# Primitive Data Types

15-110 Summer 2010

Margaret Reid-Miller

# Data Types

- Data stored in memory is a string of bits (0 or 1).
- What does `1000010` mean?

  66?

  'B'?

  9.2E-44?
- How the computer interprets the string of bits depends on the context.
- In Java, we must make the context explicit by specifying the ***type*** of the data.

# Primitive Data Types

- Java has two categories of data:
  - **primitive data** (e.g., number, character)
  - **object data** (programmer created types)

- There are 8 primitive data types:

```
byte, short, int, long, float, double,
char, boolean
```

- Primitive data are only single values; they have no special capabilities.

# Common Primitive Types

| Type | Description | Example of Literals |
|---|---|---|
| `int` | integers (whole numbers) | `42, 60634, -8, 0` |
| `double` | real numbers | `0.039, -10.2, 4.2E+72` |
| `char` | single characters | `'a', 'B', '&', '6'` |
| `boolean` | logical values | `true, false` |

# Numbers

| Type | Storage | Range of Values |
|---|---|---|
| `byte` | 8 bits | -128 to 127 |
| `short` | 16 bits | -32,768 to 32,727 |
| `int` | 32 bits | -2,147,483,648 to 2,147,483,647 |
| `long` | 64 bits | $-9 \times 10^{18}$ to $9 \times 10^{18}$ |
| `float` | 32 bits | $\pm 10^{-45}$ to $\pm 10^{38}$, 7 significant digits |
| `double` | 64 bits | $\pm 10^{-324}$ to $\pm 10^{308}$, 15 significant digits |

# Variables

- A ***variable*** is a *name* for a location in memory used to store a data value.
- We use variables to save and restore values or the results of calculations.
- The programmer has to tell Java what ***type*** of data will be store in the variable's memory location. Its type **cannot** change.
- During the program execution the **data** saved in the memory location **can** change; hence the term "variable".

# Variable Declaration

- Before you can use a variable, you must ***declare*** its type and name.
- You can declare a variable **only once** in a method.
- Examples:

```
int numDimes;
double length;
char courseSection;
boolean done;
String lastName;
```

# Declaring Variables

- Declaring a variable instructs the compiler to set aside a portion of memory large enough to hold data of that type.

```
int  count;
double length;
```


- No value has be put in memory yet. That is, the variable is ***undefined***.

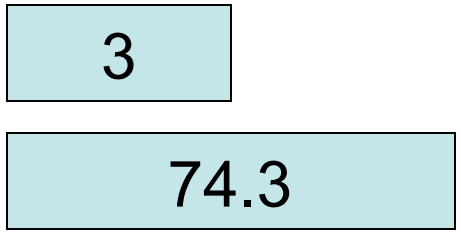
# Assignment Statements

- An ***assignment statement*** stores a value into a variable's memory location:

```
<variable> = <expression>;
```

- An ***expression*** is anything that has a value: a literal value, a variable, or a more complex calculation.
- The result of the expression is *assigned* to the variable.

```
count = 3;
length = 72.3 + 2.0;
```




- The first assignment to a variable ***initializes*** it.

# Re-Assigning Variables

- A variable must be **declared exactly once**.
- A variable can be assigned and re-assigned values many times after it is declared.

Example:

```
int x;
x = 4;
System.out.println(x);  // prints  4
x = x + 1;
System.out.println(x);  // prints  5
```

# Declaration/Initialization

- Variables can be declared and initialized in one statement:

Examples:

```
int numDimes = 4;
double length = 52.3;
char courseSection = 'J';
boolean done = true;
String lastName = "Reid-Miller";
int count = 3 + 2;
```

# Expressions

- An ***expression*** is anything that result in a value.
- It must have a type. Why?

  Example: (2 + 3) * 4

## Arithmetic operators:

| Operator | Meaning | Example | Result |
|---|---|---|---|
| + | addition | 1 + 3 | 4 |
| - | subtraction | 12 - 4 | 8 |
| * | multiplication | 3 * 4 | 12 |
| / | division | 2.2 / 1.1 | 2.0 |
| % | modulo (remainder) | 14 % 4 | 2 |

# Division and Modulo

```
int a = 40;          double x = 40.0;
int b = 6;           double y = 6.0;
int c;               double z;
```

| | | | |
|---|---|---|---|
| `c = a / b;` | `6` | `c = a % b;` | `4` |
| `z = x / y;` | `6.66666667` | `c = b % a;` | `6` |
| `c = b / a;` | `0` | `c = 0 % a;` | `0` |
| `z = y / x;` | `0.15` | `c = b % 0;` | `error` |
| `c = 0 / a;` | `6` | | |
| `c = a / 0;` | `error` | | |

# Operator Precedence

- The operators `*`, `/`, `%` are evaluated before the operators `+`, `-` because `*`, `/`, `%` have higher ***precedence*** than `+`, `-`.

  Example: `2 + 4 * 5`

  `2 + 20`

  `22`

- To change the order use parentheses:

  Example: `(2 + 4) * 5` evaluates to _____

# Evaluating expressions

- When an expression contains more than one operator with the same level of precedence, they are evaluated from left to right.
  - `2 + 2 + 3 - 1` is `(((2 + 2) + 3) - 1)` which is `6`
  - `2 * 4 % 5` is `((2 * 4) % 5)` which is 3

- `2 * 3 - 2 + 7 / 4`

  `6 - 2 + 7 / 4`

  `6 - 2 + 1`

  `4 + 1`

  `5`

# Other operators

- **Assignment operators**: =, +=, -=, *=, /=, %=

Example:

- Shortcut for `x = x + 2;` is `x += 2;`
  ("add 2 to x")
- Shortcut for `y = y * 3;` is `y *= 3;`
  ("multiply y by 3")

- **Increment / Decrement** operators: ++, --
  - Shortcut for `x = x + 1;` is `x++;` ("increment x")
  - Shortcut for `y = y - 1;` is `y--;` ("decrement y")

# Data Conversion

- ***Widening conversions*** convert data to another type that has the same or more bits of storage. *E.g.*,
  - `short` to `int`, `long` (safe)
  - `int` to `long` (safe)
  - `int` to `float`, `double` (magnitude the same but can lose precision)

- ***Narrowing conversions*** convert data to another type that has the fewer bits of storage and/or can lose information. *E.g.*,
  - `double` or `float` to any integer type
  - `double` to `float`

# Mixing Types

- When a Java operator is applied to operands of different types, Java does a widening conversion automatically, known as a ***promotion***.
- Example:
  - `2.2 * 2` evaluates to `4.4`
  - `1.0 / 2` evaluates to `0.5`
  - `double x = 2;` assigns `2.0` to `x`
  - `"count = " + 4` evaluates to `"count = 4"`

string concatenation

# Mixing Types

- Conversions are done on one operator at a time in the order the operators are evaluated.

```
3 / 2 * 3.0 + 8 / 3      5.0
```

```
2.0 * 4 / 5 + 6 / 4.0    3.2
```

# Mixing Types

- String concatenation has the same precedence as + – and is evaluated left to right.

| | |
|---|---|
| `1 + "x" + 4` | `"1x4"` |
| `"2+3=" + 2 + 3` | `"2+3=23"` |
| `1 + 2 + "3"` | `"33"` |
| `"2*3=" + 2 * 3` | `"2*3=6"` |
| `4 - 1 + "x"` | `"3x"` |
| `"x" + 4 - 1` | error |

# Type Casting

- ***Type casting*** tells Java to convert one type to another.

  Uses:

  - Convert an `int` to a `double` to force floating-point division.
  - Truncate a `double` to an `int`.

  Examples:

  - `double average = (double) 12  / 5`
  - `int feet = (int) (28.3 / 12.0)`

# Type casting

- Because type casting has high precedence, it casts the operand immediately to its right only.

Example:

```
double s = (double) 2 + 3 / 2;        3.0
double s2 = (double) (2 + 3) / 2;     2.5
double average = (double) 22 / 4;     5.5
double average2 = 22 / (double) 4;    5.5
double wrong = (double) (22 / 4);     5.0
```

# `char` data type

- A variable of type `char` holds exactly **one** (Unicode) character/symbol.
- Every character has a corresponding integer value.
- The digit characters `'0'`… `'9'` have consecutive integer values, as do the letters `'A'`… `'Z'` and `'a'`… `'z'`. We can use this ordering to sort alphabetically.
- Conversions:

```
String letter = "" + 'M';          // evaluates to "M"
int aAsInt = 'a';                  // evaluates to 97
'a' + 2;                           // evaluates to 99
char c = (char)('a' + 2); // evaluates to 'c'
```

# Operator Precedence

| Operator type | Operator | Associates |
|---|---|---|
| grouping | (*expression*) | Left to right |
| unary | ++, --, +, - | Right to left |
| cast | (*type*) | Right to left |
| multiplicative | *, /, % | Left to right |
| additive | +, – | Left to right |
| assignment | =, +=, –=, *=, /=, %= | Right to left |