Universidad
Carlos III de Madrid

Tema 6

Reutilización de código

Programación
2015-2016

Tema 6. Reutilización de código

- **Modularidad.**
- Implementación de métodos.
- Uso de métodos.

Estructuración y modularidad

- Hay varios criterios que se deben cumplir cuando se escribe un programa

- Un programa bien escrito debe ser:

  - **compacto**: no innecesariamente largo
  - **legible**: fácil a entender y utilizar por otros
  - **robusto**: no fallar con entradas inesperadas
  - **reutilizable**: se puede reutilizar parte del programa

Métodos

- Métodos en Java son módulos de código (funciones, procedimientos, rutinas) que se pueden tratar individualmente

- Un programa normalmente esta formado por varios métodos.

- Ventajas:
  - **abstracción**: el usuario de un método no necesita conocer el proceso interno, sólo la entrada y salida
  - **reutilización**: un método de un programa se puede reutilizar en otro
  - **legibilidad**: un programa es más fácil de entender y escribir si está separado en módulos
  - **compactibilidad**: no existen varias copias del mismo código

Cuando utilizar métodos

- Los métodos se utilizan para romper un problema en pedazos de forma que éste resulte mas manejable. (*Divide y Venceras*)

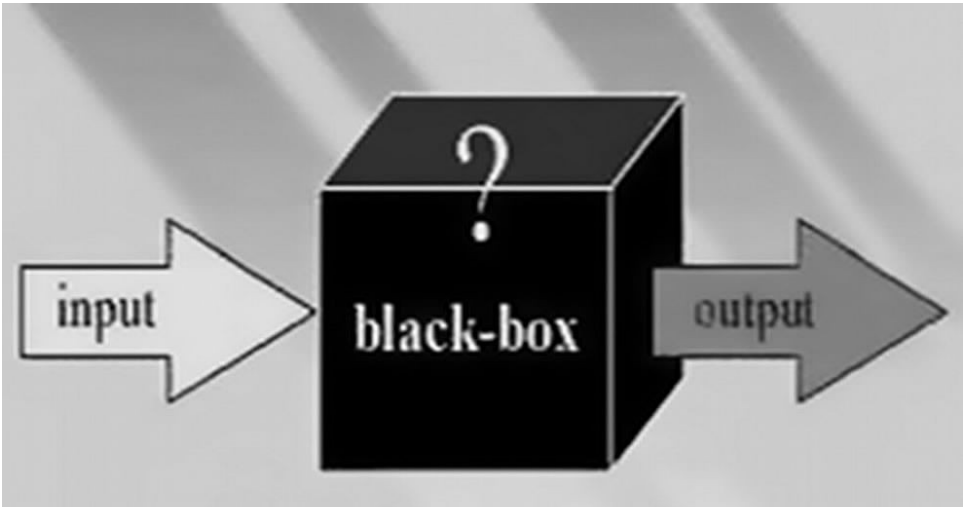
- Se utilizan también cuando tenemos grupos de instrucciones que se repiten en distintos puntos de un programa.

Funcionamiento de un método

- Podemos imaginar un método como una caja negra que procesa valores de entrada (parámetros) y origina, o no, unas salidas (valores de retorno) aunque no necesariamente sepamos cómo funciona.

- En algunos lenguajes los métodos que devuelven un valor se llaman *funciones*, mientras que los métodos que no devuelven nada se llaman *procedimientos*.

- En Java se llaman **métodos** tanto en un caso como en otro



Reutilización

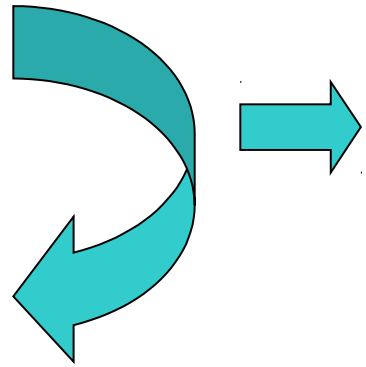
- A menudo hay que realizar una misma operación en varios programas o en distintas partes del mismo programa
  - Podemos copiar el código varias veces y manipular las entradas para que funcione en otro programa
  - No obstante, ¿qué pasa si hay que modificar ese código?
    - Habrá que cambiarlo en todos los lugares donde se encuentra
  - Por esto es mejor tener una única vez el código y poder llamarlo desde donde haga falta

Reutilización

```
public class SinMetodo2 {
    public static void main(String[] args) {
      int maximo;
      int suma;
     // Calcula la suma de los 5 primeros enteros
     maximo = 5;
     suma = 0;
     for(int i=1; i<=maximo; i++){
     suma += i;
     }
    System.out.println("La suma es: " + suma);

    // Calcula la suma de los 7 primeros enteros
    maximo = 7;
    suma = 0;
    for(int i=1; i<=maximo; i++){
    suma += i;
    }
    System.out.println("La suma es: " + suma);
}
```

Reutilización

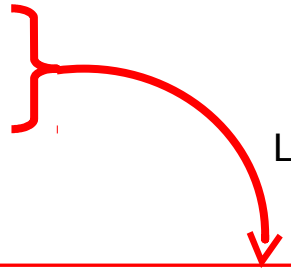
```
public class SinMetodo2 {
    public static void main(String[] args) {
      int maximo;
      int suma;
     // Calcula la suma de los 5 primeros enteros
     maximo = 5;
      suma = 0;
      for(int i=1; i<=maximo; i++){
     suma += i;
      }
     System.out.println("La suma es: " + suma);

     // Calcula la suma de los 7 primeros enteros
     maximo = 7;
     suma = 0;
      for(int i=1; i<=maximo; i++){
     suma += i;
     }
     System.out.println("La suma es: " + suma);
}
```



METODO

Reutilización

```
public class Metodo2 {

   public static void main(String[] args) {
    sumaNumeros(5);
    sumaNumeros(7);
    }

    public static void sumaNumeros(int maximo){
          int suma = 0;
          for(int i=1; i<=maximo; i++){
                suma += i;
          }
          System.out.println("La suma es: " + suma);
    }
}
```



Llamada al método

Método

Tema 6. Reutilizacion de código

- Modularidad.
- **Implementación de métodos.**
- Uso de métodos.

Declaración de métodos

- Sintaxis:

```
<acceso> <tipo> <identificador>(<entrada>) {
   <bloque de instrucciones>
   }
```

- El acceso: son modificadores o calificadores que sirven para especificar quién y como se puede acceder al método. Puede ser una o más palabras
- El tipo especifica el tipo de dato del resultado del método
- El identificador es el nombre del método
- La entrada es una secuencia de variables (parámetros) separadas por comas

Acceso a métodos

```
<acceso> <tipo> <identificador>(<entrada>){
...}
```

- A veces se desea restringir el acceso a un método
- El acceso se establece en la declaración de un método
- En nuestro caso, el acceso de un método siempre será

  `public static`
- Eso significa que se puede llamar (invocar) al método desde cualquier otra clase

Tipo de retorno del método

```
<acceso> <tipo> <identificador>(<entrada>){
...}
```

- El tipo de un método especifica de que tipo será el resultado devuelto por el método
- Por ejemplo, si el tipo de un método es **int**, el resultado del método es un número entero
- El tipo de un método puede ser cualquiera de los tipos primitivos, o también puede ser un objeto (Se verá en POO)
- Además, si un método no devuelve ningún resultado su tipo será **void**

Entrada de un método. Los parámetros

```
<acceso> <tipo> <identificador>(<entrada>){
...}
```

- Cada método tiene una entrada de datos.
- La entrada es una secuencia de variables (parámetros) separadas por comas
- La entrada puede estar vacía, es decir, no contener variables. En este caso se dejan los paréntesis vacíos
- Los parámetros especifican el tipo de información que el usuario necesita introducir en el método para poderlo utilizar
- Los parámetros no se inicializan

Salida de un método

```
<acceso> <tipo> <identificador>(<entrada>){
...}
```

- Si el tipo de retorno de un método es distinto de **`void`**, el método originará una salida
- La salida es el resultado de aplicar el método a una entrada determinada
- Para que un método devuelva el resultado se usa la palabra **`return`**
- El efecto de **`return`** es “terminar el método y devolver el valor generado por el método”

Ejemplo

- Ejemplo de un método que realiza el producto de dos números que se le pasan en la entrada:

```
public static int mult(int a, int b) {
    int resultado = a * b;
    return resultado;
}
```

Tema 6. Reutilizacion de código

- Modularidad.
- Implementación de métodos.
- **Uso de métodos.**

Llamada a métodos

- Para llamar o invocar a un método hay que especificar los valores de entrada (argumentos) que deben coincidir uno a uno con los parámetros del método.
- Sintaxis:

  ```
  <identificador>(<argumentos>)
  ```

- Los valores de los argumentos se separan con comas y sus tipos tienen que corresponderse con los tipos de los parámetros de entrada y en el mismo orden.
- Si el método no tiene parámetros no se pone nada entre los paréntesis.
- Si el método devuelve un resultado, para guardarlo hay que utilizar una variable del mismo tipo del que sea el valor devuelto

  ```
  tipo resultado = <identificador>(<argumentos>)
  ```

Llamada a métodos

- Si el método está en una clase/objeto distinta de la clase actual, es necesario especificar el nombre de su clase/objeto
- Sintaxis:

  <clase/objeto>.<identificador>(<valores>)
- Ejemplos:

```
System.out.println("Escriba un número");
int numero = teclado.nextInt();
```

Ubicación de métodos

- Todos los métodos de una clase se hallan dentro de las llaves de la clase
- ¡Nunca un método se halla dentro de otro método!
- Por lo tanto, no se hallan dentro de **`main`**, que también es un método

Ejemplo

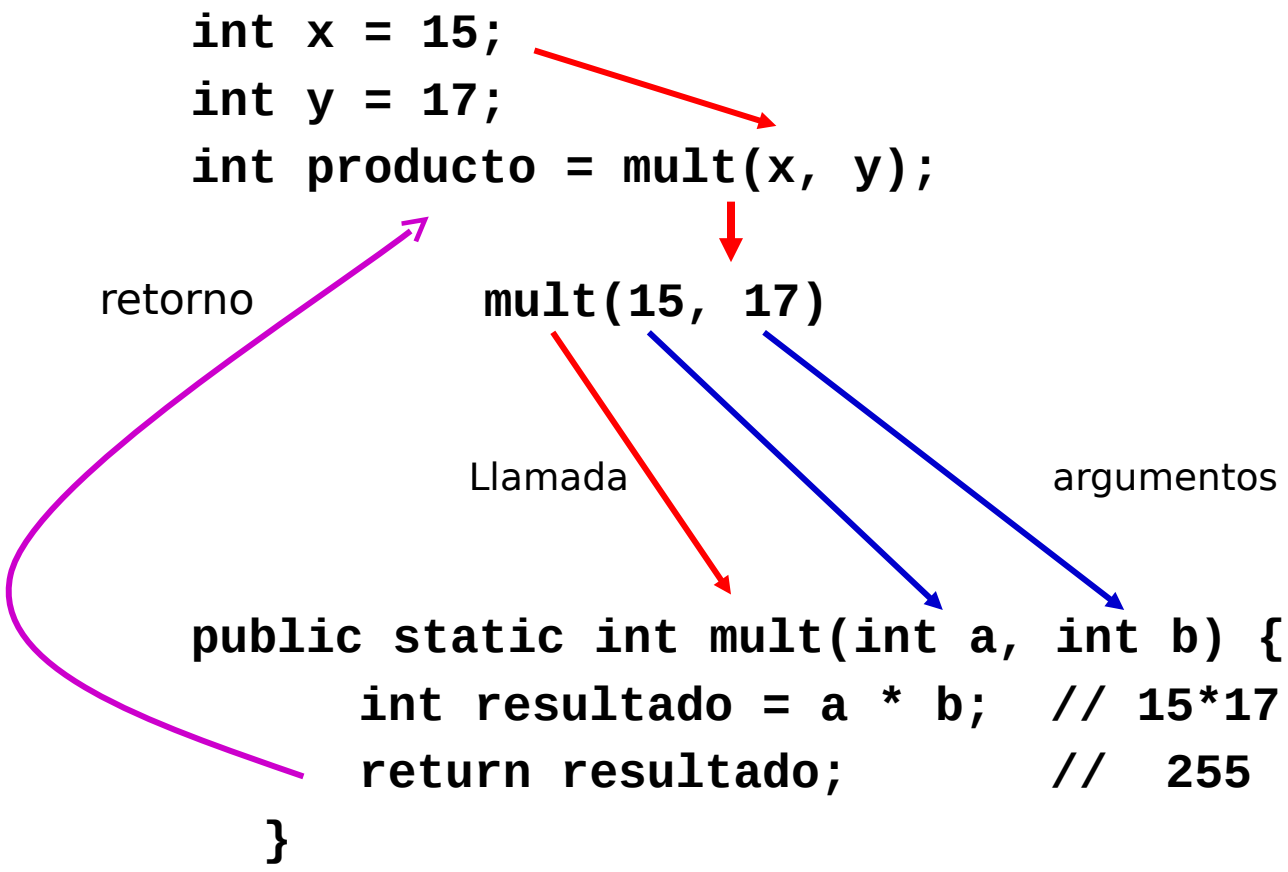
- Ejemplo de un método que realiza el producto de dos números:

```
public static int mult(int a, int b) {
   int resultado = a * b;
   return resultado;
}
```

- Se le invoca desde **main**

```
public static void main(String[] args) {
   int x = 15;
   int y = 17;
   int producto = mult(x, y);
   System.out.println(producto);
}
```

Ejemplo


int x = 15;
int y = 17;
int producto = mult(x, y);
retorno
mult(15, 17)
Llamada
argumentos
public static int mult(int a, int b) {
int resultado = a * b; // 15*17
return resultado; // 255
}

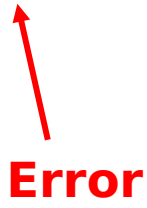
Variables de métodos

- Dentro de un método se pueden usar variables
- ¡No obstante, una variable declarada en un bloque sólo existe (sólo es visible) dentro de este bloque!
- No se puede utilizar (no es visible) dentro de otros métodos

Ejemplo

- Ejemplo de un método que realiza el producto de dos números:

```
public static int mult(int a, int b) {
   int resultado = a * b;
   return resultado;
}

public static void main(String[] args) {
   int producto = mult(15, 17);
   System.out.println(a * b);
}
```



Error

Sobrecarga de métodos

- Es un mecanismo que permite definir varios métodos con el mismo nombre.
- Un método se determina por su firma.
  - **La firma se compone del nombre del método, número de parámetros y el tipo de los parámetros.**
- Los métodos sobrecargados deben diferenciarse por su firma. En concreto por el número, o por el tipo de sus parámetros, o por el orden de éstos.
- El compilador es capaz de saber que método sobrecargado tiene que utilizar en cada momento por la firma.

Sobrecarga de métodos

- Por ejemplo supongamos que queremos escribir un método para calcular el perímetro de diferentes figuras planas:

```
public static double perimetro(double radio){
return (2*PI*radio) }     // para la circunferencia

public static double perimetro(double lado1, double lado2){
return (2*(lado1+lado2) }  // para un rectángulo

public static double perimetro(double lado, int numLados){
return (lado * numLados) } // para un polígono de lados iguales
```

- Al hacer la llamada al método el compilador sabrá que método debe usar por su firma.

Ejemplo de sobrecarga de métodos

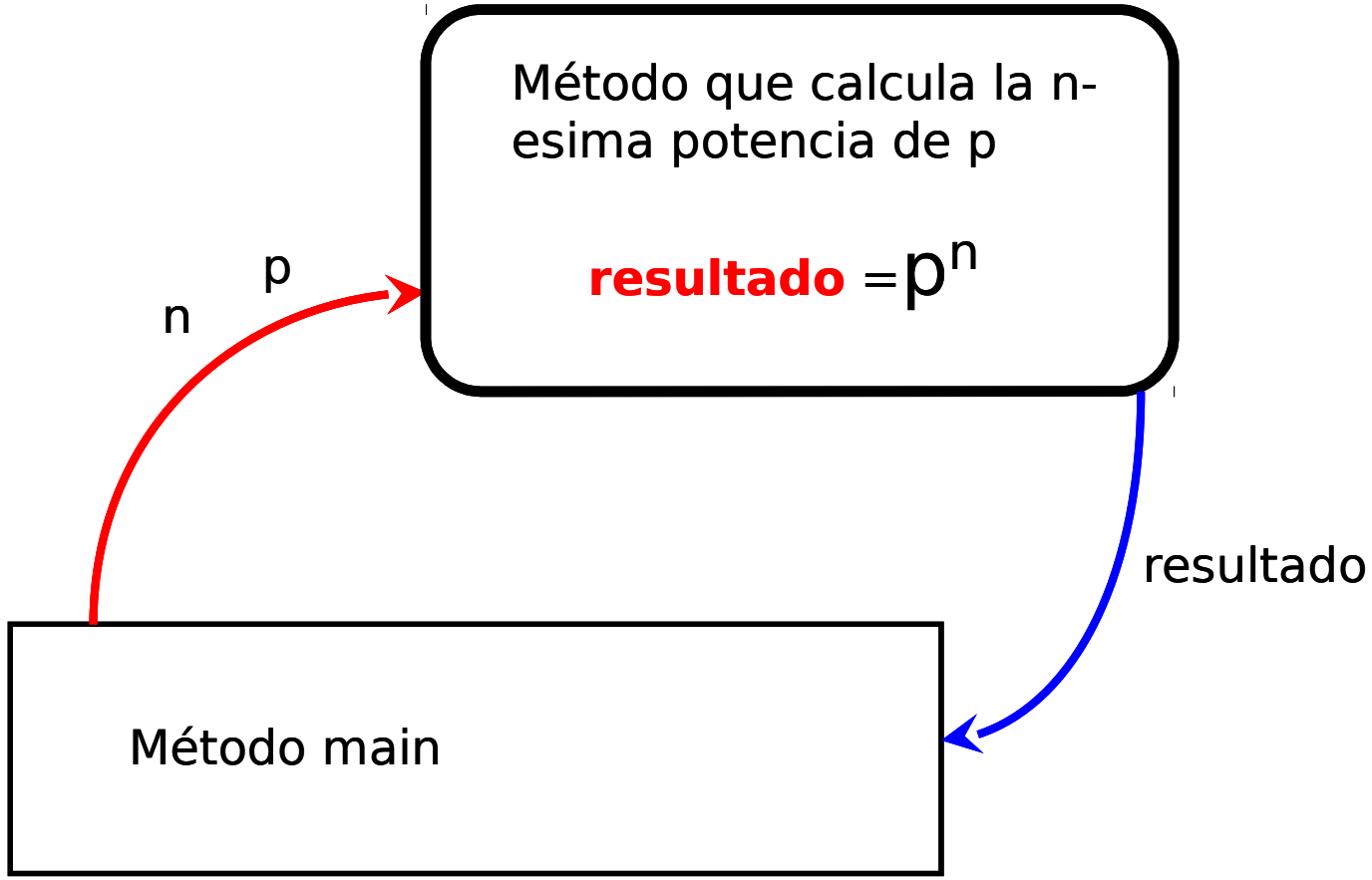
```
public class Sobrecarga1 {

   public static double perimetro(double radio){
     return(2*Math.PI*radio);
    }

   public static double perimetro(double lado1, double lado2){
     return(2*(lado1+lado2));
   }

   public static double perimetro(double lado, int numLados){
     return(lado*numLados);
   }

    public static void main(String[] args) {
     System.out.println("Longitud de la circunferencia: "+perimetro(4.5));
     System.out.println("Perimetro del rectangulo: "+perimetro(4.5, 6.5));
     System.out.println("Perimetro del poligono: "+perimetro(4.5, 10));
     }
}
```

Ejercicios

- Escribir un método para calcular la n-esima potencia de un número.

- Escribir un método que calcule el factorial de un número.

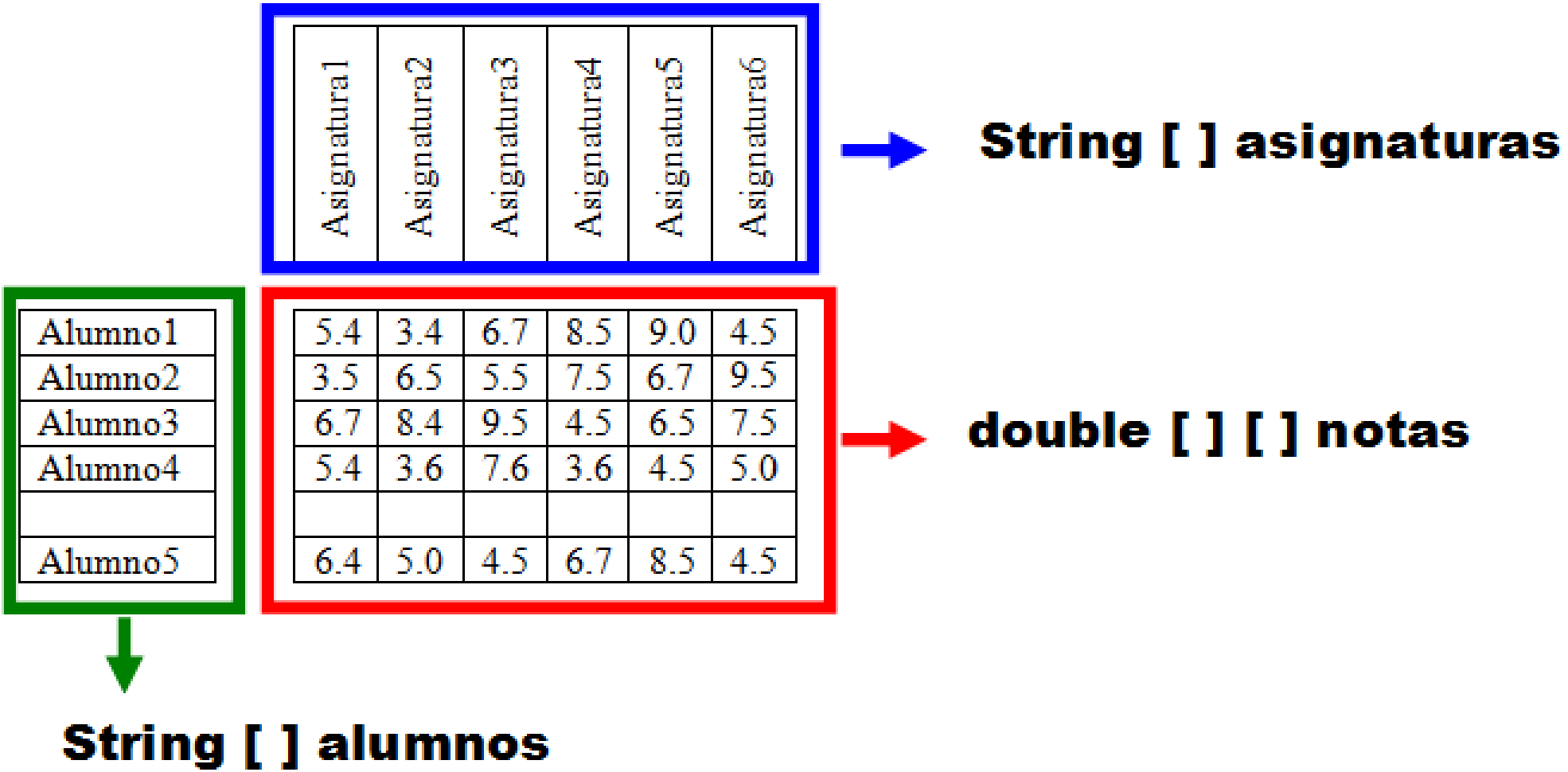
Ejercicios


Método que calcula la n-esima potencia de p
resultado = p^n
p
n
resultado
Método main

Ejercicios

- Escribir los métodos de un programa para gestionar las notas de N alumnos en M asignaturas.

Planteamiento del programa



Método main

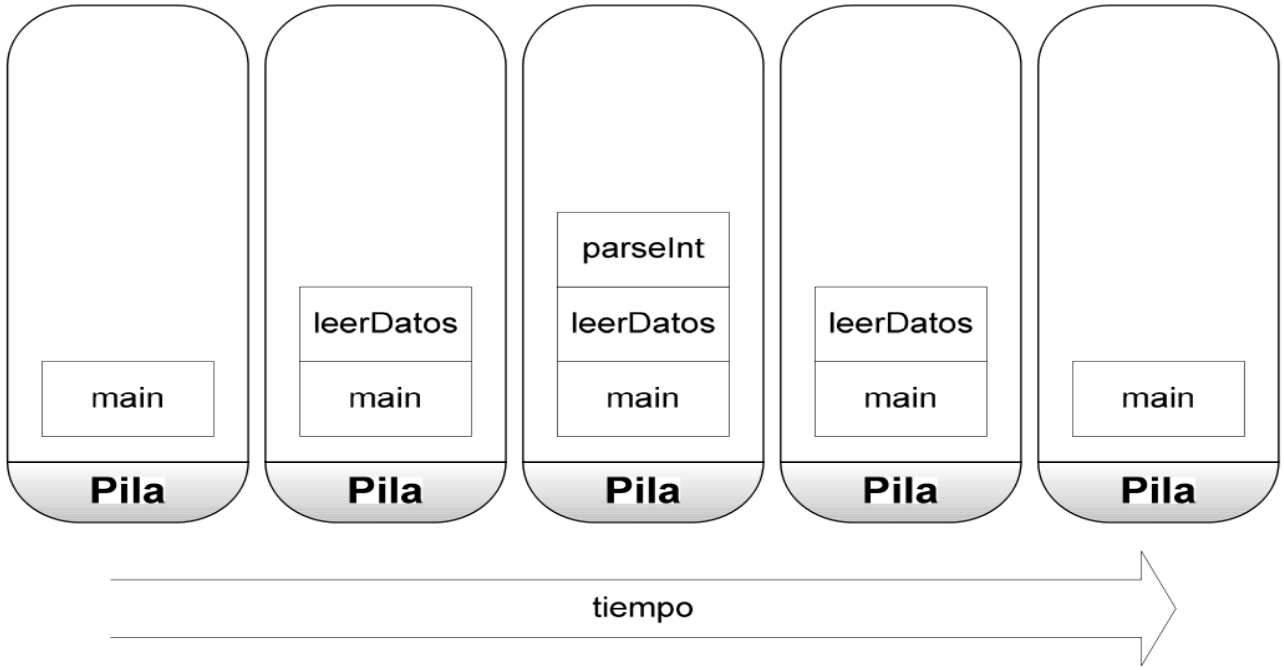
```
public static void main(String[] args) {
    String []nombres=ponNombre();
    String busca="Julio";
    int pos=hallaPosicion(busca, nombres);

    System.out.println("El nombre buscado ocupa la posición "+pos);
    double [][] lasnotas=ponNota();

    double [] sunota=buscaNota(pos, lasnotas);
    System.out.print("Las notas de "+ busca +" son \t");

    for(int j=0;j<sunota.length; j++){
    System.out.print(sunota[j]+"\t");
    }
}
```

Pila

Donde se almacenan las variables locales (y parámetros) de los métodos que se invocan.

- Cada llamada a un método provoca que se reserve espacio en la pila para almacenar sus variables locales (y los valores de sus parámetros).
- Al finalizar la ejecución del método, se libera el espacio ocupado en la pila por las variables locales del método.

Esta zona de memoria se denomina pila por la forma en que evoluciona su estado:



Recursividad

Una función que se llama a sí misma se denomina recursiva

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \cdot (n-1)! & \text{si } n > 0 \end{cases}$$

$$x^n = \begin{cases} 1 & \text{si } n = 0 \\ x \cdot x^{n-1} & \text{si } n > 0 \end{cases}$$

Definición recursiva: Véase *definición recursiva*.
[FOLDOC: Free On-line Dictionary of Computing]

Divide y vencerás

Cuando la solución de un problema se puede expresar en términos de la resolución de un problema de la misma naturaleza, aunque de menor complejidad.

Divide y vencerás:
Un problema complejo se divide en otros problemas más sencillos (del mismo tipo)

Sólo tenemos que conocer la solución no recursiva para algún caso sencillo (denominado **caso base**) y hacer que la división de nuestro problema acabe recurriendo a los casos base que hayamos definido.

Como en las demostraciones por inducción, podemos considerar que "tenemos resuelto" el problema más simple para resolver el problema más complejo (sin tener que definir la secuencia exacta de pasos necesarios para resolver el problema).

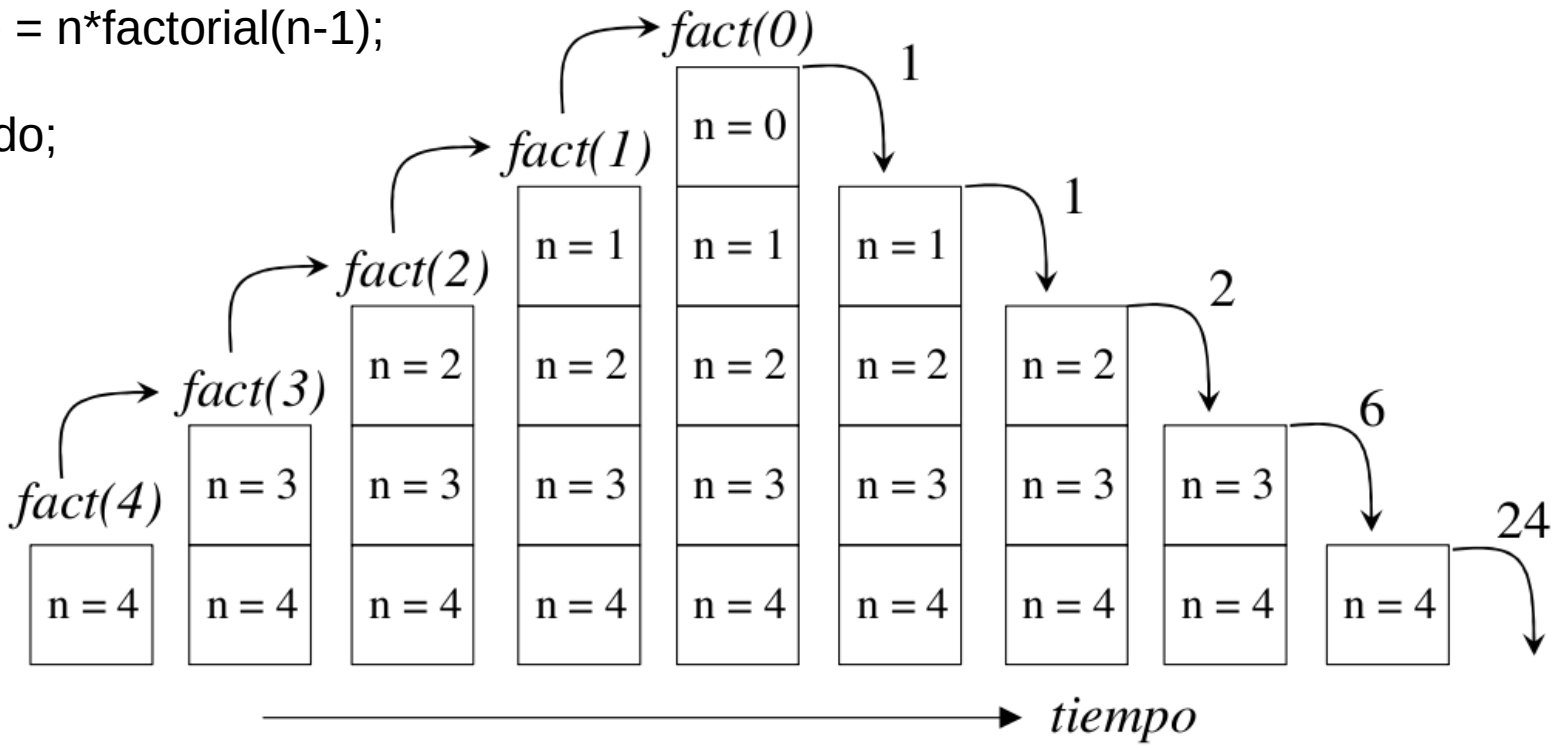
Ejemplo

¿Cuántas formas hay de colocar n objetos en orden?

- Podemos colocar cualquiera de los n objetos en la primeraposición.
- A continuación, colocamos los (n-1) objetos restantes.
- Por tanto: P(n) = n P(n-1) = n!

Factorial calculado de forma recursiva

```
static int factorial (int n)
{
        int resultado;
        if (n==0){
                // Caso base
                Resultado = 1;
        } else {
                // Caso general
                resultado = n*factorial(n-1);
        }
        return resultado;
}
```



Funcionamiento de un algoritmo recursivo

- Se descompone el problema en problemas de menor complejidad (algunos de ellos de la misma naturaleza que el problema original).
- Se resuelve el problema para, al menos, un caso base.
- Se compone la solución:

  - Resolución de problema para los casos base:
    - Sin emplear recursividad.
    - Siempre debe existir algún caso base.

  - Solución para el caso general:
    - Expresión de forma recursiva.
    - Pueden incluirse pasos adicionales (para combinar las soluciones parciales).

Casos Base

Siempre se debe avanzar hacia un caso base:
Las llamadas recursivas simplifican el problema y, en última instancia,
los casos base nos sirven para obtener la solución.

- Los casos base
  corresponden a situaciones que se pueden resolver con facilidad.

- Los demás casos se resuelven
  recurriendo, antes o después, a alguno(s) de los casos base.

De esta forma, podemos resolver problemas complejos
que serían muy difíciles de resolver directamente.

Fibonacci: Solución iterativa

Solución iterativa

```
static int fibonacci (int n)
{
	int actual, ant1, ant2;
	ant1 = ant2 = 1;
	if ((n == 0) || (n == 1)) {
		actual = 1;
	} else {
		for (i=2; i<=n; i++) {
			actual = ant1 + ant2;
			ant2 = ant1;
		ant1 = actual;
		}
	}
}
return actual;
}
```

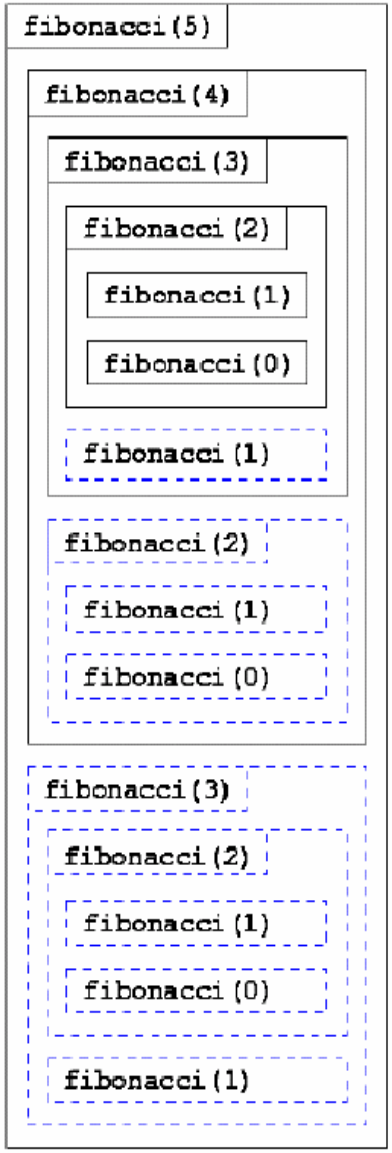
Sucesión de Fibonacci

$$Fib(0) = Fib(1) = 1$$
$$Fib(n) = Fib(n-1) + Fib(n-2)$$

Fibonacci: Solución recursiva

Solución recursiva

```
static int fibonacci (int n)
{
        if ((n == 0) || (n == 1)){
                return 1;
        } else {
                return fibonacci(n-1) + fibonacci(n-2);
        }
}
```



Ejercicio

S9-Clase: Reutilización