

Data Types

The objects about which computer programs compute is data. We often think first of integers. Underneath it all, the primary unit of data a machine has is a chunks of bits the size of a word. (The usual size of a word in hardware is 32, so $2^{32} = 4,294,967,296$ different objects can be represented.) But a language can organize the bits into varieties called data types. With different data types the same bit pattern might represent a different object. A language can provide assistance in computing with these data types giving the illusion or the abstraction of computing with complex objects like real numbers, sound, video, graphs, etc.

Data Types

- ▶ The eight primitive data type in Java
 - ▶ two's complement, IEEE754
- ▶ Definition: *objects* are everything but primitives
- ▶ The wrapper classes: `java.lang.Boolean`, `Character`, `Float`, etc.
- ▶ `java.math.BigInteger`, `java.math.BigDecimal`
- ▶ **Strings**: `java.lang.String`, and `java.lang.StringBuilder`, but do not use `java.lang.StringBuffer`.
- ▶ Arrays and its “wrapper” class `java.util.Arrays`
- ▶ Characters: Unicode

Java Data Types

In Java the data types are divided into two families: primitive data types and objects.

Java Primitive Data Types

The primitive data types (of which there are eight in Java) are those data types with simple structure and can be represented in the 32 or 64 bits of hardware, for example `int` and `long`. The operations on the primitive data types usually have hardware support. So computing with primitive data types is typically fast and convenient. These data types are used a lot in computing, though as computers get more powerful, programmers get more knowledgeable, and APIs get more expressive, it is more and more common for programs to use complex data types.

Java Objects

The other families of data types in Java are called *objects*. (This is another use of the overworked word “object.” In Java jargon an object is a specific category of data types: namely all those that are not primitive.) Common Java objects are arrays and strings.

The programmer can even define new Java objects and this extremely important. This is the subject of a later lecture.

Java Primitive Data Types

- ▶ boolean
- ▶ char
- ▶ arithmetic
 - ▶ integral (twos-complement)
 - ▶ byte
 - ▶ short
 - ▶ int
 - ▶ long
 - ▶ floating-point (IEEE)
 - ▶ float
 - ▶ double

boolean

There are only two different boolean values: true and false. The boolean data type is essential for expressions to control conditional statements and loops.

char

The data type `char` represents characters of text like the letter 'A', 'B', etc. The repertoire of characters comes from the important and well-established Unicode standard. We discuss this interesting and Byzantine collection later.

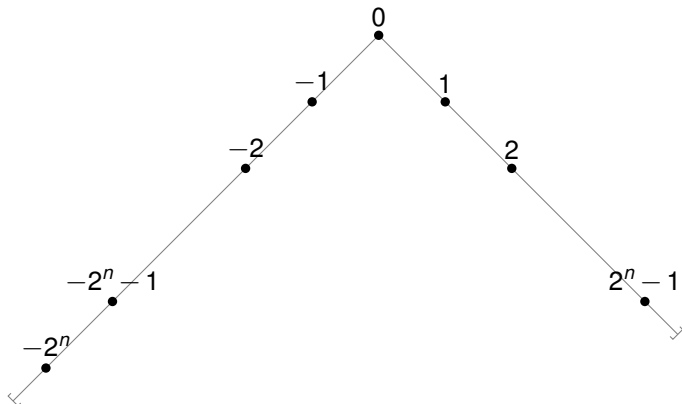
Two's complement

011111111	=	127
000000010	=	2
000000001	=	1
000000000	=	0
111111111	=	-1
111111110	=	-2
100000001	=	-127
100000000	=	-128

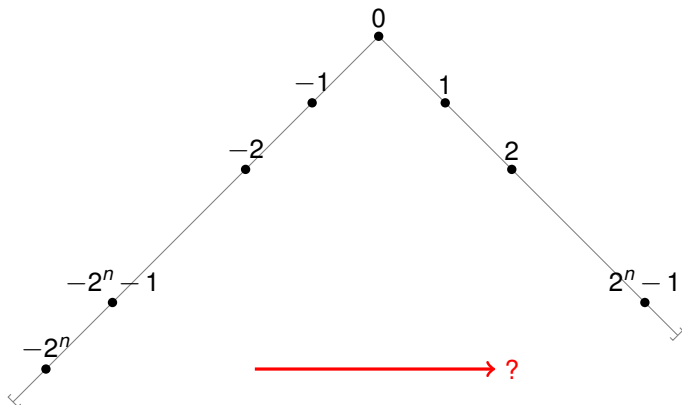
Two's complement



Two's complement



Two's complement

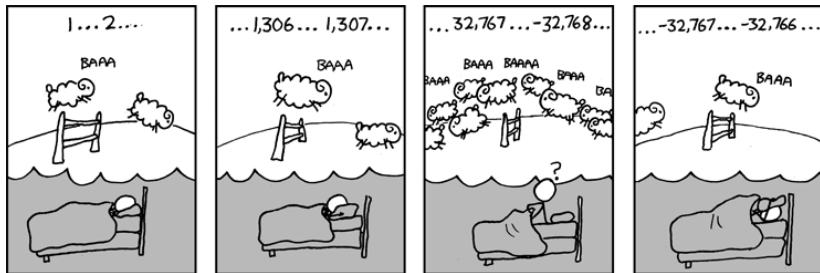


byte

The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive). The byte data type can be useful for saving memory in large arrays, where the memory savings actually matters. They can also be used in place of int where their limits help to clarify your code; the fact that a variable's range is limited can serve as a form of documentation.

short

The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive). As with byte, the same guidelines apply: you can use a short to save memory in large arrays, in situations where the memory savings actually matters.



int

The `int` data type is a 32-bit signed two's complement integer. It has a minimum value of `-2,147,483,648` and a maximum value of `2,147,483,647` (inclusive). For integral values, this data type is generally the default choice unless there is a reason to choose something else. This data type will most likely be large enough for the numbers your program will use, but if you need a wider range of values, use `long` instead.

long

The long data type is a 64-bit signed two's complement integer. It has a minimum value of -9,223,372,036,854,775,808 and a maximum value of 9,223,372,036,854,775,807 (inclusive). Use this data type when you need a range of values wider than those provided by int.

float

The float data type is a single-precision 32-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in section 4.2.3 of the Java Language Specification. As with the recommendations for byte and short, use a float (instead of double) if you need to save memory in large arrays of floating point numbers. This data type should never be used for precise values, such as currency. For that, you will need to use the `java.math.BigDecimal` class instead.

double

The double data type is a double-precision 64-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in section 4.2.3 of the Java Language Specification. For decimal values, this data type is generally the default choice. As mentioned above, this data type should never be used for precise values, such as currency.

All the primitive, numeric data in Java are limited (obviously you can only represent 2^{32} or 2^{64} numbers). This is a lot of numbers, but we have the expectation that one does not run out of numbers and this causes trouble in some contexts.

Even worse, the mathematics of computer numbers, is not the same as “real” mathematical numbers and this causes a great deal of trouble.

To ameliorate these problems the Java libraries provide arbitrary-precision signed decimal numbers `java.lang.BigDecimal` arbitrary-precision integers `java.lang.BigInteger`. There are not primitive data types, but have many of the same operations as the primitive data types.

Consider the following program.

```
public final class Monetary {  
  
    private final static String FMT =  
        "Bought %d items @ $%.2f; funds remaining $%.2f%n"  
  
    public static void main (final String args[]) {  
        final double price = 0.10;  
        double funds = 2.00;  
        int items = 0;  
        while (funds >= price) {  
            funds -= price;  
            items++;  
        }  
        System.out.format (FMT, items, price, funds);  
    }  
}
```

The output is a surprising:

```
Bought 19 items @ $0.10; funds remaining $0.10
```

because the number 0.1 cannot be represented exactly in binary.

The problem can be avoided by using `BigDecimal` which does not represent numbers in binary.

(Of course, everything is binary in a computer; it is just the `BigDecimal` uses some bit patterns to represent decimal digits and not numbers.)

```
import java.math.BigDecimal;

public final class Monetary2 {

    private final static String FMT =
        "Bought %d items @ $%s; funds remaining $%s%n";

    public static void main (final String args[]) {
        final BigDecimal price = new BigDecimal (".10");
        BigDecimal funds = new BigDecimal ("2.00");
        int items = 0;
        while (funds.compareTo(price)>=0) {
            funds = funds.subtract(price);
            items++;
        }
        System.out.format (FMT, items, price, funds);
    }
}
```

Bought 20 items @ \$0.10; funds remaining \$0.00

java.lang.String

In many cases it is more efficient to use the class `java.lang.StringBuilder` than `java.lang.String`.

`java.lang.String` is a immutable class and `java.lang.StringBuilder` is a mutable class. Immutable classes will cause fewer programming errors, mutable are more efficient to use in some circumstances. This topic will be discussed later.

java.lang.String

int	length()	string length
char	charAt(int i)	ith character
String	substring(int i,int j)	ith - j-1 characters
boolean	equals(Object o)	
int	compareTo(String s)	lexicographic ordering

java.lang.StringBuilder

int	length()	string length
char	charAt(int i)	ith character
void	setCharAt(int i)	set ith character
String	substring(int i, int j)	ith - j-1 characters
boolean	equals(Object o)	<i>pointer equality</i>

Does not implement comparable.

<http://www.javafaq.nu/java-article641.html>

java.lang.StringBuilder

- ▶ A mutable sequence of characters.
- ▶ A class in the package `java.lang`, so no need for an import statement.
- ▶ This class extends the `Object` class. The implication of class inheritance will be discussed later.
- ▶ Has several methods with the same name as those in the `String` class that perform similar operations.

<code>int</code>	<code>length()</code>	string length
<code>char</code>	<code>charAt(int i)</code>	ith character
<code>String</code>	<code>substring(int i,int j)</code>	ith - j-1 characters
<code>boolean</code>	<code>equals(Object o)</code>	

- ▶ Has numerous that change the string: `append`, `insert`, `delete`, `replace`, `reverse`, `setCharAt`, `deleteCharAt`
- ▶ BTW used it implement the '+' operator

```
new StringBuffer().append("String").append(xyz)
```

- ▶ Pitfalls. `equals()` same as `==` and does not implement comparator

```
sb1.toString().equals (sb2.toString())
```

Arrays

- ▶ Create, Triangle, Bool, Copy, Sort

Unicode

(See separate document)

(Next topic expressions, statement.)