UCA

Programación Orientada a Objetos I

JAVA – ELEMENTOS BÁSICOS (III)

Bibliotecas de clases

JAVA ofrece un conjunto de clases disponibles y garantizadas para ser usadas en cualquier ambiente JAVA.

- **Reutilización de código**: contienen código preescrito y probado. Permite reutilizar funcionalidades comunes sin reescribir código, ahorro de tiempo y esfuerzo.
- **Funcionalidad amplia**: manipulación de archivos, manejo de E/S, procesamiento de cadenas, matemáticas, redes, GUI, bases de datos, etc.
- **Consistencia**: Están diseñadas siguiendo estándares y prácticas recomendadas, lo que garantiza la consistencia y la calidad en el código.
- **Documentación**: Suelen ir acompañadas de documentación detallada que describe cómo usar sus componentes, sus parámetros y su funcionalidad. Esto facilita la comprensión y el uso de la biblioteca.
- **Amplia comunidad de usuarios**: Generalmente tienen una comunidad activa de usuarios y desarrolladores que comparten conocimientos y experiencias, lo que facilita la resolución de problemas y el aprendizaje.
- **Portabilidad**: Las bibliotecas de clases suelen estar disponibles en múltiples plataformas y sistemas operativos, lo que permite desarrollar software portátil que funcione en diferentes entornos.

Bibliotecas de clases

- **java.lang**: clases que se aplican al lenguaje mismo, entre las cuales se incluye la clases *Object*, *String* y *System*. También tiene clases para tipos primitivos (Integer, Character, Float, etc.)
- **java.util**: clases de utilería, como *Date* y *Random*, y clases de colección de datos como *Vector*
- **java.io**: clases para leer y escribir desde flujos (como las entradas y salidas estándares) y manejar archivos.
- **java.net**: clases para soporte de red, incluidos *Socket* y *URL* (una clase para representar referencias a documentos en WWW).
- **java.awt**: clases para implementar una interfaz gráfica del usuario, incluyendo clases para *Window, Menu, Button, Font, CheckBox*, etc. y también, las clases necesarias para el manejo de imágenes.
- **java.applet**: clases para implementar applets JAVA.

Bibliotecas de clases

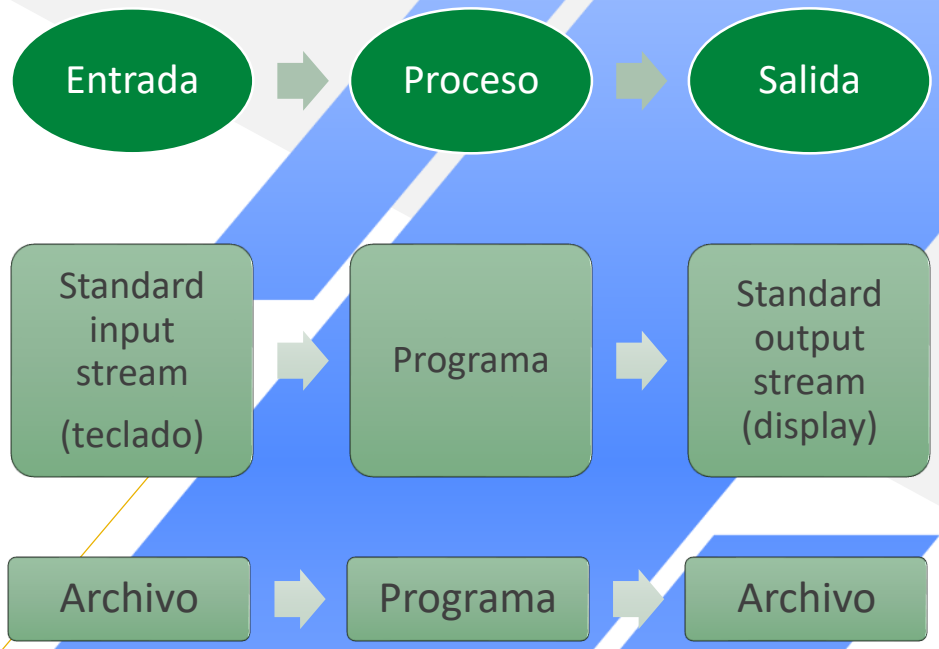
- Para utilizar en una aplicación alguna clase ya definida en JAVA, debe incluirse en la misma, el paquete de clases que la contiene.
- Esto se realiza mediante el uso de la palabra reservada *import* y el nombre del paquete de interés.
- **Ejemplos**:
  - Para incluir todas las clases de la biblioteca estándar java.io:

    *import java.io.**
  - Para incluir solo una clase específica de la biblioteca estándar *java.io*:

    import java.io.FileInputStream

Datos de entrada y salida

- *Stream*: conjunto de facilidades asociadas a los procedimientos de conversión de objetos en secuencias de caracteres y viceversa.
  Un stream es un flujo secuencial de datos.
- Flujo de datos de entrada:
  - por archivos de entrada
  - por teclado: standard input stream
- Flujo de datos de salida:
  - por archivos de salida
  - por pantalla: standard output stream



Archivos de entrada y salida

- Para leer datos desde un archivo de entrada, se debe crear un *input stream* que conecte el archivo de entrada al programa, creando objetos de la clase *FileInputStream*.

  FileInputStream InputFile = new FileInputStream (“entrada.data”)

- Para grabar datos en un archivo, se debe crear un *output stream* que envíe los datos desde el programa hacia el archivo de salida, creando objetos de la clase *FileOutputStream*.

  FileOutputStream OutputFile = new FileOutputStream (“salida.data”)

Salida estándar

- El objeto *out* (default de la clase *System*) representa la salida estándar.
- Ejemplos:

```
System.out.print (“Esto es un mensaje”);
System.out.println (a);
System.out.println (“Mensaje y variable ” + a);
System.out.println (a+b);
System.out.println (“Mensaje y suma “ + (a+b));
System.out.println (“Mensaje con tab \t “+ (a+b));
```

Salida con formato

- Para dar formato a la salida de un texto, se utiliza el método *printf()*, mediante el uso de una especificación.
- En general, el primer parámetro del método *printf()* es un **texto de formato** que puede contener los elementos que sean necesarios para describir cómo se deben mostrar los parámetros que haya a continuación.
- Las especificaciones de formato empiezan con el carácter ***%***.

```
String nombre = "Bob";
int edad = 30;
System.out.printf("Nombre: %s, Edad: %d%n", nombre, edad);
```

Salida con formato

Formato	Descripción
%c	Caracter
%d	Decimal
%o	Octal
%x	Hexadecimal
%f	Coma flotante
%s	String
%e	Científico
%n	Nueva linea
%%	Carácter %

```
double num = 123.456;
int entero = 987;
```

System.out.printf (">%.3f<%n", num);	>123,456<
System.out.printf (">%.1f<%n", num);	>123,5<
System.out.printf (">%012.4f<%n", num);	>0000123,4560<
System.out.printf (">%7.3e<%n", num);	>1,235e+02<
System.out.printf (">%d<%n", entero);	>987<
System.out.printf (">%8d<%n", entero);	> 987<
System.out.printf (">%08d<%n", entero);	>00000987<
System.out.printf (">%x<%n", entero);	>3db<
System.out.printf (">%5x<%n", entero);	> 3db<
System.out.printf ("pi = >%4.2f<%n", Math.PI);	pi = >3,14<
System.out.printf ("e = >%6.4f<%n", Math.E);	e = >2,7183<

Entrada estándar

- La clase InputStream permite el ingreso de los datos desde el teclado.
- Debe crearse un objeto de dicha clase y luego, ejecutar alguno de los método de lectura.

```
import java.util.*;
Scanner stdin = new Scanner (System.in);
double n = stdin.nextDouble();
```

Algunos Métodos:

- nextDouble()
- nextInt()
- nextLine()
- next()
- hasNextInt()
- hasNextLong()
- hasNextByte()
- hasNext()

Ejemplos:

```
double n = stdin.nextDouble();
int a = stdin.nextInt();
String str = stdin.next();
```

Funciones

- Una función es un pequeño programa en sí mismo, que puede ser llamado desde el resto del programa.
- Está compuesta por código JAVA, la cual al ejecutarse desarrolla una tarea.
- Al conjunto de código que compone la función se le asigna un nombre específico, para identificar a la función.
- En JAVA no existen funciones globales, porque cada función forma parte de la declaración de la clase con la que está relacionada.
- Pueden ser definidas antes de o después de ser utilizadas sin la necesidad de definir prototipos previamente.

Funciones

Definición de una función:

```
tipoRetornado functionName (tipo param1, tipo param2,...)
{ sentencias }
```

o bien:

```
tipoAcceso [static] tipoRetornado functionName (tipo param1, tipo param2,...)
{ sentencias }
```

Ejemplos:

```
int calcular(int a, int b)              // se asume package
public int calcular(int a, int b)
public static int calcular(int a, int b)
private int calcular(int a, int b)
private static calcular(int a, int b)
```

Funciones

Tipo de acceso o visibilidad:

- *public*: accesible desde cualquier clase u objeto.
- *private*: acceso restringido a los objetos de la clase.
- *protected*: acceso restringido a los objetos de la clase y sus derivados.
- *package*: acceso restringido a los objetos de la clase que conforman el paquete (default).

Funciones

Parámetro o argumento:

- Es un valor específico que se utiliza en las tareas de la función.
- Debe estar indicado a continuación del nombre de la misma entre paréntesis.
- Si hay más de uno, deben estar separados por coma.
- Al definir una función se debe especificar el tipo de dato en cada uno de los parámetros utilizados (no existe la propagación con el uso de la coma).
- Los argumentos y variables definidas en una función tienen alcance local exclusivamente.
- Por lo tanto, cuando un método llama a otro, los valores de los parámetros no están disponibles como tampoco las variables declaradas
- Son frecuentes, aunque no necesariamente todas las funciones aceptan parámetros o argumentos cuando son llamadas.

Funciones

Ejemplo erróneo:

```
public int calcular (int a, int b)
{ return multiplicar ()}

public int multiplicar ()
{ return a*b }
```

Debería haber sido:

```
public int calcular (int a, int b)
{ return multiplicar (a, b)}

public int multiplicar (int c, int d)
{ return c*d }
```

Funciones

Argumentos Formales:

- Son **los nombres de las variables** que se definen en la función. Ejemplo:

```
void sumar(int a, int b) { // "a" y "b" son argumentos formales
    int resultado = a + b;
    System.out.println("La suma es: " + resultado);
}
```

Argumentos Actuales:

- Son los **valores reales** que se pasan a una función cuando se llama y se asignan a los argumentos. Ejemplo:

```
int x = 5;
int y = 3;
sumar(x, y); // "x" e "y" son argumentos actuales
```

Funciones

- Generalmente, los argumentos formales representan una copia de los argumentos actuales, reservando una nueva porción de memoria para cada uno de ellos.
- JAVA es considerado un “By Value Language”, los argumentos son pasados por valor.
- Se dice que los argumentos actuales pasan por referencia, cuando los argumentos formales NO representan una copia de los actuales.
- Solamente los objetos son pasados al cuerpo de una función por referencia. Así, cualquier modificación en sus variables afecta al objeto original.
- No existe la posibilidad de utilizar argumentos por referencia con tipos primitivos.

Funciones

Parámetros dinámicos:

- O argumentos variables (*varargs*), se utilizan cuando se desconoce la cantidad de parámetros que requiere una función.
- Implementados mediante el uso de "…" (3 puntos) entre el tipo de dato y el nombre del parámetro.
- Pueden ser combinados con parámetros fijos, siendo el parámetro variable, el ultimo declarado.

Ejemplo:

```
int calculo (int a, int b, int ... numero){
    int suma = 0;
    for(int i = 0; i < numero.length; i++)
        { suma += numero[i]; }
return (a*b)+suma; }
```

Para invocarla:

```
calculo(4,5,67,9,2,4)          //dos primeros argumentos son obligatorios
calculo(14,23,3,4)
```

Funciones

- Al finalizar una función, ésta puede retornar algún resultado al programa principal.
- El tipo de dato retornado en una función, debe ser indicado en la definición de la función, delante del nombre de la misma.
- Para retornar datos en una función, se debe utilizar la palabra clave return y el dato de interés:

```
return dato;
```

- Si una función no retorna ningún valor, no es necesario el uso de la palabra clave *return*.
- En este caso el valor retornado, debe indicarse en la definición de la función usando la palabra clave *void*.
- Para llamar a una función, simplemente se debe invocar su nombre informando la misma cantidad y tipos de parámetros con los que ha sido definida.
- Si la función devuelve algún valor, la llamada a la función debe formar parte de una expresión.

Funciones

Sobre carga de funciones (overloading):

- Cuando existen múltiples definiciones para una función, se dice que está sobrecargada (overloaded).
- La sobrecarga de una función se produce al estar definida:
  - en distintas clases
  - con distintos tipos o cantidad de argumentos
- Java reconoce que definición utilizar en base a la cantidad y tipos de sus argumentos
- No importa el tipo que devuelva cada definición, lo que importa es que no haya ambigüedad entre los argumentos

Funciones

Ejemplos - Definición:

```
public void mostrar (int x)
    {System.out.println ("Valor entero: " + x); }
public void mostrar (float x)
    { System.out.println ("Valor float: " + x); }
public void mostrar ()
    { System.out.println ("Sin argumentos"); }
public int mostrar (int x)                    // ERROR ! ! !
    { return x; }
```

Ejemplos - Llamada:

```
mostrar ()
mostrar (25)
mostrar (12.5)                                // ERROR !!!
mostrar (12.5f)
mostrar (12.5, 25)                            // ERROR ! ! !
mostrar ("pepe")                              // ERROR ! ! !
```

La función *main()*

- Todo programa JAVA debe contener una función llamada main() donde se inicia la ejecución del mismo.
- Debe estar definida dentro de una clase, de la siguiente manera:

```
public static void main (String[] args)
```

public:
puede ser llamada desde cualquier otro método, incluyendo el intérprete de JAVA.

static:
es un método de la propia clase y no de la instancia.

void:
el tipo de dato que retorna la función. La función main no retorna nada

String[] args:
Los argumentos pasados al programa se encuentran en un array de String.

La función *main()*

- Para pasar argumentos a los programas, estos deben ser especificados en la línea de comandos al ejecutar la aplicación.

Ejemplo:

```
java MyPrograma ArgUno 2 Tres
java MyPrograma "Un solo Arg"
```

- Los argumentos informados se almacenan en un array de *string*, el cual pasa al método *main()* de la aplicación.
- *args* es el nombre del array de strings que contiene la lista de argumentos.
- A diferencia de C, JAVA no almacena en la primera posición del array (*args[0]*) el nombre del programa ejecutado, sino que guarda el primer argumento informado.
- Todos los argumentos son tratados como datos de tipo String.