Variables, Constants, and Data Types

- Primitive Data Types
- Variables, Initialization, and Assignment
- Constants
- Characters
- Strings
- Reading for this class: L&L, 2.1-2.3, App C

Types of Data

- In Java, you will be dealing mainly – nigh ***exclusively*** – with two types of program data:
- **Primitive** types:
  - The most basic forms that data in a Java program can take
- **Object** types:
  - Conglomerations **of other data types, both primitive and object types**

Primitive Data

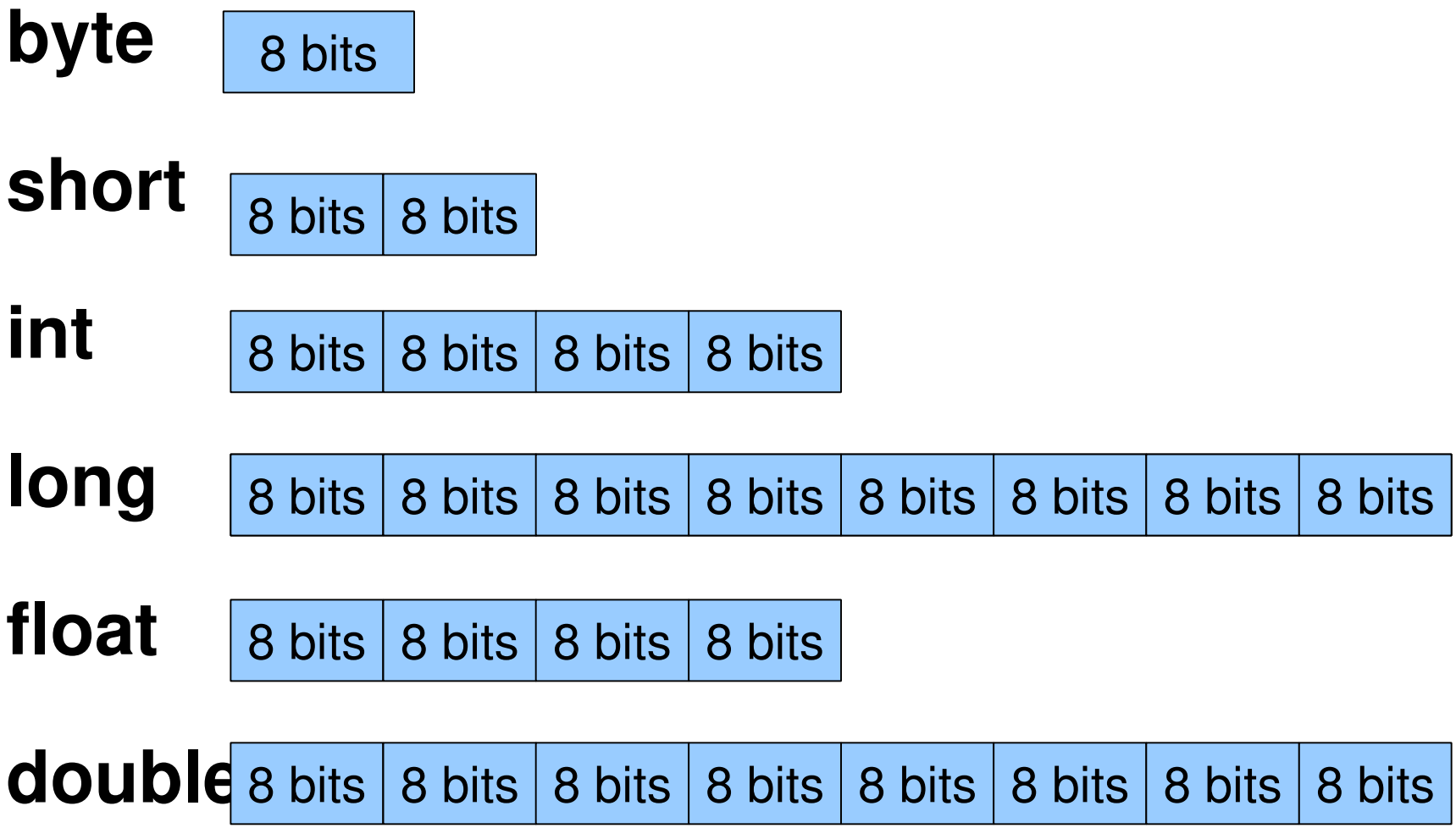
- Java has 8 primitive data types
- Four integer types:
  - `byte`, `short`, `int`, `long`
- Two decimal types:
  - `float`, `double`
- Single characters:
  - `char`
- True/false (or "boolean") values:
  - `boolean`
- For numeric types, we will primarily use the `int` and the `double` types.

Numeric Primitive Data

- The numeric types differ in size and, therefore, the values they can store:

Type	Storage	Min Value	Max Value
`byte`	8 bits	-128	127
`short`	16 bits	-32,768	32,767
`int`	32 bits	-2,147,483,648	2,147,483,647
`long`	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
`float`	32 bits	$+/- 3.4 \times 10^{38}$ with 7 significant digits	
`double`	64 bits	$+/- 1.7 \times 10^{308}$ with 15 significant digits	

Numeric Primitive Data - Visually



Boolean Primitive Data

- A `boolean` value represents a true or false condition
- `true` and `false` are reserved words and the only valid values for a boolean type

```
boolean done = false;
```

- A `boolean` variable can represent any two states such as a light bulb being on or off

```
boolean isOn = true;
```

Variable Declaration

- A *variable* is a name for a location in memory
- A variable must be *declared* by specifying its name and the type of information that it will hold

- Multiple variables of the same type can be created in one declaration:

```
int count, temp, result; boolean done, on;
```

Variable Initialization

- A variable can be initialized (given a value for the first time) at the time of declaration or later

```
int sum = 0;                    OR  int sum;
int base = 32, max = 149;           sum = 0;
```

- When a variable is referenced in a program, its current value is used
- See PianoKeys.java (page 66-67)

```
int keys = 88;
System.out.println("A piano has " + keys + " keys.");
```

- Prints as:

  A piano has 88 keys.

Constants

- A constant is an identifier that is similar to a variable except that it holds the same value during its entire existence
- As the name implies, it is constant, not variable
- In Java, we use the reserved word `final` in the declaration of a constant

```
final int MIN_HEIGHT = 69;
```

OR

```
final int MIN_HEIGHT; MIN_HEIGHT = 69;
```

- Any *subsequent* assignment statement with `MIN_HEIGHT` on the left of the = operator will be flagged as an error

Constants

- Constants are useful for three important reasons
- First, they give meaning to otherwise unclear literal values
  - For example, NUM_STATES is more meaningfult than the literal 50 – what if the country gets a 51st state?
- Second, they facilitate program maintenance
  - If a constant is used in multiple places and you need to change its value later, its value needs to be updated in only one place
  - <u>Rather than having to find and change it in multiple places!</u>
- Third, they formally show that a value should not change, avoiding inadvertent errors by other programmers

Characters

- A `char` variable stores a single character
- In Java, a character takes 2 bytes
- Character literals are delimited by single quotes:

```
'a'   'X'   '7'   '$'   ','   '\n'
```

- Example declarations:

```
char topGrade = 'A';

char terminator = ';', separator = ' ';
```

Character Sets

- A *character set* is an ordered list of characters, with each character corresponding to a unique number
- A `char` variable in Java can store any character from the *Unicode character set*
- The Unicode character set uses sixteen bits per character, allowing for 65,536 (2^16) unique characters
- It is an international character set, containing symbols and characters from many world languages

Characters

- The *ASCII character set* is older and smaller than Unicode, but is still quite popular (in C programs)
- The ASCII characters are a subset of the Unicode character set, including:

uppercase letters	A, B, C, …
lowercase letters	a, b, c, …
punctuation	period, semi-colon, …
digits	0, 1, 2, …
special symbols	&, |, \, …
control characters	carriage return, tab, ...

Value Assignment

- An *assignment statement* gives the variable an actual value in memory
- The equals sign provides this function

```
total = 55;
```

- The expression on the right is <u>evaluated</u> and the result is <u>stored</u> as the value of the variable on the left
- Any value previously stored in `total` is overwritten
- You can only assign a value to a variable that is <u>consistent</u> with the variable's declared type

YES: total = 92; NO: total = false; total = "hello";

- See Geometry.java (page 68)

Variables and Literals

```
int i = 7, j = -8, k = 9;
double d = 4.2;
char c = 'f';
boolean isItOn = true;
String str = “Hello World”;

System.out.println(str + " " + ( i + (j * -1) *
  (2.9 / k)) + c + “oo ” + (isItOn && false) +
  '\n');
```

Object Data

- In addition to the usual primitive data types, we also have object data types, of which there are very many!
- With a **primitive** type, we are dealing with **the actual value** directly. This is because any two primitive values of the same time take up the same space in memory
- However, two different **objects** of the same type may require different amounts of memory.
- Therefore, we interact with an object through a *reference* – in other words, the object's **location in memory**.

Object Data

- **The *reference* itself can come in many forms, such as:**
  - **A variable**

```
System.out.println(s);
```

  - **A literal (rare)**

```
System.out.println("Hello");
```

  - **A method call**

```
System.out.println(s.substring(0,3));
```

  - **An expression**

```
System.out.println("Hello, " + "world!");
```

Object Data

- A non-existent object reference is considered to be **`null`** (one of the Java reserved words)
  - String str1 = "Hello, world!"
  - String str2 = null;
  - str1: [obj. address], str2: null
- Objects are more complex than primitive variables:
  - Made of primitives and other objects
  - Have "features" that you can access in order to carry out tasks or get data
- Remember the distinction! **Do not** try to use primitives as you would objects – or the reverse, except in special situations

Character Strings

- A string of characters can be represented as a *string literal* by putting double quotes around the text:
- Examples:

```
"This is a string literal."      "X"
   "123 Main Street"             "" (empty string)
```

- Note the distinction between a primitive character `'X'`, which holds only one character, and a `String` object, which can hold a sequence of one or more characters
- Every character string is an **object** in Java, **defined by the `String` class**

The `println` Method

- In the `Lincoln` program from Chapter 1, we invoked the `println` method to print a character string
- The `System.out` object represents a destination (the monitor screen) to which we can send output

```
System.out.println ("Whatever you are, be a good one.");
```

object

method name

information provided to the method (parameters)

The `print` Method

The `System.out` object provides another method: `print`

- Like the `println` method, except that it does not start the next line

Therefore anything printed after the `print` method will appear on the same line (unless you ended the previous `print` command with a newline character ('\n')

- See Countdown.java (page 59)

  System.out.print (“Three… ”);

  System.out.print (“Two… ”);

- Prints as:

  Three… Two…

Combining Strings

- To combine (or "concatenate") two strings, use the plus sign

```
"Peanut butter " + "and jelly"
```

- It can also be used to append a number to a string
- A string literal cannot be broken across two lines in a program so we must add (or "concatenate") them
- See Facts.java (page 61)

```
System.out.println("We present the following " +
"facts for your extracurricular edification");
```

NOTE:
No ; here

String Concatenation

- The + operator is also used for arithmetic addition
- The function that it performs depends on the type of the information on which it operates
- If at least one operand is a string, it performs string concatenation
- If both operands are numeric, it adds them

"Hello " + 42 = "Hello 42"

4 + 42 = 46

- The + operator is evaluated left to right, but parentheses can be used to force the order
- See Addition.java (page 62)

```
System.out.println("24 and 45 concatenated: " + 24 + 45);
```

- Prints as:

```
24 and 45 concatenated: 2445
```

String Concatenation

- The + operator is evaluated left to right, but parentheses can be used to force the order
- See Addition.java (page 62)

```
System.out.println("24 and 45 added: " + (24 + 45));
```

Addition is Done first

- Prints as:

```
24 and 45 added: 69
```

Then concatenation is done

Escape Sequences

- What if we want to include the quote character itself?
- The following line would confuse the compiler because it would interpret the two pairs of quotes as two strings and the text between the strings as a syntax error:

```
System.out.println ("I said "Hello" to you.");
```

- An *escape sequence* is a series of characters that represents a special character
- Escape sequences begin with a backslash character (\)

```
System.out.println ("I said \"Hello\" to you.");
```

Escape Sequences

- Some Java Escape Sequences

Escape Sequence	Meaning
`\b`	backspace
`\t`	tab
`\n`	newline
`\r`	carriage return
`\"`	double quote
`\'`	single quote
`\\`	backslash

- See Roses.java (page 64)

```
System.out.println("Roses are red,\n\tViolets are blue,\n" +
```

- Prints as:

```
Roses are red,
      Violets are blue,
```

Escape Sequences

- To put a specified Unicode character into a string using its code value, use the escape sequence: `\uhhhh` where `hhhh` are the hexadecimal digits for the Unicode value
- Example: Create a string with a temperature value and the degree symbol:

```
double temp = 98.6;
System.out.println(
  "Body temperature is " + temp + "
 \u00b0F.");
```

- Prints as:

```
Body temperature is 98.6 °F.
```

Methods of the String class

- String is a class and classes can have methods.
- Use the Sun website link to find definitions of the methods for each standard library class
- The classes are listed in alphabetical order
- The String class has methods that can be used to find out the characteristics of a String object such as its length:

```
System.out.println("Hello".length());
```

- Prints the number 5 (for 5 characters in length)