objetos

- En el contexto de la Herencia es confuso pensar en el objeto producido por la clase derivada. Esto se debe a que hay dos clases en lugar de una.
- Desde el exterior se percibe que la nueva clase tiene la misma interfaz que la clase base y quizás algunos métodos/campos adicionales.
- Cuando se crea un objeto de la clase derivada, el mismo contiene un objeto de la clase base. Este subobjeto se encuentra *envuelto* dentro del objeto de la clase derivada.

Herencia



Inicialización de Objetos

La inicialización de los objetos creados mediante la *Relación de Herencia* se realiza en el *constructor* a través de la invocación del constructor de la clase base.

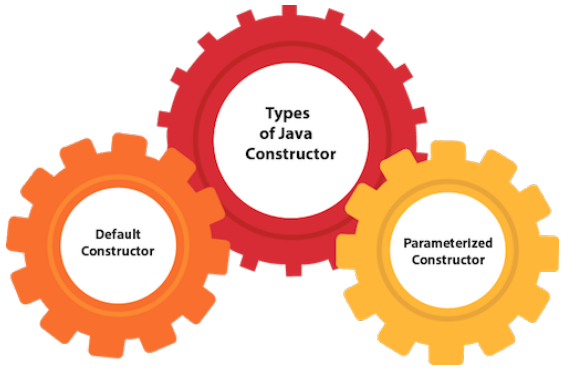
Es importante notar que: *Java inserta llamadas al constructor de la clase base en el constructor de la clase derivada*.

Herencia

Inicialización de Objetos

```
1 class Arte {
2  Arte() {
3   System.out.println("Constructor de Arte");
4  }
5 }
6
7 class Dibujo extends Arte {
8  Dibujo() {
9  System.out.println("Constructor de Dibujo");
10  }
11 }
12
13 public class DAnimado extends Dibujo {
14  public DAnimado() {
15   System.out.println("Const. de DAnimado.");
16  }
17  public static void main(String[] args) {
18   DAnimado x = new DAnimado();
19  }
20 }
```

Herencia



Constructores con Argumentos

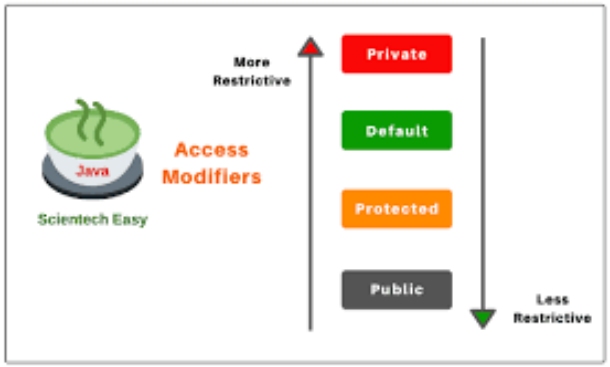
En el ejemplo anterior los *Constructores* usados son los *Constructores sin argumentos*. No obstante, si se desea llamar al constructor de la clase base se debe invocar explícitamente a dicho constructor usando la palabra *super* y la lista de argumentos apropiados.

Herencia

Ejemplo de Incialización con Constructores con Parámetros

```
1  import static net.mindview.util.Print.*;
2
3  class Game {
4   Game(int i) {
5    print("Game constructor");
6   }
7  }
8
9  class BoardGame extends Game {
10  BoardGame(int i) {
11   super(i);
12   print("BoardGame constructor");
13  }
14 }
15
16 public class Chess extends BoardGame {
17  Chess() {
18   super(11);
19   print("Chess constructor");
20  }
21
22  public static void main(String[] args) {
23   Chess x = new Chess();
24  }
25 }
```

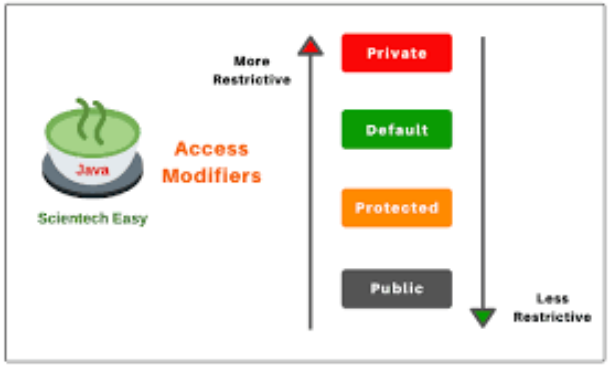
Herencia y Modificadores de Acceso



Se tiene dos niveles:

1. A nivel de clases: *public*,*paquete*, *private*.
2. A nivel de miembros: *public*,*package*,*protected*, *private*.

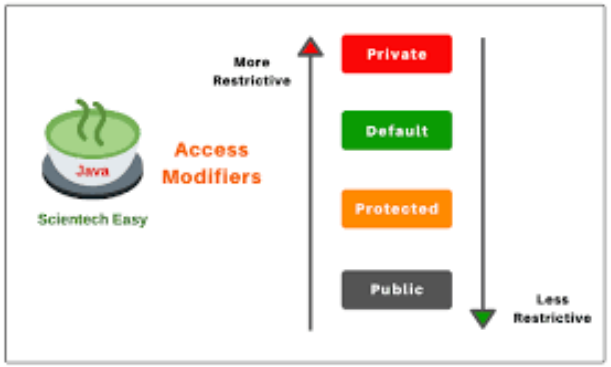
Herencia y Modificadores de Acceso



Nivel de Clases

1. Public: la clase es accesible para todas las clases en todas partes.
2. Package: la clase es accesible solo dentro de su paquete.
3. Private: se utiliza para clases internas.

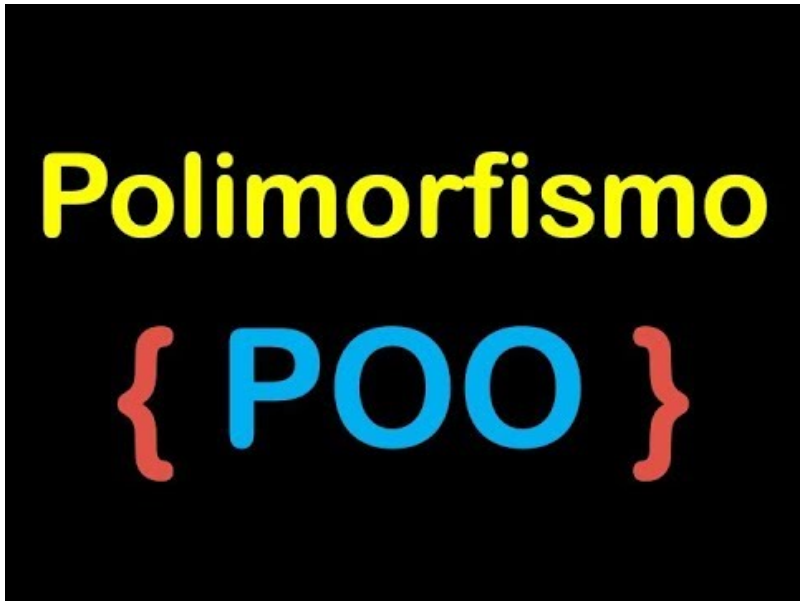
Herencia y Modificadores de Acceso



Nivel de Miembros

1. Public: el dato/método miembro es accesible para todas las clases en todas las partes.
2. Protected: el dato/método miembro es accesible dentro de su propio paquete y en sus subclases.
3. Package: el dato/método es accesible solo dentro de su paquete.
4. Private: el dato/método es accesible sólo dentro de su clase.

Herencia



La idea del *Polimorfismo* es que diferentes métodos y operadores con el mismo nombre tengan comportamientos diferente.
Una de las formas de implementar el polimorifismo es la *Sobrecarga de Métodos*. Esta técnica implica que un método tenga un mismo nombre pero se pueda distinguir por su lista de parámetros. También existe otra forma de llevar a cabo el polimorfismo que consiste en la sobre escritura de métodos.

Herencia

La sobre escritura de métodos (o anulación de métodos) es una característica que permite a una subclase proveer una implementación específica de un método que ya está implementado en una de sus superclases.

Importante

- Cuando se sobreescribe un método es conveniente usar la anotación @Override la cual informa al compilador que se está ocultando el método de la superclase. Con la anotación el compilador puede informar si la signatura de los métodos no concuerdan.
- Las anotaciones comienzan con un @ y con mayúsculas.
- El método oculto invocado se distingue por el receptor.

Implementación del Modelo