Herència en Java

Què és l'herència

L'herència és el mecanisme mitjançant el qual podem crear noves classes (classes derivades o subclasses o classes filles) aprofitant la definició d'altres classes (superclasses o classes base). L'herència és una **relació jeràrquica**.

Amb el mecanisme de l'herència, a la classe derivada només hem de programar les diferències amb la classe base. Diem que la classe derivada hereta els atributs i els mètodes de la classe base.

En llenguatge Java, per derivar una classe s'utilitza la paraula clau **extends**.

```
public class Derivada extends Base
{

}
```

Cal tenir present que, en realitat, cada objecte de la classe derivada conté un objecte sencer de la classe Base, l'accés al qual s'aconsegueix amb l'atribut super. De fet, el constructor per defecte de la classe derivada invoca al constructor per defecte de la classe base i, si definim un constructor, la primera instrucció ha de ser la invocació al constructor de la classe base.

```
public class Base
{
    public Base()
    {
        System.out.println("Nou objecte classe Base");
    }
}
public class Derivada extends Base
{
    public Derivada()
    {
        super();
        System.out.println("Nou objecte classe Derivada");
    }
}
```

Qualsevol classe pot ser base per derivar-ne d'altres, llevat de les que s'han definit amb el modificador **final**.

En canvi, les classes definides amb el modificador **abstract** han de ser necessàriament heretades per altres, atès que no es pot instanciar cap objecte d'una classe abstracta. Tota classe que contingui una definició de mètode abstracte (porta el modificador abstract en la definició del mètode) ha de ser abstracta. Els mètodes abstractes han de ser redefinits (sobreescrits) per mètodes en la classe derivada.

```
public abstract int calculaArea();
```

Exercici resolt Herencia1: Constructors sense arguments

Crear dues classes A i B amb constructors sense arguments. Els constructors només han d'anunciar la creació d'un objecte del tipus corresponent. Derivar de la classe A una classe C que contingui com a atribut una referència a objecte de classe B. La classe d'inici Herencia1 instanciarà un objecte de cada classe. Analitzar el resultat.

A continuació tenim el codi demanat. La classe C no té constructor, per tant, el compilador en genera un per defecte, sense arguments.

```
/**
 * Herencia1.java
 * Exemple de herencia i constructors
 * @author Jose Moreno
 * @version
 */
public class Herencia1 {
    public static void main(String[] args) {
        System.out.println("Creacio d'un objecte de classe A");
        A aObj = new A();
        System.out.println("Creacio d'un objecte de classe B");
        B bObj = new B();
        System.out.println("Creacio d'un objecte de classe C");
        C cObj = new C();
    }
}
class A {
    public A() {
        System.out.println("Nou objecte de tipus A");
    }
}
class B {
    public B() {
        System.out.println("Nou objecte de tipus B");
    }
}
class C extends A {
    B b;
}
```

La sortida del programa anterior ve a continuació. Podem comprovar com en instanciar cada objecte s'executa el seu constructor. En el cas de la classe C, veiem que el constructor que per defecte ha creat el compilador, executa en primer lloc el constructor de la classe base A.

```
Creacio d'un objecte de classe A
Nou objecte de tipus A
Creacio d'un objecte de classe B
Nou objecte de tipus B
Creacio d'un objecte de classe C
Nou objecte de tipus A
```

Exercici resolt Herencia2: Constructors amb arguments

Crear dues classes A i B amb constructors amb arguments. Derivar de la classe A una classe C que contingui com a atribut una referència a objecte de classe B. Definir un constructor amb arguments per a la classe C. La classe d'inici Herencia2 instanciarà un objecte de cada classe. Analitzar el resultat.

A continuació tenim el codi demanat.

```
/**
 * Herencia2.java
 * Exemple de herencia i constructors
 * @author Jose Moreno
```

```
 * @version
 */
public class Herencia2 {
    public static void main(String[] args) {
        System.out.println("Creacio d'un objecte de classe A");
        A aObj = new A(1);
        System.out.println("Creacio d'un objecte de classe B");
        B bObj = new B(2);
        System.out.println("Creacio d'un objecte de classe C");
        C cObj = new C(3,4);
    }
}
class A {
    public A(int na) {
        System.out.println("Nou objecte de tipus A "+na);
    }
}
class B {
    public B(int nb) {
        System.out.println("Nou objecte de tipus B "+nb);
    }
}
class C extends A {
    B b;
    public C(int na, int nb) {
        super(na);
        b = new B(nb);
    }
}
```

La sortida del programa anterior ve a continuació. Podem comprovar com en instanciar cada objecte s'executa el seu constructor. En el cas de la classe C, a més d'executar-se el constructor de la classe A, també s'executa a continuació el de la classe B, atès que hem instanciat un objecte de tipus B dintre del constructor de C.

```
Creacio d'un objecte de classe A
Nou objecte de tipus A 1
Creacio d'un objecte de classe B
Nou objecte de tipus B 2
Creacio d'un objecte de classe C
Nou objecte de tipus A 3
Nou objecte de tipus B 4
```

És interessant comparar aquest cas amb l'exemple anterior en el qual no s'instanciava cap objecte de classe B, sinó que l'atribut b només era una referència que calia inicialitzar a un objecte de classe B instanciat fora de la classe C.

Exercici resolt Herencia3a: Constructors

Crear una classe A i derivar-ne dues classes B i C. Els constructors d'A i C tenen arguments. El constructor de B no té arguments. La classe d'inici Herencia3a instanciarà un objecte de cada classe. Analitzar el resultat.

A continuació tenim el codi demanat.

```
/**
 * Herencia3a.java
```

```
 * Exemple de herencia i constructors
 * @author Jose Moreno
 * @version
 */
public class Herencia3a {
    public static void main(String[] args) {
        System.out.println("Creacio d'un objecte de classe A");
        A aObj = new A(1);
        System.out.println("Creacio d'un objecte de classe B");
        B bObj = new B();
        System.out.println("Creacio d'un objecte de classe C");
        C cObj = new C(3);
    }
}
class A {
    public A(int na) {
        System.out.println("Nou objecte de tipus A "+na);
    }
}
class B extends A {
    public B() {
        super(2);
        System.out.println("Nou objecte de tipus B ");
    }
}
class C extends A {
    public C(int nc) {
        super(nc);
        System.out.println("Nou objecte de tipus C ");
    }
}
```

La sortida del programa anterior ve a continuació.

```
Creacio d'un objecte de classe A
Nou objecte de tipus A 1
Creacio d'un objecte de classe B
Nou objecte de tipus A 2
Nou objecte de tipus B
Creacio d'un objecte de classe C
Nou objecte de tipus A 3
Nou objecte de tipus C
```

Podem comprovar que en instanciar objectes de classe B o C s'executa el constructor de classe A amb l'argument que se li passa amb `super()`.

Exercici resolt Herencia3b: Constructors

Crear una classe A i derivar-ne dues classes B i C. Els constructors d'A i C no tenen arguments. A la classe B no definir cap constructor. La classe d'inici Herencia3b instanciarà un objecte de cada classe. Analitzar el resultat.

A continuació tenim el codi demanat.

```
/**
 * Herencia3b.java
 * Exemple de herencia i constructors
 * @author Jose Moreno
```

```
 * @version
 */

public class Herencia3b {
    public static void main(String[] args) {
        System.out.println("Creacio d'un objecte de classe A");
        A aObj = new A();
        System.out.println("Creacio d'un objecte de classe B");
        B bObj = new B();
        System.out.println("Creacio d'un objecte de classe C");
        C cObj = new C();
    }
}
class A {
    public A() {
        System.out.println("Nou objecte de tipus A ");
    }
}
class B extends A {
}
class C extends A {
    public C() {
        super();
        System.out.println("Nou objecte de tipus C ");
    }
}
```

La sortida del programa anterior ve a continuació.

```
Creacio d'un objecte de classe A
Nou objecte de tipus A
Creacio d'un objecte de classe B
Nou objecte de tipus A
Creacio d'un objecte de classe C
Nou objecte de tipus A
Nou objecte de tipus C
```

Podem comprovar que en instanciar objectes de classe B o C s'executa el constructor de classe A, fins i tot en el cas de la classe B que no té definit cap constructor. El constructor per defecte que genera el compilador efectua la invocació al constructor de classe A.

Exercici resolt Herencia4a: Agregació i herència

Crear tres classes ComponentA, ComponentB i ComponentC amb un constructor que anunciï la instanciació. Crear una classe Base que contingui un atribut de cada una d'aquestes classes. Derivar de la classe Base una classe Derivat. Els constructors de les classes Base i Derivat només anuncien la instanciació. Programar la classe que instancia un objecte de classe Base i un altre de classe Derivat i analitzar la sortida.

A continuació tenim el codi demanat.

```
/**
 * Herencia4a.java
 * Exemple de herencia i constructors
 * @author Jose Moreno
 * @version
 */
public class Herencia4a {
```

```
    public static void main(String[] args) {
        System.out.println("Creacio d'un objecte Base");
        Base baseObj = new Base();
        System.out.println("Creacio d'un objecte Derivat");
        Derivat derivatObj = new Derivat();
    }
}
class Base {
    ComponentA aComp;
    ComponentB bComp;
    ComponentC cComp;
    public Base() {
        System.out.println("Nou objecte Base ");
    }
}
class ComponentA {
    public ComponentA() {
        System.out.println("Nou objecte ComponentA ");
    }
}
class ComponentB {
    public ComponentB() {
        System.out.println("Nou objecte ComponentB ");
    }
}
class ComponentC {
    public ComponentC() {
        System.out.println("Nou objecte ComponentC ");
    }
}
class Derivat extends Base {
    public Derivat() {
        System.out.println("Nou objecte Derivat ");
    }
}
```

La sortida del programa anterior ve a continuació. En cada instància d'objectes Derivat, s'instancia un objecte Base, tal com esperàvem. En canvi, no s'instancia cap objecte dels components, perquè els atributs només són referències.

```
Creacio d'un objecte Base
Nou objecte Base
Creacio d'un objecte Derivat
Nou objecte Base
Nou objecte Derivat
```

Exercici resolt Herencia4b: Agregació i herència

Modificar l'exemple anterior de manera que els constructors inicialitzin els components.

A continuació tenim el codi demanat.

```
/**
 * Herencia4b.java
 * Exemple de herencia i constructors
 * @author Jose Moreno
 * @version
 */
public class Herencia4b {
```

```
    public static void main(String[] args) {
        ComponentA aObj = new ComponentA();
        ComponentB bObj = new ComponentB();
        ComponentC cObj = new ComponentC();
        System.out.println("Creacio d'un objecte Base");
        Base baseObj = new Base(aObj, bObj, cObj);
        System.out.println("Creacio d'un objecte Derivat");
        Derivat derivatObj = new Derivat(aObj, bObj, cObj);
    }
}
class Base {
    ComponentA aComp;
    ComponentB bComp;
    ComponentC cComp;
    public Base(ComponentA a, ComponentB b, ComponentC c) {
        this.aComp = a;
        this.bComp = b;
        this.cComp = c;
        System.out.println("Nou objecte Base ");
    }
}
class ComponentA {
    public ComponentA() {
        System.out.println("Nou objecte ComponentA ");
    }
}
class ComponentB {
    public ComponentB() {
        System.out.println("Nou objecte ComponentB ");
    }
}
class ComponentC {
    public ComponentC() {
        System.out.println("Nou objecte ComponentC ");
    }
}
class Derivat extends Base {
    public Derivat(ComponentA a, ComponentB b, ComponentC c) {
        super(a, b, c);
        System.out.println("Nou objecte Derivat ");
    }
}
```

La sortida del programa anterior ve a continuació. Els components s'instancien en main(). Les classes Base i Derivat només reben referències als objectes.

```
Nou objecte ComponentA
Nou objecte ComponentB
Nou objecte ComponentC
Creacio d'un objecte Base
Nou objecte Base
Creacio d'un objecte Derivat
Nou objecte Base
Nou objecte Derivat
```

Sobrecàrrega i redefinició de mètodes

S'anomena **signatura** d'un mètode a la relació ordenada de tipus de paràmetre que accepta.

La **sobrecàrrega de mètodes** es produeix quan en un mateix àmbit de resolució de noms es defineixen dos o més mètodes amb el mateix identificador i diferent signatura. Els operadors, al igual que els altres mètodes, també poden ser sobrecarregats. Això no obstant, no tots els llenguatges de programació accepten la sobrecàrrega d'operadors. El llenguatge Java no admet la sobrecàrrega d'operadors, només de mètodes. L'única excepció és l'operador **+** per als String.

La relació entre la invocació a un mètode i el mètode invocat es diu **enllaç**. La sobrecàrrega es produeix en temps de compilació. En aquest cas es diu que es té **enllaç estàtic o primerenc**.

Per exemple, en Java és possible sobrecarregar un mètode dintre de la mateixa classe.

```
public class SobrecarregaTest {
    public static void main(String[] args) {
        Calculs c = new Calculs();
        System.out.println("2+3="+c.suma(2,3));
        System.out.println("2+3+4="+c.suma(2,3,4));
    }
}
class Calculs {
    int suma(int x, int y) {
        return (x+y);
    }
    int suma(int x, int y, int z) {
        return (x+y+z);
    }
}
```

La **redefinició** o **sobreescriptura** de mètodes, en canvi, es produeix quan es defineix novament un mètode (identificat amb el mateix nom, signatura i tipus de retorn) d'una classe en una classe derivada de l'anterior. La subclasse hereta els mètodes de la superclasse, però si es redefineix, la definició de la subclasse té preferència sobre la de la superclasse. El mètode de la subclasse fa invisible el de la superclasse.

La redefinició es pot produir en temps de compilació (enllaç estàtic) o en temps d'execució (**enllaç dinàmic o tardà**).