Programación

PLG
Planning and Learning Group

Universidad Carlos III de Madrid

Curso 2015-2016

Más POO: herencia y polimorfismo

Concepto de herencia

Herencia

Concepto de herencia

Herencia

- Las propiedades pasan de padres a hijos
- Relación SUBCLASE-DE y SUPERCLASE-DE
- Ejemplos

¿Por qué herencia en programas?

Reutilización y centralización de código

¿Por qué herencia en programas?

Reutilización y centralización de código

- Evita duplicidad
- Disminuye volumen de código
- Facilita mantenimiento
- Permite el polimorfismo

Herencia en Java

- Clases derivadas de otras clases que heredan sus atributos y métodos
- Se utiliza la palabra reservada `extends`

```
class Subclase extends SuperClase {
      ...
}
```

Modificador protected

```
class A {
      public int a;
      private int b;
      protected int c;
}

class B extends A {
      ... }
```

Modificador protected

```
class A {
      public int a;
      private int b;
      protected int c;
}

class B extends A {
      ... }
```

- `a` es público: accesible desde fuera de la clase y el paquete. Visible para todos, incluída la clase hija
- `b` es privado: no visible desde fuera de la clase, ni desde sus clases hijas
- `c` es protegido: accesible desde dentro del paquete. Además pasa a las clases hijas: la clase B dispone de ese atributo

Resumen restricciones de acceso

VISIBILIDAD	public	protected	nada	private
Propia clase	SI	SI	SI	SI
Mismo paquete	SI	SI	SI	NO
Otro paquete	SI	NO	NO	NO
Subclase en paquete	SI	SI	SI	NO
Subclase en otro paquete	SI	SI	NO	NO

super y super()

- La palabra reservada `this` sirve para indicar la propia clase
- La palabra reservada `super` sirve para hacer referencia a la superclase de la clase en que se utiliza

super y super()

```
class A {
      int a;
      int b;
}

class B extends A {
      String b;
}
```

super y super()

```
class A {
      int a;
      int b;
}

class B extends A {
      String b;
}
```

- Si estamos en B, `super.b` es el atributo `b` de `A`

super y super()

- Si en A hay un método `mostrarDatos()` que muestra `a` y `b`,
- y en B reimplementamos ese método para que muestre también el atributo `b` de B:

```
class B {
      ...

      void mostrarDatos () {
          super.mostrarDatos();
          System.out.println(b);
      }

}
```

super y super()

- `super()` invoca al constructor de la superclase

```
class Persona {
   String nombre;
   byte edad;
   double altura;

   Persona (String nombre, byte edad, double altura) {
      this.nombre = nombre;
      this.edad = edad;
      this.altura = altura;
   }
}
```

super y super()

- `super()` invoca al constructor de la superclase

```
class Persona {
   String nombre;
   byte edad;
   double altura;

   Persona (String nombre, byte edad, double altura) {
      this.nombre = nombre;
      this.edad = edad;
      this.altura = altura;
   }
}

class Empleado extends Persona {
   double salario;
```

super y super()

- `super()` invoca al constructor de la superclase

```
class Persona {
   String nombre;
   byte edad;
   double altura;

   Persona (String nombre, byte edad, double altura) {
      this.nombre = nombre;
      this.edad = edad;
      this.altura = altura;
   }
}

class Empleado extends Persona {
   double salario;

   Empleado (String nombre, byte edad, double altura, double salario)
      super(nombre, edad, altura);
      this.salario= salario;
   }
}
```

Sustitución de métodos y atributos

- Las clases heredan de sus superclases todos los métodos y atributos no privados
- En la subclase se pueden redefinir tanto atributos cómo métodos
  - Utilizando el mismo nombre en atributos
  - En métodos, utilizando el mismo nombre, mismo valor de retorno y mismos parámetros
- Cuando se redefine un atributo o un método, el nuevo oculta al heredado

Sustitución de métodos y atributos

Clases A y B

```
class A {
     int a = 3;
}

class B extends A {
     double a = 2.5;
}
```

Programa principal u otra clase

```
B objetoB = new B();
System.out.println(objetoB.a);
```

Sustitución de métodos y atributos

Clases A y B

```
class A {
     int a = 3;
}

class B extends A {
     double a = 2.5;
}
```

Programa principal u otra clase

```
B objetoB = new B();
System.out.println(objetoB.a); // mostrará por pantalla 2.5
```

Sustitución de métodos y atributos

Clases A y B

```
class A {
    void mostrar() {
        System.out.println("Soy una A");
    }
}

class B extends A {
}
```

Programa principal

```
B objetoB = new B();
objetoB.mostrar();
```

Sustitución de métodos y atributos

Clases A y B

```
class A {
    void mostrar() {
        System.out.println("Soy una A");
    }
}

class B extends A {
}
```

Programa principal

```
B objetoB = new B();
objetoB.mostrar(); // mostrará por pantalla soy una A
```

Sustitución de métodos y atributos

Clases A y B

```
class A {
    void mostrar() {
        System.out.println("Soy una A");
    }
}

class B extends A {
    void mostrar() {
        System.out.println("Soy una B");
    }
}
```

Programa principal u otra clase

```
B objetoB = new B();
objetoB.mostrar();
```

Sustitución de métodos y atributos

Clases A y B

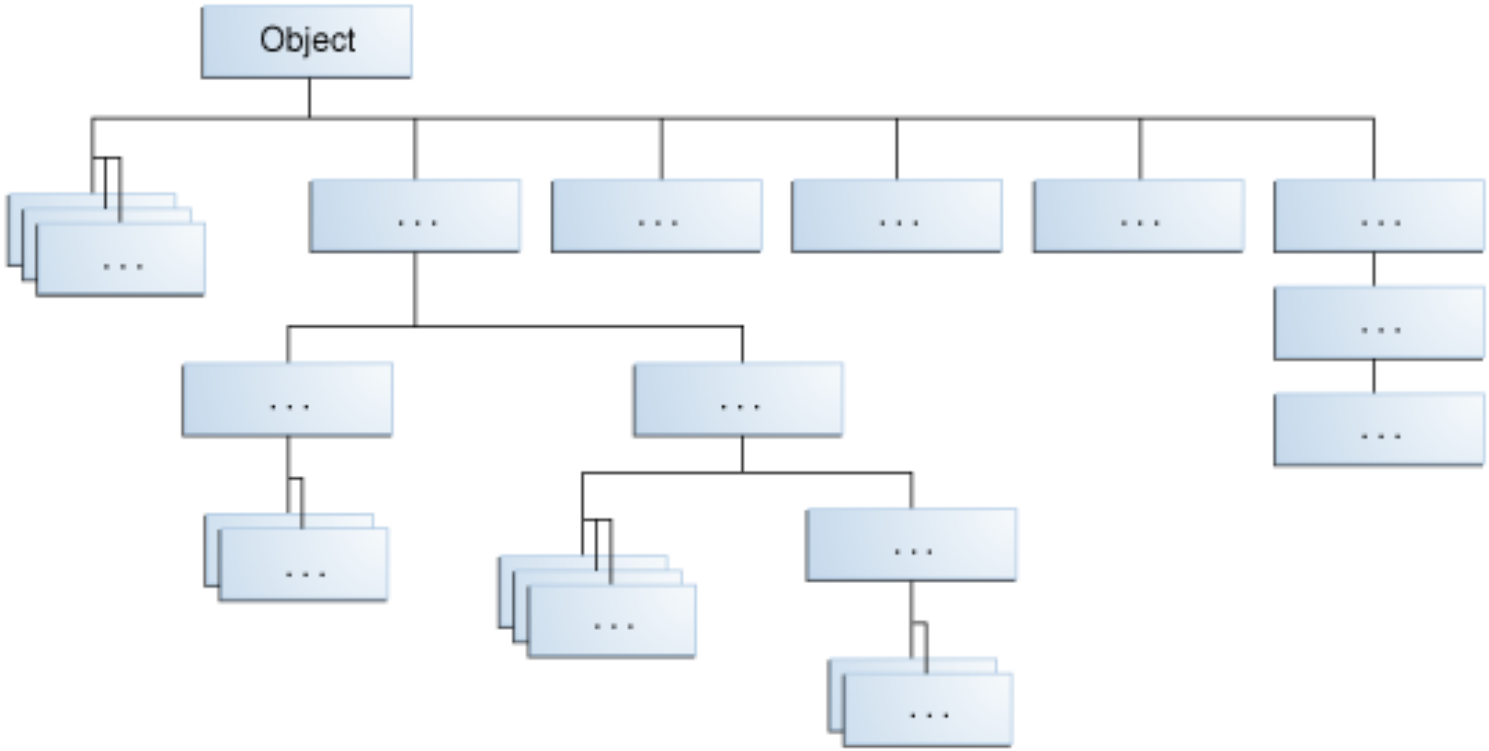
```
class A {
    void mostrar() {
        System.out.println("Soy una A");
    }
}

class B extends A {
    void mostrar() {
        System.out.println("Soy una B");
    }
}
```

Programa principal u otra clase

```
B objetoB = new B();
objetoB.mostrar(); // mostrará por pantalla soy una B
```

Clase `Object`

En Java todo objeto hereda implícitamente de la clase `Object`



- Esta clase tiene métodos que pueden ser redefinidos, por ejemplo
  - `clone`: crear una copia del objeto
  - `equals`: determinar si dos objetos son iguales
  - `toString`: devuelve un `String` que representa al objeto

Concepto de polimorfismo

Polimorfismo

Concepto de polimorfismo

Polimorfismo

- Definición de la RAE: *Cualidad de lo que tiene o puede tener distintas formas*

Concepto de polimorfismo en POO

- Hemos visto que en Java el contenido de las variables debe ser del tipo del que fueron declaradas
- Pero ...
- cuando hay herencia, ¡una variable del tipo de la Superclase puede contener también cualquier objeto del tipo de una Subclase!
- Esto se denomina polimorfismo en POO

Concepto de polimorfismo en POO

Clase Persona y subclase Empleado

```
class Persona {
    ... }

class Empleado extends Persona {
    ... }
```

Programa principal u otra clase

```
Persona persona1 = new Persona();
Empleado empleado1 = new Empleado();
```

Concepto de polimorfismo en POO

Clase Persona y subclase Empleado

```
class Persona {
    ... }

class Empleado extends Persona {
    ... }
```

Programa principal u otra clase

```
Persona persona1 = new Persona();
Empleado empleado1 = new Empleado();
Persona persona1 = new Empleado();
```

Selección dinámica de métodos

Una de las herramientas más potentes que el polimorfismo proporciona a Java es la selección dinámica de métodos en tiempo de ejecución

Selección dinámica de métodos

Clases A y B

```
class A {
    void mostrar() {
        System.out.println("Soy una A");
    }
}
class B extends A {
    void mostrar() {
        System.out.println("Soy una B");
    }
}
class C extends B {
    void mostrar() {
        System.out.println("Soy una C");
    }
}
```

Programa principal u otra clase

```
A variableA;
variableA = new A();
variableA.mostrar();
```

Selección dinámica de métodos

Clases A y B

```
class A {
    void mostrar() {
        System.out.println("Soy una A");
    }
}
class B extends A {
    void mostrar() {
        System.out.println("Soy una B");
    }
}
class C extends B {
    void mostrar() {
        System.out.println("Soy una C");
    }
}
```

Programa principal u otra clase

```
A variableA;
variableA = new A();
variableA.mostrar(); // mostrará por pantalla Soy una A
```

Selección dinámica de métodos

Clases A y B

```
class A {
    void mostrar() {
        System.out.println("Soy una A");
    }
}
class B extends A {
    void mostrar() {
        System.out.println("Soy una B");
    }
}
class C extends B {
    void mostrar() {
        System.out.println("Soy una C");
    }
}
```

Programa principal u otra clase

```
A variableA;
variableA = new A();
variableA.mostrar(); // mostrará por pantalla Soy una A
variableA = new B();
variableA.mostrar();
```

Selección dinámica de métodos

Clases A y B

```
class A {
    void mostrar() {
        System.out.println("Soy una A");
    }
}
class B extends A {
    void mostrar() {
        System.out.println("Soy una B");
    }
}
class C extends B {
    void mostrar() {
        System.out.println("Soy una C");
    }
}
```

Programa principal u otra clase

```
A variableA;
variableA = new A();
variableA.mostrar(); // mostrará por pantalla Soy una A
variableA = new B();
variableA.mostrar(); // mostrará por pantalla Soy una B
```

Selección dinámica de métodos

Clases A y B

```
class A {
    void mostrar() {
        System.out.println("Soy una A");
    }
}
class B extends A {
    void mostrar() {
        System.out.println("Soy una B");
    }
}
class C extends B {
    void mostrar() {
        System.out.println("Soy una C");
    }
}
```

Programa principal u otra clase

```
A variableA;
variableA = new A();
variableA.mostrar(); // mostrará por pantalla Soy una A
variableA = new B();
variableA.mostrar(); // mostrará por pantalla Soy una B
variableA = new C();
variableA.mostrar();
```

Selección dinámica de métodos

Clases A y B

```
class A {
    void mostrar() {
        System.out.println("Soy una A");
    }
}
class B extends A {
    void mostrar() {
        System.out.println("Soy una B");
    }
}
class C extends B {
    void mostrar() {
        System.out.println("Soy una C");
    }
}
```

Programa principal u otra clase

```
A variableA;
variableA = new A();
variableA.mostrar(); // mostrará por pantalla Soy una A
variableA = new B();
variableA.mostrar(); // mostrará por pantalla Soy una B
variableA = new C();
variableA.mostrar(); // mostrará por pantalla Soy una C
```

Selección dinámica de métodos

¡Se ejecutan distintos métodos según el tipo de objeto referenciado!

Clases abstractas

- Jerarquía de herencia de clases
  - Cuánto más abajo en la jerarquía, más específica es la clase
  - Cuanto más arriba, más general es la clase
- Si la clase de arriba es muy general puede que haya métodos que no se puedan implementar sin conocer la subclase
- Ejemplo: juego piedra, papel tijera
  - Clase Jugador, subclases JugadorEsCiclica, JugadorEsAleatoria
  - El método `Eleccion` no se puede implementar sin conocer el tipo de jugador

Clases abstractas

```
abstract class Jugador {
        int puntuacion;
        ...
        int getPuntuacion() {
             return puntuacion;
        }
        abstract int eleccion();
        // El método elección no se implementa, sólo se
        // declara como abstracto para ser implementado en las subclases
}
```

- Las clases con métodos abstractos deben ser declaradas como abstractas
- ¡No se pueden crear objetos del tipo de una clase abstracta!

Clases abstractas

```
class JugadorEsCiclica extends Jugador {
        ...
        int eleccion() {
        // < Implementación del método aquí>
        }
}

class JugadorEsAleatoria extends Jugador {
        ...
        int eleccion() {
        // < Implementación del método aquí>
        }
}
```

Clases abstractas

```
class JugadorEsCiclica extends Jugador {
        ...
        int eleccion() {
        // < Implementación del método aquí>
        }
}

class JugadorEsAleatoria extends Jugador {
        ...
        int eleccion() {
        // < Implementación del método aquí>
        }
}
```

Programa Principal

```
Jugador jugador1 = new JugadorEsCiclica();
Jugador jugador2 = new JugadorEsAleatoria();
int eleccion1 = jugador1.eleccion();
int eleccion2 = jugador2.eleccion();
System.out.println(jugador1.getPuntuacion());
System.out.println(jugador2.getPuntuacion());
```