



LENGUAJE DE PROGRAMACIÓN: JAVA

Autores:

Colectivo de Entrenadores ACM-ICPC UM

Enero / 2019

ENTRENAMIENTO PARA CONCURSANTES
ACM-ICPC DE LA UNIVERSIDAD DE
MATANZAS

Objetivos

- ▶ Familiarizar a los estudiantes con los lenguajes de programación más utilizados en los concursos ACM-ICPC.

Contenidos

Objetivos

Contenidos

Bibliografía

Lenguaje de programación

Lenguaje de programación: Java

Estructura de solución

Captura y salida de datos

Conclusiones

Bibliografía

- ▶ *Manual de preparación para concursantes ACM-ICPC de la Universidad de Matanzas.* Publicado en el Entorno Virtual de Aprendizaje de la Universidad de Matanzas (eva.umcc.cu). Sección *Bibliografía*.

Lenguajes de programación

- ▶ Un lenguaje de programación es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras.
- ▶ Es un conjunto de reglas, notaciones, símbolos y/o caracteres que permiten a un programador poder expresar el procesamiento de datos y sus estructuras en la computadora.

Lenguajes de programación: Java

- ▶ El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems.
- ▶ Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos.
- ▶ Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

Lenguajes de programación: Java

- ▶ Debido a su mecanismo de compilación la soluciones realizadas Java tiene un consumo de tiempo y memoria mayor que las hechas en C++ con similar algoritmo.
- ▶ Para desarrollar Java podemos utilizar *IDE* como *NetBeans* y *Eclipse*.

Estructura de una solución

- ▶ La primera sección está destinada para la incluir todas bibliotecas propias del lenguaje que serán utilizadas en la solución del problema.

```
import java.io.BufferedReader;  
import java.math.*;  
import java.util.Scanner;
```


Estructura de una solución

- La segunda sección va estar destinada a la solución. Diferencia de C++ donde contamos con un método main, en Java contaremos con una clase que dentro de sus métodos esta el main el cual es esttico o static.

```
class Main {  
    public static void main(String[] args) {  
        Scanner in = new Scanner(System.in);  
        int a = in.nextInt();  
        int b = in.nextInt();  
        System.out.println(a + b);  
    }  
}
```

Estructura de una solución

```
import java.io.BufferedReader;
import java.math.*;
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int a = in.nextInt();
        int b = in.nextInt();
        System.out.println(a + b);
    }
}
```

Captura y salida de datos

- ▶ Para la captura de datos en Java lo podemos hacer con *System.in* que ofrece un grupo de funcionalidades un poco primitivas para capturar datos. Mas adelante veremos opciones que facilitan el trabajo
- ▶ Para la impresión de datos en Java lo podemos hacer con *System.out* que ofrece un grupo de funcionalidades que toman de mucho de la impresión de datos del language C.

Captura de datos

- ▶ La primera variante para capturar datos en Java es utilizar la clase *Scanner*. Para utilizar basta con incluirla en su solución.

```
import java.util.Scanner;
```

- ▶ La clase *Scanner* funciona como un simple scanner de texto el cual puede parsear cadenas y tipos de datos primitivos del lenguaje usando expresiones regulares.
- ▶ La clase *Scanner* captura la entrada y la fracciona en tokens tomando como delimitadores ciertos patrones de texto (por defecto es el espacio en blanco).

Captura de datos

- ▶ Para ser uso de ella solo debemos crear una instancia u objeto de la clase como se muestra a continuación:

```
Scanner in = new Scanner(System.in);
```

- ▶ Para acceder a los tokens solo basta invocar los métodos *next* de la clase. Tener en cuenta que para cada tipo de dato la clase *Scanner* tiene un método *next*.

```
int a = in.nextInt();  
int b = in.nextInt();
```

- ▶ En caso que se necesite cambiar el patrón delimitador de los tokens la clase *Scanner* da esa posibilidad.

Captura de datos

- ▶ La segunda variante para capturar datos en Java es utilizar la clase *BufferedReader*. Para utilizar basta con incluirla en su solución.

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

- ▶ La clase *BufferedReader* es muy eficiente en la captura de los datos(puede ser la diferencia entre un Aceptado y un Tiempo Limite Excedido)

```
BufferedReader b=new BufferedReader(new  
    InputStreamReader(System.in));
```

Captura de datos

- ▶ Da tres variantes para capturar, la primera leer carácter a carácter, la segunda es leer una porción de un arreglo de caracteres y la tercera leer toda una línea (similar a `Console.ReadLine()` de C# o `raw input()` de Python).

```
String line=b.readLine();
```

- ▶ Si combinamos dos instancias, una de *BufferedReader* y otra *StringTokenizer* obtenmos una *Scanner* más rápido que el tradicional (Ver en el manual plantilla de Java con *BufferedReader*).

```
StringTokenizer cin=new StringTokenizer(b.  
    readLine());  
int N=Integer.parseInt(cin.nextToken());  
int C=Integer.parseInt(cin.nextToken());
```

Salida de datos

- ▶ En cuanto a la salida de igual manera se cuenta con dos variantes. La primera usar directamente la que proporciona el lenguaje por defecto *System.out* la cual posee funcionalidades para imprimir.

```
System.out.println(N+C);
```

- ▶ La segunda variante es utilizar la clase *PrintWriter* perteneciente al subpaquete *io* del paquete *java*. Una de las ventajas de esta variantes es la posibilidad de formatear la salida de acuerdo a un estándar o reglas impuestas por el problema.

Captura de datos

- ▶ El otro aspecto a tener en cuenta es el formato de entrada el cual puede presentar varias variantes para las cuales se debe estar preparado, a continuación se detallan los más comunes.
- ▶ Cuando me dan la cantidad de casos previamente.

```
int cases;  
Scanner in = new Scanner(System.in);  
cases=in.nextInt();  
while(cases>0)  
{  
    cases--;  
    /* Desarrollo de la solucion */  
}
```

Captura de datos

- ▶ Cuando me dan la cantidad de casos previamente y necesito saber el caso que estoy analizando.

```
int cases;  
Scanner in = new Scanner(System.in);  
cases=in.nextInt();  
for ( int i =1; i <= cases ; i ++)  
{  
    /* Desarrollo de la solucion */  
}
```

Captura de datos

- Se lee hasta que en la entrada se cumpla una determinada condición en cuanto a sus valores.

```
boolean exits = false ;
Scanner in = new Scanner(System.in);

int entrada ;
while(!exits)
{
    entrada=in.nextInt();
    if(entrada==0)
        exits = true ;
    else{
        /* Desarrollo de la solucion */
    }
}
```

Captura de datos

- ▶ Se lee hasta fin de fichero o no se sabe en que momento termina la entrada.

```
String entrada2 ;  
Scanner in = new Scanner(System.in);  
while((entrada2=in.next())!=null)  
{  
    /* Desarrollo de la solucion */  
}
```

Conclusiones

- ▶ Debido a su mecanismo de compilación las soluciones realizadas en Java tienen un consumo de tiempo y memoria mayor que las hechas en C++ con similar algoritmo.
- ▶ A favor de Java está su trabajo con cadenas y números grandes.

UNIVERSIDAD DE MATANZAS

cosechando el saber

FIN