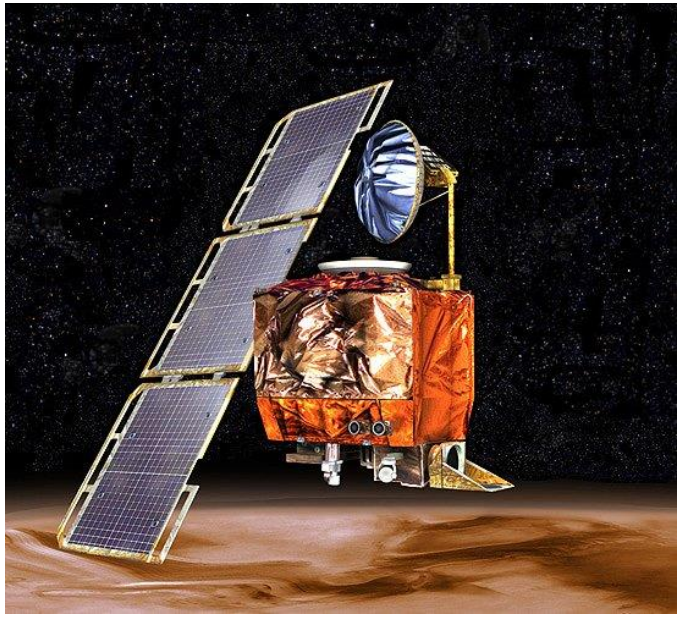
Java Class 4



NASA Mars Climate Orbiter and Polar Lander

The failure of the two-spacecraft Mars expeditions in 1998 and 1999 is attributed to errors in the software systems, which used pounds-force units for the guidance systems while programming the spacecraft to expect to metric data.

Cost of error: $327.6M.

Primitive Data

There are eight primitive data types in Java

variable → primitive value

Four of them represent integers:

- `byte, short, int, long` **`1,-5,1024`**

Two of them represent real (floating point) numbers:

- `float, double` **`2.14159, -6., 6.0`**

One of them represents a single character:

- `Char` **`'A', ' ', ':'`**

And one of them represents boolean values:

- `Boolean` **`true, false`**

Numeric Primitive Types

The difference between the numeric primitive types is their size and 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	

Usually we use `int` for integers and `double` for real numbers

Characters

A ***char*** variable stores a single character

Character literals are delimited by single quotes:

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

Example declarations:

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

Note the difference between a primitive character variable, which holds only one character, and a String object, which can hold multiple characters

Boolean

A ***boolean*** value represents a true or false condition

The reserved words true and false are the only valid values for a boolean type

```
boolean done = false;
```

A ***boolean*** variable can also be used to represent any two states, such as a light bulb being on or off

Expressions

An ***expression*** is a combination of one or more operators and operands

Arithmetic expressions compute numeric results and make use of the arithmetic operators:

Addition	**+**
Subtraction	**-**
Multiplication	*****
Division	**/**
Remainder	**%**

- If either or both operands are floating point values, then the result is a floating point value

Division and Remainder

If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)

`14 / 3` **equals** **4**

`8 / 12` **equals** **0**

- If either or both operands to the division operator (/) are floating point values, the result is a floating point value (the fractional part is kept)

`14.0 / 3` **equals** **4.67**

`8 / 12.0` **equals** **0.67**

- The remainder operator (%) returns the remainder after dividing the first operand by the second

`14 % 3` **equals** **2**

`8 % 12` **equals** **8**

Quick Check

What are the results of the following expressions?

```
    12 / 2     =  6
12.0 / 2.0     =  6.0
    10 / 4     =  2
  10 / 4.0     =  2.5
    4 / 10     =  0
  4.0 / 10     =  0.4
    12 % 3     =  0
    10 % 3     =  1
    3 % 10     =  3
```

Operator Precedence

Operators can be combined into larger expressions

```
result  =  total + count / max - offset;
```

Operators have a well-defined precedence which determines the order in which they are evaluated

Multiplication, division, and remainder are evaluated before addition, subtraction, and string concatenation

Arithmetic operators with the same precedence are evaluated from left to right, but parentheses can be used to force the evaluation order

Quick Check

In what order are the operators evaluated in the following expressions?

```
a + b + c + d + e
```

1 2 3 4

```
a + b * c - d / e
```

3 1 4 2

```
a / (b + c) - d % e
```

2 1 4 3

```
a / (b * (c + (d - e)))
```

4 3 2 1

Assignment Revisited

The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of `count`

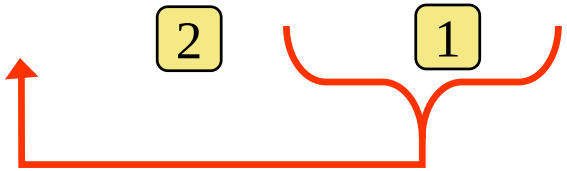
```
count  =  count + 1;
```

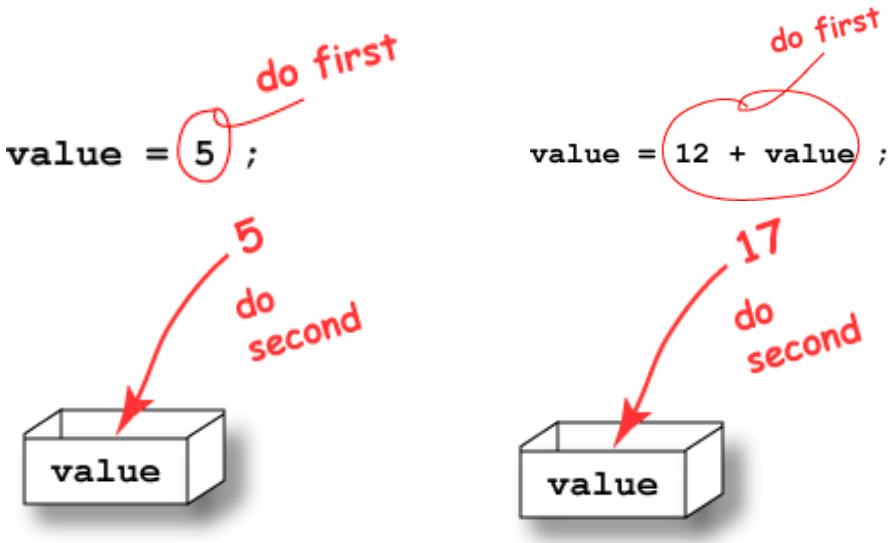
Then the result is stored back into `count` (overwriting the original value)

Assignment Revisited

The right and left hand sides of an assignment statement can contain the same variable

```
value = 5;
value = value + 12;
```





- First, 12 is added to the original value (5) of variable **value**
- Then the result (17) is stored back into **value** (overwriting the original value)

Assignment Revisited

The assignment operator has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

```
answer  =  sum / 4 + MAX * lowest;
```

4 1 3 2

Then the result is stored in the variable on the left hand side

```
//*******************************************************************
//  TempConverter.java       Author: Lewis/Loftus
//
//  Demonstrates the use of primitive data types and arithmetic
//  expressions.
//*******************************************************************

public class TempConverter
{
  //-----------------------------------------------------------------
  //  Computes the Fahrenheit equivalent of a specific Celsius
  //  value using the formula F = (9/5)C + 32.
  //-----------------------------------------------------------------
  public static void main (String[] args)
  {
    final int BASE = 32;
    final double CONVERSION_FACTOR = 9.0 / 5.0;

    double fahrenheitTemp;
    int celsiusTemp = 24;  // value to convert

    fahrenheitTemp = celsiusTemp * CONVERSION_FACTOR + BASE;

    System.out.println ("Celsius Temperature: " + celsiusTemp);
    System.out.println ("Fahrenheit Equivalent: " + fahrenheitTemp);
  }
}
```

```
run:
Celsius Temperature: 24
Fahrenheit Equivalent: 75.2
BUILD SUCCESSFUL (total time: 0
seconds)
```

Increment and Decrement

The ***increment*** (++) and ***decrement*** (--) operators use only one operand

The statement:

```
count++;
```

is functionally equivalent to:

```
count = count + 1;
```

Increment and Decrement

The increment and decrement operators can be applied in:

- *postfix form*:

```
count++;
count--;
```

- or *prefix form*:

```
++count;
--count;
```

Increment and Decrement

When used as part of a larger expression, the two forms can have different effects. Assume the value of count is 12 in both cases.

- ***postfix form*:**

```
total = count++;
```

  - *count has value 13 and total has a value 12*

    equivalent to:

```
total = count;
count = count + 1;
```

- ***prefix form*:**

```
total = ++count;
```

  - *count has value 13 and total has a value 13*

    equivalent to:

```
count = count + 1;
total = count;
```

Assignment Operators

Often we perform an operation on a variable, and then store the result back into that variable

Java provides ***assignment operators*** to simplify that process

For example, the statement

```
num += count;
```

is equivalent to

```
num = num + count;
```

Quick Check

If an integer variable `weight` holds the value 100, what are the values of `weight` and `total` after the following statements are executed?

```
weight -= 17;
```

weight = 83

```
total = weight++;
```

total = 83 , weight = 84

```
total = --weight;
```

weight = 83, total = 83

Data Conversion

Sometimes it is convenient to convert data from one type to another

For example, in a particular situation we may want to treat an integer as a floating point value

These conversions do not change the type of a variable or the value that's stored in it – they only convert a value as part of a computation

Data Conversion

Widening conversions are safest because they tend to go from a small data type to a larger one (such as a `short` to an `int`)

Narrowing conversions can lose information because they tend to go from a large data type to a smaller one (such as an `int` to a `short`)

In Java, data conversions can occur in three ways:

- assignment conversion
- promotion
- casting

Data Conversion

Widening Conversions

From	To
byte	short, int, long, float, or double
short	int, long, float, or double
char	int, long, float, or double
int	long, float, or double
long	float or double
float	double

Narrowing Conversions

From	To
byte	char
short	byte or char
char	byte or short
int	byte, short, or char
long	byte, short, char, or int
float	byte, short, char, int, or long
double	byte, short, char, int, long, or float

Assignment Conversion

Assignment conversion occurs when a value of one type is assigned to a variable of another

Example:

```
int dollars = 20;
double money = dollars;
```

happy compiler

Only widening conversions can happen via assignment

Note that the value or type of `dollars` did not change

```
double dollars = 20.;

int money = dollars;
```

unhappy compiler

Promotion

Promotion happens automatically when operators in expressions convert their operands

Example:

```
int count = 12;
double sum = 490.27;
result = sum / count;
```

The value of `count` is converted to a floating point value to perform the division calculation

Casting

Casting is the most powerful, and dangerous, technique for conversion

Both widening and narrowing conversions can be accomplished by explicitly casting a value

To cast, the type is put in parentheses in front of the value being converted

```
int total = 50;
float result = (float) total / 6;
```

Without the cast, the fractional part of the answer would be lost

Interactive Programs

Programs generally need input on which to operate

The **`Scanner class`** provides convenient methods for reading input values of various types

A **`Scanner object`** can be set up to read input from various sources, including the user typing values on the keyboard

Keyboard input is represented by the **`System.in`** object

Reading Input

The `System` and `String` classes are part of the Java.lang package (whose classes can be thought of as basic extensions to the Java language). These classes can be invoked without explicitly importing them, so we can just use

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

The `Scanner` class is part of the `java.util` class library and must be imported into a program that is using it by using the statement

```
import java.util.Scanner;
```

The `import` statement precedes the class statement

Reading Input

The following line creates a `Scanner` object that reads from the keyboard:

```
Scanner scan = new Scanner (System.in);
```

The `new` operator creates the `Scanner` object by calling a special method called a ***constructor***

Once created, the `Scanner` object can be used to invoke various input methods, such as:

```
answer = scan.nextLine();
```

The `nextLine` method reads all of the input until the end of the line is found

Reading Input

Delimiters or white space characters (space, tabs, new line) are used to separate the elements of the input – called tokens

The **`next`** method of the Scanner class reads the next token as a string

If the input consists of a series of words separated by spaces, a call to **`next`** will return the next word

Methods such as **`nextInt`** and **`nextDouble`** read data of particular types (int and double)

See page 88 for more methods in the Scanner class

```java
//*******************************************************************
//  Echo.java        Author: Lewis/Loftus
//
//  Demonstrates the use of the nextLine method of the Scanner class
//  to read a string from the user.
//*******************************************************************

import java.util.Scanner;

public class Echo
{
   //-----------------------------------------------------------------
   //  Reads a character string from the user and prints it.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      String message;
      Scanner scan = new Scanner (System.in);

      System.out.println ("Enter a line of text:");

      message = scan.nextLine();

      System.out.println ("You entered: \"" + message + "\"");
   }
}
```

Sample Run

```
Enter a line of text:
You want fries with that?
You entered: "You want fries with that?"
```

```
//********************************************************************
//  Echo
//
//  Demonstrates the use of the Scanner class to read a character string
//  to
//********************************************************************

import java.util.Scanner;

public class Echo
{
   //-----------------------------------------------------------------
   //  Reads a character string from the user and prints it.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      String message;
      Scanner scan = new Scanner (System.in);

      System.out.println ("Enter a line of text:");

      message = scan.nextLine();

      System.out.println ("You entered: \"" + message + "\"");
   }
}
```

Group Exercises

Ex: 2.1

Ex: 2.7

Ex: 2.8

Ex: 2.9

Ex: 2.10

Ex: 2.11

Ex: 2.12

Assignment for Class 5

Review TempConversion, Echo

Read Chapter 3.1, 3.2, 3.3, 3.6