Universidad Complutense de Madrid
Facultad de Informática
Grado en Ingeniería Informática, Grupo B

Tecnología de la Programación

Introducción a Java / Clases y objetos

(Tema 2 de los apuntes)

Simon Pickin,
Alberto Díaz, Puri Arenas, Yolanda García

Historia del lenguaje Java

- Finales de los 80:
  - Sun Microsystems decide introducirse en el mercado de la electrónica de consumo
    - Procesadores heterogéneos en amplio rango de pequeños aparatos y electrodomésticos
    - cada uno con su entorno de programación y sus API
- 1991:
  - James Gosling y su equipo de Sun inician el desarrollo del lenguaje “Oak”
  - Simplificación de C++ adaptado a sistemas embebidos y GUI sencilla o sin GUI
  - Pensado, en particular, para lo que ahora se llama “smart TV” y PDAs
  - Marcado objetivo crece más lento de lo previsto; Oak no se usa hasta…
- 1995:
  - Oak reorientado a la World Wide Web y renombrado Java
  - Ideal para una red de recursos distribuidos alojados en ordenadores heterogéneos
  - Java distribuido gratis e integrado en el navegador Netscape para ejecutar applets

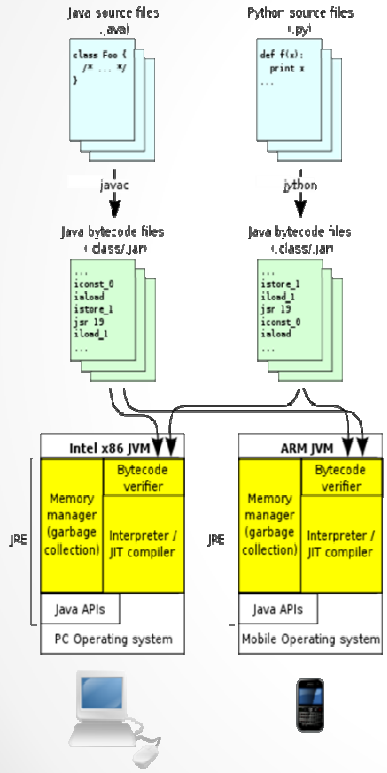
Características del lenguaje Java

- Orientado a objetos
  - Encapsulación, herencia, polimorfismo
  - Herencia múltiple solo permitida en interfaces
- Sintaxis familiar
  - similar a C++
- Tipado estático y fuertemente tipado
  - Las variables tiene tipo y el compilador verifica tipos
  - Todo valor tiene un tipo bien definido y cualquier conversión implícita de tipos es segura
- Modelo de objetos más sencillo que C o C++
  - No hay manipulación directa de punteros
- Gestión de memoria más sencilla que C o C++
  - Recolector de basura automático

Características de la plataforma Java

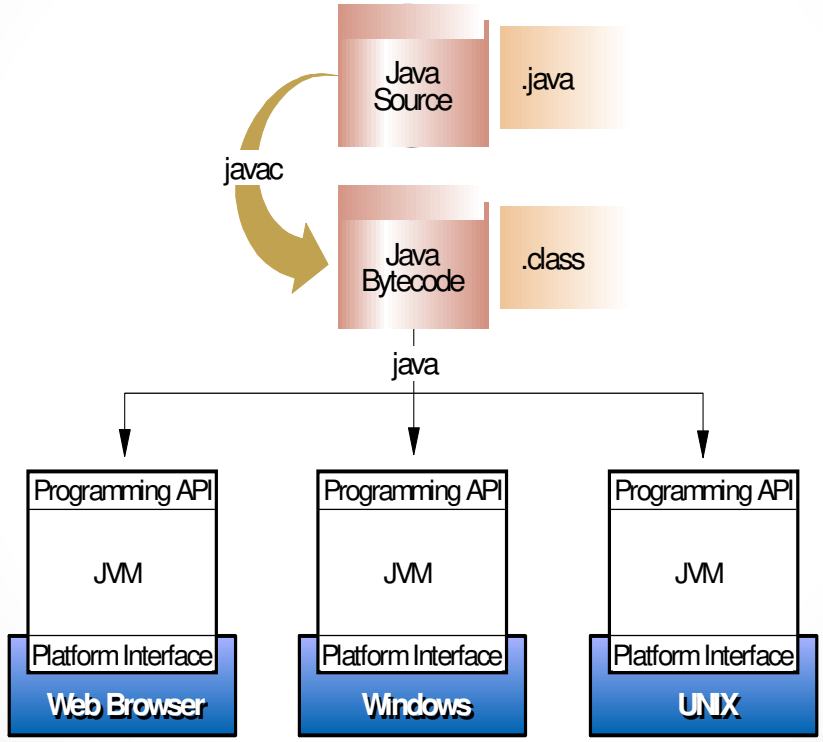
- La plataforma software Java se basa en el principio siguiente:
  - El resultado de la compilación de un programa debe poder ejecutarse en distintos ordenadores / dispositivos *sin modificación alguna*
    - Así, se puede distribuir el software como componentes binarios que pueden ejecutarse directamente en múltiples dispositivos: PCs, teles, móviles,…
    - Conocido como el principio WORA (*Write Once, Run Anywhere*)
- La arquitectura de máquina virtual
  - El lenguaje y el compilador son independientes del hardware y del SO
    - Clave para obtener WORA
  - La máquina virtual (JVM) ejecuta código máquina abstracta (bytecode)
    - El compilador Java traduce de código fuente Java a bytecode
    - El JVM traduce de bytecode a código máquina (interpretador o compilador JIT)
  - En tecnología anterior a Java, el bytecode se conocía como “p-code”

La arquitectura de la plataforma Java

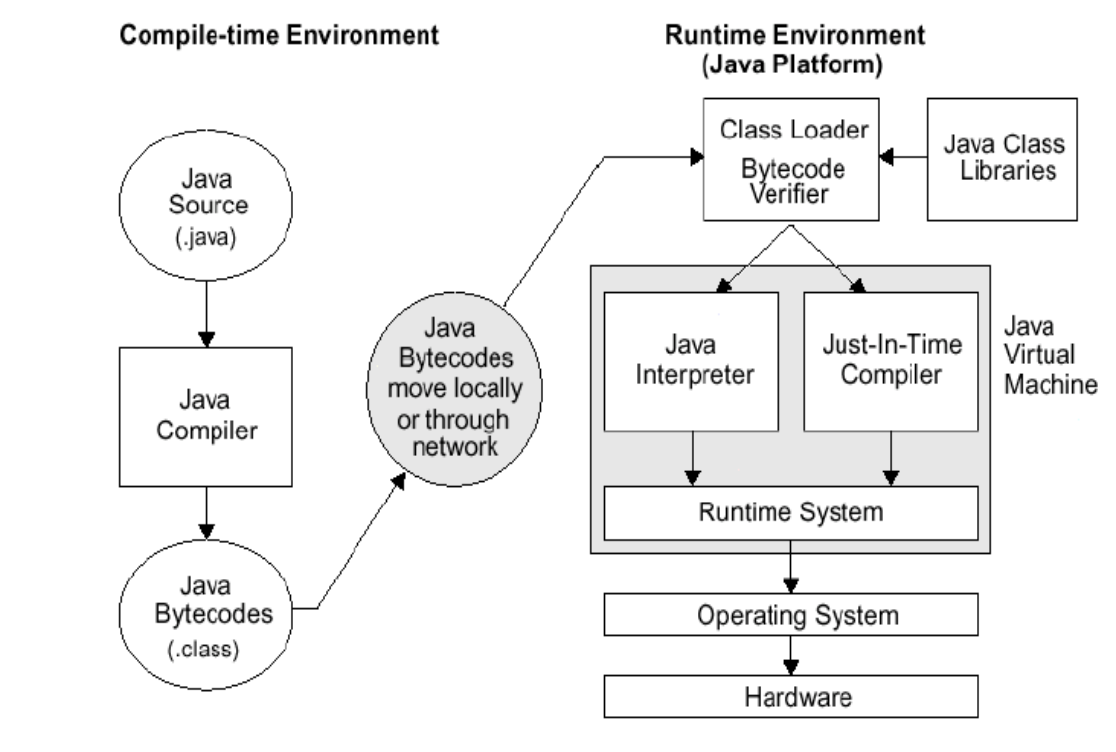


- El software Java puede ejecutarse en una amplia variedad de plataformas heterogéneas gracias a la máquina virtual Java (JVM)
- La JVM traduce instrucciones de bajo nivel independientes de plataforma (bytecode) en instrucciones de bajo nivel dependientes de plataforma (código máquina)
- La implementación de la JVM en sí es dependiente de plataforma (evidentemente!)
- Integridad: una máquina virtual puede verificar las instrucciones bytecode antes de traducirlas
- Seguridad: el enfoque máquina virtual facilita el uso de un *sandbox* en el que ejecutar código que no sea de confianza

La plataforma Java



La máquina virtual Java



La plataforma Java abierta

- La plataforma Java consta de tres elementos
  1. El lenguaje Java
  2. La máquina virtual Java (JVM)
  3. Las bibliotecas muy extensas de Java (también llamado el API Java)
- Es posible usar el 2º y 3º elemento sin usar el 1º
  - p.e. Scala y los lenguajes dinámicos Groovy, Clojure, Jruby, Jython,…
    - Pueden ser compilados a bytecode y ejecutados sobre la JVM, hacienda llamadas a las bibliotecas de Java.

Ediciones de Java

- Java SE (*Java Standard Edition*)
  - JRE (*Java Runtime Environment*) => un entorno de ejecución Java
    - JVM + API estándar (conjunto de bibliotecas)
  - JDK (*Java Development Kit*) => un entorno de desarrollo Java
    - JRE y bibliotecas extras de interés para desarrolladores
    - `javac`: compilador; `java`: lanza el JRE para ejecutar bytecodes
    - Muchos utilidades (p.e. `jdb` depurador, `javadoc` generador de documentos)
- Otras tecnologías Java:
  - Java ME (*Java Micro Edition*) : Java para pequeños dispositivos, en particular, móviles (¡que ya no son tan pequeños!)
  - Java EE (*Java Enterprise Edition*): Java para aplicaciones distribuidas comerciales, especialmente aplicaciones web
  - Java FX, Java Card, Java TV, …

Bibliotecas principales de Java SE

- `java.lang`: contiene clases básicas tales como `String`
- `java.io` & `java.nio`: contiene clases para gestionar la entrada-salida mediante flujos de datos
- `java.net`: contiene clases para gestionar el acceso a la red
- `java.util`: contiene colecciones, el modelo de eventos, facilidades horarias, facilidades de internacionalización,…
- `java.applet`: para soportar la creación de applets
- `java.awt`: la Abstract Window Toolkit (AWT) contiene clases para desarrollar interfaces de usuario gráficas (GUI) y para imprimir gráficos e imágenes; los widgets del AWT están implementados con código que depende de la plataforma.
- `javax.swing`: contiene clases para desarrollar GUI; más flexible y potente que la AWT; los widgets de swing están implementados en Java.

El entorno de programación Java

- Un entorno mínimo
  - contiene el compilador y la máquina virtual
  - se utiliza desde la línea de comandos
- El compilador en un entorno mínimo
  - se lanza con la instrucción `javac`, p.ej. `javac MyClass.java`
  - produce ficheros de bytecode con la extensión `.class`, p.ej. `MyClass.class`
- Ejecución por la JVM en un entorno mínimo
  - Después de compilar, la máquina virtual puede lanzarse y el bytecode del programa ejecutarse mediante la instrucción `java`, p.ej. `java MyClass`
- Ejecución de un applet, p.ej. para incrustarlo en el fichero HTML `page.html`
  - Lanzar un navegador web
  - Usar la instrucción `appletviewer`, p.ej. `appletviewer page.html`
    - Ejecuta en una ventana separada cada applet de la página web proporcionada como argumento

Estructura de un programa Java

```
package nombrePaquete;

// Importación de otras clases o paquetes
import java.util.Vector;
import java.util.Date;

// Declaración e implementación de la clase
public class NombreClase
{
      …
}
```

- Java distingue minúsculas y mayúsculas
  - Por convención, los nombres de las clases empiezan con mayúscula

Estructura de un programa Java

- Un programa Java consiste en una o más definiciones de clase
  - una declara el método `public static void main(String[] args)`
- Las funciones no pueden definirse fuera de clases
  - Solo se permiten métodos, cada uno de los cuales pertenece a una clase
- Un fichero fuente Java tiene la extensión `.java` y
  - es una unidad independiente de compilación
  - puede contener múltiples definiciones de clase pero max. uno `public`
  - Si tiene una clase `public` , debe tener el mismo nombre que el fichero
    - Esta restricción aumenta la eficiencia de la compilación
    - También ayuda a mejorar la legibilidad del código
- El compilador genera un fichero `.class` para cada clase
  - haya o no ficheros de código fuente que contengan múltiples clases

Ficheros y directorios

- Los ficheros `.class` de Java son parecidos a los ficheros `.o` o `.obj` generados por la compilación de C o C++
  - pero en C y C++,
    - se preconiza recoger a las declaraciones de tipos y de funciones en un fichero `.h` y luego incluirlo en el fichero (los ficheros) fuente que utiliza (utilizan) estos tipos y funciones
    - sobre todo si los mismos tipos y funciones se utilizarán en múltiples ficheros fuente
  - mientras que en Java
    - las clases simplemente importan cualquier otra clase que utilizan que no esté definida en el mismo fichero (via `import <nombrePaquete>.<nombreClase>` o `import <nombrePaquete>.*`)
    - Si se quiere declarar una interfaz explícitamente, se puede hacer más limpiamente con la construcción Java `interface`
- Para que el compilador de Java, y el interprete de Java o compilador JIT, puedan encontrar el bytecode de las clases importadas (via `import`) que no están en los paquetes del programa ni en la biblioteca estándar de Java
  - el camino de estos ficheros se coloca en la variable de entorno `CLASSPATH`
  - o se proporciona en el argumento `-classpath` (o `-cp`) a los comandos `java`, `javac`

Ejecución de programas Java

- Formas de ejecución de programas Java
  - Como aplicación de línea de comandos (con entrada / salida estándar)
  - Como aplicación con interfaz de ventanas (con una GUI)
  - Como applet incrustado en una página web
    - El uso de applets es lo que lanzó el lenguaje Java
    - Pero los applet están deprecados desde Java 9 (de sept. 2017), debido a la falta de soporte del plug-in de Java por parte de los navegadores
    - Una alternativa (si se quiere usar Java con web): Java Web Start

Ejemplo 1: primer programa Java

HolaMundo.java

```
// Este es mi primer progama en Java

public class HolaMundo
{
   public static void main(String[] args)
   {
      System.out.println("Hola Mundo");
   }
}
```

Ejemplo 1: primer programa Java

- Definimos la clase `HolaMundo`
  - Es una clase `public`, es decir, accesible desde fuera del paquete
    - defecto es “package-private”: accesible desde dentro del paquete solamente
- Tiene un solo método llamado `main`
  - Es un método `public` y `static` que no devuelve un valor
  - Contiene una sola instrucción que imprime una cadena al terminal
    - La cadena de caracteres a imprimir es el argumento del método `println`
    - Este método es un método de la clase `PrintStream` del paquete `java.io`
    - Esta clase es el tipo del atributo cuyo nombre es `out`
    - Este atributo es un atributo `static` de la clase `System` del paquete `java.lang` (este paquete se importa por defecto en todo programa Java)
  - El valor del atributo `out` encapsula el flujo de salida estándar

Ejemplo 1: compilar y ejecutar

- Para compilar desde la línea de comandos

```
javac -d <directorio donde poner los ficheros .class generados> \
-cp <classpath> miPaquete.MiClase.java
```

- Para ejecutar desde la línea de comandos

```
java -cp <classpath> miPaquete.MiClase
```

  - `MiClase` contiene el método `main`
    - es el punto de entrada de la aplicación
  - Otras clases también pueden contener un método `main`
    - puede ser útil para probar clases por separado

Ejemplo 2: primer applet Java

- Código diseñado para ser incluido dentro de páginas Web
  - Deprecado desde Java 9; incluido aquí por razones históricas
- Imprescindible incluir interfaz gráfica

```
import java.awt.*;      // contiene la clase Graphic
import javax.swing.*;   // contiene la clase Japplet

public class HolaMundoApplet extends JApplet
{
  public void paint(Graphics g)
  {
    g.drawString("Hola, mundo", 25, 50);
  }
}
```


Ejemplo 2: página web (holaMundo.html)

```
<html>
  <head>
    <title>Ejemplo de Hola Mundo</title>
  <head>
  <body>
    <applet code="HolaMundoApplet.class" width=100 height=100>
    </applet>
  </body>
</html>
```

- La etiqueta "applet" está deprecado (por la W3C) desde HTML4 pero seguía siendo la única manera sencilla de escribir HTML con applets que funcionaba en distintos navegadores (si no: `<object>` en IE, `<embed>` en Mozilla).
- Los applet están deprecados (por Oracle) desde Java 9

Ejemplo 2: compilar y ejecutar

- Para compilar desde la línea de comandos:
  - `javac HolaMundoApplet.java`
- Para ejecutar desde la línea de comandos
  - `appletviewer holaMundo.html`

Sintaxis de una clase en Java

```
public class NombreClase {
   // atributos
   // constructores
   // métodos
};
```

Atributos: visibilidad

- Los atributos definen el estado de los objetos de una clase
  - La declaración de un atributo tiene la siguiente forma: visibilidad tipo nombre;
- El campo de visibilidad puede tomar uno de los siguientes valores:
  - **`public`**: accesible desde cualquier lado
  - **`protected:`** accesible desde objetos de clases que heredan de la clase en la que se declara el atributo o desde otros objetos de clases que están en el mismo paquete.
  - El valor por defecto, "package private": accesible desde objetos de clases que están en el mismo paquete
  - **`private:`** accesible solo desde un objeto de la misma clase
- Normalmente (recuerde: ocultación de información):
  - todos los atributos son `private`
  - o posiblemente, en presencia de herencia, `protected`

Atributos

- Los atributos se pueden inicializar a la vez que se declaran

```
private int x = 2;
private String cadena = "setze jutges d'un jutjat";
private boolean fin = true;
```

- Un objeto puede referir a sus atributos mediante el prefijo `this`

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

   public int suma(){ return this.x + this.y;}
   public incr(int z){
      this.x = this.x + z;
      this.y = this.y + z;
  }
}
```

¿Qué es “this”?

- Una referencia al objeto cuyo método o constructor se está ejecutándose
- ¿Cuándo se necesita? Para desambiguar:

```
public class ClassExample {     // Una variable local no debería tener el mismo nombre que un atributo
     private int integerVar = 1;

     private void exampleOfMethod1() {
          this.integerVar = 2;          // Asignar el valor 2 al atributo integerVar
          integerVar = 3;               // Asignar el valor 3 al atributo integerVar
     }

     private void exampleOfMethod2() {
          int integerVar;               // Definir una variable local integerVar y asignarle el valor 2
          integerVar = 2;               // Modificar la variable local, no el atributo
     }

     private void exampleOfMethod3(){
          int integerVar = 2;           // Definir una variable local integerVar y asignarle el valor 2
          this.integerVar = integerVar; // Asignar el valor de la variable local al atributo
     }

}
```


Atributos constantes

- Las declaraciones de constantes en Java tienen la siguiente forma:

  public static final <tipo> <NOMBRE> = <valor>;

- El modificador `final`
  - especifica que el valor del atributo no puede cambiar
- El modificador `static`
  - especifica que el atributo forma parte del tipo definido por la clase
    - Se puede referir al atributo incluso si no se han creado un objeto de la clase
  - Todos los objetos de la clase comparten el mismo atributo
    - en vez de tener cada uno su copia
- El modificador `public`
  - especifica que el valor está disponible para todos los objetos del programa
    - Lo habitual para constantes
- Por convención, se escriben las constantes en mayúsculas

Los constructores

- Los objetos se crean invocando a sus constructoras
  - Sirven para inicializar los atributos del objeto
  - Definición de los constructores es parte importante del diseño de la clase
- Una clase puede tener múltiples constructores
  - Con tal de que el número y tipo de los parámetros de cada sean distintos
- Al contrario de los métodos normales, los constructores
  - se llaman igual que la clase cuyos objetos construyen
  - no tienen tipo de salida (ni siquiera *void*).
  - no pueden llamarse explícitamente
    - Un constructor solo se llama una vez por objeto, cuando éste se crea con **new**
  - no se heredan

Ejemplo 3: constructores

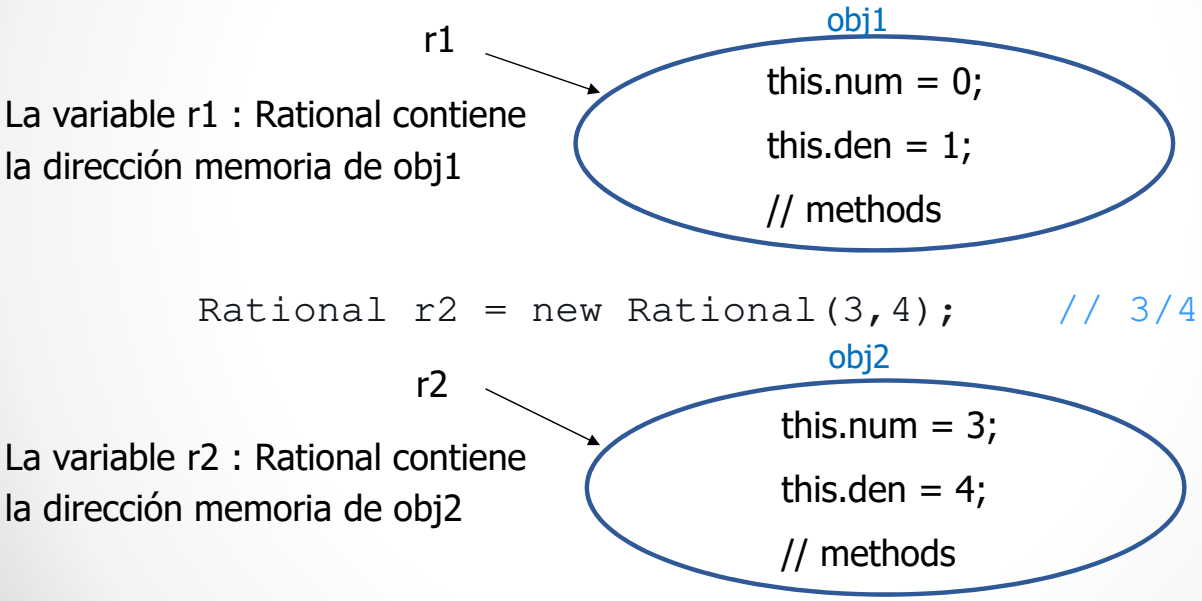
```
public class Rational {
   private int num;
   private int den;

   // constructor sin argumentos (valores por defecto)
   public Rational(){
        this.num = 0;
        this.den = 1;
   }

   // constructor con argumentos
   public Rational(int n, int d){
        this.num = n;
        this.den = d;
   }
   ...
}
```

Ejemplo 3: constructores

```
Rational r1 = new Rational();        // 0/1
```




```
Rational r2 = new Rational(3,4);     // 3/4
```


Constructores, política por defecto en Java

1. Si una clase no contiene constructor alguno
   - el compilador inserta un constructor sin argumentos vacío, luego aplica la regla 2
2. Si la primera línea de cualquier constructor no es una llamada a otro
   - el compilador inserta una llamada al constructor sin argumentos de la superclase
   - Si la superclase tiene constructor pero no un constructor sin argumentos
     - la llamada fracasa y el código no compila
   - Si la superclase no tiene constructor alguno
     - la regla 1 se aplica (al constructor de la superclase) y el compilador inserta un constructor sin argumentos y luego aplica la regla 2 (al constructor de la superclase)

- Y si no hemos especificado que la clase hereda de otra
  - hereda necesariamente de la clase del sistema llamada `Object`
  - Aplicación de la regla 2 resulta en una llamada al constructor sin argumentos de la clase `Object`

Esta información se presentará otra vez a la hora de estudiar las jerarquías de herencia.

Los métodos

- Los métodos definen la funcionalidad de los objetos
  - es decir, las operaciones que un objeto de la clase puede realizar
- Los métodos pueden definirse con las mismas cuatro visibilidades que los atributos, esto es
  - **`public`:** para métodos que forman parte de la interfaz de la clase
  - **`protected`:** como para atributos; usado en presencia de herencia,
  - Valor por defecto (*package-private*): poco usado
  - **`private`:** para métodos auxiliares llamados por otro método de la clase

Declaración de métodos

- Tiene la siguiente forma

```
modificadores tipoRetorno nombre(Lista_Parámetros){
        // cuerpo
}
```

- Los modificadores que hemos visto hasta ahora:
  - Modificadores de visibilidad y la palabra clave `static`
- Java permite *sobrecarga de métodos*
  - Una clase puede tener múltiples métodos
    - con el mismo nombre
    - y con el mismo tipo de retorno (o `void`)

  pero que difieren en el número y/o tipo de los parámetros.

Ejemplo 4: declaración de métodos

```
public class A {
  private int x;
  A(){this.x=0;}

  int incr(int z){return this.x + z;}
  int incr(){return this.x + 1;}
  int incr(double z){return 1;}
}

public class Main {

  public static void main(String[] args) {
    A a = new A();
    a.incr(2);       // ejecuta el primer método
    a.incr(2.0);     // ejecuta el tercer método
  }
}
```


Tipos de métodos públicos

- *Inicializadores*: inicializan atributos (pero no son constructores)
- *De acceso*: devuelven valores (generalmente, el valor de un atributo)
  - Se suele utilizar un nombre que empieza con "*get*"
  - Solo se debería devolver directamente valores de atributos *de tipo primitivo*
    - Para un atributo cuyo valor es un objeto, se debería devolver una copia
  - Algunos autores llaman "*getters*" a los métodos de acceso
- *Mutadores*: modifican valores (muchas veces, el valor de un atributo).
  - Se suele utilizar un nombre que empieza con "*set*"
  - Algunos autores llaman "*setters*" a los mutadores
- *Computadores*: realizan cálculos y devuelven resultados.

Ejemplo 5: distintos tipos de métodos

```
public class Rational {

    // attributos
    private int num;
    private int den;

    // constructores
    public Rational(int n, int d){
        this.num = n;
        this.den = d;
    }

    public Rational(){
        this(0,1)
    }
```


Ejemplo 5: distintos tipos de métodos

```
// métodos accesores

public int getNum(){
    return this.num;
}
public int getDen(){
    return this.den;
}

// métodos mutadores

public void setNum(int n){
    this.num = n;
}
public int setDen(int d){
    this.den = d;
}
```

Ejemplo 5: distintos tipos de métodos

```
// métodos computadores

public void ProductoEscalar(int n){
     this.num = this.num * n;
}

public void simplificar(){
     int mcd = mcd(this.num,this.den);
     this.num = this.num / mcd;
     this.den = this.den / mcd;
}
```


Ejemplo 5: distintos tipos de métodos

```
// Este método se utiliza como ejemplo de un método privado.
// En realidad, probablemente sería major como public y static
// y dentro de una clase de utilidades, siendo private el
// método "simplificar"
private int mcd(int a, int b){
     while (a != b)
     if (a > b) a = a -b; else b = b -a;
     return a;
}

public String toString(){
     return this.num + "/" + this.den;
}

}
```

Llamadas a métodos

- Las llamadas a métodos tienen la siguiente forma

```
nombreObjeto.nombreMetodo(valores-de-parámetros-separados-por-comas)
```

- `nombreObjeto` debe haber sido creado previamente
  - invocando a alguno de los constructores de la clase a la que pertenece.
- Ejemplo

```
Rational r1 = new Rational(4,8);
Rational r2 = new Rational(2,6);
r1.multiply(r2);
```


El método `main`

```
public static void main( String[] args ){
   ...
}
```

- Es el punto de inicio de toda aplicación en Java.
- Toda clase puede tener un método con este nombre y *signature*
  - es decir, `public static void main( String[] args )`
  - Puede utilizarse para comprobar el funcionamiento correcto de la misma (en fase de pruebas)
- En una aplicación Java acabada, debe haber un único punto de inicio.
  - Algunos autores lo implementan siempre en una clase llamada `Main`
- Es un método estático.
  - Se aplica por tanto a la clase y no a una instancia en particular.

Visibilidad de clases

- Las clases pueden tener las siguientes visibilidades:
  - **`public:`** los miembros (atributos y métodos) públicos de la clase son accesibles desde cualquier parte del código
  - Valor por defecto (*package-private*): los miembros(atributos y métodos) públicos de la clase son accesibles desde objetos o código estático de cualquier clase del mismo paquete
  - **`private`** y **`protected`**: solo pueden usarse para clases anidadas, (ver más adelante).

Ejemplo 6: métodos de acceso y mutadores

```
public class Primitiva {
   // atributo
   private int x;
   // constructor
   public Primitiva(int iniX){ x = iniX; }
   // métodos accesor y mutador
   public int getX(){ return x; }
   public void setX(int nuevoX){ x = nuevoX; }
 }

public class Compleja {
   // atributo
   private Primitiva p;
   // constructor
   public Compleja(Primitiva iniP){ p = iniP; }
   // metodo accesor (warning: ¡devuelve tipo no primitivo!)
   public Primitiva getP(){ return p; }
}
```

Ejemplo 6: métodos de acceso y mutadores

- Y en el `main`:

```
public static void main( String[] args ){
        Primitiva pr1 = new Primitiva(3);
        Compleja c = new Compleja(pr1);
        Primitiva pr2 = c.getP();
        pr2.setX(7);
        System.out.println(pr1.getX());
        c.getP().setX(4); // encadenamiento de métodos
        System.out.println(pr1.getX());
}
```

- ¿Qué se imprimirá en la entrada estándar?
  - ¿3 y 3, o 7 y 4?
- Observe que hemos podido modificar el valor del atributo `x` del objeto que es el valor del atributo `p` a pesar de que `p` se ha declarado como `private`,
  - Para evitar romper la encapsulación, `getP()` debería crear y devolver *una copia del objeto* valor del atributo `p` en vez de devolver el objeto mismo