Variables y Operadores

Programación de Computadoras II

Abdel G. Martínez L.

Agenda

1. Variables
2. Tipos de Datos Primitivos
3. Literales
4. Asignaciones
5. Diagramas de Estado
6. Impresión de Variables
7. Operadores
8. Operadores Aritméticos
9. Operadores Unarios
10. Números Punto-Flotante
11. Errores de Redondeo
12. Operadores para Cadenas
13. Orden de Operaciones
14. Composición
15. Tipos de Errores

Variables

- Una **variable** es una ubicación en memoria que guarda un valor.
- Este valor puede ser un número, texto, imágenes, sonidos y otros tipos de datos.
- Para almacenar un valor se debe primero **declarar** la variable:

```
String mensaje;
```

- El ejemplo anterior indica que la variable *mensaje* es de tipo *String*.
- Cada variable tiene un **tipo** que determina que clase de valor puede almacenar. Ejemplo: *int* para enteros, *char* para caracteres.
- Algunos tipos de datos inician con letra mayúsculas.

Variables

- Usualmente, los nombre de variables tienen un significado.
- Ejemplos:

```
String primerNombre;
String segundoApellido;
int hora, minuto;
```

- En el ejemplo anterior hay dos variables de tipo *String* y dos de *int*.
- Los nombres de variables son sensitivos a la mayúsculas y minúsculas. No es lo mismo *primerNombre*, que *primernombre*.
- También se pueden crear variables en una sola línea.
- No se pueden usar las palabras reservadas (total de 50).

Tipos de Datos Primitivos

byte

- Es un entero de 8-bits.
- Valor mínimo de -128
- Valor máximo de 127
- Valor por defecto es 0
- Utilizado para guardar espacio en arreglos grandes.
- Ejemplo:
  - `byte a = 100;`

short

- Es un enter de 16-bits.
- Valor mínimo de -32,768
- Valor máximo de 32,767
- Utilizado para guardar memoria ya que es 2 veces más pequeño que int.
- Ejemplo:
  - `short s = 10000;`

Tipos de Datos Primitivos

int

- Es un entero de 32-bits.
- Valor mínimo -2,147,483,846
- Valor máximo 2,147,2483,647
- Utilizado normalmente cuando no hay problemas de memoria.
- Ejemplo:
  - `int a = 100000;`

long

- Es un enter de 64-bits.
- Valor mínimo de -9,233,372,036,854,775,808
- Valor máximo de 9,233,372,036,854,775,807
- Utilizado cuando se necesita un rango mucho mayor de enteros.
- Valor por defecto es 0L.
- Ejemplo:
  - `long a = 10000L;`

Tipos de Datos Primitivos

float

- Valor flotante con precisión sencilla de 32-bits IEEE 754.
- Permite guardar memoria en arreglos grandes de flotantes.
- Su valor por defecto es 0.0f.
- No es utilizado para valores precisos como monedas.
- Ejemplo:
  - `float f1 = 235.5f;`

double

- Valor flotante con doble precisión de 64-bits IEEE 754.
- Es la opción por defecto.
- No debe ser utilizado para valores precisos como monedas.
- Su valor por defecto es 0.0d.
- Ejemplo:
  - `double d1 = 123.4;`

Tipos de Datos Primitivos

boolean

- Representa un bit de información.
- Admite dos posibles valores: *true* y *false*.
- Sirve para dar seguimiento a condiciones verdadero/falso.
- El valor por defecto es *false*.
- Ejemplo:
  - `boolean one = true;`

char

- Caracteres de 16-bits Unicode.
- Valor mínimo '\u0000' (0).
- Valor máximo '\uffff' (65,535).
- Utilizado para almacenar cualquier caracter.
- Ejemplo:
  - `char letterA = 'A'`

Literales

- Es una representación del código fuente como un valor fijo.
- Se representan directamente en el código sin necesidad de ninguna computación.
- Prefijo 0:
  - `int decimal = 100;`
  - `int octal = 0144;`
  - `int hexa = 0x64;`

Notación	Carácter Representado
\n	Nueva línea
\b	Backspace
\s	Espacio
\t	Tab
\”	Comilla
\’	Apóstrofe
\\	Backslash
\ddd	Carácter octal
\uxxxx	Carácter hexadecimal
\r	Retroceso de carro

Asignaciones

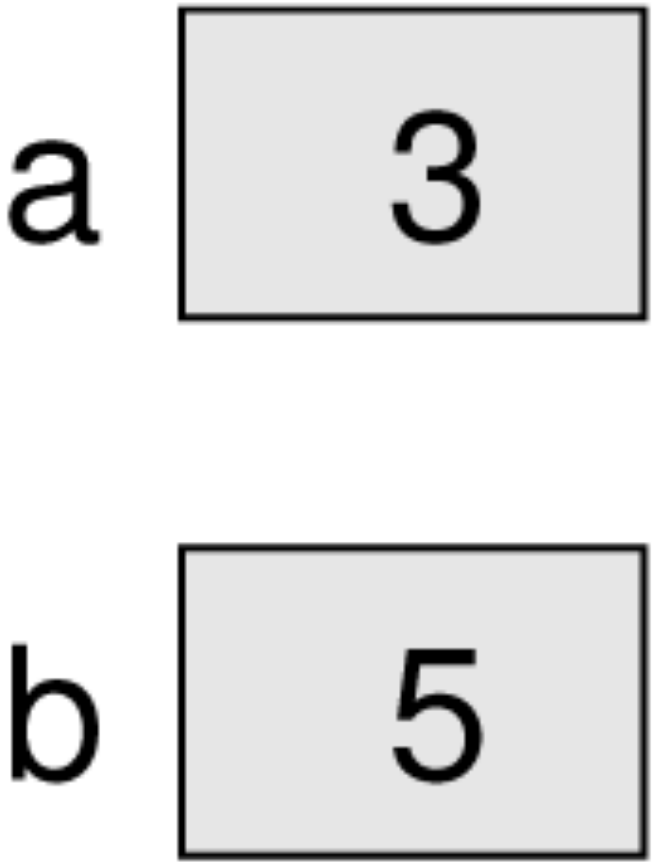
- Con variables declaradas, queremos **asignarles** valores.
- Ejemplo:
  - `mensaje = ”Hola!”;`
  - `hora = 11;`
  - `minuto = 59;`
- Al declarar una variable, se crea un nombre de almacenamiento.
- Al asignar una variable, se actualiza su valor.
- No es válido mezclar distintos tipos de datos, ejemplo: una cadena en un tipo de dato entero.
- Un punto de confusión es utilizar cadenas que parecen enteros.

Asignaciones

- No es lo mismo:
  - `mensaje = ”123”;`
  - `mensaje = 123;`
- Las variables deben ser **inicializadas** (asignadas por primera vez) antes de ser utilizadas.
- Se puede declarar una variable para asignarle un valor después.
- O bien se puede declarar e iniciarlizar en la misma:
  - `String mensaje = ”Hola!”;`
  - `int hora = 11;`
  - `int minuto = 59;`

Diagramas de Estado

- El símbolo de asignación (=) no debe confundirse con una igualdad matemática.
- La igualdad es conmutativa, la asignación no lo es.
- Ejemplo:
  - `int a = 5;`
  - `int b = a;`
  - `a = 3;`



Impresión de Variables

- Se puede mostrar el valor de una variable en consola utilizando las funciones *print* o *println*.
- Ejemplo:
  - `String primeraLinea = ”Hola, de nuevo”;`
  - `System.out.println(primeraLinea);`
- Para mostrar el nombre de la variable, debemos colocarlo entre comillas, como una cadena.
- La sintaxis para desplegar una variable es el mismo independientemente del tipo de dato.

Impresión de Variables

- Ejemplo:

```
int hora = 11;
int minuto = 59;
System.out.println(”La hora actual es ” + hora + ”:” +
minuto + ”.”);
```

- La salida del ejemplo anterior sería:

```
La hora actual es 11:59.
```

Operadores

- Los operadores son los símbolos que representan computación simple. Por ejemplo, los operadores aritméticos son la suma, resta, multiplicación, división y residuo.
- Ejemplo:

```
int hora = 11;
int minuto = 59;
System.out.print(”Minutos totales: ”);
System.out.println(hora * 60 + minuto);
```

- En este programa, *hora * 60 + minuto* es una **expresión**, el valor a ser computado. Los **operandos** serían los valores a reemplazar.

Operadores Aritméticos

Suma (+)

Resta (-)

Multiplicación (*)

División (/)

Residuo (%)

Operadores Unarios

Operador	Descripción	Ejemplo
+	Indica valor positivo	`int resultado = +1;`
-	Niega una expresión	`resultado = -resultado;`
++	Incrementa su valor en 1	`resultado++;`
--	Decrementa su valor en 1	`resultado--;`
!	Invierte el valor booleano	`boolean sal = false;` `sal = !sal;`

Números Punto-Flotante

- Normalmente se utilizan double, el cual brinda doble precisión.
- Funciona igual que con otros tipos de datos:

```
double pi;
pi = 3.14159;
```

- Java realiza una división de punto flotante cuando uno o más operandos son valores *double*.

```
double minuto = 59.0;
System.out.print(”Fraccion de hora pasada: ”);
System.out.println(minuto / 60.0);
```

- La salida del ejemplo anterior es:

```
Fraccion de la hora pasada: 0.9833333333333333333
```

Números Punto-Flotante

- Si no se manejan apropiadamente, pueden generar confusión.
- Java distingue el valor entero 1 del valor flotante 1.0, a pesar que ambos representen el mismo valor.
- Técnicamente hablando, no se pueden hacer asignaciones entre diferentes tipos de datos.
- El siguiente ejemplo mostraría error de compilación:

```
int x = 1.1;
```

- El siguiente ejemplo es legal, pero no respeta los tipos de datos (Java convierte automáticamente de 1 a 1.0):

```
double y = 1;
```

Números Punto-Flotante

- Trabajar una operación de la misma forma que la asignación del ejemplo anterior puede generar un problema de interpretación.
- Ejemplo:

```
double y = 1 / 3;
```

- En este caso, el usuario esperaría un valor 0.333333, pero en su lugar el valor es 0.0.
- En este caso, la expresión divide dos números *int*, el cual da *0*, luego hace la conversión del resultado a *double*, siendo *0.0*.
- La manera correcta de hacerlo es la siguiente:

```
double y = 1.0 / 3.0;
```

Errores de Redondeo

- La diferencia entre el número que queremos y lo que obtenemos del número de punto flotante se llama **error de redondeo.**
- Supongamos estos equivalentes:

```
System.out.println(0.1 * 10);
System.out.println(0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 +
0.1 + 0.1 + 0.1 + 0.1);
```

- Los resultados a mostrar serían:

```
1.0
0.9999999999999999
```

Errores de Redondeo

- El problema es que 0.1, el cual es una fracción de base 10, es una fracción repetitiva base 2. Por tanto, su representación es sólo un aproximado.
- En este caso se suman todas las aproximaciones y, a su vez, se contemplan todos los errores de redondeo acumulados.
- Para muchas aplicaciones, como computación gráfica, cifrado, estadística y multimedia, la aritmética de punto flotante es útil.
- Pero en aplicaciones de absoluta precisión,a solución es utilizar números enteros. Por ejemplo, un banco con números de cuenta.

Operadores para Cadenas

- No se pueden realizar operaciones matemáticas sobre las cadenas, aunque las cadenas parezcan números.
- Los siguientes ejemplos son ilegales:

```
"Hola" - 1          "Mundo" / 123          "Hola" * "Mundo"
```

- El operador + trabaja con cadenas, pero para **concatenar**, es decir, unir los valores de las cadenas.
- Los siguientes son ejemplos válidos:

```
System.out.println(1 + 2 + "Hola")   //3Hola
System.out.println("Hola" + 1 + 2)   //Hola12
```

Orden de Operaciones

- Cuando más de un operador aparece en una expresión, entonces se sigue un orden para evaluarlos.
- Java evalúa los operadores de izquierda a derecha, siguiendo las siguientes convenciones:
  - *Multiplicación y división tiene precedencia sobre suma y resta.*
  - *Si los operadores tienen la misma precedencia, se evalúa izq a der.*
  - *Si se desea sobreescribir el orden de las operaciones, se usan paréntesis.*
- No es necesario aprenderse el orden, se pueden usar paréntesis para hacer énfasis en el orden.

Orden de Operaciones

Operator	Operation
++ --	increment, decrement
+ -	unary plus, minus
!	boolean not
(<type>)	cast to <type>
* / %	multiplication, division, remainder
+ -	addition/concatenation, subtraction
< <= > >=	relational ordering
== !=	relational equality, inequality
&&	boolean and
||	boolean or
= += -= *= /= %=	assignments

Composición

- Todos los componentes de programación (variables, expresiones y sentencias) pueden trabajar en conjunto.
- Los lenguajes de programación permiten tomar pequeños bloques de construcción y componerlos.
- Por ejemplo, se pueden combinar operaciones en sentencias:

```
int porcentaje;
porcentaje = (minuto * 100) / 60;
System.out.println(hora * 60 + minuto);
```

- El lado izquierdo de la asignación debe ser un nombre de una variable, no una expresión.

Tipos de Errores

Error de Compilación

- Errores que ocurren cuando se viola las reglas de sintáxis de Java.
- El programa no puede ser compilado y el compilador muestra el error.
- Algunas veces no son fáciles de entender ya que el compilador muestra donde se detectó el error.

Error de Ejecución

- Errores que aparecen hasta después de ejecutar el programa.
- Ocurre cuando el intérprete ejecuta bytecode y algo sale mal (excepciones).
- El intérprete muestra mensajes de error de lo que ocurrió y dónde.

Error de Lógica

- El programa compila y se ejecuta sin mensajes de error, pero no genera la salida esperada.
- Identificarlos es una tarea difícil ya que requiere ingeniería en reversa.

¡HASTA LA PRÓXIMA CLASE!

Tema: Entrada y Salida