**CONVERSIÓN BETWEEN PRIMITIVE DATA TYPES**

* Antes de que un dato pueda ser almacenado en un variable, el tipo de dato debe ser compatible con la variable (tipo de dato).
* Java realiza algunas conversiones entre tipos de datos automáticamente.
* Pero no realiza conversiones en las que pueda existir perdida de datos.
* Antes de que un dato sea asignado a una variable, java verifica si el dato y el tipo de dato (variable) son compatibles. Por ejemplo:

|  |
| --- |
| **int x;**  **double y = 2.5;**  **x = y;** |

* La sentencia de asignación intenta almacenar un valor double (2.5) en una variable de tipo int.
* Cuando el compilador de java encuentra esta línea de código, responderá con un mensaje de error: ***posible lossy of conversion from double to int***.
* Sin embargo, no todas las asignaciones que mezclan tipos de datos son rechazadas por el compilador.

|  |
| --- |
| **int x;**  **short y = 2;**  **x = y;** |

* Esta asignación, la cual almacena un dato de tipo short (2) en un dato de tipo int (x), trabajará sin ningún problema.
* Entonces ¿por qué java permite almacenar un dato short en un dato int? ¿pero no permite almacenar un dato double?
* La razón más obvia es que double puede almacenar números con decimales (parte fracción) y, además, puede almacenar valores mucho más grandes de lo que puede almacenar int
* Si java permitiera asignar un dato double a un int, una pérdida de datos sería probable.
* Un tipo de dato supera a otro si puede contener (almacenar) un número mayor
* Un float supera a un int
* Un int supera a un short

|  |
| --- |
|  |

* ***Widenning Conversion:*** En asignaciones donde valores de variables pequeñas son almacenados en variables de valores más grandes, java automáticamente convierte el valor pequeño en un valor grande.

|  |
| --- |
|  |

* ***Narrowing Conversion:*** es la conversión de un valor grande a uno pequeño. Por ejemplo. Convertir un double a un int.

|  |
| --- |
|  |

* Ya que en este tipo de conversión puede haber perdida de información, java no la realiza automáticamente.

**CAST OPERATORS**

* El operador cast nos permite convertir un valor manualmente.
* Incluso si esto conlleva a una conversión de tipo ***Narrowing (grande a pequeño)***
* El operador cast es un operador que aparece como tipo de datos encerrado entre paréntesis.
* El operador indica el valor a ser convertido.

|  |
| --- |
|  |

* En esta sentencia, el operador cast, es la palabra int dentro del paréntesis.
* Retorna el valor de la variable double (grande) convertida a int(pequeño). Dicho valor luego es almacenado en la variable pequeña.
* Si el número es de tipo float o double, el valor que es retornado será truncado, es decir, la parte fraccional (decimales) será perdida.
* El valor original de la variable grande no será modificado.
* Cuando el operador cast es aplicado a una variable, no cambia su contenido.
* El operador cast solo retorna el valor convertido a un tipo específico de dato.

Más ejemplos:

|  |
| --- |
|  |

* En una división, cuando ambos operadores son enteros, el resultado será un numero entero. **División entera.**
* Esto significa que el resultado de la división será un numero entero, sin números decimales.

|  |
| --- |
|  |

* Aunque la división de 10 entre 4 da 2.5, el primer resultado almacenará el valor 2.
* Porque ambas variables, a y b son enteras, el resultado será un int, y la parte fraccionaria será desechada.
* Pero podemos modificar el código con el operador cast para que el resultado salga con el punto decimal.
* La variable a es un int que guarda el valor 10. La expresión ***double(a)*** retorna el valor de a convertido a double.
* Esto significa que uno de los dos operadores de la división es un tipo de dato double, por lo tanto, el resultado de la división será un double.
* La sentencia también se pudo haber escrito de la siguiente manera:

**double r2 = a / (double) b**

* El operador cast puede ser aplicado a una expresión entera encerrada entre paréntesis.

**double r2 = (double) (a/b)**

* Esta sentencia no convierte el valor de **a** o **b** a double, pero si convierte el resultado de la división (a/b) a double.
* Si esta expresión es utilizada, el resultado de la división será 2.0 ¿Por qué?
* El resultado de la expresión (a/b) da como resultado 2, porque ambos valores son enteros.
* Luego, al número 2, se le convierte en double, quedando 2.0
* Para prevenir este tipo de errores, lo mejor es convertir uno de los operadores a double.

**MIXED INTEGER OPERATIONS**

* Cuando los tipos de datos byte o short son utilizados en operaciones aritméticas, son temporalmente convertidos a int.
* El resultado de una operación aritmética utilizando solamente byte, short o int siempre dará como resultado un int.

|  |
| --- |
|  |

**OTHER MIXED MATHEMATICAL EXPRESSIONS**

* En situaciones donde la expresión matemática tiene uno o más valores de tipo **double, float** o **long**, java se esfuerza en convertir todos los operandos de la expresión en el mismo tipo de dato.
* Si uno de los operandos es double, el otro operando será convertido a double.
* El resultado de la operación será un double.

a = b + c

* Suponiendo que b es double, y c es int, c será convertida a double antes de que se realice la suma.
* El resultado de la suma será un double, por lo tanto, la variable a tiene que ser double.
* Si uno de los operandos es float, el valor del otro operando será convertido a float.
* El resultado de la expresión será un float.

z = x \* y

* Suponiendo que **x** es de tipo short, e **y** es float. El valor de x será convertido a float antes de realizarse la multiplicación.
* El resultado de la multiplicación será de tipo float, entonces la variable z debe ser double o float.
* Si uno de los operandos es long, el valor del otro operando será convertido a long.
* El resultado de la expresión será un dato long.

c = a – b

* Asumiendo que a es long y b es short. La variable b será convertida a long antes de que se realice la resta.
* El resultado de la resta será un long, entonces la variable c debe ser long, float o double.

***Fuente:***

***2016 - Starting out with java - Tony Gaddis – Tema 2.7 – Página 65***