Ministerio de Producción
Presidencia de la Nación

Ministerio de Educación y Deportes

Subsecretaría de Servicios Tecnológicos y Productivos

Programa 111 mil
VOS PODÉS SER UNO.

ENCADENAMIENTO DE CONSTRUCTORES

Temas

- Repaso de constructores
- Construcción de objetos con **encadenamiento** de constructores

Constructores – Repaso - 1

- Un constructor sirve para **inicializar los atributos** o estados de un objeto
  - Si una clase NO declara constructores, el compilador inserta automáticamente un constructor nulo
  - Si la clase declara al menos un constructor, con o sin argumentos, el compilador NO insertará el constructor nulo

- Una clase puede tener mas de un constructor
  → **constructores sobrecargados**

Constructores – Repaso - 2

Caso 1: Clase con constructores sobrecargados

```
public class Vehiculo {
private String nroPatente;
private String propietario;

public Vehiculo(String patente) {
this.nroPatente = patente;
}

public Vehiculo(String patente, String propietario) {
this(patente);
this.propietario= propietario;
}

public Vehiculo() { }
// métodos
}
```

Caso 2: Clase que NO declara un constructor

```
public class Vehiculo {

private String nroPatente="";
private Stri ng propietario="SinDueño";

//métodos
                El compilador agrega  public Vehiculo(){}
}
```

Creación de Objetos - 1

- La creación e inicialización de un objeto involucra los siguientes pasos:

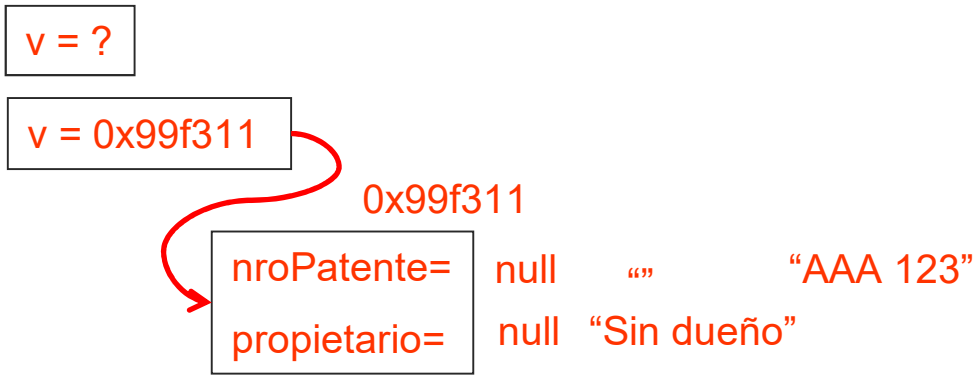
Vehiculo v= new Vehiculo(“DWL120”, “Juan García”);

1. Asignación de espacio en memoria para la variable **v** y para el objeto **Vehiculo**
2. Inicialización de las variables de instancia del objeto con los valores por defecto de acuerdo al tipo de dato
3. Re-seteo de las variables de instancia con el valor definido en la declaración (si éstos fueron definidos)
4. Ejecución del constructor. Re-seteo de las variables de instancia de acuerdo al código del constructor
5. Asignación a la variable v de la referencia del nuevo objeto

Creación de Objetos - 2

```
public class Vehiculo {
private String nroPatente="";
private String propietario="SinDueño";
public Vehiculo(String patente){
this.nroPatente = patente;
}
public Vehiculo(String patente, String propietario){
this(patente)
this.propietario= propietario;
}
public Vehiculo(){}
// métodos
}
```

Se declaran e inicializan las variables de instancia en la misma línea



Si se ejecuta el siguiente código:

Vehiculo v = new Vehiculo("AAA 123");

¿Cómo es el proceso de creación del objeto?

*1) Asignación de memoria para la variable **v** y el objeto **Vehiculo**.*

2) Inicialización de las variables de instancia. ¿Qué valores tienen nroPatente y propietario?

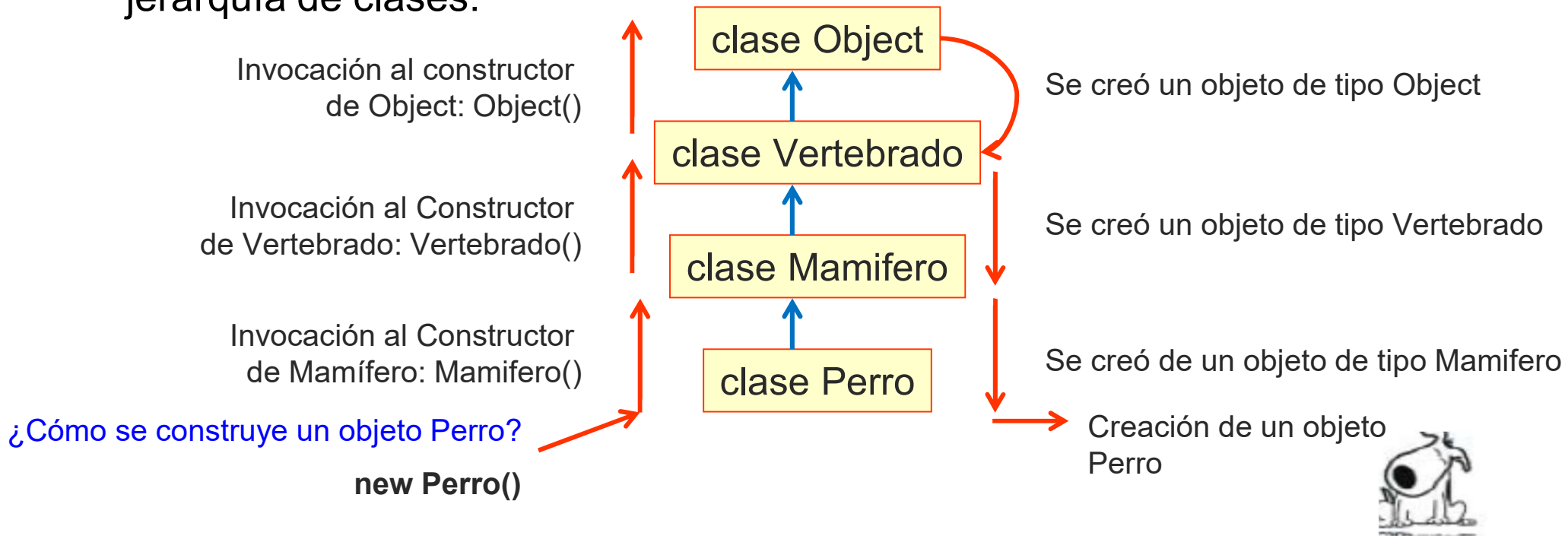
3) Re-seteo de las variables con los valores de la declaración. ¿Qué valores toman nroPatente y propietario?

4) Ejecución del constructor y re-seteo de las variables de instancia. ¿Qué valores toman nroPatente y propietario?

*5) Asignación a la variable **v** de la referencia al objeto ¿Qué valor toma **v**?*

Encadenamiento de Constructores - 1

- ¿Cómo se construye un objeto?
  - Recorriendo la jerarquía de herencia en forma ascendente e invocando al constructor de la superclase, desde cada constructor en cada nivel de la jerarquía de clases:



  - En el constructor de cada clase hay una invocación al constructor de la superclase.
  - Cada objeto contiene un referencia a un objeto de la superclase y ésta se crea en la invocación al constructor de la superclase. Mediante esta referencia, el objeto puede acceder a los métodos y variables de instancia de sus superclases.

Encadenamiento de Constructores - 2

- **Ejemplo:**
  - Disponemos de la clase **Vehiculo** y de su subclase **Automovil**. A su vez, **Automovil** define un atributo nuevo de tipo int llamado **cantidadPuertas**

```
public class Vehiculo {
private String nroPatente="";
private String propietario="SinDueño";
public Vehiculo(String patente){
this.nroPatente = patente;
}
public Vehiculo(String patente, String propietario)
{
this.nroPatente = patente;
this.propietario= propietario;
}
public Vehiculo(){}
// métodos
}
```

```
public class Automovil extends Vehiculo {
private int cantidadPuertas=4;

public Automovil(String patente, String propietario, int
puertas) {
   super(patente,propietario);

this.cantidadPuertas = puertas;

}
// métodos
```

Si en el constructor de Automovil quiero invocar al constructor de Vehiculo con los argumentos patente y propietario ¿Cómo lo hago?

Utilizo el super(). Este método es similar al this(), pero en lugar de invocar a un constructor de la misma clase, invoca a un constructor de la superclase. ¿Qué líneas de código agrego en el constructor de Automovil?

Encadenamiento de Constructores - 3

El compilador Java cuando compila una clase, **agrega el constructor nulo** (si la clase no define un constructor) e **inserta en todos los constructores una línea de código para invocar al constructor nulo de la superclase,** si es que explícitamente no se invoca a un constructor específico de la superclase.

```
public class Vertebrado {
private int cantPatas;

public Vertebrado(){
                    <-- El compilador agrega ..... super();
  System.out.println("Constructor de Vertebrado");
}

public void comer(){ }
}

public Mamifero(){  <-- El compilador agrega ..... super();

   System.out.println("Constructor de Mamifero");
}

public void comer(){  }
}
```

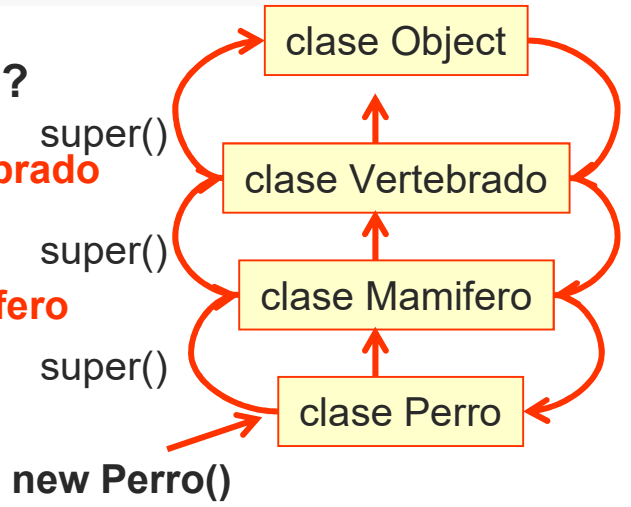
```
public class Perro extends Mamifeo{
. .
public Perro() {
                    <-- El compilador agrega ..... super();
System.out.println("Constructor de Perro");
}

public void comer(){ }
}
```

¿Cómo es la salida?

Constructor de Vertebrado

Constructor de Mamífero

Constructor de Perro



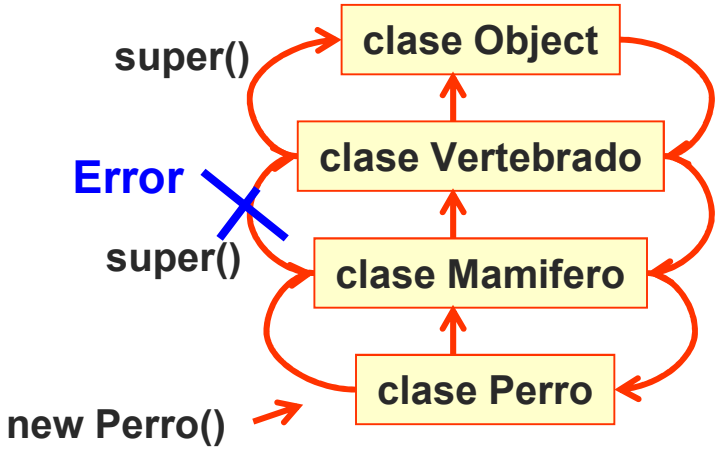
Encadenamiento de Constructores - 4

¿Qué pasa si Vertebrado declara solamente un constructor con argumentos?

```
public class Vertebrado {
private int cantPatas;
public Vertebrado(int c){ ..... >super();
   cantpatas= c;
   System.out.println("Constructor de Mamifero");
}
public void comer(){ }
}
```

```
public class Mamifero extends Vertebrado {
public Mamifero(){ ..... >super();
System.out.println("Constructor de Mamifero");
}
public void comer(){ }
}
```

```
public class Perro extends Mamifero{
public Perro(){ ..... >super();
 System.out.println("Constructor de Perro");
}
public void comer(){ }
}
```



ERROR DE COMPILACIÓN!!!

Desde el constructor de **Mamifero** NO se puede invocar al constructor nulo de Vertebrado porque no está definido

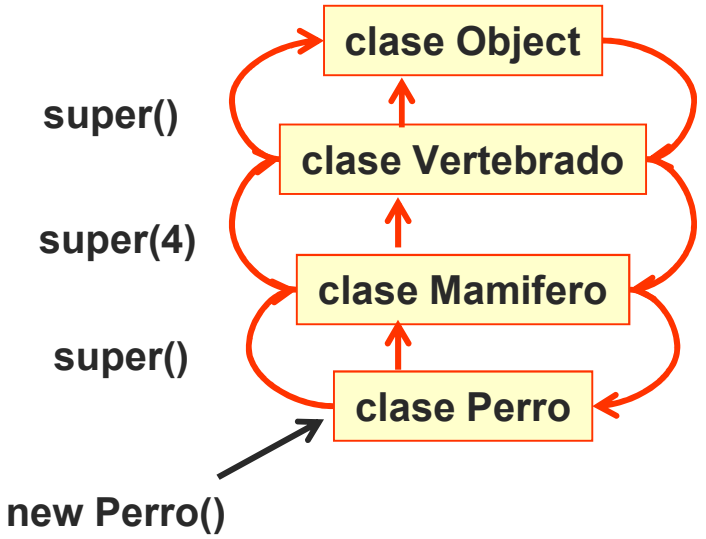
En este caso, es obligatorio invocar a un constructor definido **Vertebrado** usando la palabra clave **super(…)** y la lista de argumentos apropiada.

Encadenamiento de Constructores - 5

```
public class Vertebrado {
private int cantPatas;
public Vertebrado(int c){ ..... >super();
   cantpatas= c;
   System.out.println("Constructor de Mamifero");
}
public void comer(){ }
}
```

```
public class Mamifero extends Vertebrado {
public Mamifero(){
   super(4);
   System.out.println("Constructor de Mamifero");
}
public void comer(){ }
}
```

```
public class Perro extends Mamifero{
public Perro(){ ..... >super();
   System.out.println("Constructor de Perro");
}
public void comer(){ }
}
```



En cada constructor, el **compilador** inserta una línea de código para invocar al **constructor** nulo de la superclase: super().

En caso de no estar definido el **constructor** nulo en la superclase, es obligatorio invocar explícitamente a un constructor con argumentos de la superclase en la primera línea de código del constructor de la subclase.

Encadenamiento de Constructores - 6

super() y super(...)

- Permite invocar a un constructor de la superclase.
- La invocación al constructor de la superclase debe hacerse en la primera línea de código del constructor de la clase derivada, para garantizar que todos los datos de la superclase se inicialicen correctamente.

El código del constructor de **Perro** se terminará de ejecutar cuando se haya terminado de ejecutar el constructor de **Mamifero**.

```
public class Perro extends Mamifero {
private String sonido;
public Perro(){
   super(4);
   sonido=new String("guau");
 }
. . .
}
```

Se invoca al constructor de Mamifero con un argumento entero

¿Es posible usar **super()** en un método?¿y **this()**?

NO!!! super() y this() sólo se usan en constructores

La palabra clave super

- Permite invocar a un método sobreescrito de la superclase desde la subclase, o bien acceder a las variables de instancia “ocultas” de la superclase
- Todos los métodos de instancia disponen de la variable **super** (además de **this**) que contiene una referencia al objeto padre.

```
public class Usuario{
private String usrID;
public String getUsrID () {
return usrID;
}
public String setUsrID (String id){
usrID = id;
}
}
```

```
public class Administrador extends User{
private String pssw;
public void setPssw (){
// código que setea el pssw
}
public String getUsrID () {
return super.getUsrID() + "nombre secreto”;
}
}
```

En los métodos de la clase **Administrador**, el método **getUsrID()** de la clase **Usuario** NO es accesible usando simplemente el nombre.
Es necesario usar la palabra clave **super** para hacer referencia al método definido en el objeto padre.

```
public class Circulo {
public static final double PI= 3.14159;
public double r;
        public double circunferencia() {
            return 2 * PI * r;
        }
}
```

```
public class CirculoPlano extends Circulo {
public double r;
public double area(){
        return PI*super.r*super.r;
}
}
```

En los métodos de la clase **CirculoPlano**, la variable de instancia **r** de la clase **Circulo** NO es accesible usando simplemente el nombre.
Es necesario usar la palabra clave **super** para hacer referencia a la variable oculta del objeto padre.

Repaso de this y super

```
public class CuentaBancaria {
private int cuentaId;
private double saldo=0.0;
private String propietario="Sin Titular";
public Cuenta(int id, double saldo, String propietario){
this(id,propietario);
this.saldo=saldo;
}
public Cuenta(int id,String propietario){
this(id);
this.propietario=propietario;
}
public Cuenta(int id){
this.cuentaId=id;
}
public String toString(){
return "nroCuenta: "+this.getCuentaId()+" Propietario:
"+this.getPropietario()+" Saldo: "+this.getSaldo();
}
// setters y getters
}
```

this() para invocar a otro constructor de la misma clase

this para acceder a una variable de instancia y eliminar la ambigüedad de nombres

this para invocar a un método de instancia

```
public class CuentaCorriente extends Cuenta {
public CuentaCorriente(int id) {
super(id);
}
public CuentaCorriente(int id,String propietario) {
super(id, propietario);
}
public CuentaCorriente(int id, double saldo, String
prop) {
super(id, saldo, prop);
}
public String toString(){
return "CuentaCorriente "+super.toString();
}
}
```

super() para llamar al constructor de la superclase

super para invocar a un método sobreescrito de la superclase

```
public class TestCuenta {
public static void main(String[] args) {
CuentaCorriente c = new CuentaCorriente(1,"Federico");
System.out.println(c.toString());
}
}
```

¿Qué imprime?

CuentaCorriente nroCuenta: 1
Propietario: Federico Saldo: 0.0

¿Y si comento el método toString() de CuentaCorriente?

nroCuenta: 1 Propietario: Federico Saldo: 0.0

Referencias

- **TutorJava Nivel Básico(constructores):** http://www.programacion.com/java/tutorial/java_basico/18/
- **Tutorial de Java (inglés)**: http://java.sun.com/docs/books/tutorial/java/javaOO/constructors.html
- **Introducción a Java (constructores)**: http://www.programacion.com/java/tutorial/intjava/7/#constructores
- **Tutorial de Java en castellano**: http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/parte5/cap5-6.html
- **"Thinking in Java" de Bruce Eckel (inglés) para descarga libre**: http://www.etsimo.uniovi.es/eckel/
- **Java SE, sitio oficial de SUN:** http://java.sun.com/javase/whitepapers.jsp