Sobrecarga de constructores y métodos

Prof. Mg. Rafael Mellado S.
EII147 – Introducción a las tecnologías de información
Escuela de Ingeniería Industrial
Pontificia Universidad Católica de Valparaíso

Sobrecarga de constructores

- Una clase puede tener más de un constructor.
- Los constructores se diferencian por cantidad, tipo y orden de parámetros.
  - Ejemplo: constructores distintos de la clase Observación:

```
public Observación(){...
public Observación(int a){...
public Observación(int a, double b){...
public Observación(double a, int b){...
```

- Esto permite instanciar objetos considerando distintos tipos de datos disponibles.

208

Sobrecarga de constructores

```
public class Valor {
    private int x;
    private int y;

    public Valor(){
       x = 0;
       y = 0;
    }
    public Valor(int a, int b){
       x = a;
       y = b;
    }
    ...
}
```

Constructor sin parámetros

Constructor con dos parámetros tipo entero

209

Sobrecarga de constructores

```
public class Valor {
  private int x;
  private int y;

  public Valor(){
     x = 0;
     y = 0;
  }
  public Valor(int a, int b){
     x = a;
     y = b;
  }
  ...
}
```

Clase Valor

```
public class Ejemplo {
 public static void main...

   Valor ob1, ob2;
   ob1 = new Valor( 3, 5 );
   ob2 = new Valor();

   //Las siguientes fallan:
   Valor ob3, ob4;
   ob3 = new Valor( 2 );
   ob4 = new Valor( 2.0, 3.0);
   ...
 }
}
```

No existe el constructor de un parámetro int

No existe el constructor de dos parámetros double

Aplicación

210

Sobrecarga de métodos

- Una clase puede tener más de un método con el mismo nombre.
- Los métodos se diferencian por nombre del método, y cantidad, tipo y orden de sus parámetros. Todo esto constituye la "firma del método" (method signature).
- Ejemplo: métodos distintos de una clase:

```
public double sumaTiempo(){...
public double sumaTiempo(int a){...
public double sumaTiempo(double a){...
public double sumaTiempo(int a, double b){...
public double sumaTiempo(double a, int b){...
```

211

Sobrecarga de métodos

- Importante: el tipo de valor retornado no forma parte de la "firma del método" (no es utilizado para distinguir entre métodos).

```
public double sumaTiempo(int a){...
public double sumaTiempo(int b){...
public double sumaTiempo(int a){...
```

No existe el constructor Error Conceptual:
Java no distingue entre ellos: genera un error de compilación.

212

Sobrecarga de métodos

```
public class Persona {
  private String nombre;
  private int edad;

  public void setEdad(int x){
     edad = x;
  }
  public void setEdad(double x){
     edad = (int) x;
  }
  ...
}
```

Método setEdad(int)

Método setEdad(double)

213

Sobrecarga de métodos

```
public class Persona {

  private String nombre;
  private int edad;

  public void setEdad(int x){
     edad = x;
  }

  public void setEdad(double x){
     edad = (int) x;
  }
  ...
}
```

Clase Valor

```
public class Ejemplo {

  public static void main...

   Persona p1;
   p1 = new Persona();
   ...

   P1.setEdad( 35 );
   p1.setEdad( 35.0 );

      ...
 }
}
```

Aplicación

214

Sobrecarga de métodos y promoción de argumentos

```
public class Valor {

  private int dato;

  public void setEdad(short x){
     dato = x;
  }

  public void setEdad(int x){
     dato = x;
  }

  public void setEdad(double x){
     edad = (int) x;
  }
  ...
}
```

Clase Valor

```
public class Ejemplo {
 public static void main...

   Valor v = new Valor();
   ...

   v.setEdad( 35 );
   v.setEdad( 35.0 );

   // Aquí hay promoción:

   byte b = 3;
      v.setEdad( b );
   v.setEdad( 35f );
      ...
 }
}
```

Aplicación

215

Estructura general de una clase

public class ***IdentificadorClase*** **{**

Declaración variables de instancia

Declaración constructor 1 { cuerpo constructor 1 }
Declaración constructor 2 { cuerpo constructor 2 }
Declaración constructor *n* { cuerpo constructor *n* }

Declaración método 1 { cuerpo método 1 }
Declaración método 2 { cuerpo método 2 }
Declaración método *n* { cuerpo método *n* }

}

216

Uso de objetos

Prof. Mg. Rafael Mellado S.
EII147 – Introducción a las tecnologías de información
Escuela de Ingeniería Industrial
Pontificia Universidad Católica de Valparaíso

Uso de objetos

- Los objetos deben instanciarse.
- En la instanciación se invoca al constructor de la clase.
- Debe invocarse al constructor con los parámetros adecuados y declarados.
- Ejemplo:

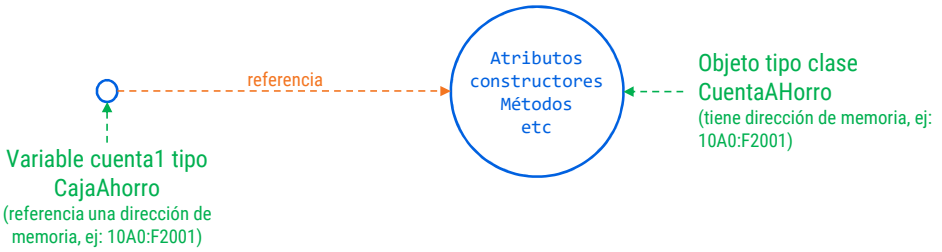
```
public class Banco
{
    public static void main(String[] args)
    {
        /*Se declara la referencia y se instancia la clase*/
        CajaAhorro cuenta1 = new CajaAhorro(2450003,1);
    }
}
```

192

Referencias

- Los objetos son manejados por referencia:

```
CajaAhorro cuenta1 = new CajaAhorro(2450003,1);
```



193

Referencias

- Ejemplos (rutear):

```
public class Banco
{
    public static void main(String[] args)
    {
        /*Crearemos dos cuentas*/
        CajaAhorro cuenta1, cuenta2, cuenta3;

        cuenta1 = new CajaAhorro(12505, 1500);
        cuenta2 = new CajaAhorro(12506, 78548);

        cuenta1.depositar(1000);
        cuenta2.depositar(500);
        cuenta3 = cuenta2;
        cuenta3.girar(50);

         System.out.println("El saldo de la cuenta 1 es: "+cuenta1.obtenerSaldo());
        System.out.println("El saldo de la cuenta 2 es: "+cuenta2.obtenerSaldo());
        System.out.println("El saldo de la cuenta 3 es: "+cuenta3.obtenerSaldo());
    }
}
```

194

Referencias

- Comparaciones:
  - La comparación de referencias permite determinar si dos variables **referencian a un mismo objeto** (comparación de direcciones).
  - No permite determinar si dos objetos son iguales.
- Una variable definida para referenciar objetos de una determinada clase, en cualquier momento puede:
  - Contener la dirección de memoria (referencia) de un objeto de la clase.
  - Contener la dirección null (dirección nula).

195

Referencias

- Pérdida de referencias y recolector de basura
  - Cuando un objeto deja de ser referenciado, se vuelve inaccesible.
  - El "recolector automático de basura" de Java (automatic garbage collector) lo destruye, liberando la memoria utilizada.
  - Los objetos que no son referenciados no pueden ser accesados nuevamente. La información almacenada se pierde con ellos.
  - El recolector de basura toma los objetos que no se encuentren referenciados y los destruye, liberando espacios de memoria.

196

Referencias

- Formas más comunes (voluntarias o involuntarias) de perder un objeto que cuenta con una única referencia:

  - Asignar a la variable de referencia otro objeto también existente (ejemplo anterior).
  - Instanciar un nuevo objeto y asignarlo a la variable de referencia.
  - Asignar la dirección **null** a la variable de referencia.

197

Referencias

- No olvidar que cuando se pierde una referencia, se pierde todo, esto puede ser a propósito, aunque parezca sin querer.

198

Arreglos de referencias objetos

- Se declaran e instancian como los arreglos de tipos primitivos:

```
Clase[] variable = new Clase[entero];
```

- Ejemplo:

```
CajaAhorro[] cuentas = new CajaAhorro[200];
```

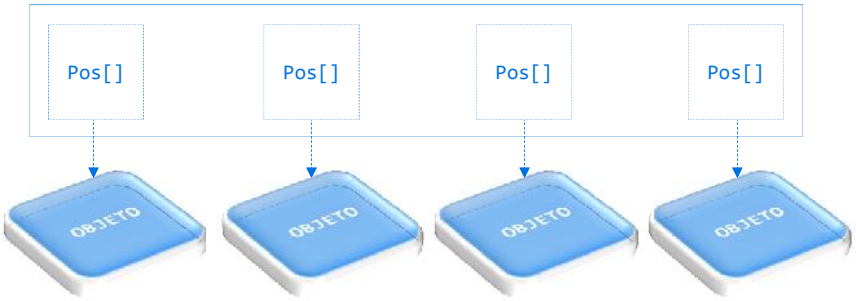
199

Arreglos de referencias objetos

- Un arreglo de objetos puede almacenar en cada posición una referencia a un objeto de la clase con que fue definido, o también la dirección **null**.
- La instanciación de un arreglo no instancia los objetos que puede referenciar.
- Los objetos a referenciar desde un arreglo deben instanciarse individualmente.

200

Arreglos de referencias objetos



201

Arreglos de referencias objetos

- Se debe instanciar cada objeto y luego asociarlo al arreglo:

```
variable = new Constructor( parámetros );
nombreArreglo[entero] = variable;
```

- También se puede instanciar y asociar inmediatamente:

```
nombreArreglo[entero] = new Constructor( parámetros );
```

- NOTA: también es posible asociar dos o más posiciones a un mismo objeto.

202

Arreglos de referencias objetos: ejemplo

- Utilizaremos la clase CajaAhorro que se ha venido utilizando:
  - Se creará un arreglo con 100 “clientes”, es decir, un arreglo que podrá contener hasta 100 referencias de tipo CajaAhorro.
  - Se llenarán 10 clientes
  - Se mostrarán por pantalla validando que exista la referencia.

Código disponible en:
https://goo.gl/q5jcgy

203

Arreglos de referencias objetos: ejemplo

```
public class CajaAhorro
{
    private int numeroCuenta;
    private int saldo;
    private int cantidadTransacciones;

    public CajaAhorro(int numeroCuenta, int saldo)
    {
        this.numeroCuenta = numeroCuenta;
        this.saldo = saldo;
        cantidadTransacciones=0;
    }

    public void depositar(int monto)
    {
        saldo=saldo+monto;
        cantidadTransacciones++;
    }

    public void girar(int monto)
    {
        saldo=saldo-monto;
        cantidadTransacciones++;
    }

    public int obtenerSaldo()
    {
        return saldo;
    }

    public int obtenerCantidadTransacciones()
    {
        return cantidadTransacciones;
    }
}
```

204

Arreglos de referencias objetos: ejemplo

```
package banco;
import java.io.*;
/**
 * @author Rafael Mellado
 * rafael.mellado@pucv.cl
 * www.rafaelmellado.cl
 */
public class Banco
{
    public static void main(String[] args) throws IOException
    {
        /*Declaración e instanciación de nuestro arreglo*/
        CajaAhorro clientes[] = new CajaAhorro[100];

        /*Llenaremos 10 clientes*/
        BufferedReader lector = new  BufferedReader( new InputStreamReader( System.in ) );
        for(int i=0; i<10 && i<clientes.length; i++)
        {
            System.out.println("Ingrese el numero de la cuenta");
            int numeroCuenta = Integer.parseInt(lector.readLine());
            System.out.println("Ingrese el saldo inicial de la cuenta");
            int saldoInicial = Integer.parseInt(lector.readLine());

            CajaAhorro nuevoCliente = new CajaAhorro(numeroCuenta,saldoInicial);
            clientes[i]=nuevoCliente;
        }

        /*Se muestran los clientes asumiendo que quedaron compactos*/
        for(int i=0; i<clientes.length && clientes[i]!=null; i++)
        {
            System.out.println("Cuenta numero: "+clientes[i].obtenerSaldo());
            System.out.println("Saldo: $"+clientes[i].obtenerSaldo());
        }
    }
}
```

Se declara e instancia el arreglo

Estas dos líneas se podrían juntar en una sola asignando a la posición el llamado al constructor

205

Desafío

- Ahora debes crear una aplicación interactiva en donde:
  - A través de un menú le pregunte al usuario que cosa quiere hacer
  - Llame a los métodos del objeto dependiendo de las opciones que el usuario seleccione.
  - Pero… esta vez con muchos clientes, por ende, deberá buscar el cliente a aplicar los métodos dentro de un total de 2000 posibles clientes.

206

Introducción a la Orientación a Objetos

Prof. Mg. Rafael Mellado S.
EII147 – Introducción a las tecnologías de información
Escuela de Ingeniería Industrial
Pontificia Universidad Católica de Valparaíso

Clases y objetos

- Una clase es un **tipo** al cual pertenecen objetos o instancias de la clase.

Clase **Persona**:
Los objetos de esta clase tienen **nombre**, capacidad de **leer**, **dormir** y **respirar**

Estas son instancias de la clase **Persona**.

Clases y objetos

- Una clase es el **"plano"** que permite **"construir"** objeto de un determinado tipo.
- Conceptualmente las clases, en la POO, especifican:

Atributos
Datos que almacena el objeto (y que caracterizan su estado).

Comportamiento
Funcionalidades o tareas que realizan los objetos

136

Definición de clases

- Ejemplo **Clase Automóvil**:

Atributos
Patente
Marca
Modelo
Año

Comportamiento
Aumentar Velocidad
Disminuir Velocidad
Frenar

137

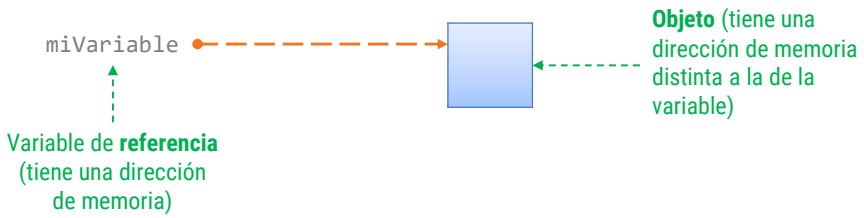
Clases y objetos

- Una clase en el lenguaje Java tiene esencialmente:
  - **Variables de instancia**: son variables que permiten almacenar los atributos de un objeto.
  - **Métodos**: procedimientos que implementan el comportamiento de los objetos de la clase.
- Además Java permite definir:
  - **Constructores**: procedimientos que se ejecutan en el momento de la instanciación del objeto (tienen el mismo nombre de la clase).

138

Clases y objetos

- Los objetos deben instanciarse (mediante el operador **new**).
- Cada objeto se accede desde una **variable de referencia**.
- Cada objeto tiene su propia identidad.



139

Instanciación de objetos

- Para utilizar un objeto primero se debe definir una variable que lo referenciará, con el formato:

```
Clase variable
```

- Luego se debe crear el objeto (instancia de clase), de la siguiente forma:

```
variable = new Constructor (lista de parámetros);
```

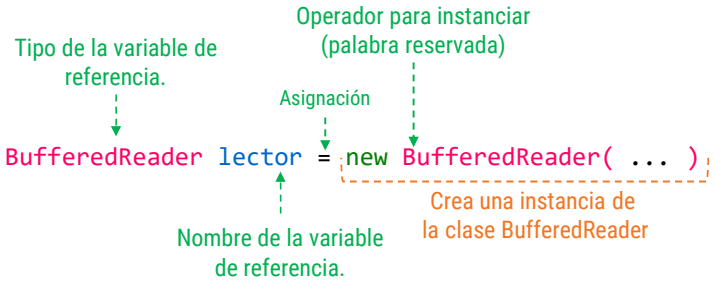
- Es posible realizar todo en una sola instrucción:

```
Clase variable = new Constructor (lista de parámetros);
```

140

Instanciación de objetos

- Ejemplo: recordemos la instanciación de un objeto de la clase *BufferedReader*:



141

Instanciación de objetos

- Importante:
  - La lista de parámetros son datos que se ha especificado como requeridos para crear el objeto.
  - Una misma clase puede tener distintas especificaciones de parámetros requeridos para su instanciación, o podría no requerirlos.
  - Los datos de la lista de parámetros se separan con comas.

142

Instanciación de objetos

- Constructor:
  - Es aquel que se invoca al momento de instanciar un objeto:

Aquí estamos llamando al constructor de la clase BufferedReader

```
BufferedReader lector = new BufferedReader( ... )
```

143

Instanciación de objetos

- Suponer la clase **CajaAhorro** que permite mantener el registro de depósitos y giros de una cuenta.
- Tiene los siguientes métodos:
  - depositar(int monto) : permite abonar el valor de monto a la cuenta.
  - girar(int monto): permite registrar un giro por el valor de monto.
  - obtenerSaldo(): retorna el saldo de la cuenta (valor int).
  - obtenerTransacciones(): retorna la cantidad total de transacciones (giros y depósitos) que se han hecho sobre la cuenta (valor int).

144

Instanciación de objetos

- Y el siguiente constructor:
  - CajaAhorro() : inicializa la cuenta con saldo y contador de transacciones en cero.
- Se creará una instancia de CajaAhorro y se accesarán sus métodos.

145

Ejemplo CajaAhorro

- Resolución:
  - Primero comenzaremos por la definición de la clase CajaAhorro.
  - Se identificarán 3 bloques: Atributos, Constructor y Métodos.
  - Posteriormente se organizarán en la clase
  - Finalmente se creará una aplicación (main) que utilizará esta clase

Código disponible en:
https://goo.gl/VjfH75

146

Ejemplo CajaAhorro

Esta es la clase **CajaAhorro**, dentro de ella (de su bloque de llaves) debe ir todo lo correspondiente a la clase

```
public class CajaAhorro
{

}
```

Estos son los **atributos** de la clase. Es importante que siempre debe existir un atributo que hará que el objeto sea único dentro de todos los objetos

```
private int numeroCuenta;
private int saldo;
private int cantidadTransacciones;
```

Este es el constructor de la clase CajaAhorro. Esto se llama (utiliza) automáticamente cuando se crea un objeto del tipo clase CajaAhorro

Estos son los parámetros del constructor

```
public CajaAhorro(int numeroCuenta, int saldo)
{
    this.numeroCuenta = numeroCuenta;
    this.saldo = saldo;
    cantidadTransacciones=0;
}
```

El operador this se utiliza para identificar que nos estamos refiriendo al atributo, ya que el parámetro se llama igual

147

Ejemplo CajaAhorro

- La definición de la clase queda así:

```
public class CajaAhorro
{
    private int numeroCuenta;
    private int saldo;
    private int cantidadTransacciones;

    public CajaAhorro(int numeroCuenta, int saldo)
    {
        this.numeroCuenta = numeroCuenta;
        this.saldo = saldo;
        cantidadTransacciones=0;
    }
}
```

- Posteriormente, se deben agregar los métodos.

148

Ejemplo CajaAhorro

- Ahora, tenemos los **métodos** solicitados

Recibe el monto a depositar y: (1) suma dicho monto al saldo actual, (2) aumenta la cantidad de transacciones

```
public void depositar(int monto)
{
    saldo=saldo+monto;
    cantidadTransacciones++;
}

public int obtenerSaldo()
{
    return saldo;
}
```

Devuelve (retorna) el valor del atributo saldo del objeto

Recibe el monto a girar y: (1) resta dicho monto al saldo actual, (2) aumenta la cantidad de transacciones

```
public void girar(int monto)
{
    saldo=saldo-monto;
    cantidadTransacciones++;
}

public int obtenerCantidadTransacciones()
{
    return cantidadTransacciones;
}
```

Devuelve (retorna) el valor del atributo cantidadTransacciones del objeto

149

Ejemplo CajaAhorro

```
public class CajaAhorro
{
    private int numeroCuenta;
    private int saldo;
    private int cantidadTransacciones;

    public CajaAhorro(int numeroCuenta, int saldo)
    {
        this.numeroCuenta = numeroCuenta;
        this.saldo = saldo;
        cantidadTransacciones=0;
    }

    public void depositar(int monto)
    {
        saldo=saldo+monto;
        cantidadTransacciones++;
    }

    public void girar(int monto)
    {
        saldo=saldo-monto;
        cantidadTransacciones++;
    }

    public int obtenerSaldo()
    {
        return saldo;
    }

    public int obtenerCantidadTransacciones()
    {
        return cantidadTransacciones;
    }
}
```

Atributos

Constructor

Métodos

150

Instanciación de objetos

```
package cajaahorro;

/**
 * @author Rafael Mellado
 * rafael.mellado@pucv.cl
 * www.rafaelmellado.cl
 */
public class Banco
{
    public static void main(String[] args)
    {
        /*Crearemos dos cuentas*/
        CajaAhorro cuenta1;
        CajaAhorro cuenta2;

        /*Se instancia la clase CajaAhorro*/
        cuenta1= new CajaAhorro(2450003,1);
        cuenta2= new CajaAhorro(12552,1500);

        /*Se utilizan los métodos*/
        cuenta1.depositar(500);
        System.out.println("El saldo de la cuenta 1 es: "+cuenta1.obtenerSaldo());

        cuenta2.girar(500);
        System.out.println("El saldo de la cuenta 2 es: "+cuenta2.obtenerSaldo());
    }
}
```

151

Instanciación de objetos

- Ahora debes crear una aplicación interactiva en donde:
  - A través de un menú le pregunte al usuario que cosa quiere hacer
  - Llame a los métodos del objeto dependiendo de las opciones que el usuario seleccione.

152

Posibilidades de la POO

- Usar de forma repetida clases previamente implementadas. Ejemplos:
  - BufferedReader
  - String
- Definir e implementar nuevas clases.

153

Pilares fundamentales

154

Abstracción

- Un objeto es capaz de desempeñar una función de forma completamente independiente del contexto en que éste es utilizado.
- En otras palabras, en cualquier ámbito (incluso diferente a aquel en que fue creado), un objeto expone las mismas propiedades y sus operaciones se comportan de la misma forma.

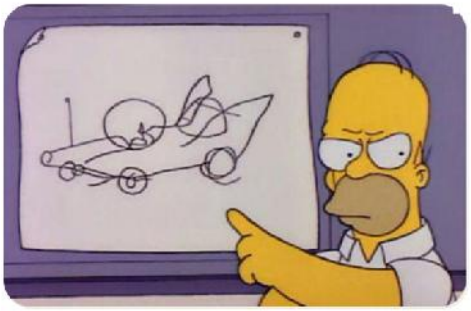
155

Abstracción

- Por ejemplo, Goofy se comporta como Goofy en la pelicula de Goofy y también en la película La Sirenita.

156

Abstracción



Énfasis en el ¿qué hace? mas que en el ¿cómo lo hace?

Homero Simpson construyendo el auto de sus sueños

157

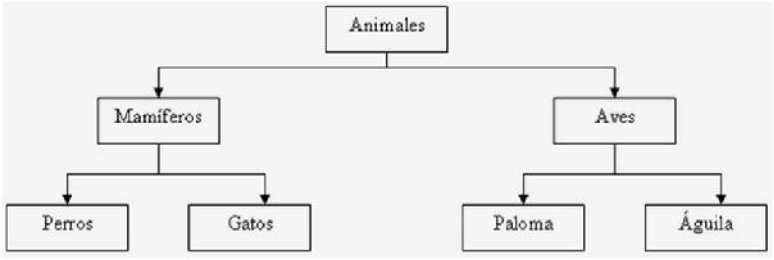
Encapsulamiento

- Un objeto es capaz de responder a peticiones sin exponer la forma en que estas son ejecutadas.
- En otras palabras, la invocación de una operación sobre un objeto gatilla un proceso cuyo efecto es logrado sin dar a conocer sus estructuras internas, ni sus algoritmos.

158

Herencia

- Una clase puede ser generada a partir de otra clase preexistente, heredando las propiedades de esta última.



159

Polimorfismo

- Objetos de un mismo (súper)tipo pueden realizar una misma operación de forma distinta.

160

Ejercicio grupal

- Crear una aplicación que sea capaz de crear una serie de personas (nombre, apellido, edad).
  - Debe crear la clase persona con su constructor respectivo.
- Leer sus datos y guardar sólo la que tenga mayor edad. (Esto requiere que no almacene todas las personas)
- Muestre sus datos por pantalla.

Implementación y uso de clases

Prof. Mg. Rafael Mellado S.
EII147 – Introducción a las tecnologías de información
Escuela de Ingeniería Industrial
Pontificia Universidad Católica de Valparaíso

Componentes de una clase

- Una clase en Java se compone de:
  - Variables de instancia
  - Constructor
  - Métodos
- A los anteriores se les conoce también como **miembros de la clase**.

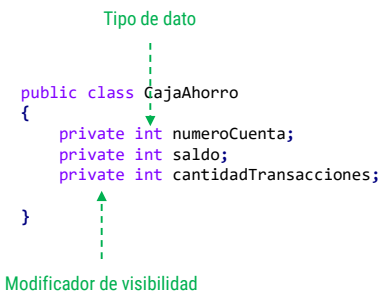
Variables de instancia

- Son los atributos de la clase.
- Se declaran fuera de cualquier constructor o método.
- Dependen de la naturaleza del problema que se esté resolviendo.
- Ejemplo, en la clase persona:
  - RUT, nombre, fecha de nacimiento.
- Ejemplo, en la clase automóvil:
  - Patente, marca, modelo, año fabricación.

164

Variables de instancia

- Declaración:

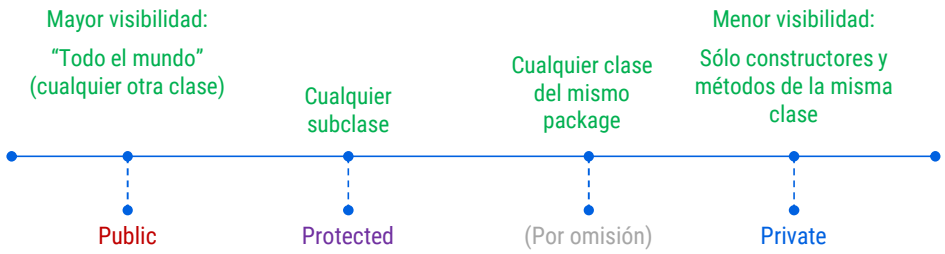


- Se declaran fuera de cualquier constructor o método.

165

Variables de instancia

– El "modificador de visibilidad" determina quién puede tener acceso a la variable de instancia:



166

Variables de instancia

– Acceso a una variable public desde una aplicación u otra clase:

varRefObjeto.varInstancia

```
public class Punto {
 public int x;
 public int y;
 public Punto(int a, int b){
    x=a;
    y=b;
 }
 ... Métodos...
}
```

Clase Punto

```
...
Punto origen = new Punto(0,0);
...
origen.x = 20;
origen.y = -30;
System.out.println(origen.x);
int z = origen.x * origen.y;
origen.y = origen.x;
...
```

Fragmento aplicación

167

Variables de instancia

– Variables private no pueden ser accesadas desde otras clases o aplicaciones.

```
public class Punto {
 public int x;
 public int y;
 public Punto(int a, int b){
    x=a;
    y=b;
 }
 ... Métodos...
}
```

Clase Punto

```
...
Punto origen = new Punto(0,0);
...
origen.x = 20;
origen.y = -30;
System.out.println(origen.x);
int z = origen.x * origen.y;
origen.x = origen.x;
...
```

Fragmento aplicación

168

Variables de instancia

Public

1. Pueden ser accesadas desde cualquier parte.
2. Acceso directo (más simple)
3. Las variables pueden recibir cualquier valor del tipo correspondiente, incluso valores "no aceptables".
4. El objeto puede resultar "inconsistente".

Private

1. Sólo las pueden acceder constructores y métodos del objeto.
2. Acceso restringido
3. Las variables sólo reciben valores que los constructores y métodos permitan.
4. Si los métodos están correctos y hacen las validaciones adecuadas, el objeto no cae en inconsistencias

169

Constructores

- Contienen las instrucciones que se ejecutan al momento de crear una instancia de clase.
- Tienen el mismo nombre que la clase.
- Normalmente se utilizan para inicializar las variables de instancia.
- Pueden recibir valores por parámetro.
- Todas las clases deben tener un constructor, el que se utilizará en la instanciación de objetos de la clase.

170

Constructores

- Si se omite la implementación del constructor de una clase, Java proporciona automáticamente un constructor sin parámetros y sin instrucciones.
- Ejemplo constructor con y sin parámetros:

```
public CajaAhorro(int numeroCuenta, int saldo)
{
    this.numeroCuenta = numeroCuenta;
    this.saldo = saldo;
    cantidadTransacciones=0;
}
```

```
public CajaAhorro( )
{
    numeroCuenta = 0;
    saldo = 0;
    cantidadTransacciones= 0;
}
```

171

Métodos

- Un método es un conjunto de instrucciones que permiten a un objeto realizar una tarea que le es propia.
- En los primeros lenguajes orientados a objeto se hablaba de los “métodos para...”.
- Ejemplo:
  - método para abonar
  - método para girar

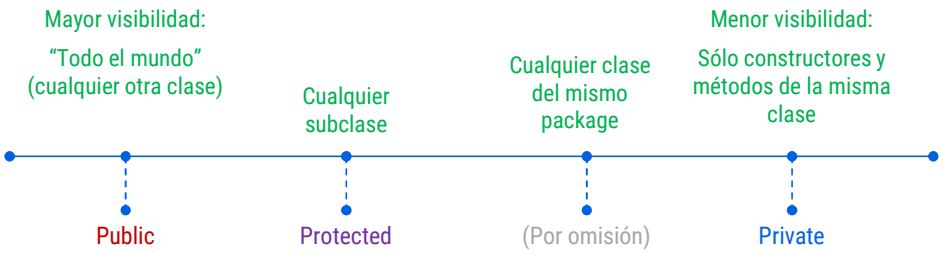
172

Métodos

173

Métodos

- El “modificador de visibilidad” determina quién puede invocar el método:



174

Métodos

- Los métodos pueden o no retornar un valor.
- El modificador de tipo de valor retornado puede ser:
  - void: el método no retorna valor.
  - Un tipo primitivo: short, byte, int, long, char, boolean, float, double.
  - Una clase: String, CuentaCorriente, Pez, etc.
  - Un arreglo de tipos primitivos o clases: int[ ], String[ ], String[ ][ ][ ], Pez[ ], etc.
- Cada método puede retornar como máximo un único tipo de valor.

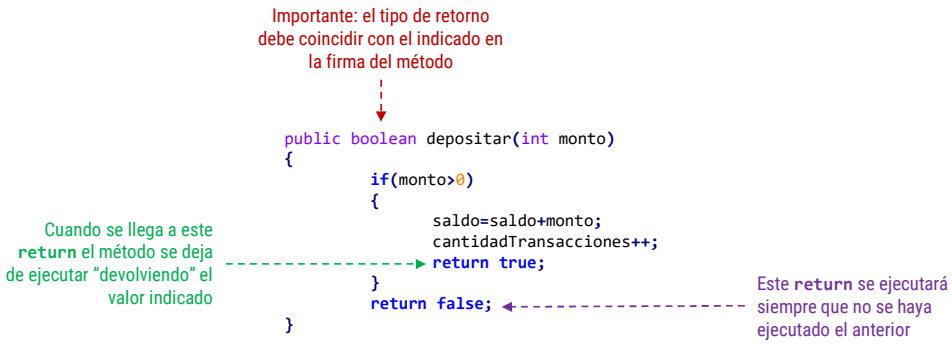
175

Métodos

- Para retornar valores se utiliza la instrucción `return`. Formato:
  - `return valor;`
  - `return variable;`
- El flujo de un método termina cuando se alcanza una instrucción return.
- Todos los flujos de control de un método no void deben terminar en un return que retorne un valor del tipo indicado en la declaración del método.
- Ejemplo, validaremos si el monto a depositar es positivo. Si lo es, entonces retornaremos true sino false.

176

Métodos



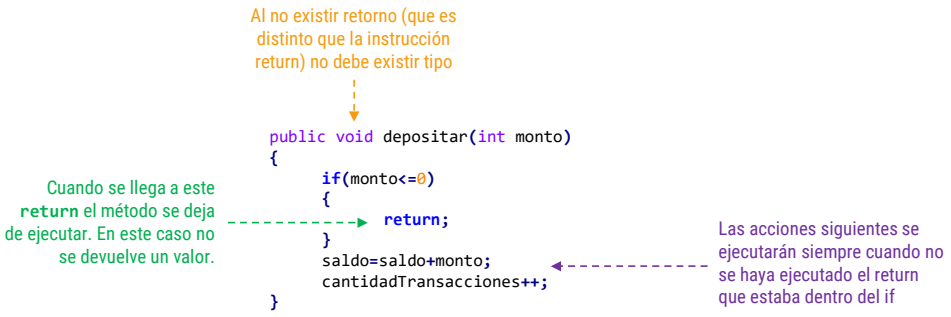
177

Métodos

- Un método void puede incluir la sentencia `return` sin valor:
  - `return;`
- Esto permite interrumpir la ejecución del método cuando la instrucción `return` se alcanza.
- Para el ejemplo anterior, asumiremos un cambio en la estructura del método, por ende, no devolverá un valor sino que cuando no cumpla la condición el método se interrumpirá.

178

Métodos



179

Métodos

- El nombre de cada método es arbitrario.
- Debe ser representativo de la función que cumple. Por ejemplo abonar es el "método para abonar dinero en la CuentaAhorro".
- Convención:
  - `primeraLetraEnMinúscula`
- Ejemplo: consultarSaldo, inscribirCurso, determinarSiguienteTrabajo.

180

Métodos

- Son variables que reciben valores que el constructor o método requiere para cumplir su función, y que no correspondan a variables de instancia.
- No todos los constructores, ni todos los métodos, requieren parámetros.
- Cada parámetro es de un tipo primitivo, clase o arreglo.

181

Métodos

- Formato para declarar parámetros:
  - (tipo1 var1, tipo2 var2, ..., tipoN varN)
- Por ejemplo:
  - `public String determinarNombre(String rut)`
  - `public boolean crearFicha(int codigo, String nombre)`
  - `public void asignarCoordenada(float x, float y )`
  - `public void asignarCoordenada(float x, y ) : Error!`

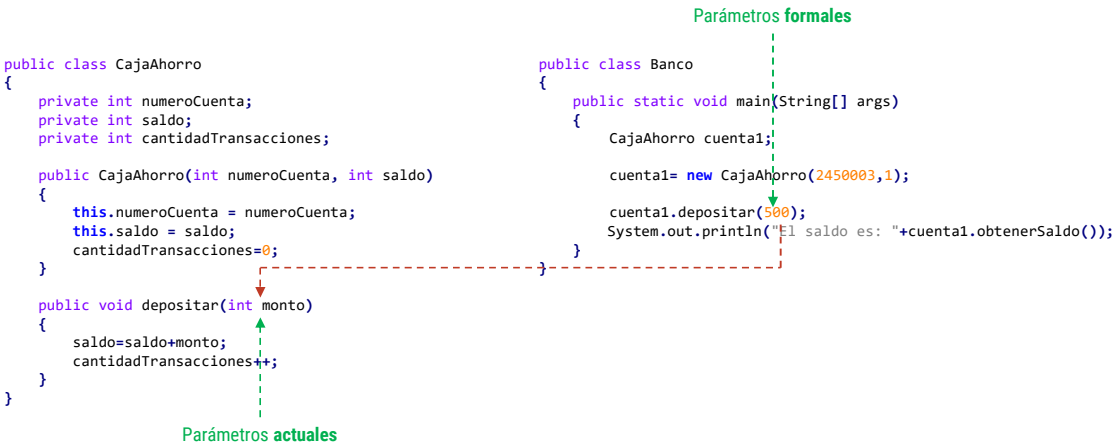
182

Parámetros formales y actuales

- Formales: son las variables que se especifican en la lista de parámetros de cada constructor o método.
- Actuales: son las variables o valores que se indican en el punto de llamado de un constructor o método.
- Parámetros formales y actuales deben coincidir en cantidad, tipo y significado.

183

Parámetros formales y actuales



Parámetros formales y actuales

- Los parámetros permiten a los métodos recibir valores traspasados desde el punto de llamado (traspaso de parámetros por valor).
- Si un constructor o método tiene parámetros denominados igual que variables de instancia, los primeros ocultan a estas últimas al interior del referido constructor o método.

Parámetros formales y actuales

- Haremos un cambio al constructor de la clase CajaAhorro para ejemplificar:

```
public class CajaAhorro
{
    private int numeroCuenta;
    private int saldo;
    private int cantidadTransacciones;

    public CajaAhorro(int numeroCuenta, int saldo)
    {
        numeroCuenta = numeroCuenta;
        saldo = saldo;
        cantidadTransacciones=0;
    }
}
```

Los valores de los parámetros que se llaman igual a los atributos se asignan a ellos mismos

186

Palabra reservada this

- La palabra reservada **this** permite hacer una autorreferencia al objeto, y acceder a variables de instancia ocultas por coincidencia de nombre con parámetros en constructores o métodos.

```
public class CajaAhorro
{
    private int numeroCuenta;
    private int saldo;
    private int cantidadTransacciones;

    public CajaAhorro(int numeroCuenta, int saldo)
    {
        this.numeroCuenta = numeroCuenta;
        this.saldo = saldo;
        cantidadTransacciones=0;
    }
}
```

Los valores de los parámetros se asignan a las variables de instancia

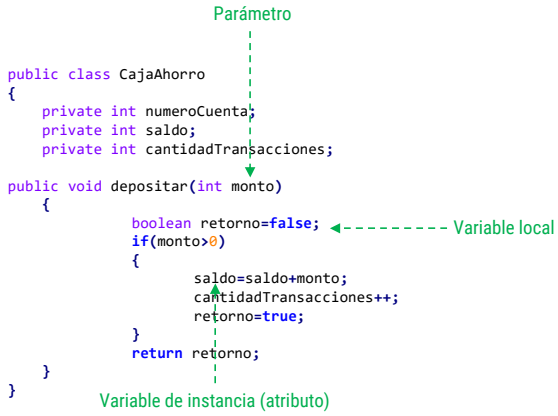
187

Variables en constructores y métodos

- Las variables que constructores y métodos pueden utilizar pueden ser:
  - Variables de instancia del objeto, sean public, private o protected.
  - Parámetros del constructor/método.
  - Variables locales: variables declaradas dentro del cuerpo del constructor o método. Se crean y utilizan en cada ejecución del constructor o método. No existen fuera de él (sus valores se pierden al terminar el constructor o método que las creó).

188

Variables en constructores y métodos



189

Variables locales

- Las variables locales son variables de función auxiliar dentro de constructores y métodos.
- Criterio elemental de diseño: las variables auxiliares siempre deben declararse como variables locales, nunca como variables de instancia... aunque “igual funcione”.

190