Tema 3: Clases y Objetos

Introducción a la Programación
Grado en Ingeniería Informática, EPI Gijón

{jdiez,oluaces,juanjo}@uniovi.es

Departamento de Informática - Universidad de Oviedo en Gijón

Objetivos

- Comprender la diferencia entre los conceptos clase y objeto.
- Utilizar objetos de clases ya implementadas.
- Ser capaz de diseñar un clase sencilla, con atributos y métodos.
- Entender la diferencia entre la representación en memoria de los tipos básicos y los tipos referenciados.
- Comprender las ventajas que ofrece la encapsulación y cómo se logra al programar una clase.

Contenidos

1 Introducción

2 Clases y objetos

3 Atributos

4 Métodos

5 Uso de objetos

6 Encapsulación: público/privado

7 Documentación Javadoc

8 Reutilización

9 Representación en memoria

Programación Orientada a Objetos

Principios básicos

- Analiza los objetos que intervienen en un programa:
  1. cómo son, y
  2. cómo se comportan.
- Busca en los enunciados los sustantivos que aparecen (serán los objetos) y los verbos (las acciones que podrán realizar).
- Todos los objetos que presentan un mismo comportamiento son de la misma clase.
- La clase es el elemento central de la Programación Orientada a Objetos.
- Dado un problema, se construye una colección de clases.
- El programa usará objetos, instancias de esas clases, para cumplir con sus requisitos de funcionamiento.

Programa: Área de un círculo

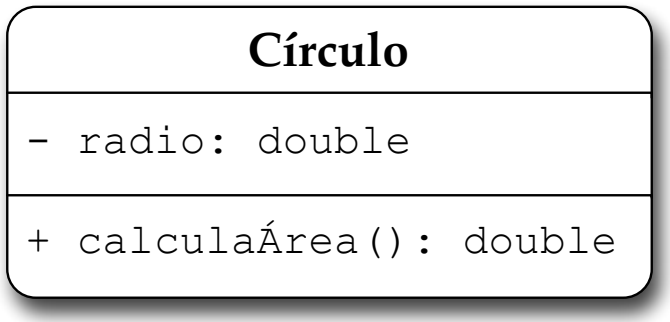
Enunciado y análisis (orientado a objetos)

Enunciado

Realizar un programa que permita al usuario introducir el radio de un círculo (número real) y muestre su área en la pantalla.

En el análisis nos centraremos en identificar:

- Las **clases de objetos** que se manejan → `Círculo`
- Cada clase se definen por dos elementos principales:
  1. Los **datos** que definen esos objetos → `radio`
  2. Las **acciones** que pueden realizar → `calculaÁrea`



Clases

Definición de un nuevo tipo de objetos

Clase

Representación abstracta de un conjunto de objetos que se comportan igual.

- Una clase es la definición de un nuevo tipo de objetos.
- Determina qué **datos** los describen y las **funciones** que hacen.
- Puede entenderse como una extensión de los tipos de datos
  - definen un nuevo tipo de dato,
  - con las operaciones que pueden hacer gracias a sus métodos.
- Informalmente, una clase puede verse como el molde que nos permite crear objetos del mismo tipo.

Declaración de una clase

```
public class nombre {
    elementos que contiene la clase (atributos+métodos)
}
```

Objetos

Instancias de una clase

Objeto

Cada una de las instancias creadas a partir de una clase.

- Un objeto es cada variable cuyo tipo es una clase.
- El objeto tendrá todos los elementos que define su clase:
  1. sus atributos o campos, y
  2. los métodos o acciones que realiza.
- A los objetos también se denomina instancias, y al hecho de crear un objeto a partir de una clase, instanciación.
- Para poder usar un objeto en un programa, primero hay que crearlo.
- Se crean de forma similar a las variables, salvo que su tipo, en lugar de ser un tipo básico, será el nombre de su clase.

```
9        Círculo c = new Círculo();
```

Atributos

Propiedades que definen un objeto concreto

Atributos o Campos

Son los datos que tienen los objetos de una clase.

Declaración de un atributo

[`private` | `public`] [`static`] `tipo nombre`;

- Se declaran fuera de todos los métodos.
- Se deben declarar como privados usando la palabra reservada `private`. Solo serán accesibles **dentro** de la clase.
- No suelen ser estáticos, cada objeto tendrá su atributo.
- Puede dárseles un valor inicial, aunque no se suele hacer.
- La clase también puede tener constantes (estáticas).

```
5     /**Valor del radio del objeto Círculo*/
6     private double radio;
7     /**Constante matemática pi */
8     private static final double PI=3.1416;
```

Métodos

Acciones que puede realizar un objeto de la clase

Métodos

Son las acciones que pueden realizar los objetos de una clase.

Declaración de métodos

```
[private|public] [static] tipo nombre (lista par.) {
    código
}
```

- Se suelen declarar como públicos, usando la palabra reservada `public`. Serán accesibles **fuera** de la clase.
- Un método estático (`static`) no necesitará de un objeto para poder ser llamado.
- **tipo**: es el tipo del valor que devuelve el método. Si no devuelve nada se pone la palabra reservada `void`.
- **parámetros**: lista (separada por comas) con los parámetros que el método necesita. Por cada uno se indica tipo y nombre.

Métodos que devuelven un resultado

Sentencia return

1. Métodos que no producen resultados (métodos `void`).
2. Métodos que devuelven un resultado, un valor. Su tipo de retorno será algún tipo básico o el nombre de una clase.

Sentencia `return`

`return` *expresión*;

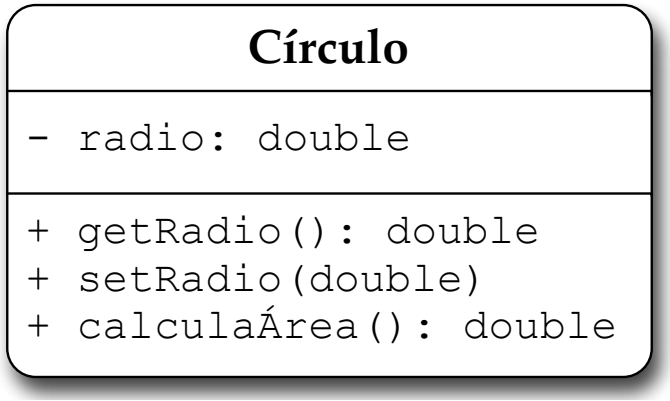
- La sentencia `return` hace que el método acabe.
- Por ello suele colocarse al final del código del método.
- Puede haber más de una sentencia `return` en un método.
- La *expresión* tiene que ser del mismo tipo que el tipo escogido en la declaración como tipo de retorno o de un tipo que se pueda convertir de forma automática a él.

```
25      public double calculaÁrea() {
26          return PI*radio*radio;
27      }
```

Métodos get() y set()

Acceso a los atributos privados

- Los métodos `get()` y `set()` permiten a los programas que usan objetos de una clase poder ver y modificar sus atributos.
- Proporcionan la forma de trabajar con los atributos privados.
- Se define un método `get()` y `set()` por cada atributo.
- Su nombre siempre debe empezar por `get()` o `set()`, seguido del nombre del atributo.
- A veces también se crea un método `set()` para poder cambiar a la vez varios atributos de una objeto.



Métodos get()

Obtener el valor de un atributo privado

- Su misión es permitir a los programas **obtener** el valor de un atributo.
- Típicamente consta de una única sentencia en la que se retorna el valor de dicho atributo.
- El valor de retorno del método `get()` debe ser el mismo que el del atributo para el que está hecho.
- No necesita parámetros.

```
10      /**Devuelve el valor del radio del objeto Círculo
11       * @return el radio del objeto */
12      public double getRadio() {
13          return radio;
14      }
```

Métodos set()

Modificar el valor de un atributo privado

- Su misión es permitir a los programas **modificar** el valor de un atributo.
- El valor de retorno del método `set()` es `void`, ya que no tiene que devolver nada.
- Necesita un parámetro que sea del mismo tipo que el del atributo con el que está asociado.
- Su objetivo es garantizar cambios seguros del atributo y asegurar que el objeto siempre está en un estado correcto.

```
16      /**Cambia el valor del radio del objeto, para que valga r
17       * @param r nuevo valor para el radio del objeto
18       * @return nada  */
19      public void setRadio(double r) {
20          radio=r;
21      }
```

Métodos de la Clase Círculo

Código Fuente

```
10      /**Devuelve el valor del radio del objeto Círculo
11       * @return el radio del objeto */
12      public double getRadio() {
13          return radio;
14      }

16      /**Cambia el valor del radio del objeto, para que valga r
17       * @param r nuevo valor para el radio del objeto
18       * @return nada  */
19      public void setRadio(double r) {
20          radio=r;
21      }

23      /**Devuelve el área del objeto Círculo
24       * @return el valor del área del Círculo*/
25      public double calculaÁrea() {
26          return PI*radio*radio;
27      }
```

El operador new

Cómo se crean objetos

Creación de un objeto (por defecto)

```
clase nombre = new clase()
```

- Si solamente se declara el objeto, se reserva el espacio para la variable que se referirá a él, pero NO se crea el objeto.

```
Círculo c;     //El objeto NO se crea
```

- Para crearlo hay que hacer un new y asignárselo a la variable.
- Después del nombre de la clase se indica cómo se inicializa el objeto entre paréntesis.
- Si simplemente se ponen `()` el objeto se inicializa por defecto. Los atributos:
  - de los tipos básicos $\rightarrow 0$
  - de una clase (tipos referenciados) $\rightarrow$ `null` (sin crear)

```
9         Círculo c = new Círculo();
11        Scanner teclado= new Scanner(System.in);
```

El operador punto .

Cómo se utilizan los objetos

Operador punto .

```
objeto.elemento_publico
```

- El elemento de la izquierda debe ser un objeto de una clase.
- El elemento de la derecha (ya sea un atributo o un método) debe ser público (`public`).
- Si el elemento fuera privado (`private`) se produciría un error en tiempo de compilación.
- El operador `.` es uno de los que más precedencia tiene, por lo que se accederá al elemento antes de realizar otras operaciones.

```
14          c.setRadio(teclado.nextDouble());
16          System.out.printf("El área es %f\n",c.calculaÁrea());
```

Llamadas a métodos

Los argumentos deben casar con los parámetros

Llamadas a métodos

`objeto.metodo` (*lista de argumentos*)

- Si el método no tiene parámetros, se indica su nombre y los paréntesis.
- Los argumentos se pasan separados por comas.
- Se asignan a los parámetros en orden.
- La lista de argumentos debe coincidir en número con la lista de parámetros que tenga el método.
- Pueden ser tanto variables, como expresiones que involucren variables, constantes y operadores.
- Lo importante es que el tipo de cada argumento sea "compatible" con el tipo de su parámetro correspondiente.
- Se realizan las conversiones automáticas oportunas en caso de que un argumento y un parámetro no sean del mismo tipo.

Ejemplo 1 - Clase Círculo

Código fuente

```
1  import java.util.Scanner;
2
3  /** Ejemplo 1 de uso de la clase Círculo
4   *  @author los profesores de IP  */
5  public class Ejemplo1Círculo {
6
7      public static void main(String[ ] args) {
8          //Objeto de la clase Círculo
9          Círculo c = new Círculo();
10         //Objeto Scanner asociado con el teclado
11         Scanner teclado= new Scanner(System.in);
12         //Leemos el radio
13         System.out.print("Introduce el radio: ");
14         c.setRadio(teclado.nextDouble());
15         //Mostramos el área en la pantalla
16         System.out.printf("El área es %f\n",c.calculaÁrea());
17     }
18 }
```

Encapsulación

Ocultar cómo es por dentro una clase

Encapsulación

Consiste en ocultar al programador que usa una clase los detalles de cómo está programada internamente dicha clase.

Implica varias ideas:

- Una clase es una unidad de código con unas funcionalidades proporcionadas a través de su interfaz (los métodos que tiene).
- El programador que use la clase debe servirse de esas funcionalidades.
- No necesita saber cómo está programada por dentro la clase.
- Una clase debe proporcionar una unidad de código sin errores, eficiente y que además sea resistente a los errores que puedan cometer los programadores que la usen.

Encapsulación

Proteger el acceso a los atributos de la clase

- ¿Qué ocurriría si declaráramos los atributos como públicos?

```
c.radio = -1;  //El radio no puede ser negativo!!!
```

- Por eso los atributos se hacen privados.
- Se evita que los usuarios de la clase puedan usarla mal. Se les protege de ellos mismos.
- Ni siquiera necesitan saber los atributos que tiene una clase.
- Los métodos `set()` serán la forma de garantizar que el objeto de una clase esté siempre en un estado correcto.
- Usaremos sentencias condicionales para comprobar que el valor que se pretende asignar a un atributo es válido.
- Solamente si lo es, el valor del atributo se actualizará.

Javadoc

Documentación de una Clase

- Se genera con un comentario /** . . . */ justo antes de la clase.
- En las primeras líneas se puede poner una descripción genérica de la clase.
- Después se incluyen diversas etiquetas (comandos @):
  1. **@author**: nombre del autor. Se pueden incluir varias de estas etiquetas, una por autor.
  2. **@version**: número de la versión de la clase.
  3. **@since**: fecha de creación de la clase.
  4. **@see**: nombre de otras clases que el usuario podría necesitar consultar. Pueden ponerse varias etiquetas @see.
- No es obligatorio poner todas las etiquetas, solamente las necesarias.
- Se puede ver cómo queda en el Eclipse (pestaña Javadoc).

```
1 /** Representa objetos Círculo, con un campo Radio
2  *  @author los profesores de IP
3  *  @version 1.0  */
4 public class Círculo {
```

Javadoc

Documentación de Atributos y Métodos

- En ambos casos, lo primero debe ser incluir una descripción del elemento.
- Los atributos solamente pueden tener etiquetas @see.
- Los métodos incluyen además otro tipo de etiquetas:
  1. **@param**: una por cada uno de los parámetros.
  2. **@return**: explicación de lo que el método retorna como resultado.
  3. **@see**: información adicional que el usuario podría necesitar consultar.
- Las etiquetas @param y @return sirven para **especificar formalmente** los métodos.

```
 6     /**Valor del radio del objeto Círculo*/
 7     private double radio;
16     /**Cambia el valor del radio del objeto, para que valga r
17      * @param r nuevo valor para el radio del objeto
18      * @return nada  */
19     public void setRadio(double r) {
```

Reutilización

Hacer una clase y utilizarla en muchos programas

Reutilización

Consiste en diseñar código que pueda (re)usarse en muchos programas.

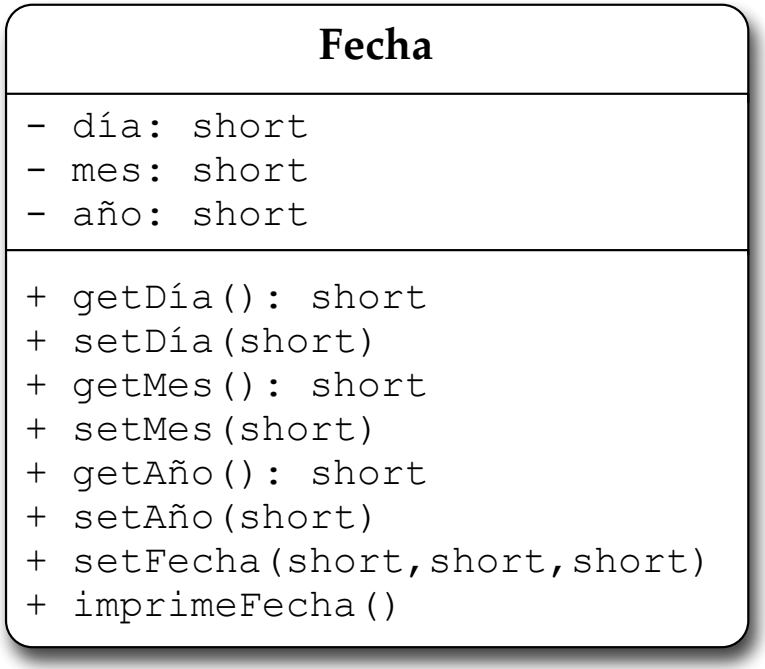
- A veces el objetivo no es crear un programa.
- El objetivo puede ser crear clases para que otros programadores las usen.
- Lo habitual es crear librerías o bibilotecas de clases que están relacionadas entre sí.
- Por ejemplo: una librería para tratar imágenes.
- En este caso, nuestros usuarios son programadores.
- Creamos un código que será **reutilizado** muchas veces.
- Ventajas: reducción de costes, código libre de errores y eficiente, ...

Clase Fecha

El objetivo es solamente crear la clase, no hacer un programa

Enunciado

Realizar una clase para representar fechas, que incluya un método para poder imprimirlas en formato dd/mm/aaaa.



Ejemplo 1 - Clase Fecha

Podrían usar nuestra clase sin saber cómo está programada

```
1  import java.util.Scanner;
2
3  /** Ejemplo 1 de uso de la clase Fecha
4   *  @author los profesores de IP  */
5  public class Ejemplo1Fecha {
6
7      public static void main(String[ ] args) {
8          //Objeto de la clase Fecha
9          Fecha f = new Fecha();
10         //Objeto Scanner asociado con el teclado
11         Scanner teclado= new Scanner(System.in);
12         System.out.print("Introduce la fecha (día, mes y año): ");
13         //Variables para leer la fecha y lectura de los datos
14         short día=teclado.nextShort();
15         short mes=teclado.nextShort();
16         short año=teclado.nextShort();
17         //Cambiamos el objeto fecha con los datos leídos
18         f.setFecha(día, mes, año);
19         //Mostramos la fecha en la pantalla
20         f.imprimeFecha();
21     }
22 }
```

Clase Fecha

Atributos y métodos get()

```
 4 public class Fecha {
 5     /**Valor del día del objeto Fecha*/
 6     private short día;
 7     /**Valor del mes del objeto Fecha*/
 8     private short mes;
 9     /**Valor del año del objeto Fecha*/
10     private short año;

14     public short getDía() {
15         return día;
16     }

27     public short getMes() {
28         return mes;
29     }

40     public short getAño() {
41         return año;
42     }
```

Clase Fecha

Métodos set() e imprimeFecha()

```
21      public void setDía(short d) {
22          día=d;
23      }

34      public void setMes(short m) {
35          mes=m;
36      }

47      public void setAño(short a) {
48          año=a;
49      }

56      public void setFecha(short d, short m, short a) {
57          setAño(a);
58          setMes(m);
59          setDía(d);
60      }

64      public void imprimeFecha() {
65          System.out.printf("%02d/%02d/%04d",getDía(),getMes(),
                getAño());
66      }
67  }
```

Recapitulando

Todas las cosas que hay que tener en cuenta

- Los atributos de la clase son privados para impedir accesos incorrectos a dichos campos.
- Los métodos son públicos, ya que proprocionan las acciones que pueden hacer los objetos.
- Incluimos métodos `get()` y `set()` para facilitar una interfaz con los atributos.
- Solamente los métodos `set()` modifican los atributos.
- El acceso a sus valores se realiza mediante los métodos `get()`.
- Se incluye una documentación apropiada de la clase para facilitar su uso.
- Hemos creado una trozo código con una funcionalidad clara y bien definida, reutilizable y libre de errores.

Importante

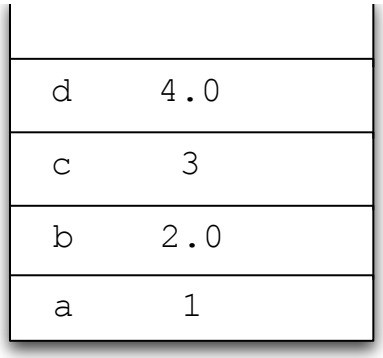
Concentrar el acceso a los atributos en los métodos `get()`/`set()` facilita la depuración de errores y el mantenimiento de la clase

Representación en Memoria: Tipos básicos

Guardan el valor de la variable

- Se guardan en una estructura denominada **Pila** (*Stack* en inglés).
- Lo que contiene la posición de memoria que representa la variable es el valor de la misma.

```
int  a = 1;
double b = 2.0;
int c = 3;
double d = 4.0;
```



- Cuando se realiza una asignación entre dos variables, simplemente se copia el valor de una en la otra.
- Si luego cambiamos una de ellas, la otra no se ve afectada.

```
c = a;      // c pasará a valer 1
a = 7;      // c sigue valiendo 1
```

Representación en Memoria: Tipos referenciados

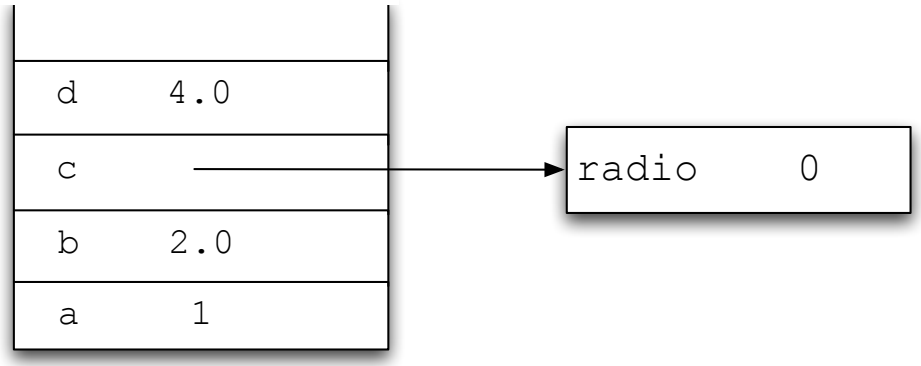
Guardan la referencia del objeto

Tipo referenciado

Un objeto o variable será de un tipo referenciado cuando se cree con el operador new.

- También se guardan en la Pila.
- Pero la variable solo contiene una referencia (dirección) a la posición de la memoria donde está realmente el objeto.
- Al crearse el objeto con new, este se guarda en otra zona de la memoria llamada **Montón** (*Heap* en inglés).

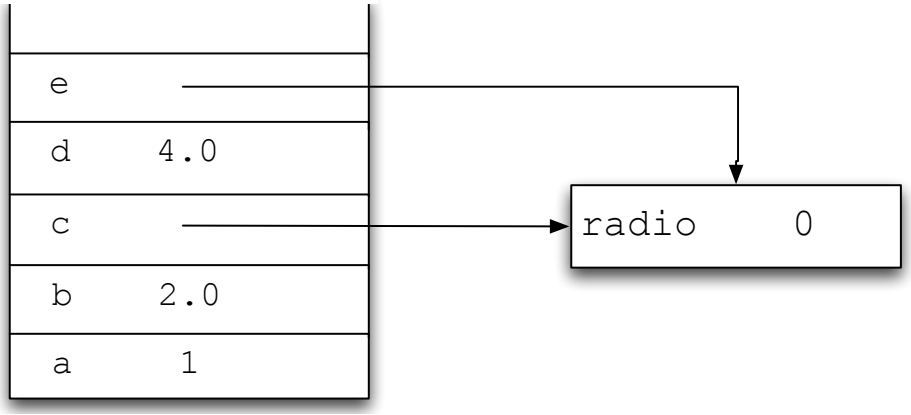
```
int  a = 1;
double b = 2.0;
Circulo c=new Circulo();
double d = 4.0;
```



Asignación de tipos referenciados

Apuntan al mismo objeto

```
Círculo e;
e = c;
```




```
e.setRadio( 5 );
```

Importante

Estaríamos cambiando el `radio` *del objeto* `c`

Paso de parámetros

Se pasan lo valores de los argumentos

- La semántica de una llamada a un método es la inicialización de sus parámetros.
- En Java se inicializan con el **valor** de sus argumentos.
- Es lo que se conoce como **paso por valor**.
- Si tenemos en cuenta la representación en memoria:
  1. variables de tipos básicos: se pasa el valor que contienen,
  2. tipos referenciados (p. ej. objetos de clases): se pasa la referencia del objeto al que apuntan.
- Esto implica que el método **puede cambiar** un objeto que reciba.
- La ventaja que se obtiene es de eficiencia, ya que ningún elemento grande (p. ej. un objeto con muchos campos) se duplica en memoria.

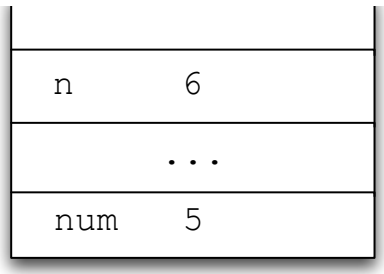
Paso de parámetros: Tipos básicos

El valor del argumento no puede ser cambiado por el método

```
 5      public static void incrementaInt(int n) {
 6          n=n+1;
 7      }

13      public static void main(String[ ] args) {
14          //Variable entera
15          int num=5;
16          System.out.printf("\nAntes de la llamada num=%d",num);
17          incrementaInt(num);
18          System.out.printf("\nDespués de la llamada num=%d",num);

30      }
```



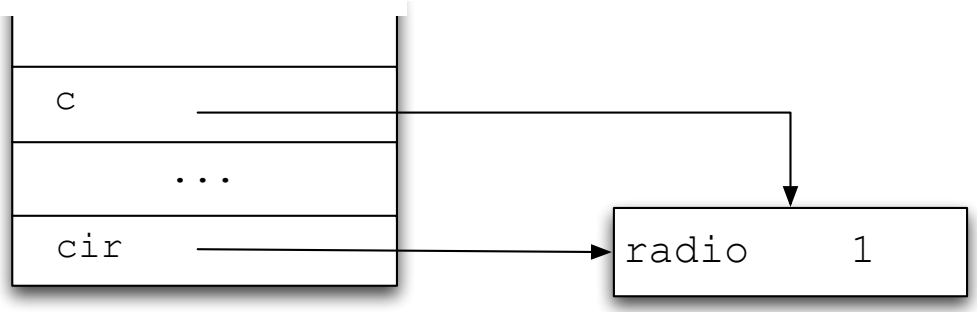
Importante

Una variable de un tipo básico, usada como argumento en una llamada a un método, no cambia NUNCA tras dicha llamada

Paso de parámetros: Tipos referenciados

El valor del argumento puede ser cambiado por el método

```
 9      public static void incrementaRadio(Círculo c) {
10          c.setRadio( c.getRadio()+1 );
11      }
13      public static void main(String[ ] args) {
20          //Objeto de la clase Círculo
21          Círculo cir = new Círculo();
22          System.out.printf("\nAntes de la llamada radio=%f",cir.getRadio());
23          incrementaRadio(cir);
24          System.out.printf("\nDespués de la llamada radio=%f",cir.getRadio());
30      }
```



Importante

Un objeto de una clase, y en general de un tipo referenciado, puede cambiar si se pasa como argumento en una llamada a un método

Métodos que reciben objetos de la propia clase

Acceso a cada uno de los objetos

- Objeto que invoca el método: se pone simplemente el nombre del atributo o del método.
- Objeto pasado como parámetro: es necesario anteponer el nombre del objeto y el operador punto.

```
33      public void copiaRadio(Círculo c) {
34          setRadio(c.getRadio());
35      }
```

Al hacer la llamada necesitamos dos objetos:

```
27          Círculo otro = new Círculo();
28          otro.copiaRadio(cir);
29          System.out.printf("\nradio=%f",otro.getRadio());
```