Recall the primitive types we use for most of our programming:

int (32-bits, integer), double (64-bits, floating point number), char (16-bits, a character), boolean (1-bit (\*), true or false)

You saw some strange behavior with the int type in lab 1. Recall:

> Integer.MAX\_VALUE

2147483647

> 2147483647 + 1

-2147483648

Why does Integer.MAX\_VALUE + 1 become a large negative number?

It is because the int type is only 32-bits so there is a limit to the number of values it can store. The reason it "wraps" to negative numbers

is a trick of the binary representation currently used for int.

For positive numbers, it uses a straight binary-to-decimal conversion. (We abbreviate below to avoid writing all 32 bits.)

0000....0000 = 0

0000....0001 = 1

0000....0010 = 2

...

0111....1111 = 2147483647 = 2^(31) - 1

If we add 1 to this, we get

1000....0000

and by designer decision, this will binary value will represent -2^(31). Now, if we keep adding 1 we get:

1000....0000 = -2^(31)

1000....0001 = -2^(31) - 1

1000....0010 = -2^(31) - 2

...

1111....1111 = -1

If we add 1 to this, we keep carrying a 1, but the last carried bit is thrown away (because we only have 32 bits), and we get

0000....0000 = 0

So, with the int (as well as the byte, char, short, and long) type, if we keep adding 1, the values will cycle.

Here is another strange behavior, but this time with the double type:

> 1.3 + 1.3

1.6

> 1.3 + 2.3

3.5999999999999996

Why does 1.3 + 1.3 equal 2.6, but 1.3 + 2.3 does not equal 3.6?

The numbers are represented in binary, and some fraction numbers (such as 3/10) can not be represented exactly.

This is similar to how 1/3 and 1/7 can not be represented exactly in decimal.

If the error in the representation is small enough, the Java routine that prints the numbers will round the printed number to a "nice" decimal value.

For 1.3 + 2.3, the errors in the representation just happen, when added, to be big enough so that the result is not "close enough" to 3.6.

Why should you care?

Mathematics in Java is not the same as "real" mathematics. Errors will crop up due to the fixed size of the numeric data types that would not appear if we

were manipulating the numbers using "real" mathematics. A programmer must always be alert to situations where such errors could occur so that they do not cause

strange program behavior.

Typecasts (type conversions)

Java allows the programmer to change the type of an expression by using a typecast.

To change the type, you place the desired type in parentheses immediately to the left of the value.

Ex: (double)3 changes the int value 3 to the double value that is the closest to 3. In this case, the value is 3.0.

(int)3.7 changes the double value 3.7 to an int type with value 3 by truncating everything after the decimal point

You can change between all numeric (i.e. not boolean) primitive types.

Note that with primitive types, changing the type also changes the value.

Introduction to Variables

A variable is the name given to a location in memory. The variable has a type associated with it.

A variable is used to store data.

"The only way to remember a value in a Java program is to store the value in a variable."

Creating Variables

A declaration has the form:

type name

Two examples:

int x

double temperature

Java now sets aside a chunk of memory for each varialble. The first is a 32-bit chunk that is associated with the name "x".

The area of memory can store values that represent int.

The second is a 64-bit chunk of memory that is associated with the name "temperature", and we can store values of type double in it.

Storing Values in Variables

We call storing a value "assigning a variable".

Some variables will get default values when created, but for many variables, the value initially in the variable is whatever happened to be stored in that

memory location previously. It is an error to try to access the variable before explicitly storing a value.

To store a value in the variable, you use the = operator.

variable = value

For example:

x = 100

This stores the int value 100 into the memory location with the name "x".

The value can be any expression of the appropriate type.

temperature = 75.0 \* 9.0 / 5.0

will store the double value 135.0 in the memory location with the name "temperature".

The type of value you store in the variable must match the type of the variable:

x = 100.0

Java will give an error. Why? The type of the variable and the type of the value do not match!

More on Type conversions (typecasts)

Values are automatically converted (called "type coercion") from "narrower" to "wider" types.

Values must be explicitly converted by the programmer when changing from a "wider" to a "narrower" type.

Widest: double, float, long, int, short/char, byte : Narrowest

int x

double temperature

x = 100.0 ILLEGAL. The double 100.0 is a wider type than the int type we indicated would be stored in x

x = (int)100.0 LEGAL

temperature = 75 LEGAL The int type 75 is automatically converted to the wider double

temperature = 'A' LEGAL! Even though it makes no sense, the char type 'A' is narrower than the double type of the temperature variable.

The type rules of Java are meant to prevent errors by making sure that data is always used appropriately. However, in the last example, we can see that the rules are not foolproof.

Type Rules and Expressions

Java code fragments that have a value are called "expressions". Because they have values, expressions also have types.

An expression can be used -anywhere- in the code where a value of that type may be used.

Type rule for the = operator.

The type of the = expression is the type of the variable.

The value of the = expression is the value being assigned in the variable.

Is this legal?

int x

double temperature

temperature = x = 6

YES! We have to consider how = works. The compiler will verify that the type of the variable on the left of the = matches the value on the right.

In this case, the variable on the left is temperature storing double.

The value on the right is the expression "x = 6" which has type int. int is narrower than double.

When run, first 6 will be stored in x and then 6.0 will be stored in temperature.

How about this?

x = temperature = 6

NO! The compiler will verify that the type of the variable on the left of the = matches the type on the right.

In this case, the variable on the left is x storing int.

The value on the right is the expression "temperature = 6" which has type double, but double is wider than int.

To make the above work, we must typecast the value. Typecasts have very high precedence, and so we must use parentheses to indicate the value that we want to typecast.

In this case, we want to typecast the value of the assignment operator = from double to int.

x = (int)(tempterature = 6)

The Laws of Variables:

First Law of Variables: You must declare (create) a variable before you can use it.

Second Law of Variables: The first use of a variable must be to assign (store) a value.