Programación

PLG
Planning and Learning Group

Universidad Carlos III de Madrid

Funciones

¿Qué es una función?

- Fragmento de código etiquetado con un nombre
- ¿Por qué usar funciones?
  - Evitar duplicidad en el código. Código más organizado
  - Facilitar el mantenimiento. Las modificaciones sólo se realizan en un lugar.

Llamadas a funciones

1. El programa principal se ejecuta normalmente hasta que encuentra una función
2. Entonces salta al código de la función
3. Ejecuta la función
4. Cuando la ejecución termina, el programa continua a partir del punto en el que se invocó la función

Valor de retorno de una función

- Las funciones pueden devolver un resultado cuyo tipo debe coincidir con el tipo de retorno de la función

```
<tipo> NombreFuncion (){
        <cuerpo de la función>
        return resultado
}
```

- Si la función no devuelve nada:
  - El tipo de retorno es `void`
  - La función no tiene `return`

Ejemplo

```
void saludo (){
    System.out.println("Hola");
}
```

Parámetros de una función

- Permiten pasar a la función datos de entrada

Parámetros de una función

- Permiten pasar a la función datos de entrada

```
<tipo> NombreFuncion (<tipo> parametro1, <tipo> parametro2, ...){
      <cuerpo de la función>
      return resultado
}
```

Parámetros de una función

- Permiten pasar a la función datos de entrada

```
<tipo> NombreFuncion (<tipo> parametro1, <tipo> parametro2, ...){
      <cuerpo de la función>
      return resultado
}
```

- Son variables del ámbito (bloque) de la función que ya están inicializadas

Parámetros de una función

- Permiten pasar a la función datos de entrada

```
<tipo> NombreFuncion (<tipo> parametro1, <tipo> parametro2, ...){
      <cuerpo de la función>
      return resultado
}
```

- Son variables del ámbito (bloque) de la función que ya están inicializadas

```
int Suma (int a, int b){
     int suma = a + b;
     return suma;
}
```

- Se inicializan en la llamada a la función

Parámetros de una función

- Permiten pasar a la función datos de entrada

```
<tipo> NombreFuncion (<tipo> parametro1, <tipo> parametro2, ...){
      <cuerpo de la función>
      return resultado
}
```

- Son variables del ámbito (bloque) de la función que ya están inicializadas

```
int Suma (int a, int b){
     int suma = a + b;
     return suma;
}
```

- Se inicializan en la llamada a la función
- Por ejemplo, en el programa principal:

```
int sumando1 = 3;
int sumando2 = 4
int resultadoSuma =  Suma (sumando1, sumando2);
```

¿Funciones en java?

- Programación estructurada: tipos/estructuras de datos, estructuras de control (condicionales y bucles) y funciones

¿Funciones en java?

- Programación estructurada: tipos/estructuras de datos, estructuras de control (condicionales y bucles) y funciones
- Programación orientada a objetos (POO)
  - El código se organiza de otra manera
  - Uno de los objetivos es que los datos estén junto a las funciones que los manipulan

¿Funciones en java?

- Programación estructurada: tipos/estructuras de datos, estructuras de control (condicionales y bucles) y funciones
- Programación orientada a objetos (POO)
  - El código se organiza de otra manera
  - Uno de los objetivos es que los datos estén junto a las funciones que los manipulan
- En Java (POO) NO existen funciones como tal, el código se organiza en clases

Clases, objetos y métodos

En este tema

Funciones

Clases, objetos y métodos

- Clases y objetos
- Métodos
  - Constructores
  - Resto Métodos

Clases

- Una clase define de forma abstracta:
  - Un conjunto de variables, cada una con su tipo y nombre: atributos
  - Un conjunto de funciones que por lo general manipulan esos datos: métodos

Clases

- Una clase define de forma abstracta:
  - Un conjunto de variables, cada una con su tipo y nombre: atributos
  - Un conjunto de funciones que por lo general manipulan esos datos: métodos
- Una clase se define en un fichero .java cuyo nombre debe coincidir con el nombre de la clase
- Los atributos pueden ser de distinto tipo
- El ámbito de los atributos es global a toda la clase

Ejemplo de definición de una clase y sus atributos

```
public class Fecha {
    public int dia;
    public String mes;
    public int anyo;
    public String diaSemana;
    public boolean festivo;
}
```

Objetos

- Una vez definida la clase, se pueden declarar variables del *“tipo de datos”* que define la clase
- Una variable cuyo tipo es una clase contiene una referencia a un objeto
- Un objeto es un elemento concreto que pertenece a esa clase: una instancia de esa clase

Declaración de variables de una clase

- Se hace en el código de una clase distinta
- Se expresa como si se declarara un tipo de dato cualquiera
- Hay que crear el objeto reservando el espacio de memoria necesario con el operador new

```
Fecha hoy = new Fecha();
Fecha ayer;

ayer = new Fecha();
```

Declaración de variables de una clase

- Se hace en el código de una clase distinta
- Se expresa como si se declarara un tipo de dato cualquiera
- Hay que crear el objeto reservando el espacio de memoria necesario con el operador new

```
Fecha hoy = new Fecha();
Fecha ayer;

ayer = new Fecha();
```

- Nos referiremos a las variables que contienen referencias a objetos directamente como objetos

Acceso a atributos

- La sintaxis para acceder a los atributos es:
  `<nombreObjeto>.<nombreAtributo>`
- Ejemplo:

```
hoy.dia = 24;
hoy.mes = “Octubre”;
hoy.anyo = 2015;

ayer.dia = hoy.dia - 1;
ayer.mes = hoy.mes;
ayer.anyo = hoy.anyo;
```

Valores Iniciales de los Atributos

- Cuando se crea un objeto, los atributos tienen valor por defecto.
  - 0 para los números
  - `false` para los `boolean`
  - la cadena vacía para los `char`
  - `null` para los `String`

Valores por Defecto de los Atributos

- También se puede declarar la clase con valores por defecto para los atributos

```
public class Fecha {
    public int dia = 1;
    public String mes = "Enero";
    ...
}
```

- En este caso todos los objetos que se creen de esa clase, tendrán esos valores por defecto en sus atributos

Asignación y Copia de Objetos

- Un objeto es un puntero a una dirección de memoria que contiene los atributos del objeto y alguna otra información adicional
- Asignar objetos con '=' copia el puntero
- Comparar objetos con '==' compara el puntero
- Para copiar un objeto en otro hay que crear el nuevo objeto y copiar atributo a atributo
- Para comparar objetos hay que comparar atributo a atributo

En este tema

Funciones

Clases, objetos y métodos

- Clases y objetos
- Métodos
  - Constructores
  - Resto Métodos

Métodos

- Los métodos son funciones que se implementan dentro de una clase
- Los métodos implementan operaciones comunes a todos los objetos de la clase
- Los atributos son accesibles a todos los métodos de una clase

En este tema

Constructores

- Los constructores son métodos especiales que sirven para asignar valores a los atributos de un objeto en el **momento en que éste se crea con** `new`
- Los constructores tienen el mismo nombre que la clase

Constructores

- Los constructores son métodos especiales que sirven para asignar valores a los atributos de un objeto en el **momento en que éste se crea con** `new`
- Los constructores tienen el mismo nombre que la clase
- Existen dos tipos:
  - Constructor por defecto: no tiene argumentos: se utiliza si los valores de los argumentos se obtienen mediante cálculos
  - Constructor completo: sus argumentos representan el valor de los atributos al crear el objeto

Ejemplo de constructor

```
public class Fecha {
    public int dia;
    public String mes;
    public int anyo;
    public String diaSemana;
    public boolean festivo;

    public Fecha (int d, String m, int a, String ds, boolean f) {
        dia = d;
        mes = m;
        anyo = a;
        diaSemana = ds;
        festivo = f;
    }
}
```

Otros ejemplos de constructor

```
public class Fecha {
    public int dia;
    public String mes;
    public int anyo;
    public String diaSemana;
    public boolean festivo;

    public Fecha (int dia, String mes, int anyo, boolean festivo) {
        this.dia = dia;
        this.mes = mes;
        this.anyo = anyo;
        this.festivo = festivo;

    public Fecha (int anyo) {
        this(1,”Enero”,anyo,,true);
    }
}
```

En este tema

Otros Métodos

- Se sitúan dentro de la clase
- Habitualmente después de los *atributos*
- Y antes del *main* si existe

Métodos – estructura básica: cabecera + cuerpo

```
<visibilidad> <modificador-static> <tipo> nombreMetodo (<tipo> param1, <tipo> param2, ...)
      <cuerpo del método>
      return resultado
```

Métodos – estructura básica: cabecera + cuerpo

```
<visibilidad> <modificador-static> <tipo> nombreMetodo (<tipo> param1, <tipo> param2, ...)
      <cuerpo del método>
      return resultado
```

- Son funciones comunes a todos los objetos de la clase:
  - Tipo de retorno: tipo del valor que devuelve el método
  - Nombre del método: en minúsculas (*camelCase*) por convención
  - Parámetros del método: lista de parámetros con su tipo

Métodos – estructura básica: cabecera + cuerpo

```
<visibilidad> <modificador-static> <tipo> nombreMetodo (<tipo> param1, <tipo> param2, ...)
      <cuerpo del método>
      return resultado
```

- Son funciones comunes a todos los objetos de la clase:
  - Tipo de retorno: tipo del valor que devuelve el método
  - Nombre del método: en minúsculas (*camelCase*) por convención
  - Parámetros del método: lista de parámetros con su tipo
- Con algunos elementos más específicos de POO:
  - <visibilidad>
  - <modificador-static>

Ejemplo de Métodos – en una clase cualquiera

```
public class Fecha {
    public int dia;
    public String mes;
    public int anyo;
    public String diaSemana;
    public boolean festivo;
```

Ejemplo de Métodos – en una clase cualquiera

```
public class Fecha {
    public int dia;
    public String mes;
    public int anyo;
    public String diaSemana;
    public boolean festivo;

    /** CONSTRUCTOR **/
    public Fecha (int dia, String mes, int anyo, boolean festivo) {
        this.dia = dia;
        this.mes = mes;
        this.anyo = anyo;
        this.festivo = festivo;
```

Ejemplo de Métodos – en una clase cualquiera

```
public class Fecha {
    public int dia;
    public String mes;
    public int anyo;
    public String diaSemana;
    public boolean festivo;

    /** CONSTRUCTOR **/
    public Fecha (int dia, String mes, int anyo, boolean festivo) {
        this.dia = dia;
        this.mes = mes;
        this.anyo = anyo;
        this.festivo = festivo;

    /** OTROS MÉTODOS **/
    public int anyoSiguiente () {
        return anyo+1;
    }
```

Ejemplo de Métodos – en una clase cualquiera

```
public class Fecha {
    public int dia;
    public String mes;
    public int anyo;
    public String diaSemana;
    public boolean festivo;

    /** CONSTRUCTOR **/
    public Fecha (int dia, String mes, int anyo, boolean festivo) {
        this.dia = dia;
        this.mes = mes;
        this.anyo = anyo;
        this.festivo = festivo;

    /** OTROS MÉTODOS **/
    public int anyoSiguiente () {
        return anyo+1;
    }

    public void printDia () {
        System.out.println("El día es: " + dia);

    }
}
```

Ejemplos de Métodos – en la clase del programa principal

```
public class  ProgramaPrincipal {
```

Ejemplos de Métodos – en la clase del programa principal

```
public class  ProgramaPrincipal {

     static int  maximo (int a, int b) {
         int  max;
         if  (a > b)
             max = a;
         else
             max = b;
         return max;
      }
```

Ejemplos de Métodos – en la clase del programa principal

```
public class  ProgramaPrincipal {

     static int  maximo (int a, int b) {
         int  max;
         if  (a > b)
             max = a;
         else
             max = b;
         return max;
      }

     public static void main (String [] args) {
         int a = 2, b = 3, c = 4, max1, max2, max3;
         max1 = maximo (a, b);
         max2 = maximo (b, c);
         max3 = maximo (max1, max2);
      }
 }
```

Valor de retorno

- Es `void` en caso de que no devuelva nada (método sin `return`)
- El `return` sólo puede devolver un único dato

Paso de parámetros

- Los parámetros son variables locales al método que ya están inicializadas
- Se inicializan en la llamada al método
- Paso de parámetros por valor vs. paso de parámetros por referencia

Paso de parámetros

Paso de parámetros POR VALOR

- El parámetro contiene una copia del valor, variable o expresión que se utiliza en la llamada
- Cuando se modifica un parámetro se modifica la copia y no el dato original. ¡Los cambios no permanecen al salir de la función!
- Los tipos básicos y *Strings* en Java se pasan por valor

Paso de parámetros

Paso de parámetros POR REFERENCIA

- La variable que se utiliza en la llamada contiene una dirección (referencia)
- El parámetro contiene esa referencia
- La referencia permite acceder al valor original. ¡Los cambios permanecen al salir de la función!
- Los *arrays* y objetos se pasan por referencia

Ejemplo – paso de parámetros

```
public static int SumaUno(int a) {
        a++;
        return a; }
```

Ejemplo – paso de parámetros

```
public static int SumaUno(int a) {
        a++;
        return a; }

public static int [] SumaUno(int [] a) {
        a[0]++;
        return a; }
```

Ejemplo – paso de parámetros

```
public static int SumaUno(int a) {
        a++;
        return a; }

public static int [] SumaUno(int [] a) {
        a[0]++;
        return a; }

public static void main(String[] args)
        int a = 3;
        int [] b = {3};
        System.out.println("a: " + a);
        System.out.println("b[0]: " + b[0]);
        SumaUno(a);
        SumaUno(b);
        System.out.println("a: " + a);
        System.out.println("b[0]: " + b[0]);
```

Documentación Javadoc

Se deben documentar los métodos

- Explicar qué hace el método y qué devuelve
- Explicar qué representan los parámetros

Ejemplo – comentarios Javadoc

```
/**
* The AddNum program implements the addition of two given integers.
*
* @author  Zara Ali
* @version 1.0
* @since   2014-03-31
*/
public class AddNum {

   /**
   * This method is used to add two integers.
   * @param numA This is the first paramter to addNum method
   * @param numB  This is the second parameter to addNum method
   * @return int This returns sum of numA and numB.
   */
   public static int addNum(int numA, int numB) {
      return numA + numB;   }
   /**
   * This is the main method which makes use of addNum method.
   * @param args Unused.
   * @return Nothing.
   */
   public static void main(String args[]) {
     int sum = addNum(10, 20);
     System.out.println("Sum of 10 and 20 is :" + sum);
  }
}
```

Ejemplo – comentarios Javadoc

```
/**
* The AddNum program implements the addition of two given integers.
*
* @author  Zara Ali
* @version 1.0
* @since   2014-03-31
*/
public class AddNum {

   /**
   * This method is used to add two integers.
   * @param numA This is the first paramter to addNum method
   * @param numB  This is the second parameter to addNum method
   * @return int This returns sum of numA and numB.
   */
   public static int addNum(int numA, int numB) {
      return numA + numB;   }
   /**
   * This is the main method which makes use of addNum method.
   * @param args Unused.
   * @return Nothing.
   */
   public static void main(String args[]) {
     int sum = addNum(10, 20);
     System.out.println("Sum of 10 and 20 is :" + sum);
  }
}
```

Sobrecarga de métodos

- Dos o más métodos con el mismo nombre en el mismo ámbito
  - En una clase
  - En el programa principal
- Se distinguen por su número de parámetros y el tipo de los mismos
- Los métodos sobrecargados pueden devolver distinto tipo
- Aunque el tipo de retorno no se usa para diferenciarlos

Ejemplos – métodos sobrecargados

```
public class  ProgramaPrincipal {

     static int  maximo (int a, int b) {
         int  max;
         if  (a > b)
             max = a;
         else
             max = b;
```

Ejemplos – métodos sobrecargados

```
public class  ProgramaPrincipal {

     static int  maximo (int a, int b) {
         int  max;
         if  (a > b)
             max = a;
         else
             max = b;
         return max;
      }

     static int  maximo (int a, int b, int c) {
```

Ejemplos – métodos sobrecargados

```
public class  ProgramaPrincipal {

     static int  maximo (int a, int b) {
         int  max;
         if  (a > b)
             max = a;
         else
             max = b;
         return max;
      }

     static int  maximo (int a, int b, int c) {
         return maximo(maximo(a,b), maximo(b,c));
      }
```

Ejemplos – métodos sobrecargados

```
public class  ProgramaPrincipal {

     static int  maximo (int a, int b) {
         int  max;
         if  (a > b)
             max = a;
         else
             max = b;
         return max;
      }

     static int  maximo (int a, int b, int c) {
         return maximo(maximo(a,b), maximo(b,c));
      }

     public static void main (String [] args) {
         int a = 2, b = 3, c = 4, max1, max2, max3;
         max3 = maximo (a, b ,c);
      }
 }
```

Métodos estáticos (`<modificador-static>`)

- Cada objeto de una clase tiene una copia de los atributos y una copia de los métodos
- Si un método no utiliza los atributos puede ser `static`: no hace falta una copia en cada objeto
- Un método **static** es un método de clase
- ¡No hace falta un objeto para invocarlo!

Invocación de métodos y acceso a atributos en objetos

- Una vez creado un objeto podemos acceder a sus atributos y métodos utilizando un punto:

<objeto>.<atributo o método>

Ejemplo – acceso métodos y atributos

Fecha.java

```
public class Fecha {
    public int dia;
    public String mes;
    public int anyo;
    public String diaSemana;
    public boolean festivo;
```

Ejemplo – acceso métodos y atributos

Fecha.java

```
public class Fecha {
    public int dia;
    public String mes;
    public int anyo;
    public String diaSemana;
    public boolean festivo;

    public Fecha (int dia, String mes, int anyo, boolean festivo) {
        this.dia = dia;
        this.mes = mes;
        this.anyo = anyo;
        this.festivo = festivo;
```

Ejemplo – acceso métodos y atributos

Fecha.java

```
public class Fecha {
    public int dia;
    public String mes;
    public int anyo;
    public String diaSemana;
    public boolean festivo;

    public Fecha (int dia, String mes, int anyo, boolean festivo) {
        this.dia = dia;
        this.mes = mes;
        this.anyo = anyo;
        this.festivo = festivo;

    public int anyoSiguiente () {
        return anyo+1;} }
```

Ejemplo – acceso métodos y atributos

Fecha.java

```
public class Fecha {
    public int dia;
    public String mes;
    public int anyo;
    public String diaSemana;
    public boolean festivo;

    public Fecha (int dia, String mes, int anyo, boolean festivo) {
        this.dia = dia;
        this.mes = mes;
        this.anyo = anyo;
        this.festivo = festivo;

    public int anyoSiguiente () {
        return anyo+1;} }
```

ProgramaPrincipal.java

```
public class  ProgramaPrincipal {
    public static void main (String [] args) {
        Fecha miFecha= new Fecha(19, ''Mayo'', 2015, false);
        int anyo = miFecha.anyo();
        int anyoSig = miFecha.anyoSiguiente(); }
```

Modificadores de acceso (<visibilidad>)

- Por defecto, los atributos y métodos son accesibles (visibles) dentro del paquete en el que se define la clase
- Para limitar este acceso se pueden utilizar modificadores de acceso
  - `public`
  - `private`

Modificadores de acceso (`<visibilidad>`)

- Modificador **public**
  - En una clase, significa que es accesible fuera del paquete donde está definida. Para acceder a ella hay que importarla mediante `import`
  - En un atributo o método, significa que es accesible donde lo sea la clase. Si no se pone nada el acceso es `public` por defecto

Modificadores de acceso (<visibilidad>)

- Modificador **private**
  - Sólo en atributos y métodos
  - En atributos
    - No permite que se acceda a ellos desde fuera de la clase ni para leer ni para modificar el atributo
    - El acceso a atributos privados se puede realizar mediante métodos públicos
  - En métodos impide invocarlos desde fuera de la clase

¿Por qué existen los modificadores de acceso?

¿Por qué existen los modificadores de acceso?

Encapsulación

Características de la POO

- Reutilización
- Organización del código
- Específicas de la POO
  - Encapsulación
  - Herencia
  - Polimorfismo

Ejemplo – modificador `private` en atributos

```
public class SerieAritmetica {
```

Ejemplo – modificador `private` en atributos

```
public class SerieAritmetica {

     private int a1; // primer término
     private int inc;//incremento
```

Ejemplo – modificador `private` en atributos

```
public class SerieAritmetica {

     private int a1; // primer término
     private int inc;//incremento

     SerieAritmetica (int a1, int inc){
         this.a1 = a1;
         this.inc = inc; }
```

Ejemplo – modificador `private` en atributos

```
public class SerieAritmetica {

      private int a1; // primer término
      private int inc;//incremento

      SerieAritmetica (int a1, int inc){
          this.a1 = a1;
          this.inc = inc; }

      public int Suma (int n){
          int suma = 0;
          for(int i = 0; i< n; i++)
                suma += a1 + i * inc;
          return suma;}

      }
```

Métodos comunes – acceso a atributos privados

- Método **set**: para dar valor a un atributo privado
- Método **get**: para obtener el valor de un atributo privado

Ejemplo – métodos `set` y `get`

```
public class SerieAritmetica {
     private int a1; // primer término
     private int inc;//incremento

     SerieAritmetica (int a1, int inc){
         this.a1 = a1;
         this.inc = inc; }
```

Ejemplo – métodos `set` y `get`

```
public class SerieAritmetica {
     private int a1; // primer término
     private int inc;//incremento

     SerieAritmetica (int a1, int inc){
         this.a1 = a1;
         this.inc = inc; }


      /* EJEMPLO set  */
     public void seta1 (int a1){
         this.a1 = a1;}
```

Ejemplo – métodos `set` y `get`

```
public class SerieAritmetica {
     private int a1; // primer término
     private int inc;//incremento

     SerieAritmetica (int a1, int inc){
         this.a1 = a1;
         this.inc = inc; }


      /* EJEMPLO set  */
     public void seta1 (int a1){
         this.a1 = a1;}

      /* EJEMPLO get */
     public int geta1 (){
         return a1;}

     }
```

Métodos comunes – escribir un objeto en un String

- Método **toString**: para escribir un objeto en un `String`
- Se suele utilizar para imprimir el objeto en un formato adecuado

Ejemplo – métodos `toString`

```
public class SerieAritmetica {
      private int a1; // primer término
      private int inc;//incremento

      SerieAritmetica (int a1, int inc){
          this.a1 = a1;
          this.inc = inc; }


       /* EJEMPLO toString */
      public String toString (){
          String string = "";
          for(int i = 0; i< n; i++)
                string += a1 + i * inc+ " ";
          return string;}

      }
```

Métodos comunes – comparación de objetos

- Método **equals**: para determinar si dos objetos son iguales

Ejemplo – métodos `equals`

```
public class SerieAritmetica {
     private int a1; // primer término
     private int inc;//incremento

     SerieAritmetica (int a1, int inc){
         this.a1 = a1;
         this.inc = inc; }

      /* EJEMPLO equals */
     public boolean equals (SerieAritmetica otraSerie){
            if (a1 == otraSerie.a1 && inc== otraSerie.inc)
                  return true;
            return false;}
    }
```