Resumen Programación 1ºEVA

Tema 1: Introducción a la orientación a objetos. Tipos de datos primitivos

Programa

Conjunto de instrucciones, un programa de compone de clases. Siempre debe existir una clase que contenga el método principal

```
1   public static void main (String [] args) {}
```

Clase

Cada clase está formada por:

- ***Atributos** (variables miembro):* *No hace falta darles un valor inicial (no es necesario inicializarlas)*
- ***Métodos o funciones:*** *Conjuntos de instrucciones que realizan una tarea determinada reflejando el comportamiento de los objetos o el desarrollo del programa (pueden retornar un valor o array)*
- ***Dato o variable:*** *Dirección simbólica de memoria (el nombre de una variable se denomina identificador)*

Identificadores

Nombre que los programadores dan a las variables, funciones, clases,... Se deben seguir las siguientes reglas para establecer un nombre correcto:

1. *El primer carácter tiene que ser una letra o carácter especial (_, **$** o **ç**)*
2. *Puede estar acentuado y llevar ñ*
3. *No permite espacios en blanco*
4. *Pueden ser escritos en mayúsculas y minúsculas*
5. *Largo que se desee*
6. *Se aconseja poner en caso de varias palabras la primera letra en mayúscula*
7. *No se admiten palabras reservadas*

Variables

Espacio de memoria identificado por un nombre, sirven para guardar un dato que puede variar en la ejecución del programa o guardar una dirección de memoria

A. Su tipo indica:

1. *Los valores que se pueden guardar dentro de ese espacio*
2. *Las operaciones que se pueden realizar sobre esos valores*
3. *La cantidad de memoria reservada para guardar el dato* ***B. Según donde esté declarada una variable:***

1. *Variable miembro de una clase: Son atributos, se declaran en una clase pero fuera de los métodos, suelen declararse como private (para limitar el uso dentro*

de la clase y son compartidas por métodos de esa clase), no hace falta asignarle un valor previo

2. *Variables locales: Se declaran dentro de un método, donde se podrán utilizar y "es necesario darles un valor previo"*

C. Tipos de datos: *Indican al compilador:*

1. *Cuanta memoria le debe asignar*
2. *El tipo de datos que se puede almacenar*
3. *Operaciones que se pueden realizar*

Variables de tipo primitivo

Tipo		Tamaño (bytes)	Rango
Numéricos enteros	byte	1	-128 a 127
	short	2	-32768 a 32767
	int	4	-2_{31} a 2_{31-1}
	long	8	-2_{63} a 2_{63-1}
Numéricos decimales	float (7 decimales)	4	$-3{,}4*10^{38}$ a $3{,}4*10^{38}$
	double (entre 16 y 17 decimales)	8	$-1{,}7*10^{308}$ a $1{,}7*10^{308}$
Carácter (no pueden intervenir en operaciones aritméticas)	char	2	Conjunto de caracteres
Tipo lógico (boleano)	boolean	1	True/false

Variable de referencia

No guardan un valor real sino que almacenan una dirección de memoria (indica donde están guardados los datos)

- **Objeto:** *Variable cuyo tipo no es primitivo, sino que es una clase creada por el programador o definida en una API de java, el contenido puede estar formado por un conjunto heterogéneo de datos*
- **Array:** *Variable cuyo tipo no es primitivo, sino que es una referencia a un conjunto de datos homogéneos*

Resumen Programación 1ºEVA

Literales

Valor que se le asigna a una variable dentro del propio código

Constantes

*Espacios de memoria identificados por un nombre, cuyo valor permanece **invariable** a lo largo de la ejecución del programa. Si se tratase de **cambiar su valor inicializándolo de nuevo se produce un error***

*Se utiliza la palabra **final** en el momento de la declaración de la variable para convertirla en constante*

Tipos de expresiones y operadores

Expresiones

- ***Numéricas:*** *En ellas se combinan operadores y operandos, obteniendo resultados numéricos* 🡺 *nota1 + nota2*
- ***Alfanuméricas:*** *Producen resultados de tipo String, se construyen con el operador +* 🡺 *"Juan" + "Carlos"*
- ***Booleanas:*** *Obtienen resultados cierto o falso, se construyen mediante los operadores relacionales o lógicos. Su valor por defecto es verdadero o true*

  🡺 *eleccion >= 0 || eleccion <= 5*

Operadores

Se usan para construir las expresiones

- ***Operadores aritméticos:*** *Son operadores binarios, es decir requieren dos operandos (+, -, *, /, % (resto de una división), para las potencias y raíces usamos la clase math)*
- ***Operadores de asignación:*** *Permiten asignar un valor a una variable mediante =*
  - ✓ *dondeGuardamos = loQueQueremosGuardar;*
- ***Operadores incrementales:*** *Son unarios porque actúan sobre un único operando (i++, i-- o bien ++i, --i si queremos que primero se realice la operación y luego incrementemos)*

Operadores relacionales

Su resultado es verdadero o falso, se usan para realizar comparaciones (<, <=, >, >=, == o !=). Los operandos pueden ser de tipo entero o real

Operadores lógicos

A veces es necesario combinar dos o más expresiones de relación, disponemos de los siguientes:

- ***&& (and):*** *Si **ambos** operandos son iguales nos devuelve verdadero en caso contrario retorna falso*
- ***|| (or):*** *Si **uno** de los operadores es distinto nos devuelve verdadero en caso contrario retorna falso*
- ***!= (not):*** *Si el operando es verdadero lo convierte en falso, y viceversa*

Resumen Programación 1ºEVA

Operadores concatenadores de cadenas de caracteres

El operador usado para concatenar cadenas de caracteres es + ("Juan" + "Carlos")

Prioridad de los operadores

1. ***()*** *=> paréntesis*
2. ***i++, i--*** *=> incrementos y decrementos*
3. ***new, casting*** *=> instanciar un objeto o forzar una conversión*
4. ****, /, %*** *=> de izquierda a derecha*
5. ***+, -*** *=> de izquierda a derecha 6.* ***<, >, <=,...*** *=> operadores relacionales*
7. ***instanceof*** *=> ???*
8. ***&&*** *=> and*
9. ***||*** *=> or*
10. ***i -= a, i += a,...*** *=> asignaciones*

Conversiones implícitas y explicitas

Es posible transformar el tipo de una variable u objeto en otro diferente al original con el que fue declarado, no todas las conversiones son posibles, ejemplo:

```
3   int i;
4   float j;
5
6   i = (float) j;
```

- ***Conversiones explícitas:*** *En el caso que la variable origen sea de tamaño mayor, que la variable receptora, la conversión automática es imposible, produciéndose un error, que el compilador avisa como error de* conversión necesita un "cast"
- ***Conversiones implícitas:*** *Se realizan de forma automática. Condiciones:*
  - ✓ *Los tipos de las variables son compatibles (numéricos o alfanuméricos)*
  - ✓ *Siempre que la variable receptora sea de mayor o igual tamaño que la variable origen*

Comentarios

*Son ayuda para quienes tengan que leer dicho código, no se ejecutan ni compilan, es únicamente para aclarar las sentencias, existen comentarios de línea (**//**) o de varias líneas (**/* ... */**)*

Tema 2: Clases, atributos, métodos y visibilidad. Tipos de datos referenciados. Clase String

Definición de clase

Es un tipo de dato definido por el programador ó por el API de java, donde se agrupan las propiedades que nos interesan sobre algo, así como las operaciones que se pueden realizar sobre dichos datos. Está formada por atributos que son los datos y por métodos ó funciones que definen el comportamiento de la clase, permiten cambiar los datos de los atributos de una clase

Variables miembro o atributos

Son los datos que contiene una clase, puede tener cualquier número o bien ninguno, se declaran con un nombre o atributo y el tipo que le corresponda

Funciones miembro

Definen el comportamiento de una clase, representan las operaciones que se pueden realizar, permiten cambiar los datos de los atributos que forman la clase

Modificadores de visibilidad de los atributos y métodos

Cualquier método de una clase puede acceder a cualquier miembro de dicha clase, pero para poder acceder a métodos ó atributos de otras clases, dependerá de los modificadores de visibilidad que tengan, hay cuatro niveles:

- ***Público (public):*** *Si los miembros de una clase tienen este modificador, significa que cualquier clase puede acceder a ellos*
- ***Paquete (package, no poner nada):*** *Podrán ser utilizados por los métodos de otras clases, que estén en el mismo paquete*
- ***Protegido (protected):*** *Podrán acceder a dichos miembros cualquier clase que esté en el mismo paquete así como cualquier clase que herede dicha clase*
- ***Privado (private):*** *Solo podrán acceder a ellos los métodos de la propia clase*

Modificadores de visibilidad de las clases

- ***Pública:*** *Se podrá usar fuera del paquete al que pertenece*
- ***Package (sin modificador):*** *Significa que solo es visible en el mismo paquete*

Clase final

Tanto los métodos como las clases se pueden definir como final

Cuando una clase se declara cómo final, estamos impidiendo que de ella se deriven subclases, y además todos sus métodos se convierten automáticamente en final (no varían sus datos). Con esto estamos sacrificando una de las características más importantes de la POO: la reutilización de código

Resumen Programación 1ºEVA

Objeto

Es la unidad básica de la programación orientada a objetos. Se define como una variable donde el tipo es una clase, son considerados como instancias de una clase

- ***El estado:*** *Valores concretos de un objeto en cada uno de sus atributos*
- ***El comportamiento:*** *Formado por los métodos que son los que fijan las operaciones que puede realizar un objeto*
- ***La identidad:*** *Nombre ó identificador que se le ha dado a un objeto*

*Ejemplo (**get** y **set**):*

```
23      public String getCliente() {
24          return cliente;
25      }
26
27      public void setCliente(String cliente) {
28          this.cliente = cliente;
29      }
```

Mensajes

Llamada o invocación a los métodos de una clase, los mensajes tienen que estar creados dentro de una clase y constan de una cabecera llamada prototipo o declaración que consta de:

1. *Modificador de visibilidad*
2. *Si es método de clase ó de objeto, valor de retorno*
3. *Identificador ó nombre del método y entre paréntesis los parámetros ó variables que recibe del exterior por ejemplo de otros métodos*

Ejemplos:

```
8   public static int calcular (int suma, int suma2) {
9       ... cuerpo/implementación ...
10  }
```

1. => método de clase

```
12  public void calcular (int suma, float media) {
13      ... cuerpo/implementación ...
14  }
```

2. => método de objeto

Invocación o llamada

Ejemplos:

```
17  int b;
18
19  b = calcularMedia (suma, media);
```

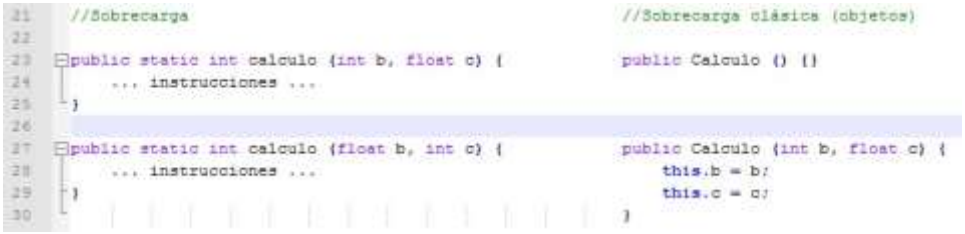
1. => si está en la misma clase

```
17  int b;
18
19  b = Calcular.calcularMedia (suma, media);
```

2. => si está en otra clase

Sobrecarga de métodos y constructores

Se produce cuando declaramos varios métodos con el mismo identificador, pero que se diferencian en los parámetros que reciben, no se considera sobrecarga si tienen el mismo nombre y distinto valor de retorno Ejemplo:



```
//Sobrecarga

public static int calculo (int b, float c) {
    ... instrucciones ...
}

public static int calculo (float b, int c) {
    ... instrucciones ...
}
```

```
//Sobrecarga clásica (objetos)

public Calculo () {}

public Calculo (int b, float c) {
    this.b = b;
    this.c = c;
}
```

Constructor/es

Métodos que se llaman igual que la clase y que sólo pueden recibir valores pero nunca retornar ni siquiera void, gracias a ellos se inicializan atributos de objeto (constructor sin parámetros), o bien construye el objeto con datos (constructor con parámetros)

Ejemplo: Ver imagen anterior en la segunda columna

Entrada por teclado

- *Scanner*

```
import java.util.Scanner;

public static void main (String [] args) {
    Scanner lee = new Scanner(System.in);

    int a = lee.nextInt();
    char letra = lee.next().charAt(0);
}
```

- *BufferedReader*

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public static void main (String [] args) throws IOException {
    BufferedReader lee = new BufferedReader(new InputStreamReader(System.in));

    String nombre = lee.readLine();
    int edad = Integer.parseInt(lee.readLine());
    float peso = Float.parseFloat(lee.readLine());
    char letra = lee.readLine().charAt(0);
}
```

Encapsulamiento y visibilidad

Un objeto es el encapsulamiento de métodos y datos, con lo cual la encapsulación consiste en agrupar atributos (variables o constantes) y métodos en una clase

La encapsulación está relacionada con la visibilidad, por eso es aconsejable que los datos sean privados y que se pueda acceder a ellos a través de los métodos de dicha clase que serán declarados como públicos

Resumen Programación 1ºEVA

Modularidad

Consiste en agrupar código en bloques para poder reutilizarlo, con la POO tenemos la ventaja que en los bloques se agrupan los datos y métodos, consiguiendo ahorrar muchos envíos de datos a través de los llamados parámetros

Abstracción

Consiste en extraer del mundo real los datos que nos interesan y su comportamiento para formar clases

Tipos de datos referenciados

Strings

También llamados literales o cadenas de caracteres, nos permite declarar, definir y operar con palabras o frases, podemos concatenarlos usando + (ejemplo: System.out.println("Frases: " + frase1 + " " + frase2);)

Si comparamos dos Strings con == solo serán iguales si apuntan a la misma estructura de datos, ejemplo

1. *String frase1 = "hola";*
2. *String frase2 = frase1;*
3. *(frase1 == frase2); => true*

Si queremos comparar por el contenido y no por la referencia podemos usar los métodos de comparación:

- ***equals:*** *Compara carácter a carácter los contenidos de las dos referencias suministradas*
- ***compareTo:*** *Compara lexicográficamente el String especificado con el que recibe el método (la parte del compareTo(String)) el resultado es un número entero*
  - *Si el número es menor que 0, el String que recibe como parámetro es menor que el otro String*
  - *Igual a 0 ambos Strings son iguales*
  - *Mayor que 0 el String que recibe como parámetro es mayor que el otro String*
    - *Tanto en el caso de menor que 0 y mayor que 0 lo que sabremos es el orden alfabético de los Strings*

Otros métodos que se usan con String:

- ***public int length();*** *=> Devuelve la longitud del String*
- ***public String toLowerCase();*** *=> Devuelve un String basado en el objeto base convertido en minúsculas*
- ***public String toUpperCase();*** *=> Devuelve un String basado en el objeto base convertido en mayúsculas*

- ***public String substring(intComienzo, int Final);*** *=> Devuelve el substring formado por los caracteres situados entre las posiciones Comienzo y Final-1*

  (posiciones numeradas a partir de 0)

Arrays

Conjunto de datos del mismo tipo, que ocupan posiciones de memoria consecutivas, ejemplo:

```
2   int [] nota= new int [4];
3   int [] nota = {5, 2, 10, 3};
4   String [] nombre = new String [4];
5   String [] nombre = {"Juan", "Carlos", "Maria", "Lucia"};
```

Este sería un conjunto de cuatro posiciones de memoria de tipo entero, la dirección de comienzo es a ó a[0],que apunta al espacio de memoria creado con new, las cuatro posiciones ,además de ser consecutivas se diferencian por el índice: nota[0]...nota[3], al empezar en la cero acaba en la posición tres

Objetos

Conjuntos de datos heterogéneos, es decir de distinto tipo, cuyo tipo es una clase determinada. Son tipos de datos referenciados y existen también arrays de objetos

Tema 3 y 4: Estructuras de control: Alternativas y Repetitivas

Estructuras alternativas

Instrucción condicional IF

Las instrucciones condicionales nos permiten ejecutar distintas instrucciones en base a la evaluación de condiciones, esto implica la toma de decisiones durante la ejecución del programa, podemos anidar varios ifs

Sintaxis:

```
if (condición) {
    instrucciones;
}
else {
    instrucciones;
}
```

Anidamiento de ifs:

```
if (condición) {
    instrucciones;
}
else {
    if (condición) {
        instrucciones;
    }
    else {
        instrucciones;
    }
}
```

Instrucción condicional SWITCH

La instrucción condicional switch permite definir un número ilimitado de ramas basadas en una misma condición (solo permite trabajar con char, byte, short e int)

Sintaxis:

```
switch (variable) {
    case 0:
        instrucciones;
        break;
    ...
    case n:
        instrucciones;
        break;
    default:
        instrucciones;
        break;
}
```

Si varias opciones del switch tienen que realizar las mismas instrucciones se seguirá la siguiente sintaxis:

```
switch (variable) {
    case 0:
        instrucciones;
        break;
    case 1:
    case 2:
        instrucciones;
        break;
    ...
    case n:
        instrucciones;
        break;
    default:
        instrucciones;
        break;
}
```

También es posible anidar las sentencia switch:

```
switch (variable) {
    switch (variable) {
        case 0:
            instrucciones;
            break;
        ...
        case n:
            instrucciones;
            break;
        default:
            instrucciones;
            break;
    }
    case 0:
        instrucciones;
        break;
    ...
    case n:
        instrucciones;
        break;
    default:
        instrucciones;
        break;
}
```

Estructuras repetitivas

Instrucción FOR

Cuando deseamos ejecutar un grupo de instrucciones un número determinado de veces, la instrucción for es la que mejor se adapta a esta tarea. La sintaxis de esta instrucción es:

```
for (inicialización; condicionContinuidad; incrementoDecremento) {
    ... instrucciones ...
}
```

Bucles WHILE y DO WHILE

El bucle while nos permite repetir la ejecución de una serie de instrucciones mientras que se cumpla una condición de continuidad. Su uso resulta recomendable cuando no conocemos a priori el número de iteraciones que debemos realizar Sintaxis:

```
6   while (condicionContinuidad) {
7       ... instrucciones ...
8   }
```

Otra forma de escribir la sintaxis de while es:

```
10  while (condicionContinuidad) {
11      ... instrucciones ...
12      incremento
13  }
```

Sintaxis do while:

```
15  do {
16      ... instrucciones ...
17  }
18  while (condicionContinuidad);
```

Tema 5: Arrays y Matrices

Arrays unidimensionales (vectores)

En muchas ocasiones es necesario hacer uso de un conjunto ordenado de elementos del mismo tipo. Cuando el número de elementos es grande, resulta muy pesado tener que declarar, definir y utilizar una variable por cada elemento

Para solucionar este tipo de situaciones, los lenguajes de programación poseen estructuras de datos que permiten declarar/utilizar con un solo nombre un conjunto de variables ordenadas de un mismo tipo, estas estructuras de datos son los arrays y matrices

Sintaxis:

1. *tipo [] nombre= new tipo [numeroElementos];*
2. *tipo [] nombre = {valor0, valor1,..., valorN};*
   A. *nombre [0] = valor0;*
   B. *nombre [1] = valor1; Esto se denomina acceso a datos del array*
   C. *...*
   D. *nombre [n] = valorN;*

Arrays bidireccionales (matrices)

Siguen una sintaxis parecida a la de los arrays unidimensionales Sintaxis:

1. *tipo [][] nombre= new tipo [numeroFilas][numeroColumnas];*
2. *tipo [][] nombre = {valor0.0, valor0.1,..., valor0.N}...{valorN.0, valorN.1,... valorN.N};*
   A. *nombre [0][0] = valor0.1;*
   B. *nombre [0][1] = valor0.2;*
   C. *...*
   D. *nombre [0][n] = valor0.N; Esto se denomina acceso a datos de la matriz*
   E. *...*
   F. *nombre [n][0] = valorN.1;*
   G. *nombre [n][1] = valorN.2;*

H. ...

I. *nombre [n][n] = valorN.N;*

Métodos para ordenar Arrays y Matrices

Método de la Burbuja

```java
//Burbuja
public static void burbuja (String nombres, float notas) {

    int i;
    int j;
    float aux;
    String aux2;

    for (i = 0; i < notas.length - 1; i++) {
        for (j = (i + 1); j < notas.length; j++) {
            if (notas [i] < notas [j]) { //Ordena de forma ascendente (de 1 a 9) en forma descendente seria de 9 a 1
                aux = notas [i];
                notas [i] = notas [j];
                notas [j] = aux;

                aux2 = nombres [i];
                nombres [i] = nombres [j];
                nombres [j] = aux2;
            }
        }
    }
}
```

Método del Intercambio

```java
//Intercambio
public static void intercambio (String nombres, float notas) {

    int i;
    int b = 0;
    float aux;
    String aux2;

    while (b == 0) {
        b = 1;

        for (i = 0; i < nombres.length - 1; i++) {
            if (nombres [i].compareToIgnoreCase(nombres [i + 1]) > 0) { //Ordena de forma ascendente (de A a Z) en forma descendente seria de Z a A
                aux = notas [i];
                notas [i] = notas [i + 1];
                notas [i + 1] = aux;

                aux2 = nombres [i];
                nombres [i] = nombres [i + 1];
                nombres[i + 1] = aux2;

                b = 0;
            }
        }
    }
```

Métodos para buscar elementos en Arrays y Matrices

Búsqueda Secuencial

```java
//Secuencial
import java.util.Scanner;

public static void busquedaSecuencial (String nombres, float notas, String nombresBusq, float notasBusq) {

    int c = 0;
    int b = 0;
    int i;
    String nombreBusq;

    Scanner lee = new Scanner(System.in);

    System.out.println("\n Introduce el nombre a buscar: \n");
    nombre = lee.next();

    for (i = 0; i < nombres.length; i++) {
        if (nombreBusq.compareToIgnoreCase(nombres [i]) == 0) {
            nombresBusq [c] = nombres [i];
            notasBusq [c] = notas [i];
            b = 1;
            c++;
        }
    }

    if (b == 1) {
        System.out.print("\n");
        for(i = 0; i < c; i++) {
            System.out.println("   " + nombresBusq [i] + " " + notasBusq [i] + "\n");
        }
    }
    else {
        System.out.println("\n El alumno " + nombreBusq + " no ha sido encontrado \n");
    }
```

Búsqueda Dicotómica

Es el método más rápido, consiste en dividir el array en mitades hasta que encuentre el elemento deseado. Para poder realizar este tipo de búsqueda el array por el que se realiza dicha búsqueda tiene que estar ordenado, en caso contrario no funciona

```java
//Dicotomica
import java.util.Scanner;

public static int busquedaDicotomica (String nombres, float notas) {

    int b;
    int i = 0;
    int f =  nombres.length - 1;
    int m;
    String nombreBusq;

    Scanner lee = new Scanner(System.in);

    System.out.println("\n Introduce el nombre del alumno que deseas buscar: \n");
    nombreBusq = lee.next();

    while (i < f) {
        m = (i + f) / 2;
        if (nombreBusq.compareToIgnoreCase(nombres [m]) == 0) {
            i = m;
            f = m;
        }
        else {
            if (nombreBusq.compareToIgnoreCase(nombres [m]) > 0) { //Esto es así porque ordenamos de forma descendente (en la ordenacion pusimos <)
                f = m - 1;
            }
            else {
                i = m + 1;
            }
        }
    }
public static void visualiza (String nombres, float notas, int b) {

    if (b != - 1) {
        System.out.println("\n  El alumno ha sido encontrado \n" +
                           "\n  Nombre: " + nombres [b] +
                           "\n  Nota: " + notas [b] + "\n");
    }
    else {
        System.out.println("\n  El alumno no ha sido encontrado \n");
    }
}
```