Herencia y Polimorfismo en JAVA

CREACIÓN DE JERARQUÍAS DE CLASES EN JAVA MEDIANTE HERENCIA. USO DE POLIMORFISMO.

José Luis Redondo García.
Grupo Quercus Engineering Software, UEX

Herencia en el mundo real

Cosa

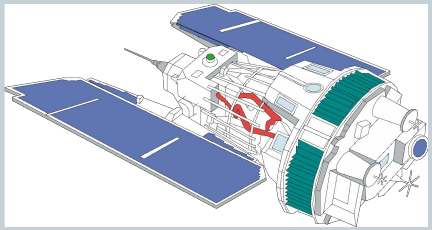
Medio de transporte

Medio de tele-comunicación

Objeto de oficina

Coche

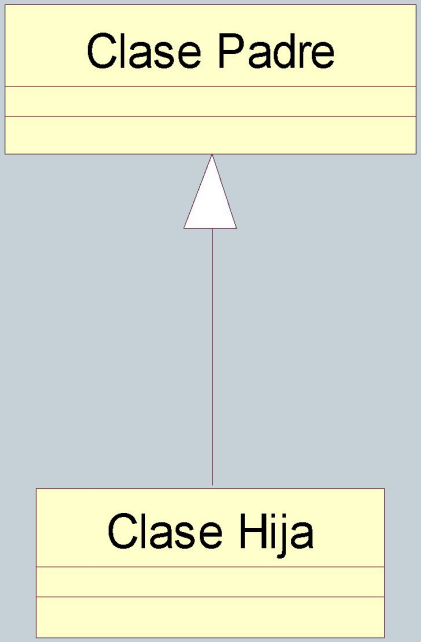
Vehiculo aéreo



Herencia en Java

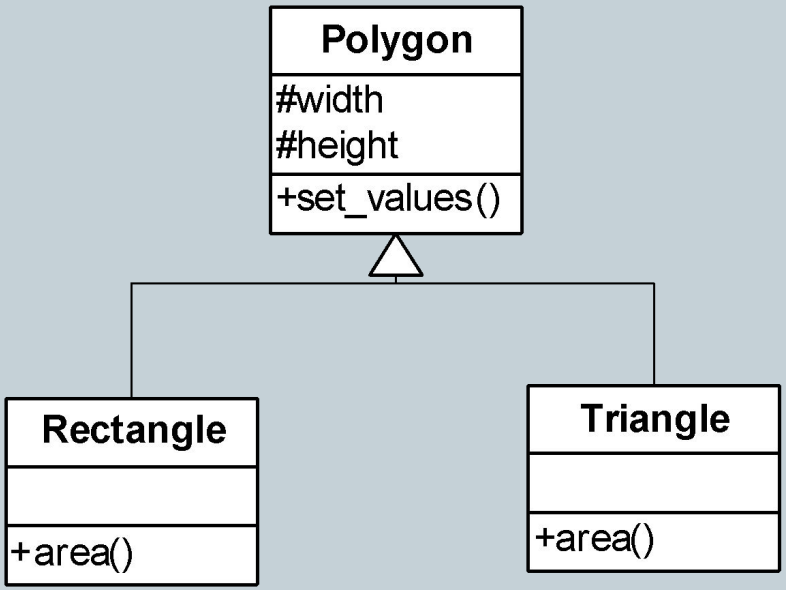
- Java permite definir una clase como subclase de una clase padre.

class *clase_hija* ***extends***
clase_padre
{
.....
}



Ejemplo de Herencia


```
class Polygon {
 protected int width, height;
 public void set_values (int a, int b) {
 width=a; height=b;} }
class Rectangle extends Polygon {
 public int area() { return (width * height); } }
class Triangle extends Polygon {
 public int area() { return (width * height / 2); }
 public static void main(String[] args) {
 Rectangle rect; Triangle trgl;
 rect = new Rectangle(); trgl = new Triangle();
 rect.set_values (4,5); trgl.set_values (4,5);
 System.out.print("area" + rect.area() + '\n' +
 trgl.area() + '\n');
}}
```



Constructores y Herencia

- Cuando se declara un obj de una clase derivada, se ejecutan los conts siguiendo el orden de derivación, es decir, primero el de la clase base, y después los conts de las clases derivadas de arriba a abajo.
- Para pasar parámetros al constructor de la clase padre:

  super (para1, para2, …, paraN)

Ejemplo de super


```
class Persona {
    private String nombre;
    private int edad;
    public Persona() {}
    public Persona (String n, int e)
    { nombre = n; edad = e; }
}
class Alumno extends Persona {
    private int curso;
    private String nivelAcademico;
    public Alumno (String n, int e, int c, String nivel) {
      super(n, e);
      curso = c; nivel_academico = nivel;
    }
    public static void main(String[] args) {
      Alumno a = new Alumno("Pepe", 1, 2, "bueno");
    }
}
```

Redefinir f. miembros de la clase padre


```
class Persona {
    private String nombre;
    private int edad;
    ..................
    public String toString() { return nombre + edad; }
    public void setEdad(int e) { edad = e; }
}
class Alumno extends Persona {
    private int curso;
    private String nivelAcademico;
    ..................
    public String toString() {
      return super.toString() + curso + nivelAcademico;
    }
    public void setCurso(int c) { curso = c; }
}
```

Referencias a objetos de clases hijas

- Si tenemos *ClaseHijo hijo = new ClaseHijo(…);*
- entonces es posible *padre=hijo* donde *padre* es una variable de tipo *ClasePadre*.
  - pero no es posible!! *hijo=padre* (sí que es posible con casting *hijo= (ClaseHijo) padre*)
- Ahora bien:
  - Con *padre* sólo podemos acceder a atributos y métodos def. en la clase padre.

Referencias a objetos de clases hijas


```
public static void main(String[] args) {
    Persona p;
    Alumno a = new Alumno("pepe",23,1,"universitario");
    p=a; //ref padre señala al objeto hijo
    // acceso al objeto hijo mediante la ref padre
    p.setEdad(24);
    /* no es posible acceder a un miembro de la clase hija
    usando una ref a la clase padre*/
    p.setCurso(88); // ERROR
    }
```

Ejemplo


```
class Persona { ..... }
class Alumno extends Persona {
    ......................
    public String toString() {
      return super.toString() + curso + nivelAcademico;
    }
}
class Profesor extends Persona {
    private String asignatura;
    public Profesor (String n, int e, String asign) {
      super(n, e);
      asignatura = asign;
    }
    public String toString() {
      return super.toString() + asignatura;
    }
}
```

Polimorfismo

- Una misma llamada ejecuta distintas sentencias dependiendo de la clase a la que pertenezca el objeto al que se aplica el método.
- Supongamos que declaramos: *Persona p;*
- Podría suceder que durante la ej. del programa, p referencie a un profesor o a un alumno en distintos momentos, y
- Entonces:
  - Si p referencia a un alumno, con p.toString(), se ejecuta el toString de la clase Alumno.
  - Si p referencia a un profesor, con p.toString(), se ejecuta el toString de la clase Profesor.
- **Enlace dinámico**: Se decide en **tiempo de ejecución** qué método se ejecuta.
- OJO!: Sobrecarga de fs => enlace estático (t. de compilación).

Ejemplo de Polimorfismo


```
public static void main(String[] args) {
  Persona v[]=new Persona[10];
// Se introducen alumnos, profesores y personas en v
  for (int i=0 ; i<10; i++)
    /* Se piden datos al usuario de profesor, alumno o persona */
    switch (tipo) {
      case /*  profesor */:  v[i] = new Profesor (....); break;
      case /* alumno */: v[i] = new Alumno(...); break;
      case /* persona */: v[i] = new Persona(...); break;
      default: /* ERROR */ }
    }
  for (int i=0 ; i<10; i++)
    System.out.println(v[i]);  // enlace dinámico con toString()
}
```

Métodos abstractos

- Tenemos un método f() aplicable a todos los objetos de la clase A.
  - Área de un polígono.
- La implementación del método es completamente diferente en cada subclase de A.
  - Área de un triángulo.
  - Área de un rectángulo.

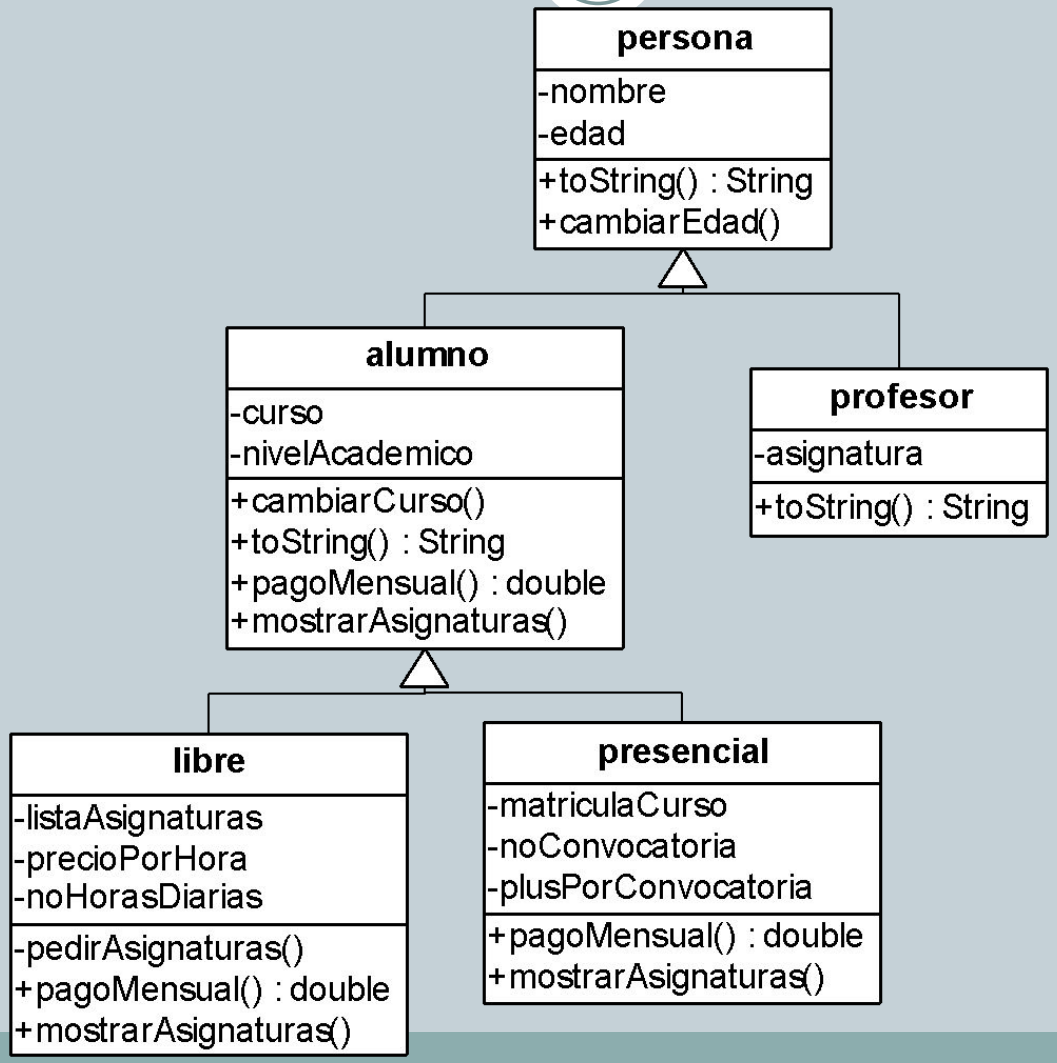
  .....
- Para declarar un método como abstracto, se pone delante la palabra reservada *abstract* y no define un cuerpo:
  abstract tipo nombreMétodo(....);
- Luego en cada subclase se define un método con la misma cabecera y distinto cuerpo.

Clases Abstractas

- Si una clase contiene al menos un método abstracto, entonces es una clase abstracta.
- Una **clase abstracta** es una clase de la que no se pueden crear objetos, pero puede ser utilizada como clase padre para otras clases.
- Declaración:

```
abstract class NombreClase {
    ..............
}
```

Ejemplo de clase abstracta



Ejemplo de clase abstracta


```
abstract class Alumno extends Persona {
    protected int curso;
    private String nivelAcademico;
    public Alumno (String n, int e, int c, String nivel) {
      super(n, e);
      curso = c; nivelAcademico = nivel;
    }
    public String toString() {
      return super.toString() + curso + nivelAcademico;
    }
    abstract double pagoMensual();
    abstract String getAsignaturas();
}
```

Ejemplo de clase abstracta


```
class Libre extends Alumno {
    private String []listaDeAsignaturas;
    private static float precioPorHora=10;
    private int noHorasDiarias;
    private void pedirAsignaturas() {}// se inicializa listaDeAsignaturas
    public double pagoMensual() {
      return precioPorHora*noHorasDiarias*30; }
    public String getAsignaturas() {
       String asignaturas="";
       for (int i=0; i<listaDeAsignaturas.length; i++)
             asignaturas += listaDeAsignaturas[i] + ' ';
       return asignaturas;
    }
    public Libre(String n, int e, int c, String nivel, int horas)
     {super(n,e,c,nivel); noHorasDiarias = horas; pedirAsignaturas(); }
}
```

Ejemplo de clase abstracta


```
class Presencial extends Alumno {
    private double matriculaCurso;
    private double plusPorConvocatoria;
    private int noConvocatoria;
    public double pagoMensual()
    { return
   (matriculaCurso+plusPorConvocatoria*noConvocatoria)/12; }
    public String getAsignaturas(){
      return "todas las del curso " + curso;
    }
    public Presencial(String n, int e, int c, String nivel,
            double mc, double pc, int nc) {
      super(n,e,c,nivel);
      matriculaCurso=mc;
      plusPorConvocatoria=pc;
      noConvocatoria=nc;
    }
}
```

Ejemplo de clase abstracta

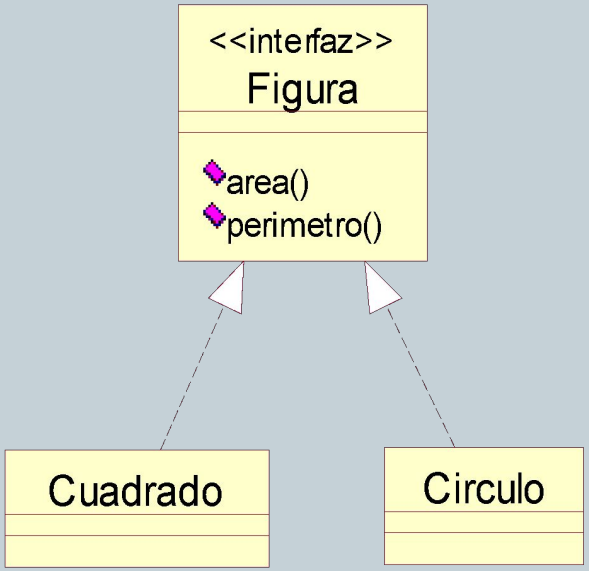

```
// FUNCIONES GLOBALES
void mostrarAsignaturas(Alumno v[]) {
 for (int i=0; i<v.length; i++)
  System.out.println(v[i].getAsignaturas());
  // enlace dinámico
}
double totalPagos(Alumno v[]) {
 double t=0;
 for (int i=0; i<v.length; i++)
  t += v[i].pagoMensual(); // enlace dinámico
 return t;
}
```

Interfaces

- Podría suceder que los objetos de varias clases compartan la capacidad de ejecutar un cierto conjunto de operaciones.
- Y dependiendo de la clase de objeto, cada operación se realice de diferente manera.
- Ejemplo:
  - Clases: Circulo, Elipse, Triangulo, ...
  - Todas esas clases incluyen los métodos: área, perimetro, cambiarEscala, etc.
- Podríamos definir una interfaz común que agrupe todos los métodos comunes (como métodos abstractos).
- Y luego definir varias clases de modo que implementen una misma interfaz.

Ejemplo de Interfaz


```
public interface Figura {
    abstract double area();
    abstract double perimetro();
}
public class Circulo implements Figura {
    private double radio;
    private static double PI=3.1416;
    ..............
    public double area() { return PI*radio*radio;
    }
    public double perimetro() { return
    2*PI*radio; }
}
public class Cuadrado implements Figura {
    private double lado;
    ..............
    public double area() { return lado*lado; }
    public double perimetro() { return 4*lado; }
}
```



Ejemplo de Interfaz

- Una interfaz puede incluir también definiciones de constantes a parte de métodos abstractos.
- Una misma clase puede implementar más de una interfaz ⇒ Herencia múltiple de interfaces
- Se pueden crear jerarquías de interfaces (con extends!!).
- Se pueden declarar referencias a objetos que implementen una cierta interfaz.

```
double totalArea(Figura v[]) {
 double t=0;
 for (int i=0; i<v.length; i++)
  t += v[i].area(); // enlace dinámico
 return t;
}
```

Igualdad y Asignación entre objetos

- El operador de asignación no sirve para crear una copia de un objeto.
- ¿Cómo crear una copia a nivel de bits?
  - Solución: utilizar el método *clone()*.
  - Para poder utilizarlo con los objetos de una clase A, la clase A debe implementar la interfaz *Cloneable* y se debe incluir el siguiente método clone() en la clase A:

```
public Object clone(){
    Object obj=null;
    try{
       obj=super.clone();
    }catch(CloneNotSupportedException ex){
       System.out.println(" no se puede duplicar");
    }
    return obj;
 }
```

Ejemplo con clone()


```
class Date implements Cloneable {
    .....
    public Object clone(){
            Object obj=null;
          try{
             obj=super.clone();
           }catch(CloneNotSupportedException ex){
             System.out.println(" no se puede duplicar");
           }
           return obj;
      }
    public static void main(String[] args) {
       Date ob1, ob2;
       ob1 = new Date(12, 4, 96);
       ob2 = (Date) ob1.clone(); // ob2 es una copia de ob1
       // las alias de ob1 y ob2 son diferentes
       System.out.println(ob1 == ob2);
       // el contenido de ob1 y ob2 es el mismo
       System.out.println(ob1.equals(ob2));
    }
}
```

Problemas con el clone()

- Si se quiere hacer una copia de un objeto que contiene otros objetos, no sirve el clone() que proporciona Java.
  - Ejemplo: clase Persona con un atributo fecha de nacimiento.
  - Solución: redefinir el método clone() para la clase Persona de modo que haga una copia del objeto fecha de nacimiento.

Ejemplo con clone() redefinido


```
public class Persona implements Cloneable {
    private String nombre;
    private Date fechaNacimiento;
    public Persona(String nombre, Date fechaNacimiento){
      this.nombre = nombre;
      this.fechaNacimiento = fechaNacimiento;
    }
    public Object clone(){
      return (new Persona(nombre, (Date)
    (fechaNacimiento.clone())));
    }
    public static void main(String[] args) {
      Persona p1, p2;
      p1 = new Persona("Pepe", new Date(1,1,2006));
      p2 = (Persona) p1.clone();
    }
}
```