Programación Avanzada
Material adicional 2013

Juan Manuel Fernández Peña

Arreglos

Uso de arreglos

- Un arreglo es una colección de datos del mismo tipo, a los que se puede acceder mediante un nombre común y un índice para seleccionar el elemento específico
- Para usarlos se usan tres etapas:
  - Declararlos
  - Iniciarlos
  - Emplearlos

Declaración de archivos

- <Tipo de los datos> [ ] <nombre del arreglo>

- private int [ ] datos;
- private String [ ] nombres;
- p4rivate Cuenta [ ] misCuentas;

Iniciar arreglos

- <nombre del arreglo> = new <tipo>[<número máximo de elementos] ;

- Datos = new int[100];
- Nombres = new String [20];
- misCuentas = new Cuenta[1000];

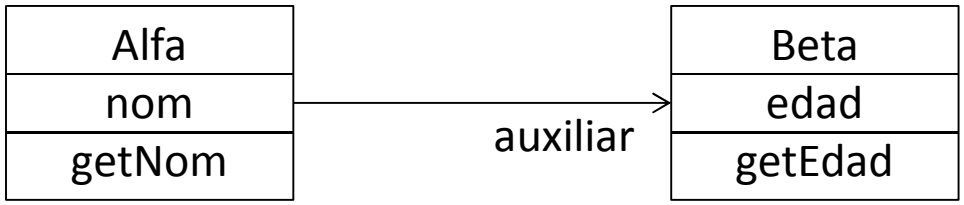
Emplear los arreglos

- <nombre del arreglo>[índice seleccionado]
- El índice va de cero al máximo menos uno

- datos[45] = 28; (se guarda un valor 28 en el lugar 45 del arreglo)
- nombres[0] = “El primer elemento”;
- misCuentas[17] = unaCuenta;
- misCuentas[18] = new Cuenta …
- misCuentas[3].getSaldo()

Codificación de relaciones

Relación simple (1)




```
public class Alfa{
  private String nom;
  Beta auxiliar;
  public Alfa(){
    auxiliar = new Beta();
  }
  public boolean algo(){
    int res = auxiliar.getEdad();
    if (res >18)
      return true;
    else
      return false;
  }
}
```

Declara un objeto Beta

Lo inicializa

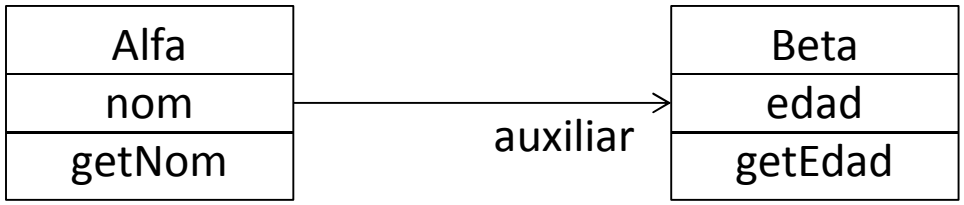
Lo usa

```
public class Beta{
  private int edad;

  public Beta(int e){
    edad = e;
  }
  public int getEdad(){
    return edad;
  }
}
```

Alfa sabe de Beta; Beta no sabe de Alfa.
Si Alfa muere, muere Beta

Relación simple (2)




```
Public class OtraClase{
  Beta unaB = new Beta();
  Alfa alef(unaB);
}
```

```
public class Alfa{
  private String nom;
  Beta auxiliar;
  public Alfa( Beta b){
    auxiliar = b;
  }
  public boolean algo(){
    int res = auxiliar.getEdad();
    if (res >18)
      return true;
    else
      return false;
  }
}
```

Declara un objeto Beta

Lo recibe de fuera

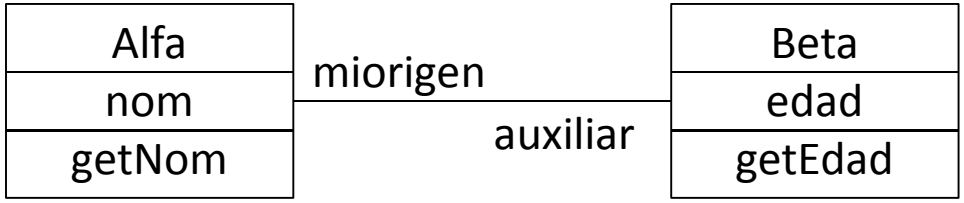
Lo usa

```
public class Beta{
  private int edad;

  public Beta(int e){
    edad = e;
  }
  public int getEdad(){
    return edad;
  }
}
```

Alfa sabe de Beta; Beta no sabe de Alfa.
Si Alfa muere, Beta sigue existiendo

Relación bidireccional (1)




```
public class Alfa{
  private String nom;
  Beta auxiliar;
  public Alfa(){
    auxiliar = new Beta(this);
  }
  public boolean algo(){
    int res = auxiliar.getEdad();
    if (res >18)
      return true;
    else
      return false;
  }
}
```

Declara un objeto Beta

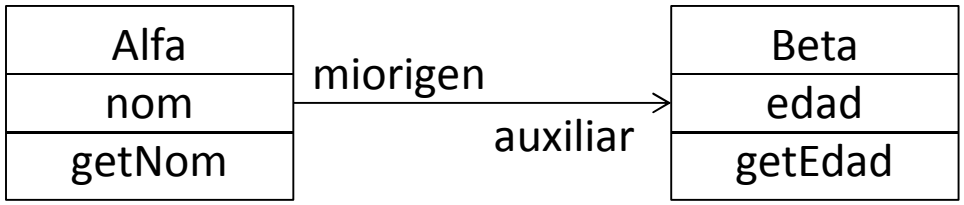
Lo inicializa y le manda su nombre

Lo usa

```
public class Beta{
  private int edad;
  Alfa miorigen;
  public BetaAlfa a, int e){
    miorigen = a;
    edad = e;
    a.algo();
  }
  public int getEdad(){
    return edad;
  }
}
```

Alfa sabe de Beta; Beta sabe de Alfa.
Si Alfa muere, muere Beta

Relación bidireccional (2)




```
Public class OtraClase{
   Beta unaB = new Beta(34);
   Alfa alef();
   unaB.presenta(alef);
   alef.presenta(unaB);
}
```

```
public class Alfa{
  private String nom;
  Beta auxiliar;
  public Alfa(){
    …
  }
  public boolean algo(){
    int res = auxiliar.getEdad();
    if (res >18)
      return true;
    else
      return false;
  }
  public void presenta(Beta b){
    auxiliar = b;
  }
}
```

Declara un objeto Beta

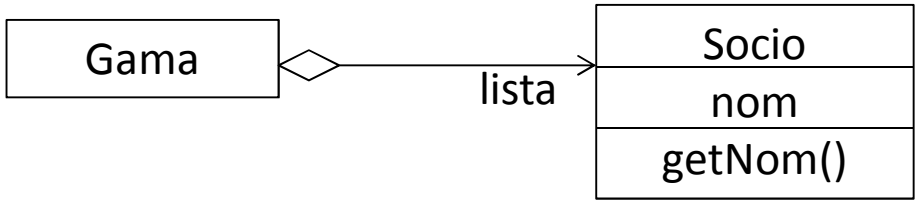
Lo usa

Lo recibe de fuera

```
public class Beta{
  private int edad;
  Alfa miorigen;
  public Beta(int e){
    edad = e;
  }
  public int getEdad(){
    return edad;
  }
  public void presenta(Alfa a){
    miorigen = a;
  }
}
```

Alfa sabe de Beta; Beta sabe de Alfa.
Si Alfa muere, Beta sigue existiendo

Relación múltiple (1)

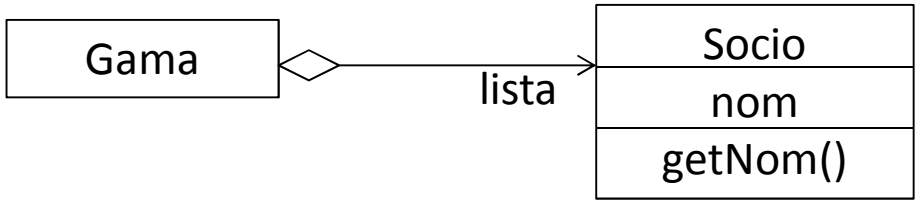



```
public class Gama{
  LinkedList<Socio> lista;
  public Gama(){
     lista = new LinkedList<Socio>();
  }
  public void agrega(String n){
    lista.add(new Socio(n));
  }
}
```

```
public class Socio{
  private String nom;
  public Socio(String n){
     nom = n;
  }
}
```

Los objetos de tipo Socio se crean en Gama; si Gama desaparece, desaparecen todos los objetos de su lista.

Relación múltiple (2)




```
public class Gama{
  LinkedList<Socio> lista;
  public Gama(){
     lista = new LinkedList<Socio>();
  }
  public void agrega(Socio s){
    lista.add(s);
  }
}
```

```
public class Socio{
  private String nom;
  public Socio(String n){
     nom = n;
  }
}
```

Los objetos de tipo Socio se en otra parte y se le pasan a Gama; si Gama desaparece, los objetos de su lista siguen existiendo, siempre que haya alguna otra referencia de ellos.

```
Public class OtraClase{
  Gama grupo = new Gama();
  Socio unS = new Socio("algo");
  grupo.agrega(unS);
}
```