1.00Clase12

Herencia

Herencia

- La herencia permite escribir nuevas clases basadas ensuperclases existentes
  – Hereda los métodos y datos de la superclase
  – Añade nuevos métodos y datos
- Permite una reutilización considerable del códigoJava
  – Al ampliar elsoftware,a menudo hay que escribir nuevo código que invoque al antiguo(librerías,etc.)
  – A veces necesitamos que el código antiguo llame al nuevo (incluso a código que ni siquiera se había imaginado cuando se escribió el antiguo),sin cambiar (o incluso teniendo) el código antiguo.
    - P. ej.Un programa de dibujo debe gestionar una nueva forma
  – ¡La herencia nos permite hacerlo!

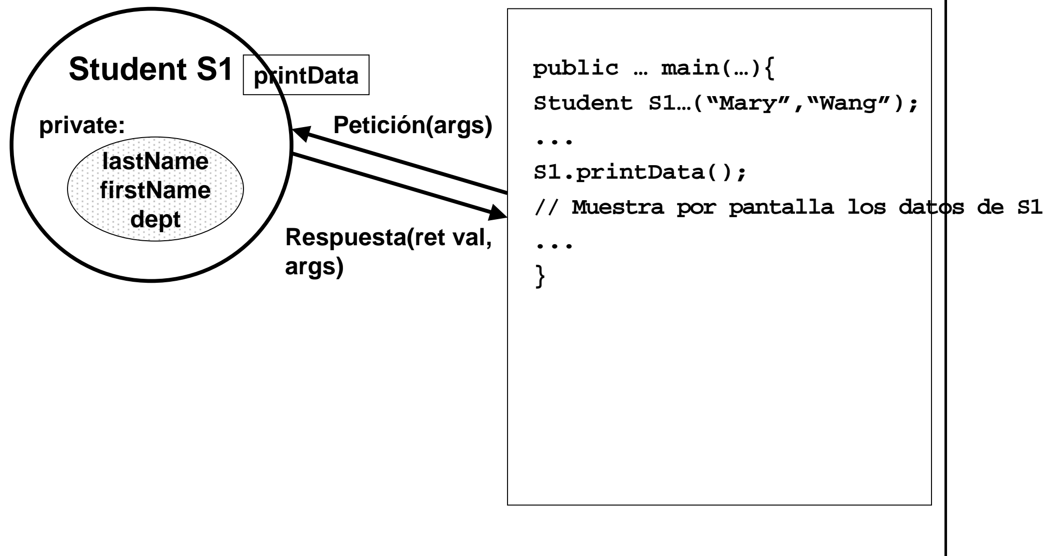
Miembros de una clase

- Una clase puede contener varios miembros (métodos o datos)de tipo:
  - Private:
    - Accesiblesólo para los métodos de la clase
  - Protected(apenas utilizado enJava;no es muy seguro)
    - Accesible para:
      - Los métodos de la clase
      - Los métodos de clases heredadas,llamadassubclases
      - Las clases del mismo paquete
  - Package:
    - Accesiblepara los métodos de las clases que pertenecen al mismo paquete
  - Public:
    - Accesiblepara todas las clases desde cualquier parte del programa

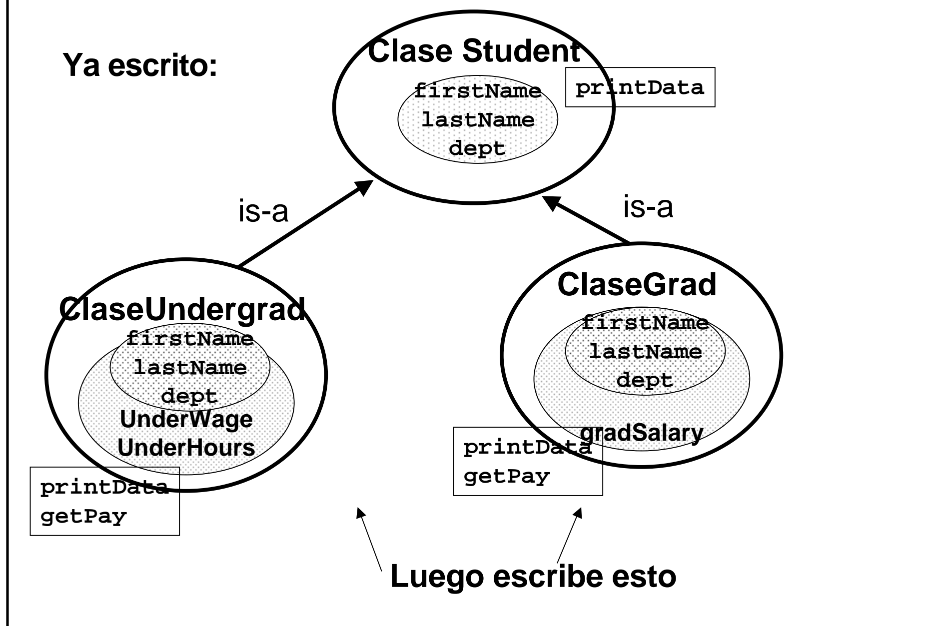
Un proyecto de programación

- Un departamento tiene un sistema con la claseStudent
  - Tiene varios datos(nombre,ID,cursos,año …) de los estudiantes que necesita utilizar/mostrar
  - El dpto. desea gestionar mejor los proyectos de investigación
    - Los estudiantes de licenciatura y los de posgrado tienen distintos papeles
      - Puestos,cr éditos/notas,pagos…
  - Desea reutilizar la claseStudent ,pero tiene que añadir datos y métodos muy diferentespara los estudiantes de licenciatura y para los de posgrado
    - Suponga quealguien escribió Studenthace 5años sin saber que podría ser utilizada para gestionar proyectos de investigación

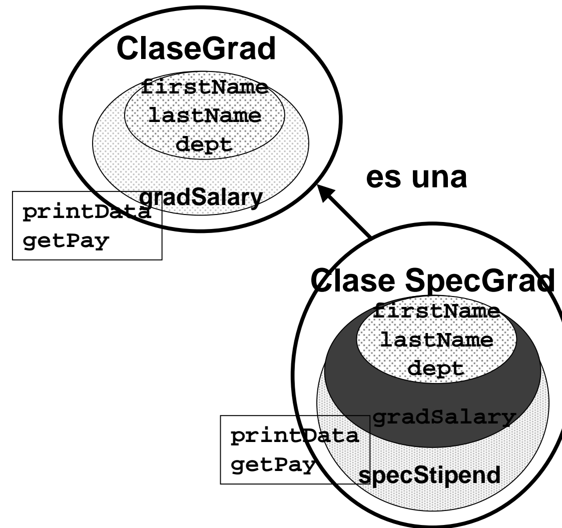
Clasesyobjetos



Herencia



Herencia 2



Ejemplo:clase Student

```
class Student {
    // Constructor
    public Student(String fName, String lName) {
        firstName= fName; lastName= lName;}
    // Método
    public void printData() {
        System.out.println(firstName + " " + lastName);}
    // Datos
    private String firstName;
    private String lastName;
}
```

Esta es la superclase o clase base

Clase Undergradderivada de Student

```
class Undergrad extends Student {        // 'Extends': palabra clave
    // Constructor: llama al constructor de la superclase indicada en la 1ª línea
    public Undergrad(String fName, String lName,
                     double rate, double hours) {
        super(fName, lName);      // Constructor de Student
        UnderWage= rate;
        UnderHours= hours; }

    public double getPay() {
        return UnderWage * UnderHours; }

    public void printData() {
        super.printData();        // Método getData de Student
        System.out.println("Paga semanal: $" + UnderWage *
                              UnderHours); }

    private double UnderWage;
    private double UnderHours;
}
```

Subclaseo clase (directamente) derivada

Clase Gradderivada de Student

```
class Grad extends Student {
    public Grad(String fName, String lName, double salary) {
        super(fName, lName);
        GradSalary= salary; }

    public double getPay() {
        return GradSalary; }

    public void printData() {
        super.printData();
        System.out.println("Salario mensual: $" + GradSalary); }

    private double GradSalary;
}
```

Subclaseo clase (directamente) derivada

Clase SpecGradderivada de Student

```
class SpecGrad extends Grad {
   public SpecGrad(String fName, String lName, double stipend) {
       super(fName, lName, 0.0);         // Zero monthly salary
       SpecStipend= stipend; }

   public double getPay() {
       return SpecStipend; }

   public void printData() {
       super.printData();
       System.out.println("Remuneración semestral: $" + getPay()); }
       // ¡Es peligroso utilizar getPay! Si se define una subclase,
       // ésta invocará a su propio getPay, no al de SpecGrad

   private double SpecStipend;
}
```

Clase derivada de otra clase derivada

Método Main

```
public class Student2 {
 public static void main(String[] args) {
   Undergrad Ferd= new Undergrad("Ferd", "Smith", 12.00, 8.0);
   Ferd.getData();
   Grad Ann= new Grad("Ann", "Brown", 1500.00);
   Ann.getData();
   SpecGrad Mary= new SpecGrad("Mary", "Barrett", 2000.00);
   Mary.getData();
   System.out.println("\n");

   // Polimorfismo, or late binding
   Student[] team= new Student[3];
   team[0]= Ferd;
   team[1]= Ann;
   team[2]= Mary;
   for (int i=0; i < 3; i++)
      team[i].printData();
    }
 }
```

Java conoce el tipo de objeto y elige el método apropiado en tiempo de ejecución

Salida delmétodo Main

```
Ferd Smith
Paga semanal: $96.0
Ann Brown
Salario mensual: $1500.0
Mary Barrett
Salario mensual: $0.0
Remuneración semestral: $2000.0
```

```
     Observe que no podemos escribir:
           team[i].getPay();
porque getPay() no es un método de la
superclase Student. Por el contrario, printData() es un método
de Student, por lo que Java puede hallar la versión apropiada.

     Tendríamos los mismos problemas con un método como isUROP
     que sólo se definiría para universitarios y no en Student.
```

Ejercicio

- Proyecte una jeraquía para:
  - Reino Plantae o vegetal: fotosintético y multicelular
  - Árboles:de madera perenne con tronco y co pa
  - Flores: polinizadoras, con estambre, pistilo y flor
  - Rosas: géneroRosa,hojas con recortes,tallo espinoso
  - Pinos:géneroPino,hojas en forma de aguja,formato de cono
- Pasos:
  - Dibuje primero el árbol de herencia.
  - Definaentre2 y 4campos de datosy un constructor para cada clase
    - ¿Qué tiene cada planta en común?Coloque eso en la superclase Plant.No use solamente las definiciones anteriores.
    - ¿Qué tiene cada subclaseen común?Colóquelas en la subclase apropriada;no utilice solamente las definiciones anteriores.
    - Tal vez necesite reorganizar cada información a medida que procede.
    - No defina ningún otro método aparte del constructor.
  - Si le sirve de ayuda, dibuje un "huevo" a medida que procede.

Ejercicio

```
class Plant {
  ____________________________
  ____________________________
  ____________________________
  ____________________________
  public Plant(_____________________________) {
  ____________________________
  ____________________________
  ____________________________
  ____________________________
  }
}
class Tree extends ______ {
  ____________________________
  ____________________________
  ____________________________
  ____________________________
  public Tree(_____________________________) {
  ____________________________
  ____________________________
  ____________________________
  ____________________________
  }
}
```

Ejercicio

```
class Flower extends ______ {
  ____________________________
  ____________________________
  public Flower(_____________________________) {
  ____________________________
  ____________________________
  }
}
class Rose extends ______ {
  ____________________________
  ____________________________
  public Rose(_____________________________) {
  ____________________________
  ____________________________
  ____________________________
  }
}
class Pine extends _____ {
  ____________________________
  ____________________________
  public Rose(_____________________________) {
  ____________________________
  ____________________________
  ____________________________
  }
}
```

Ejemplo de solución

```
class Plant {
    private String kingdom;
    private String genus;
    private String species;
    private boolean annual;
    public Plant(String g, String s, boolean a) {
        kingdom= "Plantae";
        genus= g;
        species= s;
        annual= a;
    }
}
class Tree extends Plant {
    private double crownSize;
    private double trunkSize;
    public Tree(String g, String s, double cs, double ts) {
        super(g, s, false);
        crownSize= cs;
        trunkSize= ts;
    }   }
```

Ejemplo de solución,p.2

```
class Flower extends Plant {
    private String blossomColor;
    public Flower(String g, String s, String bc, boolean a) {
        super(g, s, a);
        blossomColor= bc;
     }  }
class Rose extends Flower {
    private double thornDensity;
    public Rose(double td, String g, String s, String bc) {
        super(g, s, bc, false);
        thornDensity= td;
    }   }
class Pine extends Tree {
    private String needleType;
    private String coneType;
    public Pine(String g, String s, double cs, double ts,
                String nt, String ct) {
        super(g, s, cs, ts);
        needleType= nt;
        coneType= ct;
    }   }
```

Ejemplo de solución,p.3

```
public class PlantTest {
    public static void main(String[] args) {
        Plant p= new Plant("PGenus", "PSpecies", false);
        Tree t= new Tree("TGenus", "TSpecies", 15.0, 2.0);
        Flower f= new Flower("FGenus", "FSpecies", "rojo", true);
        Rose r= new Rose(1.0, "RGenus", "RSpecies", "amarillo");
        Pine pi= new Pine("PiGenus", "PiSpecies", 10.0, 1.0,
              "delgado", "grueso");
        System.exit(0);
    }
}

// Avance en el depurador para ver cómo se llama a los constructores
```

Constructores

- **La subclase hereda los constructores de la superclase**
  - **Los constructores son llamados en orden de herencia**

```
class Base{
    public Base() {
        System.out.println("Base"); }  }
class Derived extends Base {
    public Derived() {
        System.out.println("Derived"); }  }
class DerivedAgain extends Derived {
    public DerivedAgain() {
        System.out.println("Derived Again"); }  }
public class Constructor1 {
    public static void main(String[] args) {
            DerivedAgain Object1= new DerivedAgain();}}
```

```
Salida:
  Base
  Derived
  DerivedAgain
```

Constructor por defecto invocado a menos que otro sea llamado explícitamente. Algún constructor debe ser invocado.