2.D. Conversiones de tipos.

Sitio: Aula Virtual
Curso: Programación_DAM
Libro: 2.D. Conversiones de tipos.
Imprimido por: LUIS PUJOL
Día: miércoles, 20 de noviembre de 2019, 02:33

Tabla de contenidos

1 Introducción

Imagina que queremos dividir un número entre otro ¿tendrá decimales el resultado de esa división? Podemos pensar que siempre que el denominador no sea divisible entre el divisor, tendremos un resultado con decimales, pero no es así. Si el denominador y el divisor son variables de tipo entero, el resultado será entero y no tendrá decimales. Para que el resultado tenga decimales necesitaremos hacer una **conversión de tipo**.

Las conversiones de tipo se realizan para hacer que el resultado de una expresión sea del tipo que nosotros deseamos, en el ejemplo anterior para hacer que el resultado de la división sea de tipo real y, por ende, tenga decimales. Existen dos tipos de conversiones:

· **Conversiones automáticas**. Cuando a una variable de un tipo se le asigna un valor de otro tipo numérico con menos bits para su representación, se realiza una conversión automática. En ese caso, el valor se dice que es promocionado al tipo más grande (el de la variable), para poder hacer la asignación. También se realizan conversiones automáticas en las operaciones aritméticas, cuando estamos utilizando valores de distinto tipo, el valor más pequeño se promociona al valor más grande, ya que el tipo mayor siempre podrá representar cualquier valor del tipo menor (por ejemplo, de `int` a `long` o de `float` a `double`).

Aquí se hace una conversión automática, también llamadas conversiones explícitas:

```
public class Principal3 {


   public static void main(String[] args) {

      int num1=9;
      float num2=9.3f;
      float suma;
      suma=num2+num1;

   }

}
```

Como num2 es float y num1 es int, Java lo convierte todo a float y lo guarda en la variable suma (que es float).

Prueba que pasaría si suma fuera int.

Otro ejemplo (de conversión implícita):

```
import java.util.Scanner;


public class conversionImplicita1 {


   public static void main(String[] args) {
      //conversiones implícitas
      int num;
      float numAñadido;
```

```
        short numCorto=9;
        byte numC=90;

        Scanner scanner=new Scanner(System.in);
        System.out.print("Inserte un número: ");
        num=scanner.nextInt();
        numAñadido=num+0.2f;
        System.out.println("El valor del número insertado, habiéndole sumado previamente la cantidad
de     0.2, es "+numAñadido);
 ;

    }

}
```

· **Conversiones explícitas**. Cuando hacemos una conversión de un tipo con más bits a un tipo con menos bits. En estos casos debemos indicar que queremos hacer la conversión de manera expresa, ya que se puede producir una pérdida de datos y hemos de ser conscientes de ello. Este tipo de conversiones se realiza con el **operador** `cast`. El operador `cast` es un operador unario que se forma colocando delante del valor a convertir el tipo de dato entre paréntesis. Tiene la misma precedencia que el resto de operadores unarios y se asocia de izquierda a derecha.

Debemos tener en cuenta que **un valor numérico nunca puede ser asignado a una variable de un tipo menor en rango, si no es con una conversión explícita**. Por ejemplo:

```
int a;
```

```
byte b;
```

```
a = 12;              // no se realiza conversión alguna
```

```
b = 12;              // se permite porque 12 está dentro
```

```
                     // del rango permitido de valores para b
```

```
b = a;               // error, no permitido (incluso aunque
```

```
                     // 12 podría almacenarse en un byte)
```

```
byte b = (byte) a;   // Correcto, forzamos conversión explícita
```

En el ejemplo anterior vemos un caso típico de error de tipos, ya que estamos intentando asignarle a `b` el valor de `a`, siendo `b` de un tipo más pequeño. Lo correcto es promocionar `a` al tipo de datos `byte`, y entonces asignarle su valor a la variable `b`.

Otro ejemplo de conversiones explícitas:

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

```
public class ConversionesExplicitas1 {

   public static void main(String[] args) {
      int nota1, nota2;
      float media;
      Scanner scanner = new Scanner(System.in);

      System.out.print("\n\tInserte la primera nota: ");
      nota1 = scanner.nextInt();

      System.out.print("\n\tInserte la segunda nota: ");
      nota2 = scanner.nextInt();

      media = (float) ((nota1 + nota2) / 2);

      System.out.println("\n\tLa nota media es " + media);
   }

}
```

2 Reglas de Promoción de tipos de Datos.

Cuando en una expresión hay datos o variables de distinto tipo, el compilador realiza la promoción de unos tipos en otros, para obtener como resultado el tipo final de la expresión. Esta promoción de tipos se hace siguiendo unas reglas básicas en base a las cuales se realiza esta promoción de tipos, y resumidamente son las siguientes:

- Si uno de los operandos es de tipo **double**, el otro es convertido a **double**.
- En cualquier otro caso:
  - Si el uno de los operandos es **float**, el otro se convierte a **float**
  - Si uno de los operandos es **long**, el otro se convierte a **long**
  - Si no se cumple ninguna de las condiciones anteriores, entonces ambos operandos son convertidos al tipo **int**.

<table>
<tr><th colspan="3">Tabla sobre otras consideraciones con los Tipos de Datos</th></tr>
<tr><th>Conversiones de números en Coma flotante (float, double) a enteros (int)</th><th>Conversiones entre caracteres (char) y enteros (int)</th><th>Conversiones de tipo con cadenas de caracteres (String)</th></tr>
<tr>
<td>Cuando convertimos números en coma flotante a números enteros, la parte decimal se trunca (redondeo a cero). Si queremos hacer otro tipo de redondeo, podemos utilizar, entre otras, las siguientes funciones:

• Math.round(num): Redondeo al siguiente número entero.
• Math.ceil(num): Mínimo entero que sea mayor o igual a num.
• Math.floor(num): Entero mayor, que sea inferior o igual a num.

double num=3.5;
x=Math.round(num); // x = 4
y=Math.ceil(num); // y = 4
z=Math.floor(num); // z = 3</td>
<td>Como un tipo char lo que guarda en realidad es el código Unicode de un carácter, los caracteres pueden ser considerados como números enteros sin signo.

Ejemplo:

int num;
char c;

num = (int) ‘A’;
//num = 65
c = (char) 65; // c = ‘A’</td>
<td>Para convertir cadenas de texto a otros tipos de datos se utilizan las siguientes funciones:

num=Byte.parseByte(cad);
num=Short.parseShort(cad);
num=Integer.parseInt(cad);
num=Long.parseLong(cad);
num=Float.parseFloat(cad);
num=Double.parseDouble(cad);

Por ejemplo, si hemos leído de teclado un número que está almacenado en una variable de tipo String llamada cadena, y lo queremos convertir al tipo de datos byte, haríamos lo siguiente:

byte n=Byte.parseByte(cadena);</td>
</tr>
</table>

3 Conversión de tipos de datos en Java.

En Java se podrán realizar conversiones de tipos de datos de acuerdo a la siguiente tabla:

Tabla de Conversión de Tipos de Datos Primitivos									
		Tipo destino							
		boolean	**char**	**byte**	**short**	**int**	**long**	**float**	**Double**
Tipo origen	boolean	-	N	N	N	N	N	N	N
	char	N	-	C	C	Cl	Cl	Cl	Cl
	byte	N	C	-	CI	Cl	Cl	Cl	Cl
	short	N	C	C	-	Cl	Cl	Cl	Cl
	int	N	C	C	C	-	Cl	Cl*	Cl
	long	N	C	C	C	C	-	Cl*	Cl*
	float	N	C	C	C	C	C	-	Cl
	double	N	C	C	C	C	C	C	-

Explicación de los símbolos utilizados:

N: Conversión no permitida (un **boolean** no se puede convertir a ningún otro tipo y viceversa).
CI: Conversión implícita o automática. Un asterisco indica que puede haber posible pérdida de datos.
C: Casting de tipos o conversión explícita.

El asterisco indica que puede haber una posible pérdida de datos, por ejemplo al convertir un número de tipo **int** que usa los 32 bits posibles de la representación, a un tipo **float**, que también usa 32 bits para la representación, pero 8 de los cuales son para el exponente.

En cualquier caso, las conversiones de números en coma flotante a números enteros siempre necesitarán un **Casting**, y deberemos tener mucho cuidado debido a la pérdida de precisión que ello supone.