Unidad 4

Funciones

Contenidos

- **Conceptos básicos**
- **Comportamiento**
- **Ventajas**
- **Ámbito de las variables**
- **Parámetros**
- **Paso por valor**
- **Retorno de la función**
- **Sobrecarga**
- **Recursividad**

Conceptos básicos

El Contexto

Cuando un programa adquiere cierta complejidad, nos encontramos con:

- Fragmentos de códigos con distintas funcionalidades mezclados entre ellos. Lo que lo hace más difícil de entender.
- Fragmentos de código con la misma funcionalidad repetidos en distintas partes del programa. Duplicando código.

Conceptos básicos

El problema

Lo visto anteriormente genera los siguientes problemas:

- Código menos legible: difícil de entender.
- Duplicidad del código: aumentando el tamaño del código.
- Dificultad en el mantenimiento:
  - El cambio de una funcionalidad deberemos saber a qué parte del código afecta.
  - En el caso de tener repetida dicha funcionalidad en diferentes partes, tendremos que realizar la modificación en cada uno de los fragmentos en los que aparece dicha funcionalidad.

Conceptos básicos

Definición

> Una función es un conjunto de instrucciones que ejecutan una tarea determinada y que hemos encapsulado en un formato estándar para que nos sea muy sencillo de manipular y reutilizar.

Esta función podrá ser invocada una o varias veces para ejecutar dichas instrucciones.

Por tanto, de esta forma tenemos bloques de códigos identificados que hacen una función concreta y que pueden ser reutilizados.

Ejemplo

Repitiendo el bloque de código

```
public static void main(String[] args) {
   ... //código
   System.out.println("Voy a saludar tres veces:"); //fragmento repetido
   for(int i = 0; i < 3; i++) {
      System.out.println("Hola.");
   }
   ... //más código
   System.out.println("Voy a saludar tres veces:"); //fragmento repetido
   for(int i = 0; i < 3; i++) {
      System.out.println("Hola.");
   }
   ... //otro código
   System.out.println("Voy a saludar tres veces:"); //fragmento repetido
   for(int i = 0; i < 3; i++) {
      System.out.println("Hola.");
   }
   ... //resto del código
}
```

Con función

```
public static void main(String[] args) {
   ... //código
   tresSaludos(); //sustitución por una función
   ... //más código
   tresSaludos(); //sustitución por una función
   ... //otro código
   tresSaludos(); //sustitución por una función
   ...//resto del código
}
```

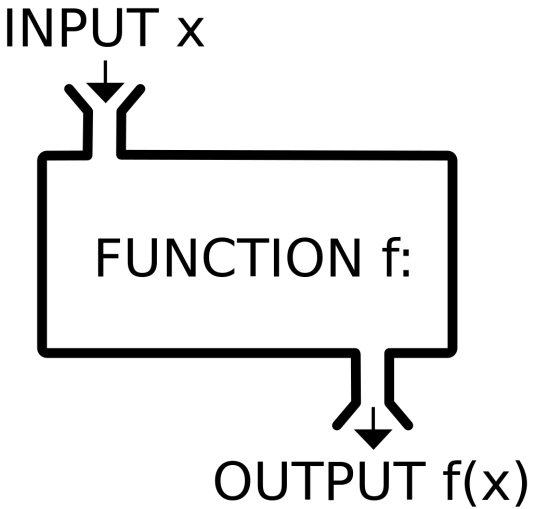
Conceptos básicos

Una función normalmente tiene unos datos de entrada.

Realiza unas operaciones con dichos datos.

Y genera unos datos de salida.

Aunque no siempre es obligatorio que haya datos de entrada y/o salida.



Conceptos básicos

Conceptos necesarios

- **Llamada a la función:** Las funciones tienen un nombre (identificador) para poder invocarlas. Se les invoca usando dicho nombre seguido de paréntesis **()** como si fuera una instrucción más. Dentro de los paréntesis van los datos que se pasan a la función.

  *int resultado = **factorial(5)**;*

- **Prototipo de la función:** Es la declaración de la función, donde se especifica su **nombre**, el **tipo** que devuelve y, entre paréntesis, los **parámetros de entrada** que utiliza.

  ***static int factorial(int numero)** { ...*

Conceptos básicos

Conceptos necesarios

- **Cuerpo de la función:** Es el bloque de código que ejecuta la función cada vez que se invoca y que aparece entre llaves después del prototipo

```
static int factorial(int numero) {
        int resultado = 1;
        for (int i=2; i<=numero; i++) {
                resultado = resultado * i;
        }
        return resultado;
}
```

Conceptos básicos

Conceptos necesarios

Definición de la función: Prototipo + cuerpo de la función.

```
static int factorial(int numero) {
    int resultado = 1;
    for (int i=2; i<=numero; i++) {
        resultado = resultado * i;
    }
    return resultado;
}
```

Prototipo

Cuerpo

Comportamiento

```
// Programa principal
public static void main(String[] args) {

      int solucion1 = factorial(5);
      System.out.println(solucion1);

      int x = 7;
      int solucion2 = factorial(x);
      int operacion = solucion2 - solucion1;

      int dato = sc.nextInt();
      int solucion3 = factorial(dato);
      System.out.println(solucion3);

}
```

Ejecuta el cuerpo de la función con ***numero***=5

Ejecuta el cuerpo de la función con ***numero***=7

Ejecuta el cuerpo de la función con el ***numero*** que el usuario ha introducido por consola

```
// Función factorial
static int factorial(int numero) {
   int resultado = 1;
   for (int i=2; i<=numero; i++) {
      resultado = resultado * i;
   }
   return resultado;
}
```

Aclaraciónes

- Realmente en JAVA no usamos funciones sino **métodos**.
- Aunque el comportamiento es similar, la diferencia es conceptual y se verá cuando se de el tema de Programación Orientada a Objetos.
- La declaración de una función puede hacerse antes o después del main()
- Los nombres de las funciones siguen el estilo Camel: suma(), calculaRaiz()...

Ámbito de las variables

El **ámbito de una variable** es dónde puede utilizarse. Es decir, en qué bloque o bloques de código podemos usarla.

- **Variables locales**: aquellas que se declaran en el cuerpo de una función. Su ámbito es el de la propia función. No se pueden usar fuera del cuerpo de la función donde se declaran.
- **Variables de bloque**: son las que se declaran dentro de una estructura (if, else, while, for, ...). Es decir, las que se declaran dentro de un bloque de código { }. El ámbito es dicho bloque de código y no fuera de él.

En resumen, podemos indicar que si se declara un variable dentro de un bloque, la variable estará disponible dentro de ese bloque (y los bloques anidados dentro), pero no se podrá usar fuera de dicho bloque.

Parámetros

Como entrada de una función se pueden pasar diferentes valores que pueden ser:

- **Literales**: factorial(**5**)
- **Expresiones**: factorial(**3+2**)
- **Variables**: factorial(**x**)
- **O una combinación**: factorial(3+x)

Antes de llamar a la función se debe resolver las expresiones para calcular exactamente qué valor se pasa a la función.

A las entradas de las funciones se les llama parámetros.

La función puede definirse para recibir tantos parámetros como necesite: 0, 1, 2, 3, 4, ...

```
static tipo nombre(tipo1 parametro1, tipo2 parametro2, …)
```

El número de parámetros de la definición determina el número de valores que hay que utilizar en cada llamada.

Se han de pasar el mismo número de valores que de parámetros hay definidos en la función y en el mismo orden.

Ejemplo parámetros

```
// Programa principal

public static void main(String[] args) {

      int dado1 = aleatorio(1,6);



      int carta2 = aleatorio(1,12);

      //Hora aleatoria de la tarde.
      int hora_tarde = aleatorio(14,20);
}
```

```
// Función que devuelve un número
// aleatorio comprendido entre un rango
// de valores.

static int aleatorio(int inicio, int fin) {
   int valores = fin - inicio + 1;
   int resultado = (int)Math.random()*valores+inicio;
   return resultado;
}
```

Parámetros de entrada

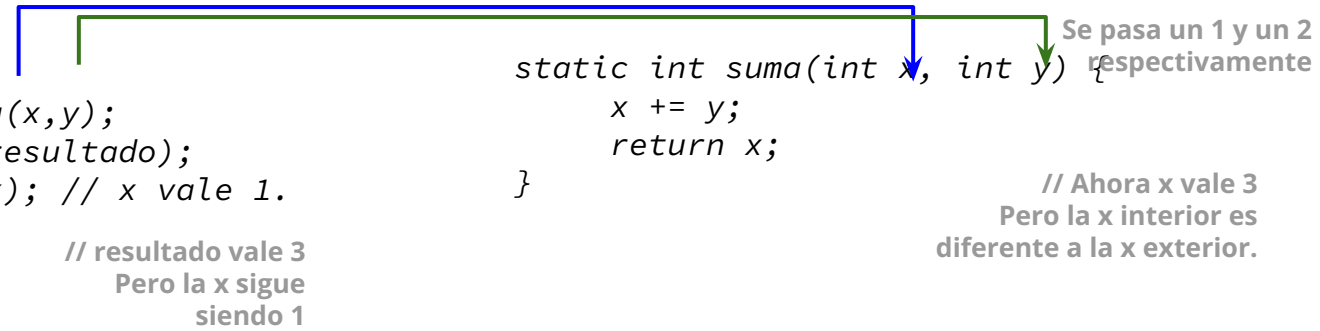
Valor devuelto por la función

Paso por valor

En Java los parámetros toman su valor como una copia de la expresión o variable utilizada.

Es decir, que no se está pasando directamente ninguna variable sino que se copia el valor que estemos pasando.

Aunque en la definición se pongan los mismos nombres de las variables, realmente son variables diferentes y no afectan al valor de la variable externa.



Retorno de la función

La salida de la función consiste en devolver un dato al bloque de código que la ha invocado.

Con esto se consigue que la llamada a una función se convierta en un valor.

Por ejemplo:

```
suma(1,2);
```

vale 3 y puede ser almacenado en una variable o usado en una expresión.

```
int resultado = suma(1,2);
System.out.println(2 * suma(1,2));
```

Siempre se ha de devolver algo y se indica el tipo de lo que se va a devolver en el prototipo de la función.

```
static int factorial(int numero) {
```

Para indicar que se devuelve se usa la palabra reservada ***return*** en el bloque. Que debe ser del mismo tipo que lo indicado en el prototipo.

return fuerza el fin de la función. Por tanto, es recomendable que sea la última instrucción.

Si una función no devuelve nada, se indica con el tipo ***void*** en el prototipo.

Sobrecarga

En Java puedes tener dos o más funciones con el mismo nombre (identificador) siempre que tengan diferentes parámetros. Puede ser diferente número de parámetros o diferentes tipos de datos.

Java sabrá qué función tiene que ejecutar de entre todas las que tienen el mismo nombre basándose en los parámetros que se usan a la hora de invocar a la función.

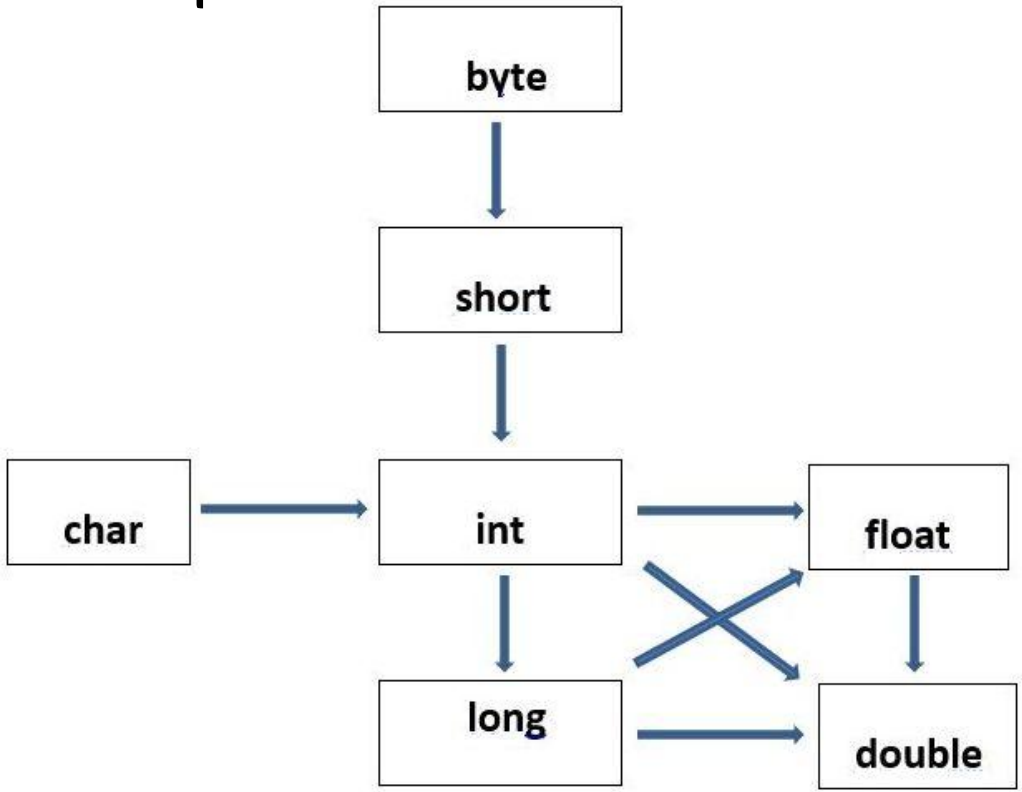
Ejemplo Sobrecarga

```
static double calcularPrecio(double precio) {
    float impuesto = 0.07;
    precio += precio * impuesto;
    return precio;
}

static double calcularPrecio(double precio, double impuesto) {
    precio += precio * impuesto;
    return precio;
}

static double calcularPrecio(double precio, double impuesto, double descuento) {
    precio -= precio * descuento;
    precio += precio * impuesto;
    return precio;
}
```

Sobrecarga conversión de tipos.

Java proporciona ciertas conversiones de tipo automáticas.

Estas conversiones también se aplican a los parámetros de métodos sobrecargados.



Recursividad

Una función puede ser invocada desde cualquier lugar:

- Desde el programa principal.
- Desde otra función.
- **Desde la propia función.** En este caso se dice que es una función **recursiva**.

```
static int recursiva() {
    …
    recursiva();     // Llamada recursiva.
    …
}
```

Recursividad infinita

Si no se controla las llamadas recursivas, se da el caso que cada vez que se vaya a ejecutar el bloque de la función, se haga una llamada recursiva y se vuelva a empezar otra vez el bloque de la función, que a su vez tiene otra llamada recursiva a la función y así infinitas veces como si se tratara de un bucle infinito.

Esto produce el **bloqueo del programa** y el error final por **falta de recursos**.

Por tanto, hay que **controlar la llamadas recursivas** mediante condiciones que se pasan como parámetro y que van guiando las llamadas recursivas hasta un **caso base** que devolverá un valor sin volver a hacer una llamada recursiva.

Ejemplo de función recursiva

La función factorial se puede resolver de forma recursiva.

Por ejemplo: 5! = 5 * 4 * 3 * 2 * 1. Que a su vez se puede expresar 5 * 4!

Por tanto: n! = n * (n-1)!

```
static int factorial(int numero) {
   int resultado;
   if (numero == 0) {     // caso base
       resultado = 1;
    } else {
    resultado = numero * factorial(numero -1);
   }
   return resultado;
}
```

Ejemplo de Sobrecarga con “Recursividad”

```
static double calcularPrecio(double precio) {
    return calcularPrecio(precio, 0.07);
}

static double calcularPrecio(double precio, double impuesto) {
    return calcularPrecio(precio, impuesto, 0);
}

static double calcularPrecio(double precio, double impuesto, double descuento) {
    precio -= precio * descuento;
    precio += precio * impuesto;
    return precio;
}
```

Realmente **no hay recursividad** porque no se está llamando a la misma función.