Types and type int

A *type* is a set of values together with the (primitive) operations on them. Memorize this definition! The term will be used over and over again when dealing with Java.

Type *integer* is used frequently in mathematics, the sciences, and engineering:

Type *integer*:
values: the set of integers { …, -3, -2, -1, 0, 1, 2, 3, …}
operations: *negation*, or *unary minus* –b
addition b + c,
subtraction b – c,
multiplication b * c,
division /.

Integers have to be stored in memory locations on a computer, which have finite capacity. Therefore, Java, like most programming languages, does not use type *integer*. Instead, it has a type **int**, in which the values are restricted to the range -2^{31} to $2^{31}-1$. This range is chosen because of the way the numbers are stored in today's computers, in a sequence of 32 bits, or 4 bytes. You don't have to know exactly how the values are stored in 4 bytes.

Operations on type int

The operations of type **int** are those of type integer, except that a new one, remainder %, has been added. We will go over these in a minute. But before we do, we mention an important principle decided upon by the designers of Java:

Principle: Each primitive operation of type **int** must yield an **int**.

We use this principle to help answer questions about type **int** and its operations.

We now use DrJava's interactions pane to investigate some properties of type **int**. First, unary minus, addition, subtraction, and multiplication work as you would expect them to.

But what about division? According to the principle that **int** operations must yield **int**s, 6 / 4 cannot yield 1.5. There are two possible choices for the value of 6 / 4: either truncate to 1 or round up to 2. The Java designers chose truncation toward zero, so 6 / 4 = 1 and –6 / 4 = –1.

The percent sign % is used for the remainder operation 6 % 4 is the remainder when 6 is divided by 4. So, 6 % 4 = 2 and 6 % 3 = 0.

Names for the minimum and maximum values of type int

Java has names for the smallest and largest values of type **int**:

Integer.MIN_VALUE and Integer.MAX_VALUE

Later, we will see why it has these values have these strange names, with a period in them. For now, just use them.

Since Integer.MAX_VALUE is the largest value of type **int**, what do you think adding 1 to Integer.MAX_VALUE yields? There are several possibilities, and the designers of Java had to choose one. What do you think it should be? Note that, there is no "wrong" answer. There are reasons for each of several choices, and the designers just chose one of them.

One choice would be to have evaluation of Integer.MAX_VALUE + 1 give a error and halt execution, so that the programmer, or user, knows that some mistake has been made. Another might be to yield Integer.MAX_VALUE itself, and one might even say it should be 0. Java chose another answer; Integer.MAX_VALUE + 1 yields the smallest value of type **int** —the values "wrap around". To know why this value was chosen, one has to know something about arithmetic is implemented in the computer, and we don't go into that here.

The programs you write in this course will exhibit this wrap-around, and you need not worry about this kind of "overflow" now.

Types and type int

Java actually has four different integer types, each with a different range of values, so that programmers can control to some extent the memory used by a program. We have seen type **int**. A value type int uses 32 bits, or 4 bytes.

- Values of type **byte** occupy one byte (8 bits) of space.
- Values of type **short** occupy 2 bytes (16 bits).
- Values of of type **long** occupy 8 bytes (64 bits).

You do not have to remember all these details about the integer types in Java. In fact, in this course, we will be using only type **int**. However, if you ever need information about any of these types, turn to Chapter 6 of the text for this course. Use this chapter often as a reference, whenever you have questions about the primitive types in Java.

The ProgramLive CD has an extensive, hypertexted glossary of definitions, which can be extremely useful in finding out information about Java and programming. For example, here is page 6-1 of ProgramLive. If we click on a blue underlined word, the glossary opens to the corresponding entry for that word. This glossary provides an efficient, effective way for you to explore the meanings of words related to programming.