Programación I
Grado en Informática
Curso 2018 / 2019

MANEJO DE VARIABLES
Tipos básicos, operadores, conversiones de tipo y cadenas de caracteres

Tabla de contenidos

1. Introducción

Una **variable** se entiende como un espacio en memoria dónde se guarda una información. El formato, tamaño y manejo de dicha variable depende del tipo asociado a la variable. El lenguaje Java proporciona variables de dos tipos:

- Tipo básico. Permite almacenar constantes y literales que representan valores específicos.
- Tipo no básico. Permiten almacenar referencias a objetos que representan entidades.

Junto a su concepción teórica, es necesario poner en práctica los conocimientos teóricos adquiridos a fin de garantizar la correcta utilización de las variables. Basándose en esta premisa, la presente práctica propone la realización de una serie de ejercicios sobre el manejo de variables.

1.1. Objetivos

El objetivo principal de la práctica es poner en uso los conocimientos sobre variables adquiridos en clase de teoría. Para ello se deberán resolver en lenguaje JAVA una serie de ejercicios sobre tipos de datos básicos, operadores, conversiones de tipo y cadenas de caracteres.

Junto al objetivo general, la práctica tiene una serie de objetivos específicos. A saber:

- Aprender a resolver errores de compilación.
- Conocer la forma adecuada de mostrar el contenido de una clase por pantalla.

2. Entorno de trabajo

Como paso previo a la realización de los ejercicios, se exponen las condiciones del entorno de trabajo en el cual se realizarán las prácticas. El objetivo es homogenizar la forma de desarrollo de las mismas.

2.1. Entorno de desarrollo

Sun Microsystems proporciona dos herramientas fundamentales en la plataforma Java SE (*Standard Edition*):

- *Java SE Runtime Environment* (JRE) .Entorno de ejecución de Java SE. El JRE proporciona las librerías, máquina virtual y otros componentes necesarios para la ejecución de aplicaciones desarrolladas en lenguaje Java. El entorno de ejecución, dado su tamaño, puede distribuirse junto con la aplicación para hacer éste independiente del sistema.
- *JavaSE Development Kit* (JDK) .Entorno de desarrollo de Java SE. El JDK incluye el entorno de ejecución más herramientas que permiten el desarrollo de aplicaciones: compilador, depurador, generador de documentación, etc.

La implementación de la aplicación se realizará mediante el lenguaje de programación Java, utilizando como entorno de desarrollo el JDK 8. Para cualquier duda sobre este entorno se recomienda consultar la documentación disponible en:

```
http://docs.oracle.com/javase/8/docs/api/
```

2.2. Herramienta de programación

Para la edición y programación del código se puede utilizar cualquier editor de texto. No obstante, se recomienda utilizar editores de texto que, al menos, coloreen la sintaxis de java y tabulen el código. Las tareas de compilación, ejecución y generación de la documentación se llevará a cabo desde la línea de comandos siguiendo las instrucciones de utilización del código de prueba que acompaña la práctica.

3. Ejercicios

La realización de la práctica implicará la resolución de una serie de ejercicios de programación. Los ejercicios se presentan agrupados por contenidos y deberán resolverse completando los métodos correspondientes de las clases aportadas.

3.1.1. Tipos de datos básicos

Ejercicio 1

Dado el siguiente fragmento de código:

```
Int entero = 6;
long otroEntero = 1.000;
long decimal = 7.0;
double otroDecimal = 7,0;
byte enteroDe8Bits = 10000;
char caracter = a;
char otroCaracter = "a";
boolean booleano = "true";
short enteroDe16Bits = 50000;

byte static = 5;
byte int = 3;
double _otra-Variable = 2.0;
```

Se pide modificar el código a fin de eliminar los errores de compilación existentes. Los errores de compilación tienen que ver con el manejo de tipos de datos básicos. Las modificaciones se realizarán sobre el método `ejercicio01()` de la clase `Apartado030101`.

Ejercicio 2

Dado el siguiente fragmento de código:

```
variable1 = 637;
variable2 = 637L;
variable3 = 6.37;
variable4 = 6.37f;
variable5 = 6.37d;
variable6 = '6';
variable7 = "6.37";
variable8 = 'a';
variable9 = "a";
variable10 = true;
```

Se pide completar el código a fin de determinar el tipo de dato más adecuado para cada literal. Las modificaciones se realizarán sobre el método `ejercicio02()` de la clase `Apartado030101`.

Ejercicio 3

Dado el siguiente fragmento de código:

```
//Numero de asignaturas de un curso
//Nota media de la asignatura
//Edad de una persona
//Salario mensual de un empleado
//Nombre de una asignatura
//Constante PI
//Constante VERDADERO
//Portal de la direccion de una vivienda
//Piso de la direccion de una vivienda
//Puerta la direccion de una vivienda;
```

Se pide definir variables que permitan representar la información referida en los comentarios. Las modificaciones se realizarán sobre el método `ejercicio03()` de la clase `Apartado030101`.

Ejercicio 4

Dado el siguiente fragmento de código:

```
double valor1 = 2.8;
double valor2 = 1.5;

double resultado = valor1 - valor2;
System.out.println(
      valor1 + " - " + valor2 + " = " + resultado);
```

Se pide:

- Compilar y ejecutar el método `ejercicio04()` de la clase `Apartado030101`.
- Analizar los resultados obtenidos.
- Explicar en el fichero LEEME.TXT el porqué de los resultados.

Ejercicio 5

Dado el siguiente fragmento de código:

```
BigDecimal valor1 = new BigDecimal("2.8");
BigDecimal valor2 = new BigDecimal("1.5");

System.out.println( valor1 + " - " + valor2 + " = " +
                    valor1.subtract(valor2) );
```

Se pide:

- Compilar y ejecutar el método `ejercicio05()` de la clase `Apartado030101`.
- Analizar los resultados obtenidos.
- Explicar en el fichero LEEME.TXT el porqué de los resultados.

3.1.2. Operadores

Ejercicio 1

Dado el siguiente fragmento de código:

```
final int CONST=128;
int op1;
int op2;
int resultado;
//Preincrementa op1 y multiplicalo por 12
//El valor de op2 es la suma op1 predecrementado con CONST
//Halla el resto de dividir op2 entre op1 y guardalo en resultado
//Muestra por pantalla los valores de op1, op2 y resultado
```

Se pide:

Realizar las operaciones aritméticas indicadas en cada comentario.

Las modificaciones se realizarán sobre el método `ejercicio01()` de la clase `Apartado030102`.

Ejercicio 2

Dado el siguiente fragmento de código:

```
int edad;
int numeroPartes;
boolean deportivo;
boolean rebaja;
// rebaja=expresión booleana
System.out.println("Rebaja= "+rebaja);
```

Se pide:

Construir una expresión booleana que permita identificar si un cliente de un seguro de automóvil tiene derecho a una rebaja a la hora de la renovación anual, sabiendo que la rebaja se da si se cumple una de las siguientes condiciones:

- Si tiene entre 40 y 60 años y menos de 3 partes.
- Si es mayor de 20, tiene como mucho un parte y su coche no es deportivo.

Inicializa las variables con valores para comprobar el correcto funcionamiento de la expresión.

Las modificaciones se realizarán sobre el método `ejercicio02()` de la clase `Apartado030102`.

Ejercicio 3

Dado el siguiente fragmento de código:

```
int segundos, horas, minutos;
int totalSegundos=56000;
// Realización de cálculos
System.out.println(horas+"h "+minutos+"m "+segundos+"s ");
```

Se pide:

Calcular cuantas horas, minutos y segundos hay en 56000 segundos.

Las modificaciones se realizarán sobre el método `ejercicio03()` de la clase `Apartado030102`.

3.1.3. Operaciones matemáticas. La clase "*Math*"

Ejercicio 1

Consultar la clase "Math" del API de Java y programar el código necesario para realizar la operación: obtener la raíz cuadrada de 256

Las modificaciones se realizarán sobre el método `ejercicio01()` de la clase `Apartado030103`.

Ejercicio 2

Consultar la clase "Math" del API de Java y programar el código necesario para realizar la operación: obtener el resultado de elevar al cubo el número 9

Las modificaciones se realizarán sobre el método `ejercicio02()` de la clase `Apartado030103`.

Ejercicio 3

Consultar la clase "Math" del API de Java y programar el código necesario para generar un número aleatorio contenido entre 5 y 10. Ejecutar varias veces el método para comprobar que la generación se realiza correctamente.

> **IMPORTANTE**
>
> Si tenemos números aleatorios decimales entre (0,1) y deseamos generar números en otro intervalo, únicamente necesitamos multiplicar por el tamaño del nuevo intervalo, e incrementar el valor del límite inferior del mismo.

Las modificaciones se realizarán sobre el método `ejercicio03()` de la clase `Apartado030103`.

Ejercicio 4

Consultar la clase "Math" del API de Java y programar el código necesario para conocer la superficie de un círculo de diez unidades de radio.

Las modificaciones se realizarán sobre el método `ejercicio04()` de la clase `Apartado030103`.

3.1.4. Conversiones de tipo

Ejercicio 1

Dado el siguiente fragmento de código:

```
byte varByte;
short varShort;
int varInt;
long varLong;
float varFloat;
double varDouble;
char varChar ;
boolean varBoolean;

varByte = 50;
varShort = 1500 ;
varInt = 1500000 ;
varLong = 65000000 ;
varFloat = 20.0E4F ;
varDouble = 0.123456789e9 ;
varChar = 'H' ;
varBoolean = true ;

varInt    = varShort;
varDouble = varFloat;
varFloat  = varLong;
varLong   = varInt;
varLong   = 9223372036854775807L;
varFloat  = varLong;
varByte   = varShort;
varShort  = varInt;
```

Se pide:

Comprobar cuales de las conversiones implícitas realizadas son correctas y comentar las incorrectas.

Las modificaciones se realizarán sobre el método `ejercicio01()` de la clase `Apartado030104`.

Ejercicio 2

Dado el siguiente fragmento de código:

```
byte varByte;
short varShort;
int varInt;
long varLong;

varLong=35000L;
```

Se pide:

Asignar varLong al resto de variables realizando las conversiones explícitas necesarias. Imprime por pantalla el resultado de dichas conversiones y explica en el fichero LEEME.TXT el significado de los valores obtenidos.

Puedes consultar esta referencia:
http://www.cs.grin.edu/~rebelsky/Espresso/Readings/binary.html

Las modificaciones se realizarán sobre el método `ejercicio02()` de la clase `Apartado030104`.

Ejercicio 3

Dado el siguiente fragmento de código:

```
byte varByte;
short varShort;
int varInt;
long varLong;
float varFloat;
double varDouble;
varFloat= 123.1f;
```

Se pide:

Asignar varFloat al resto de variables realizando las conversiones necesarias. Imprime por pantalla el resultado de dichas conversiones.

Las modificaciones se realizarán sobre el método `ejercicio03()` de la clase `Apartado030104`.

Ejercicio 4

Dado el siguiente fragmento de código:

```
double dGigante, dNormal, dMinimo;
float  fGigante, fNormal, fMinimo;

dGigante = 1.766e289;
dNormal  = 35.987654321;
dMinimo  = 0.2E-256;

fGigante = (float)dGigante;
fNormal  = (float)dNormal;
fMinimo  = (float)dMinimo;

System.out.println("Gigante: " + fGigante);
System.out.println("Normal : " + fNormal);
System.out.println("Minimo : " + fMinimo);

byte b = (byte)130;
short s = (short)32770;
int i = (int)21474836501;

System.out.println("Byte  : " + b);
System.out.println("Short : " + s);
System.out.println("Int   : " + i);

float f = 1.3e22;
System.out.println("f: " + f);
```

Se pide:

Arreglar los posibles errores de compilación y explicar en el fichero LEEME.TXT los resultados.

Las modificaciones se realizarán sobre el método `ejercicio04()` de la clase `Apartado030104`.

3.1.5. Cadenas de caracteres. La clase "*String*"

Para realizar los siguientes ejercicios es conveniente consultar la clase "String" del API de Java.

Ejercicio 1

Dado el siguiente fragmento de código:

```
String cadena="En un lugar de la Mancha";
```

Se pide añadir el código necesario para realizar las siguientes tareas:

- Obtener el número de caracteres de la cadena.
- Calcular la posición intermedia de la cadena.
- Extraer el carácter que ocupa dicha posición.
- Mostrar por pantalla dicho carácter y el código que lo representa.

Las modificaciones se realizarán sobre el método `ejercicio01()` de la clase `Apartado030201`.

Ejercicio 2

Dado el siguiente fragmento de código:

```
String cadena="Viaje al Parnaso";
String otraCadena="ViAje al pArnaso";
```

Se pide añadir el código necesario para realizar las siguientes tareas:

- Comparar las dos cadenas para ver si son iguales y mostrar por pantalla el resultado de la comparación.
- Volver a compararlas pero ahora sin tener en cuenta si están en mayúsculas o minúsculas y mostrar por pantalla el resultado de la comparación.
- Convertir las dos cadenas a minúsculas, volver a compararlas y mostrar por pantalla el resultado de la comparación.

Las modificaciones se realizarán sobre el método `ejercicio02()` de la clase `Apartado030201`.

Ejercicio 3

Dado el siguiente fragmento de código:

```
String cadena="Viaje al Parnaso";
String otraCadena="Persiles y Segismunda";
```

Se pide añadir el código necesario para realizar las siguientes tareas:

- Concatenar las dos cadenas formando una tercera usando el operador +.
- Concatenar las dos cadenas formando una tercera usando el método concat.
- Mostrar los resultados por pantalla.

Las modificaciones se realizarán sobre el método `ejercicio03()` de la clase `Apartado030201`.

Ejercicio 4

Dado el siguiente fragmento de código:

```
String cadena="Viaje al Parnaso";
```

Se pide añadir el código necesario para realizar las siguientes tareas:

- Comprobar si la cadena termina con la palabra Parnaso utilizando endsWith.
- Comprobar si la cadena empieza con la palabra Viaje utilizando startsWith.
- Mostrar los resultados por pantalla.

Las modificaciones se realizarán sobre el método `ejercicio04()` de la clase `Apartado030201`.

Ejercicio 5

Dado el siguiente fragmento de código:

```
String cadena="Viaje al Parnaso";
```

Se pide añadir el código necesario para realizar las siguientes búsquedas en cadena utilizando indexOf:

- Buscar si el carácter p está en la cadena y mostrar el resultado por pantalla.
- Buscar si la cadena Par está en la cadena y mostrar el resultado por pantalla.
- Buscar la última ocurrencia de la letra a en la cadena y mostrar el resultado por pantalla.
- Buscar la letra a empezando por la posición 3 y mostrar el resultado por pantalla.

Las modificaciones se realizarán sobre el método `ejercicio05()` de la clase `Apartado030201`.

Ejercicio 6

Dado el siguiente fragmento de código:

```
String cadena="Viaje al Parnaso";
```

Se pide añadir el código necesario para realizar las siguientes tareas:

- Reemplazar las ocurrencias de la letra a por * y mostrar el resultado por pantalla.
- Reemplazar las ocurrencias de la palabra Parnaso por Olimpo y mostrar en resultado por pantalla.

Las modificaciones se realizarán sobre el método `ejercicio06()` de la clase `Apartado030201`.

Ejercicio 7

Dado el siguiente fragmento de código:

```
String cadena="Viaje al Parnaso";
```

Se pide añadir el código necesario para realizar las siguientes tareas:

- Obtener la subcadena que va desde la mitad al final.
- Obtener la subcadena que empieza en la primera j y termina antes de la primera s.

Las modificaciones se realizarán sobre el método `ejercicio07()` de la clase `Apartado030201`.

Ejercicio 8

Dado el siguiente fragmento de código:

```
String cadena=" La Galatea   ";
```

Se pide añadir el código necesario quitar los espacios sobrantes al principio y al final.

Las modificaciones se realizarán sobre el método `ejercicio08()` de la clase `Apartado030201`.

Ejercicio 9

Dado el siguiente fragmento de código:

```
double numero=1.12e12;
boolean expresion=true;
long enteroGrande=1231231L;
```

Se pide añadir el código necesario convertir las variables a String utilizando el método estático String.valueOf. Mostrar el resultado por pantalla.

Las modificaciones se realizarán sobre el método `ejercicio09()` de la clase `Apartado030201`.

Ejercicio 10

Dado el siguiente fragmento de código:

```
String cadena=“Viaje al Parnaso”;
String otraCadena=”Viaje al Olimpo”;
```

- Se pide comparar las dos cadenas lexicograficamente y mostrar el resultado por pantalla. Explica el resultado de esta comparación en el fichero LEEME.TXT

Las modificaciones se realizarán sobre el método `ejercicio10()` de la clase `Apartado030201`.

3.1.6. Cadenas de caracteres. La clase “*StringBuffer*”

Para realizar los siguientes ejercicios es conveniente consultar la clase “StringBuffer” del API de Java.

Ejercicio 1

Dado el siguiente fragmento de código:

```
StringBuffer cadena=new StringBuffer(“Viaje al Parnaso”);
System.out.println(cadena.length());
System.out.println(cadena.capacity());
boolean logica=true;
String otraCadena=”Cervantes”;
int año=1616;
```

Se pide:

- Anexar a la cadena cada una de las variables definidas en el código, mostrando por pantalla el resultado de cada operación.
- Explicar en el fichero LEEME.txt la diferencia entre lenght() y capacity().

Las modificaciones se realizarán sobre el método `ejercicio01()` de la clase `Apartado030202`.

Ejercicio 2

Dado el siguiente fragmento de código:

```
StringBuffer cadena=new StringBuffer(“Viaje al Parnaso”);
// Modificaciones

System.out.println(cadena.toString());
```

Se pide:

- Insertar en la posición 9 del StringBuffer un String con valor “`mitico`”.
- Establecer en la posición 9 del StringBuffer una letra M.

Las modificaciones se realizarán sobre el método `ejercicio02()` de la clase `Apartado030202`.

Ejercicio 3

Dado el siguiente fragmento de código:

```
StringBuffer cadena=new StringBuffer("Viaje al Parnaso");
// Modificaciones

System.out.println(cadena.toString());
```

Se pide:

- Reemplaza el texto "Parnaso" por "Castalia".
- Borra la letra l.

Las modificaciones se realizarán sobre el método `ejercicio03()` de la clase `Apartado030202`.

Ejercicio 4

Dado el siguiente fragmento de código:

```
StringBuffer cadena=new StringBuffer("Viaje al Parnaso");
// Modificaciones

System.out.println(cadena.toString());
```

Se pide:

- Invertir el texto almacenado en cadena.

Las modificaciones se realizarán sobre el método `ejercicio04()` de la clase `Apartado030202`.

Ejercicio 5

Dado el siguiente fragmento de código:

```
StringBuffer cadena=new StringBuffer("Viaje al Parnaso");
String otraCadena=new String("Viaje desde Arcadia");
```

- Se pide realizar las tareas necesarias para que cadena almacene el valor "Viaje desde Arcadia al Parnaso".
- Mostrar el resultado por pantalla.

Las modificaciones se realizarán sobre el método `ejercicio05()` de la clase `Apartado030202`.

3.1.7. Cadenas de caracteres. Método "to*String()*"

IMPORTANTE

El método `toString()` es un método ejecutado directamente por el interprete de Java cuando se realiza un `System.out.println()`

Todas las clases de Java tienen dicho método; sin embargo, exceptuando en las clases predefinidas del API, este método se implementa del siguiente modo: muestra el nombre de la clase y la posición de memoria ocupada por el objeto a imprimir. Si se desea obtener otro resultado es necesario rescribir el método.

Ejercicio 1

Dado el siguiente fragmento de código:

```
Alumno alumno = new Alumno("1000011111","Pepe");
alumno.asignarNota(6.0f);

System.out.println(5);
System.out.println(10.5);
System.out.println("Cadena de caracteres");
System.out.println(alumno);
```

Se pide:

- Compilar y ejecutar el método `ejercicio01()` de la clase `Apartado030203`.
- Analizar los resultados obtenidos.
- Eliminar los comentarios del método `toString()` del fichero `Alumno.java`
- Compilar la clase `Alumno`.
- Ejecutar el método `ejercicio01()` de la clase `Apartado030203`.
- Analizar los resultados obtenidos.
- Explicar en el fichero LEEME.TXT el porqué de los resultados.

Ejercicio 2

Dado el siguiente fragmento de código:

```
Profesor profesor = new Profesor("4000011111","Juan");
System.out.println(profesor);
```

Se pide modificar el fichero `Profesor.java` a fin de mostrar por pantalla la información que define al profesor.

4. Material proporcionado

Para la realización de la práctica se proporciona, junto a este enunciado, un enlace a un repositorio de github. un fichero comprimido con el material necesario para su correcta realización. Una vez descomprimido el fichero, se creará un directorio `/p-var` con la siguiente estructura de trabajo:

- **src.** Ficheros fuente de la aplicación. Contiene el programa principal de prueba. Ver punto 4.2 para más información.
- **classes.** Ficheros compilados. Contiene los *bytecode* de los ficheros fuente.
- **etc.** Ficheros de configuración. Contiene el esbozo del fichero LEEME.TXT a completar.

4.1. Definición de ejercicios

Dentro del directorio `/src`, se encuentra la definición de los métodos que deberán completarse de acuerdo a los enunciados previamente expuestos. La definición de los métodos recoge la signatura completa, incluyendo comentarios Javadoc que reproducen el contenido del enunciado correspondiente. Al realizar la implementación de los métodos se deberá respetar la información contenida en estos comentarios.

4.2. Casos de prueba

Para probar el resultado de la práctica, se proporciona una clase de prueba. Esta clase ejecuta los distintos ejercicios en función de los parámetros recibidos por la línea de comandos. Se proporciona un build.xml que solicitará, una vez que la compilación tenga éxito, la entrada por consola de dos parámetros que corresponden con:

[*apartado*] Apartado de la práctica al cual pertenece el ejercicio (solo el número, ej: 030101).

[*ejercicio*] Número de ejercicio (solo el número, ej. 01).

4.3. Resolución de errores

Al ejecutar la clase de prueba pueden producirse excepciones originadas por el código del alumno. Estos errores serán emitidos por la clase de prueba de la siguiente manera:

```
Resultado de la ejecución

java.lang.reflect.InvocationTargetException
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native
Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown
Source)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
        at java.lang.reflect.Method.invoke(Unknown Source)

        at PruebaPractica.main(PruebaPractica.java:36)
Caused by: java.lang.NullPointerException
        at Apartado030201.ejercicio02(Apartado030201.java:75)
        ... 5 more
```

En este caso, a fin de saber la causa del error, el alumno debe fijarse en la parte final del mensaje emitido por pantalla: a partir de la línea que indique **`PruebaPractica.main.`** Así, en el ejemplo mostrado, el error se origina en la línea 75 de la clase `Apartado030201`. Se trata de un error producido al ejecutar un objeto declarado pero no creado; es decir, que apunta a "null".