Primitive operators and types:

In general, and with only a few exceptions, the different Java operators only work on primitive types.

The operators are created to work with any type for which the operation is reasonable.

The resulting type is usually reasonable given the types of the operands; however, most operators do not return values of narrower type than int.

Below are the specifics for most of the operators of Java. A few additional "shortcut" operators will be discussed around midterms.

Primitive operations:

Following are the type rules for the different primitive operators. Please see your textbook for precedence and associativity rules.

1) arithmetic operators: +, -, \*, /, %

2) unary arithmetic operators: +, -

3) binary operators (not used in EECS 132, they manipulate the binary representations): &, |, ^, ~, <<, >>, >>>

The above operators have the same type rules:

a) The two operands (or one operand for the unary operators) must be a numeric primitive type (i.e. not boolean)

b) The result is the narrowest of double, float, long, or int that is at least as wide as the widest operand.

c) Any operand narrower than the result type is automatically widened before the operation is performed.

ex: 5.0 + 3 (the result type is double because of the 5.0, before the addition is performed, the 3 is widened to 3.0)

'a' + 'b' (the result type is int and both operands are widened to int before the addition is performed)

4) comparison operators: >, <, >=, <=

The type rules:

a) The operands must be a numeric type

b) The result type is boolean

c) The narrower operand type is automatically widened before the operation is performed

5) equality operators: ==, !=

The type rules:

a) If used with primitive types, the operands must either be both numeric types or both boolean

b) For primitive numeric types, the narrower type is automatically widened before the operation

c) If used with non-primitive types, either one operand must be null (not an address), or the two types must be comparable in the hierarchy or one type must extend the other

d) The result is boolean

ex:

5 == 5.0 <- first converts 5 to 5.0 before comparing

JFrame j1 = new JFrame(); JFrame j2 = new JFrame(); j1 == j2

<- legal and returns false. == does a comparison of the binary values stored in j1 and j2. These are both addresses, and j1 and j2 store different addreses.

- for non-primitives, == returns true if the two values are the same object.

6) boolean operators: &&, ||, !, &, |

The type rules:

a) The operands (or operand for !) must be boolean

b) The result is boolean

How &&, ||, and ! work (the other three operators are not often used and so will be skipped)

NOT: !

operand result

true false

false true

AND: &&

left-operand right-operand result

true true true

true false false

false NOT EVALUATED false

OR: ||

left-operand right-operand result

true NOT EVALUATED true

false true true

false false false

7) assignment opertor: (not really a primitive operator)

x = 5;

a) the type of the expression on the right must be the same or narrower as the type of the variable on the left

EXCEPTION: if the type on the right is int, the variable on the left is narrower than int, and the expression is on the right is ONLY constants and primitive operators,

and the value "fits" inside the range of values for the narrower type, then the value on the right is automatically converted to the narrower type of the variable.

char c = 10 + 5 <- legal

char c = 10 + x <- not legal (no matter what the type of x is --- do you see why?)

char c = Character.MAX\_VALUE + 1 <- not legal because the result of the + is too large to fit in a char type.

int x = Integer.MAX\_VALUE + 1 <- legal because the result of + is int

b) the type of the result is the type of the variable (the value is the value assigned to the variable)

A Short Quiz on Types, Assignment and Primitive Operators

Suppose we have the following variables:

double x

int y

char c

For each of the following expressions, is the expression legal and what does it do?

EXPRESSION 1:

x = 2.5

Is this legal?

Yes, the type of 2.5 is double and the type of x is double. The types match.

What does the expression do?

Stores 2.5 into variable x (or more precisely, the memory location referred to by the name x)

What is the type and value of this expression? (Note that if we ended the line with a semicolon, then it would be a statement, not an expression. Statements do not have types or values.)

The type of = is the type of the variable. The value if the value assigned to the variable.

The type is double the value is 2.5.

EXPRESSION 2:

y = 5

Is this legal?

Yes, the type of 5 is int and the type of y is int. The types match.

What is the type and value of this expression?

The type is int and the value is 5.

EXPRESSION 3:

c = 'z'

Is this legal?

Yes, the type of 'z' is char and the type of c is char. The types match.

What is the type and value of this expression?

The type is char and the value is 'z'.

EXPRESSION 4:

x = 5

Is this legal?

Yes, the type of 5 is int and the type of x is double. The types do not match but double is wider than int so the 5 is converted from int to double.

What is the type and value of this expression?

The type is double and the value is 5.0.

EXPRESSION 5:

y = 2.5

Is this legal?

No, the type of 2.5 is double and the type of y is int. int is narrower than double and so the assignment is not allowed.

How can we fix it? With a typecast.

y = (int)2.5

Is this legal?

Yes, the type of 2.5 is double. The typecast request a change to type int. double and int are both numeric primitives and so the conversion is allowed. The type if (int)2.5 is int and the type of y is int.

What is the type and value of this expression?

The type is int and the value is 2. (All data after the . is truncated.)

EXPRESSION 6:

c = 65

Is this legal?

Yes, the type of 65 is int and the type of c is char. char is narrower than int BUT the special rule of = states that if both types are non-floating point numeric, the right side only contains numbers and

primitive operators, and the value will fit in the variable type, the assignment is allowed.

What is the type and value of this expression?

The type is char and the value is 'A'. 'A' is the char representation of the number 65 in Java.

EXPRESSION 7:

short s = 1000000

Is this legal?

No, 1000000 is an int. The special rule for = does not apply because 1000000 does not fit in the range for short (approx +/- 32,000)

EXPRESSION 8:

short s = 1000000 / 10000

Is this legal?

Yes, 1000000 / 10000 is an int, but expression is only constants and primitive operators, the resulting value is 100, and 100 fits in a short type.

EXPRESSION 9:

c = 'A' + 1

Is this legal?

Yes, the type of 'A' is char and the type of 1 is int so the type of 'A' + 1 is int. However, by the special rule of =, 'A' + 1 only has numbers and operators and the value will fit in a char type.

What is the type and value of this expression?

The type is char and the value is 'B'.

EXPRESSION 10:

c = c + 1

Is this legal?

No, the type of c is char and the type of 1 is int so the type of c + 1 is int. The special rule does not apply because c + 1 contains a variable, not just numbers and operators.

EXPRESSION 11:

c = +c

Is this legal?

No, the type pf +c is int!

EXPRESSION 12:

c = (char)c + (char)1

Is this legal?

No, the type of c is char, adding the typecast has no affect. The type of 1 is int, but it is typecast to char. However, the type rules for the arithmetic operators states that char + char gives int.

So, the type of (char)c + (char)1 is int and int is wider than char.

EXPRESSION 13:

c = (char)(c + 1)

Is this legal?

Yes, the type of (c + 1) is int. int and char are both numeric primitives and so they can be converted. The type of (char)(c + 1) is char.

What is the type and value of the expression?

The type is char and the value is 'C' (assuming c was originally storing 'B')

EXPRESSION 14:

y = 5 + 4.0/2.0

Is this legal?

No. Even though the math value is 7, which is a whole number, the types do not match. The first operation is /, and double / double gives double. The second operation is + and int + double gives double.

y has type int, but int is narrower than double.

EXPRESSION 15:

float f = 10.0

Is this legal?

No! 10.0 is a double which is wider than float. Although 10.0 can fit in the float type, the "special rule" for assignment does not apply to non-integer types.

EXPRESSION 16:

x = y = c

Is this legal?

Yes. The first operation is y = c. The type of c is char, the type of y is int. The second operation is x = (y = c). The type of x is double and the type of (y = c) is int.

What is the type and value of the expression?

The type is double. Assuming c is storing 'C' which has value 67, the value of the expression is 67.0.

EXPRESSION 17:

c = y = x

Is this legal?

No. The first operation is y = x. The type of x is double, the type of y is int, and int is narrower than double.

EXPRESSION 18:

x == y == c

Is this legal?

No. The first operation is x == y. The type of x is double and the type of y is int. Both int and double are numeric primitives and so they can be compared. The type of (x == y) is boolean.

The second operation is (x == y) == c. The type of (x == y) is boolean and the type of c is char. char is a numeric primitive but boolean is a non-numeric primitive. The type types cannot be compared.

EXPRESSION 19:

(y = 7) != 8 && 7 < 5

Is this legal?

Yes.

What is the result of the expression?

false

What is stored in y after execution of the expression?

7

The left operand of && is evaluated first. HOW DO I KNOW IT IS && AND NOT != or < ?? SEE THE PRECEDENCE RULES IN YOUR TEXTBOOK.

The left operand of != is evaluated. It is an assignment statement. 7 is stored in x and the result of the = operator is the value stored, 7.

Now, 7 != 8 is evaluated, and the result is true.

Because the left operand evaluates to true, the right operand is evalated. 7 < 5 is false.

true AND false is false.

EXPRESSION 20:

7 < 5 && ((y = 10) != 6)

Is this legal?

Yes.

What is the result of the expression?

false

What is stored in y after execution of the expression?

7 (assuming 7 is was previously stored in y from Expression 15)

The left operand of && is evaluated first.

7 < 5 evaluates to false so the right operand is not evaluated. As a result the assignment never happens and x still stores the previous value of 7.