Tema 8: Manejo de Métodos

Miembros de una clase

- Una clase puede contener variables y métodos. Las variables pueden ser tipos primitivos (int, char, etc.) Los métodos son funciones.

- Ejemplo:

```
public MiClase {
    int i;
    public MiClase() {
        i = 10;   }
    public void Suma_a_i( int j ) {
        i = i + j;
    }
}
```

Métodos

- Un método debe ser definido dentro de una clase.

- Sintaxis

```
[m. acceso] [m. Comport.] tiporetorno
    nombreMetodo([tipo arg1, tipo arg2, ...])
            [throws Excepcion_1, ... , Excepcion_n] {
//  declaraciones, Cuerpo de método
}
```

- Invocación

```
//Desde otra clase diferente o desde el main:
Variableobjeto.nombreMetodo([argumentos]);
// Si es dentro de la misma clase :
nombreMetodo ([argumentos]);
```

- Ejemplo:

```
int producto ( int a, int b ) {
        int c = a * b;
        return c;
    }  // Invocación:  int x=objeto,producto(5,4);
```

Características

- No hay límite en el número de llamadas a un método.
- La invocación puede hacerse dentro de la misma clase o desde otra.
- La invocación de métodos puede ser en cualquier orden no necesariamente como estás listados en la clase.
- Paso de argumentos

  Deberá de existir un método con la misma cantidad y orden del tipo de dato que se recibe como parámetros.
- Retorno de valor
  - ✓ Un método sólo puede retornar un valor (o varios si es un String, un array o una colección).
  - ✓ El compilador no asume un tipo de retorno por default.

Argumentos

- Paso de argumentos por valor

  Al ser una copia, el dato original no se altera en caso de que el método cambiara el valor de la copia.

  En Java los tipos primitivos siempre se pasan por valor.

- Ejemplo

```
void a(int n){  // a recibe un valor en el argumento n
    n=n+1;      //  a modifica su copia.
}

void b(){
    int n=0;   // n vale 0
    a(n);      // pasa n en 0 como argumento real
   System.out.println(n);  // n sigue con valor 0
}
```

Argumentos

- Paso de argumentos por referencia

  Cuando se pasa una referencia al dato. Aunque el método no puede alterar la referencia propiamente dicha, si puede alterar aquello a que se refiere la referencia.

  Java pasa todo por valor aquello que no sean primitivos como un arreglo y objetos.

- Ejemplo

```
void a(int datos[ ]){  // recibe la referencia de un array
    datos[0]=1;      //  modifica el array compartido
}

void b(){
    int datos[] =new int[2];
    datos[0]=0;    // la 1ra posicion contine 0;
    a(datos);       // se pasa a a una referencia al array
   System.out.println(datos[0]);  // imprime 1
}
```

Ejemplo

```
void a(int datos[ ]){  // recibe la referencia de un
     array
   datos=new int[2];  // crea su propia copia
    datos[0]=1;      //  modifica el array compartido
}

void b(){
    int datos[] =new int[2];
    datos[0]=0;    // la 1ra posicion contine 0;
    a(datos);    // se pasa a a una referencia al array
   System.out.println(datos[0]);  // imprime 0
}
```

Constructor

- Definición

- Un constructor es un método que lleva el mismo nombre de la clase.
- Se invoca automáticamente cada vez que se crea un nuevo objeto (por default).
- Sirve para inicializar las variables de instancia del objeto.
- Un constructor puede incluir parámetros. Estos parámetros se enlazan con los argumentos especificados al crear un objeto con **new**.
- Una clase puede definir varios constructores con distintos tipos de argumentos (sobrecarga).

Constructor

- Sintaxis

```
[modificadores de aceso]
    Nombre_Clase([argumentos]) {

   // Acciones

}
```

Ejemplo 1

```
class Datos {
   String name;
   int height;
   int weight;
 Datos() { name = “Luis"; height = 70; weight = 75; }
 Datos(String theName) {
     name = theName; height = 70; weight = 75; }

 public static void main (String args[]) {
     Datos d = new Datos(); // llama al 1er constructor
     System.out.println(“d.nombre = " + d.name);
     System.out.println(“d.altura = " + d.height);
     System.out.println(“d.peso = " + d.weight);

     Datos p = new Datos(“Francisco"); // llama al 2do
     constructor
     System.out.println("p.nombre = " + p.name);
     System.out.println("p.altura = " + p.height);
     System.out.println("p.peso = " + p.weight);
  // Datos err = new Datos(37.2,”Manuel”);
  // No hace match con algún constructor
   }
 }
```

Ejemplo 2

```
public class Canario{
   public void vuela(){
         System.out.println(“Estoy volando”);
}
//Está la llama del constructor por default que es
    vacío y sin parámetros.
public static void main(Sting[] args){
Canario piolin=new Canario();
piolin.vuela();
}
```

Ejemplo 3

```
public class Canario{
   public void vuela(){
         System.out.println(“Estoy volando”);
}
public Canario(){
System.out.println(“Mensaje en creacion”);
}

public static void main(Sting[] args){
Canario piolin=new Canario();
piolin.vuela();
}
```

Recordar

- A manera de resumen, considerar los puntos siguientes:

  - ✓ Una Clase es una definición de un nuevo Tipo, al que se da un nombre.
  - ✓ Una Clase contiene miembros: Datos y Métodos que conforman el estado y las operaciones que puede realizar.
  - ✓ Un Objeto es la instanciación (materialización) de una clase. Puede haber tantos objetos de una clase como sea necesario.
  - ✓ Los Objetos se crean (asignación memoria) con el operador **new**.
  - ✓ Los Objetos se manipulan con referencias.
  - ✓ Una Referencia es una variable que apunta a un objeto.
  - ✓ El acceso a los elementos de un Objeto (Datos o métodos) se hace con el operador **.** (punto) : *nombre_referencia.miembro*

Sobrecarga

- Definición y características
- Es definir más de una vez el mismo método (incluyendo al constructor).
- Con diferentes argumentos (tipo, número, y orden) y pueden o no tener diferente tipo de retorno.
- La sobrecarga se da en la misma clase.
- El modificador no interfiere en la sobrecarga.
- Puede haber o no herencia.

Sobrecarga de constructores

- Ejemplos:

```
class Ejemplo1{
   public void metodo (int a){ }
   public int metodo (int b){ }
}
class Ejemplo2{
   public void metodo (int a,float b){ }
   public int metodo (float a,int b){ }
}
```

- También se puede invocar un constructor dentro de otro.

Ejemplo:

```
class Point {
   int x, y;
   Point(int x, int y) {
          this.x = x;
          this.y = y;  }
    Point() {
     x = -1; y = -5; }
 }
class PointCreate {
    public static void main (String args[]) {
      Point p = new Point(10, 20);
      System.out.println("x = " + p.x + " y
   = " + p.y);
      Point p1 = new Point();
      System.out.println("x = " + p1.x + " y
   = " + p1.y);
   }
}
```

Sobrecarga de métodos

- Existe la sobrecarga de métodos usando el mismo nombre de un método pero dependiendo del número, tipo y orden de los argumentos (signature) es la invocación de un método.

- Los métodos pueden tener el mismo o diferente valor de retorno.

- Ejemplo:

  System.out.printl(xxx);

Sobrecarga de métodos

- Ejemplo (pagina 10-16): class Statistics

  Se muestra el uso de diferente número de argumentos del mismo tipo

- Ejemplo 2

```
class SumaGenerica {
    int suma (int a, int b) {
        return a+b; }
    double suma (double a, double b) {
        return a+b; }
 ... }
```

Varargs

- Uso desde la ver. 1.5
- Permiten escribir métodos mas genéricos, llamada también variable arguments.

- Sintaxis:

  tipo … Identificador

  ✓ int... x;
  ✓ int ...x;
  ✓ int ... x;
  X int . . . x;

- Todos los argumento son del mismo tipo.

Varargs

- Notas:

1 En la sobrecarga si hay un método que haga exactamente match se toma en cuenta dicho método y no el que tenga las vararg.

2 También puede usarse en la sobreescritura haciendo primero match con algún método del padre sino con uno del hijo.

Ejemplo Varargs

```
class VarArgs1 {
   static void vaTest(int ... v) {
    System.out.print("vaTest(int ...): " +
                     "Numero de args: " + v.length +
                     " Contenido: ");
    for(int x : v)
      System.out.print(x + " ");
      System.out.println();
  }

  static void vaTest(boolean ... v) {
    System.out.print("vaTest(boolean ...) " +
                     "Numero de args: " + v.length +
                     " Contenido: ");
    for(boolean x : v)
      System.out.print(x + " ");
    System.out.println();
  }
```

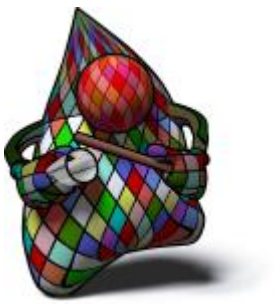
Ejemplo... Varargs

```
//continuación

  static void vaTest(String msg, int ... v) {
    System.out.print ("vaTest(String, int ...): " +
     msg + v.length + " Contenido: ");

  for(int x : v)
      System.out.print(x + " ");
   System.out.println();
  }

  public static void main(String args[])
  {
      vaTest(1, 2, 3);
      vaTest(“Prueba: ", 10, 20);
      vaTest(true, false, false);
  }
}
```

Implementando encapsulación

- Encapsulación se refiere a ocultar datos sin una clase y hacerlos disponibles a través de métodos públicos, esto con el fin de que los datos no sean modificados inapropiadamente.

- Implementación es/son los detalles de cómo una clase manipula o trabaja a partir de los métodos públicos y variables públicas.



Métodos get y set

- Accesar a atributos y métodos privados es posible siempre y cuando se tengan métodos públicos para obtener y devolver un valor privado.

- Métodos de acceso (getters)

  Se dice de aquellos métodos que devuelven el valor de un campo del objeto.

  Por convenio se escriben iniciando con la palabra get seguida del nombre del campo. El tipo de retorno es del tipo de dato que se accede.

- Métodos de carga(setters)

  Se dice de aquellos que cargan el valor de un campo de objeto.

Ejemplos get y set

- Método de acceso:

```
class Circulo{
    private double radio; //. . .
     double getRadio(){
         return radio;
  }
}
```

- Método de carga:

```
class Circulo{
    private double radio; // . . .
     double setRadio(double radio){
         return this.radio=radio;
  }
}
```

static

- Todo lo declarado como static se crea una vez.
- Si un atributo (variable) es declarado como “static” se vuelve variable de clase.
- Si un método se declara “static” se vuelve método de clase.
- Un objeto static es compartido por todas las demás instancias, si una de esas instancias modifica su estado se habrá modificado para todas las demás instancias que lo usen.

Métodos static

- Todos los métodos estáticos pueden usar métodos y atributos estáticos así como sus variables locales.
- No se pueden sobreescribir métodos estáticos.
- El método main siempre será estático.
- Puede ser invocado sin una instancia de la clase.

```
NombreClase.metodoStatico();
```

- Un método estático no puede accesar a miembros no estáticos directamente, tiene que crear primero un objeto?.

Métodos static

- Ejemplo 1

Pagina 7-23

- Ejemplo 2

```
class Punto {
        int x , y ;
        static int numPuntos = 0;

        Punto ( int a , int b ) {
            x = a ; y = b;
            numPuntos ++ ;
        }

        static int cuantosPuntos() {
            return numPuntos;
        }
    }
//La llamada es usando sólo el nombre de la clase:

int totalPuntos = Punto.cuantosPuntos();
```

Ejemplos

- Ejemplo 3

```
static void Inc2() {
    cv++;
 }

 static void Inc3() {
        iv++;
} // error iv es de un objeto

 static void Inc4() {
   Inc(); // error Inc necesita un obj.
   Inc2(); // Ok, porque Inc2 es static
}
```

Variables static

- El modificador de comportamiento permite que la variable actúe como “global”.
- Dentro de la misma clase no se coloca el nombre de la clase, únicamente el nombre de la variable.
- Sintaxis declaración:

  static tipo identificador;
- Invocación

  NombreClase.identificador;

Ejemplo

```
class VarNativos{
    public static boolean vB;
    private static short vS;
    private static int vI;
    public float vF;
}

public class Main{
    public static void main(String[] args){
    System.out.println(VarNativos.vB);
    //System.out.println(vS); //error porque es privado
    VarNativos x=new  VarNativos();
    System.out.println(x.vF);

    }
}
```

Bloques static

- El bloque estático se ejecuta cuando la clase es cargada. Si hay muchos bloques estáticos en una clase se ejecutan en orden de aparición.
- Ejemplo:

```
class StaticBlock {
   static int i = 20;
   static {
      System.out.println(“El bloque static dice:
    i = " + i);
      System.out.println(“Yo soy mas rápido
    que el main !!");
       i *= 2; }
  public static void main(String args[]) {
    System.out.println(“Main() dice: i = " + i);
    System.out.println(“pensé que era el
    primero...");
  }
 }
```

Bloques static

- Usados también para inicializar campos static (v. de clase).

- Ejemplo:

```
class Diccionario {
   public static final String  diccionario[ ][ ];
   static {
      diccionario=new String[2][2];
      diccionario[0]=new String[ ] {“Hola”,”hi” };
      diccionario[1]=new String[ ] {“Adios”,”bye”
    };
}
```

Ejemplo static

```
class Estatica {
    static int a = 3;
    static int b;
    static void show(int x) {
            System.out.println("x = " + x);
            System.out.println("a = " + a);
            System.out.println("b = " + b);
    }
    static {
            System.out.println("Bloque estático
     inicializado");
             b = a * 4; // aunque declarada como static
             //b se inicializa en tiempo de ejecución
     }
    public static void main(String args[]) {
            show(15);
    }
  }
```

Métodos final

- final es un modificador de comportamiento.
- Cuando un método es final se indica que no puede ser sobreescrito.
- Si este modificador se aplica a una clase ésta no puede heredar.

Variables final

- Este tipo de variables indican que su valor no podrá ser modificado tras su carga inicial.

- Si una variable es public static final además se dice que dicha variable es una cosntante.

- Sintaxis:

[modificador] **final** tipo_dato NOMBRE[=valor];

- Ejemplo

```
public static final int  VAR=18;
```

Ejemplos

- blank final variable. Es una variable final que no ha sido inicializada, puede ser asignada en el constructor pero sólo UNA vez.

- Ejemplo

```
 final float vLuz; ...
// sentencias del programa...
   vLuz = 3.1+ 5F;  //inicialización
   otraVar= vLuz * 2;
```

- Ver otros ejemplos en el tema 11(herencia)

Invocación de un método dentro de otro

- I

```
public class Canario{
   public void come(){
        System.out.println(“Comiendo alpiste”);
}
   public void vuela(){
          System.out.println(“Estoy volando”);
   }
    public Canario(){
              System.out.println(“Me pareció ver un lindo
     gatito”);
     come();
   }
   public static void main(Sting[] args){
            Canario piolin=new Canario();
            piolin.vuela();
   }
}
```

Método recursivo

- Un método recursivo es aquel que se llama (directa o indirectamente) a sí mismo.

- Debe contener:

  Un condicional:

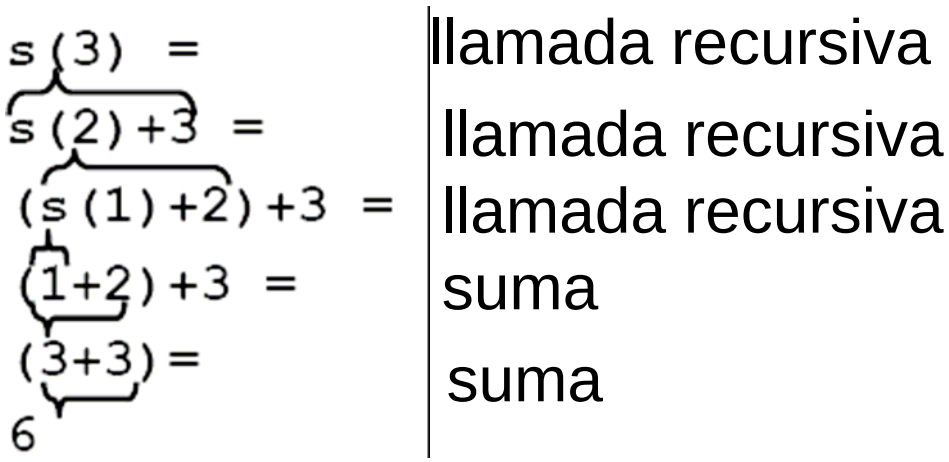
  ✓ Un caso no recursivo

  ✓ Un caso recursivo (que se aproxima a la condición de parada)

Ejemplo1 recursión

```
public static long metodo1(int s){
    if (s == 1)
        return
    else
        return  metodo1(s-1)+n;
}
```

Si s vale 3 :



Ejemplo2 recursión

```
public static long ack(int n, int m){
      if (n == 0)
            return (m+1);
      else if (m == 0)
            return  ack(n-1,1);
      else
            return ack(n-1,ack(n,m-1);
 }
```

Alcance de las variables

- Los objetos tienen un tiempo de vida y consumen recursos durante el mismo. Cuando un objeto no se va a utilizar más, debería liberar el espacio que ocupaba en la memoria de forma que las aplicaciones no la agoten (especialmente las grandes).

- Recordar que en Java, la recolección y liberación de memoria es responsabilidad del GC.

- De acuerdo al tipo de variable es el tiempo de vida que ocupan durante la ejecución del programa.

Alcance: variables locales

- Se crean cada vez que se ejecuta al bloque, pero sólo a partir del punto en el que se declararán. Desaparece cuando el bloque termina.

- Cuando se definen en el cuerpo de un método (o como parámetros) y si hay 25 llamadas, se crean 15 veces. Si un método se llama así mismo, se generan nuevas variables en el nuevo método. Desaparecen cuando el método devuelve su resultado.

Alcance: variables de referencia

- Variables de instancia.

  Llamadas también de objeto. Se crean al crear al objeto (con new) y desaparecen cuando el objeto deja de utilizarse.

- Variables de clase.

  Llamadas también static. Se crean al arrancar el programa y se destruyen cuando termina el programa.
  Siempre están disponibles