Sheridan

PROG 10082

Object Oriented Programming 1

Recap: Structure of a Java Program

- Class name
- Main method
- Statements
- Statement terminator
- Reserved words
- Comments
- Blocks

Class Name

Every Java program must have at least one class. Each class has a name. By convention,

- class names start with an uppercase letter. In this example, the class name is Welcome.
- must be public to be available to run;
- must be declared as a class;
- **must have the same name as the file**

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Statement

A statement represents an action or a sequence of actions.

System.out.println: The java language contains API's (Application Programme Interface). These are libraries of useful programmes. To print on the screen you use the println programme which is found in the out directory which is found in the System directory of system programmes.

System.out.println("Welcome to Java!"):

is a statement to display the greeting "Welcome to Java!“

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

- For simple programme development there are two parts to the **Java Development Kit (jdk);**

- **javac - compiler and java - run-time programme**
  Each is run from the command line.

- **<u>javac:</u>** compiles the xxx.java file into the xxx.class file. The class file will contain bytecode. This is a common first stage compilation.

- **java:** This runs the Java Virtual Machine (jvm) which loads the xxx.class file and then tests for validity of the code and for security violations. It will then convert the code to native code and run it.

Special Symbols

Character	Name	Description
{}	Opening and closing braces	Denotes a block to enclose statements.
()	Opening and closing parentheses	Used with methods.
[]	Opening and closing brackets	Denotes an array.
//	Double slashes	Precedes a comment line.
" "	Opening and closing quotation marks	Enclosing a string (i.e., sequence of characters).
;	Semicolon	Marks the end of a statement.

Programming Style and Documentation

- Appropriate Comments
- Naming Conventions
- Proper Indentation and Spacing Lines
- Block Styles

Why Standards?

- Professional! Industry Standard!
  - Easier to read
  - Easier to maintain
  - Easier to debug and avoid errors
  - Makes code more robust, reliable
- See chapter 1.10 for some guidelines
- Also see Expanded Guidelines Supplement

Documentation

- Code should be well documented
- Describe what your statements are doing
- NOTE: Javadoc comments
  - Covered in term 2.
- You should also add documentation at the top of your program
  - See standards, also see assignment standards

Appropriate Comments

Include a summary at the beginning of the program to explain what the program does, its key features, its supporting data structures, and any unique techniques it uses.

Include your name, class section, instructor, date, and a brief description at the beginning of the program.

Naming Conventions

- Choose meaningful and descriptive names.

  - Name should tell you what the identifier

    is/contains/does/etc

- Class names:
  - Capitalize the first letter of each word in the name. For example, the class name `ComputeExpression`.
- Classes start with upper-case letter
  - Everything else starts with lower-case letter
  - Constants are all upper-case letters

Proper Indentation and Spacing

- Indentation
  - Indent in every block
    - 4 spaces per indent
  - Multi-Line Statements
    - Indent the 2nd and subsequent lines once
- Spacing
  - Use blank line to separate segments of the code.
  - Between related groups of code
  - Around operators, except unary operators, brackets

Block Styles

Use end-of-line style for braces.

Next-line style

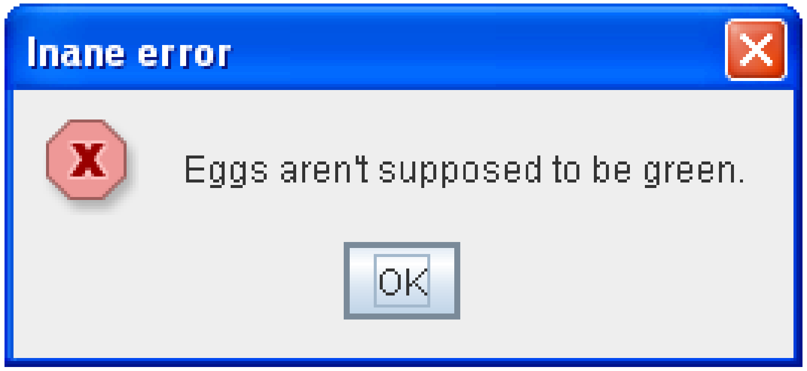
```
public class Test
{
  public static void main(String[] args)
  {
    System.out.println("Block Styles");
  }
}
```

End-of-line style

```
public class Test {
  public static void main(String[] args) {
    System.out.println("Block Styles");
  }
}
```

Message Boxes

- Mini dialog box
  - Gives simple information to user
  - Contains OK button
  - Contains an informative icon
  - Contains a title


Inane error

Eggs aren't supposed to be green.

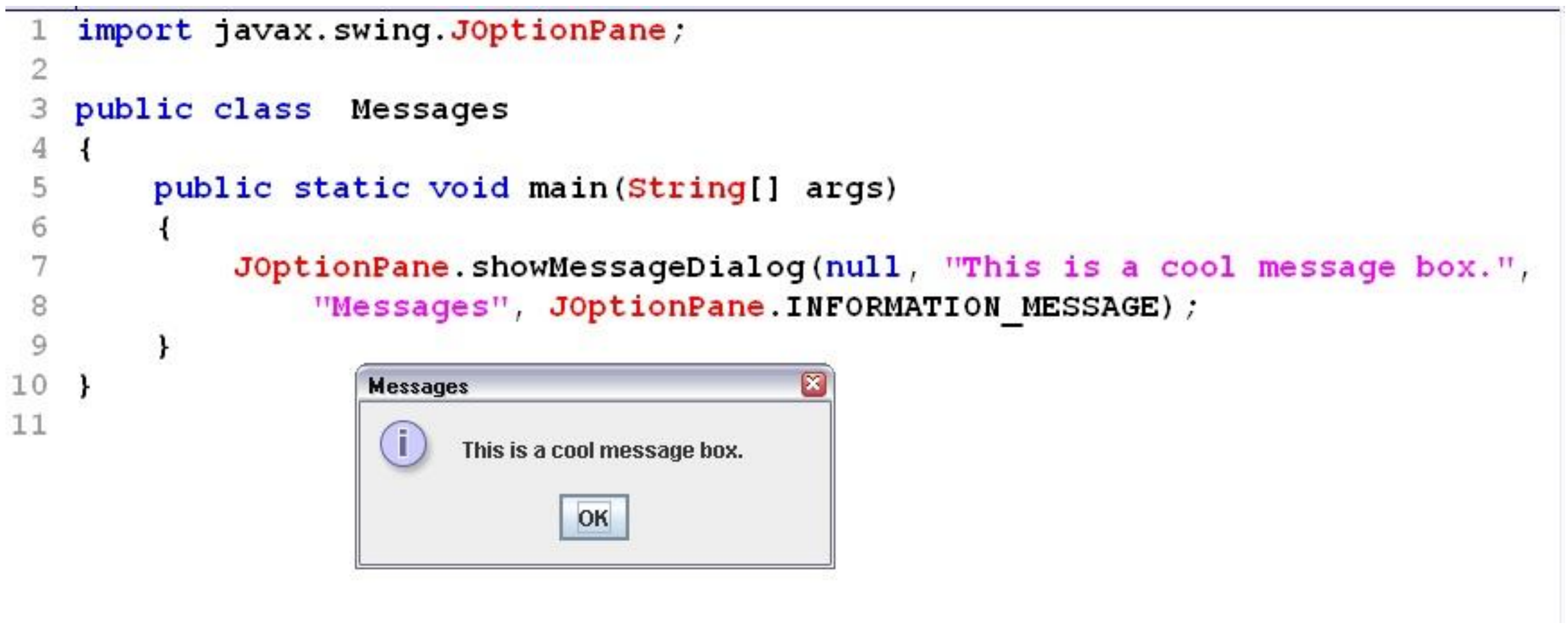
JOptionPane

- JOptionPane
  - A class that contains methods for showing dialogs
  - JOptionPane.showMessageDialog() arguments:
    - null (parent object)
    - The message
    - The title in the title bar
    - The icon style

JOptionPane Icons

Type of Icon	Constant Name	Java Look & Feel	Windows Look & Feel
Question	QUESTION_MESSAGE	❓	❓
Information	INFORMATION_MESSAGE	ℹ	ℹ
Warning	WARNING_MESSAGE	⚠	⚠
Error	ERROR_MESSAGE	❌	❌

JOptionPane Code



```java
import javax.swing.JOptionPane;

public class  Messages
{
    public static void main(String[] args)
    {
        JOptionPane.showMessageDialog(null, "This is a cool message box.",
            "Messages", JOptionPane.INFORMATION_MESSAGE);
    }
}

```

Messages

This is a cool message box.

Packages

- JOptionPane gives you an error when you compile
- Java doesn’t know where it is!
- Most classes are in a different **package**
  - Classes like System are in java.lang
  - All programs have access to this
- To use another package, you need to **import** it

Import a Package

```
import javax.swing.*;
```

Or

```
import javax.swing.JOptionPane;
```

- Either is acceptable
  - .* will NOT import everything!
  - It only tells Java where to look
- Must be first statement in your code (for now)

Exercise- 1

1. Reformat the following code segments by hand so that they follow proper standards:

a)

```
public class BadCode { public static void main(String[]
args){ System.out.println("Flowers for Algernon");}}
```

b)

```
public class
BadCode { public static void main(String[]
args){ System.out.print("6/4*2 is ");System.
out.println(6/4*2);}}
```

?

2. Add documentation to each of the corrected code segments above.

Identifiers

- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).
- An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
- An identifier cannot be a reserved word. (See Appendix A, "Java Keywords," for a list of reserved words).
- An identifier cannot be `true`, `false`, or `null`.
- An identifier can be of any length.

Numeric Data Types

There are two categories of numeric types in Java:

integers and **floating point** numbers

- Integers are what you might commonly know as "whole numbers".
- Integers can be positive or negative, but they have no fractional or decimal portion.
- Integers are often used for counting things because we generally count with whole numbers (e.g. 1, 2, 3, 4...).
- Integers should only be used for data when you know there is no way for it to contain a fractional or decimal portion.
- For example, you wouldn't use an integer for the price of a product, even if the price were ten dollars (10 is an integer); if the price went up by 50 cents, you'd have a price of 10.50 (10.5 can't be represented as an integer value).

- Floating point numbers may have a fractional or decimal portion, so sometimes they are referred to as "decimal numbers".
- In programming we find this confusing, because when referring to numbers, the term "decimal" refers to the number system.
- The number system we use in every-day life is the decimal number system (or "base 10" number system).
- In technology, you might also use the binary number system, hexadecimal number system, or octal number system.
- Therefore, when referring to data that can (or could) contain decimal values, we prefer to call it a floating-point number, and not a decimal number.

- Precision is a measure of how many digits you include in a number.

For example, 3.14159 is less precise than 3.141592653589793.

-There are two kinds of floating point numbers in Java:

Single-precision floating-point (**float**), and

double-precision floating point (**double**).

Declaring Variables

```
int x;            // Declare x to be an
                  // integer variable;
double radius;    // Declare radius to
                  // be a double variable;
char a;           // Declare a to be a
                  // character variable;
```

byte, **short**, **int** and **long** `//used for storing whole numbers.`

boolean // `used for variables that holds either true or false.`

Integer Literal

Integer literals are assigned to the variables of data type byte, short, int and long.

```
byte b = 100;
 short s = 200;
int num = 13313131;
long l = 928389283L;
```

Float Literals

Used for data type float and double.

```
double num1 = 22.4;
float num2 = 22.4f;
```

Note: Always suffix float value with the “f” else compiler will consider it as double.

Char and String Literal

Used for char and String type.

```
char ch = 'Z’;
String str = "BeginnersBook";
```

byte:

This can hold whole number between -128 and 127. Mostly used to save memory and when you are certain that the numbers would be in the limit specified by byte data type.

Default size of this data type: 1 byte.

Default value: 0

Example:

```
class JavaExample {
public static void main(String[] args){
         byte num;
         num = 113;
         System.out.println(num);
  }
}
```

Output:

```
113
```

short:

This is greater than byte in terms of size and less than integer. Its range is -32,768 to 32767.
Default size of this data type: 2 byte

```
short num = 45678;
```

int:

Used when short is not large enough to hold the number, it has a wider range:
-2,147,483,648 to 2,147,483,647

Default size: 4 byte
Default value: 0

Example:

```
    class JavaExample {
    public static void main(String[] args) {
    short num;
    num = 150;
    System.out.println(num);
     }
    }
```

Output:

```
150
```

The byte data type couldn't hold the value 150 but a short data type can because it has a wider range.

long:

Used when int is not large enough to hold the value, it has wider range than int data type,

ranging from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

size: 8 bytes
Default value: 0

Example:

```
class JavaExample {
public static void main(String[] args) {
 long num = -12332252626L; System.out.println(num);
 }
}
```

Output:

```
-12332252626
```

double:

Sufficient for holding 15 decimal digits
size: 8 bytes

Example:

```
class JavaExample {
public static void main(String[] args) {
double num = -42937737.9d;
System.out.println(num);
 }
}
```

Output:

```
-4.29377379E7
```

float:

Sufficient for holding 6 to 7 decimal digits
size: 4 bytes

```
class JavaExample {
public static void main(String[] args)
{
float num = 19.98f;
System.out.println(num);
 }
}
```

Output:

```
19.98
```

boolean:

holds either true of false.

```
class JavaExample {
public static void main(String[] args) {
boolean b = false;
System.out.println(b);
 }
}
```

Output:

```
false
```

char:
holds characters.

size: 2 bytes

```
class JavaExample {
 public static void main(String[] args) {
  char ch = 'Z’;
  System.out.println(ch);
  }
}
```

Output:
```
Z
```

Numerical Data Types

Name	Range	Storage Size
`byte`	-2^7 to $2^7 - 1$ (-128 to 127)	8-bit signed
`short`	-2^{15} to $2^{15} - 1$ (-32768 to 32767)	16-bit signed
`int`	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed
`long`	-2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
`float`	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
`double`	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

Variables

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
System.out.println("The area is “ +
  area + " for radius "+radius);

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
System.out.println("The area is “ +
  area + " for radius "+radius);
```

Assignment Statements

```
x = 1;              // Assign 1 to x;

radius = 1.0;       // Assign 1.0 to radius;

a = 'A';            // Assign 'A' to a;
```

Declaring and Initializing in One Step

- **`int x = 1;`**
- **`double d = 1.4;`**

Naming Conventions

- Choose meaningful and descriptive names.
- Variables and method names:
  - Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name. For example, the variables `radius` and `area`, and the method `computeArea`.

Naming Conventions, cont.

- Class names:
  - Capitalize the first letter of each word in the name. For example, the class name `ComputeArea`.
- Constants:
  - Capitalize all letters in constants, and use underscores to connect words. For example, the constant `PI and` MAX_VALUE

Introducing Programming with an Example

Listing 2.1 Computing the Area of a Circle

This program computes the area of the circle.

ComputeArea

Run

Note: Clicking the green button displays the source code with interactive animation. You can also run the code in a browser. Internet connection is needed for this button.

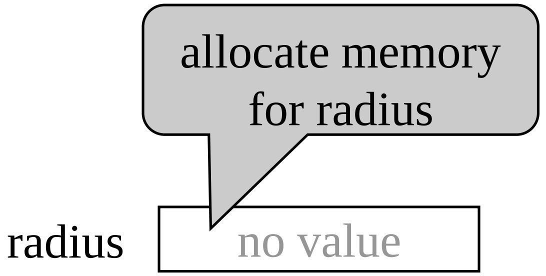
Note: Clicking the blue button runs the code from Windows. If you cannot run the buttons, see

IMPORTANT NOTE: If you cannot run the buttons, see www.cs.armstrong.edu/liang/javaslidenote.doc.

animation

Trace a Program Execution

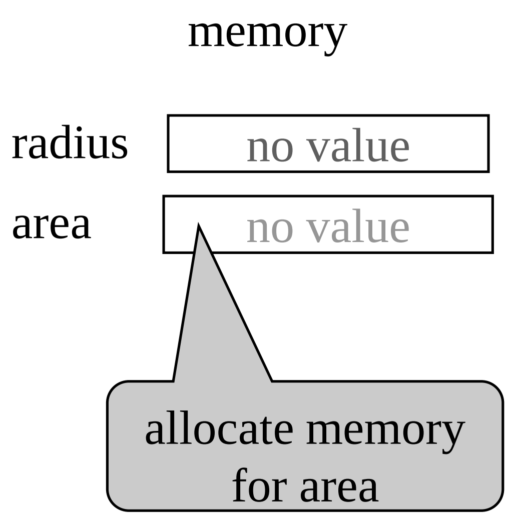
```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



animation

Trace a Program Execution

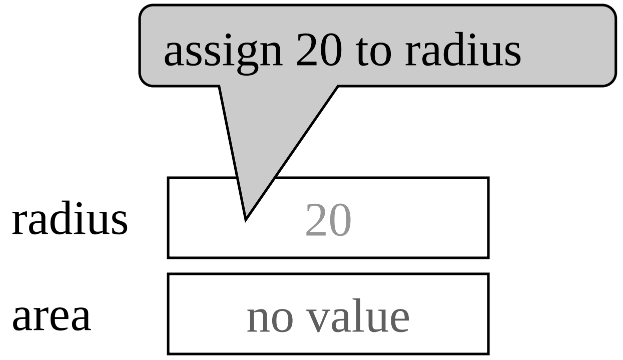
```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



animation

Trace a Program Execution

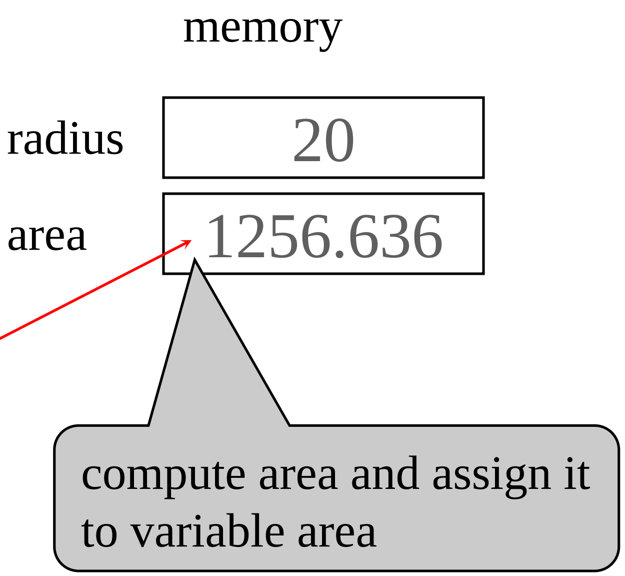
```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



animation

Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



animation

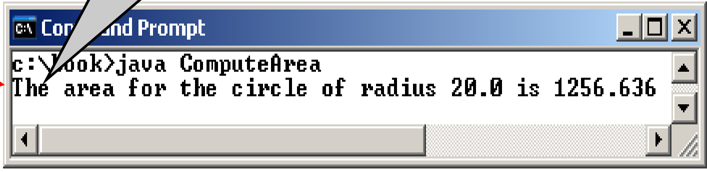
Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```

memory

radius	20
area	1256.636

print a message to the console



```
c:\book>java ComputeArea
The area for the circle of radius 20.0 is 1256.636
```

Numeric Operators

Name	**Meaning**	**Example**	**Result**
+	Addition	34 + 1	35
-	Subtraction	34.0 – 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

Dividing integers

- If you divide 2 integers, the result is an integer, and the floating point gets truncated:
  - 3/2 = 1 8/3 = 2
- To get a floating-point result, one of the numbers has to be a floating point
  - 3.0/2 = 1.5   8.0/3 = 2.667

Dividing integers

- So when dividing 2 integers, result is an integer
- It performs whole number division
  - 5/2 = 2 10/5 = 2 9/2 = 4
- To get the remainder for the division you use the % operator
  - 5%2 = 1 10%5 = 0 9%2 = 1

Integer Division

+, -, *, /, and %

5 / 2 yields an integer 2.

5.0 / 2 yields a double value 2.5

5 % 2 yields 1 (the remainder of the division)

Modulus (%) - This operator performs a division operation but yields the remainder instead of the quotient.

Example:

- 45 % 6 = 3 (6 goes into 45 seven times; 7 * 6 is 42. This leaves a remainder of 3)
- In a modulus expression, the left-hand operand is called the dividend and the right-hand operand is called the divisor. To find the result of the expression, you determine what's left over (the remainder) after dividing the divisor into the dividend. More examples:
- 17 % 4 = 1 (4 goes into 17 4 times, leaving a remainder of 1)
   45 % 5 = 0 (5 goes into 45 exactly 9 times, leaving no remainder)
   9 % 5 = 4 (5 goes into 9 once, leaving a remainder of 4)
   5 % 9 = 5 (9 goes into 5 zero times, leaving a remainder of 5)

- Modulus works with negatives. If the dividend (the # to the left of the % symbol) is negative, the result is negative. Otherwise, the result is positive:

  -17 % 4 = -1
  -9 % -5 = -4
  11 % -3 = 2

Remainder is very useful in programming.

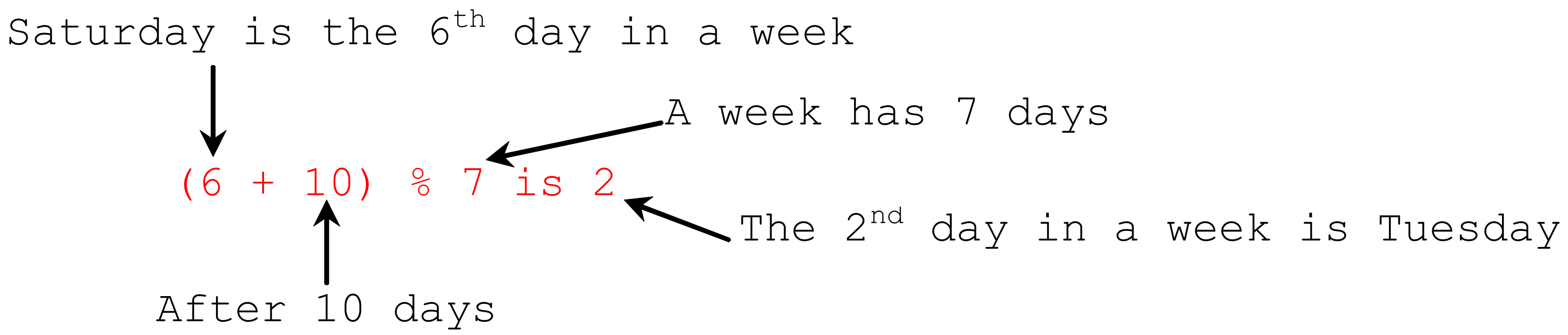
Modulus is often used to find out if a number is even or odd. How would this work?

Answer: Any number N % 2 will be equal to 0 if the number is even, and equal to 1 if the number is odd.

Remainder Operator

Remainder is very useful in programming.

Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:


Saturday is the 6th day in a week
A week has 7 days
(6 + 10) % 7 is 2
The 2nd day in a week is Tuesday
After 10 days

Unary vs binary operator

- The + and – operators can be both unary and binary
  - Unary – has only one operand
    - -5 (unary operator and used to negate 5)
  - Binary – has 2 operands
    - 4 – 5 (binary operator, subtract 5 from 4)

NOTE

Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy. For example,

System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);

displays 0.5000000000000001, not 0.5, and

System.out.println(1.0 - 0.9);

displays 0.09999999999999998, not 0.1. Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.

When writing arithmetic expressions in Java, you have to recall not only the special symbols used, but also that you can't use shorthand notation like you do on paper.

For example, the expression **xy + 2** would have to be written as **x * y + 2**.

Also, exponents in Java are not written with the ^ or ****** symbols as they are in other programming languages.

Exercises

Rewrite each of the following expressions in proper Java syntax:

- ❑ **3 x 4**
- ❑ **(x + 2)(x - 3)**
- ❑ **10 ÷ 2**
- ❑ **2x - 4**

Solutions

Rewrite each of the following expressions in proper Java syntax:

a. **3 x 4**

Answer: 3 * 4

b. **(x + 2)(x - 3)**

Answer: (x + 2) * (x - 3)

c. **10 ÷ 2**

Answer: 10 / 2

d. **2x - 4**

Answer: 2 * x - 4

Number Literals

A *literal* is a constant value that appears directly in the program.

For example,

34, 1,000,000, and 5.0 are literals in the following statements:

```
 int i = 34;
long x = 1000000;
double d = 5.0;
```

Integer Literals

An integer literal can be assigned to an integer variable as long as it can fit into the variable.

A compilation error would occur if the literal were too large for the variable to hold.

For example, the statement

byte b = 1000

would cause a compilation error, because:

- 1000 cannot be stored in a variable of the byte type.

- Too large to fit in a <u>byte</u> type

To denote an integer literal of the long type, append it with the letter L or l. L is preferred because l (lowercase L) can easily be confused with 1 (the digit one).

Integer Division

- The simplest arithmetic expressions in Java are made up of two **operands** and one **operator**. As a programmer, it is important that you not only know the data type of the operands, but also the data type of the expression's result. The rules for this are:

1. **If either operand is a floating-point value, the result is a floating-point value.**
2. **If both operands are a floating-point value, the result is a floating-point value.**
3. **If both operands are an integer value, the result is an integer value.**

Floating-Point Literals

Floating-point literals are written with a decimal point.

By default, a floating-point literal is treated as a double type value.

For example, 5.0 is considered a double value, not a float value.

You can make a number a float by appending the letter f or F, and make a number a double by appending the letter d or D.

For example, you can use 100.2f or 100.2F for a float number, and 100.2d or 100.2D for a double number.

double vs. float

The double type values are more accurate than the float type values. For example,

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

displays `1.0 / 3.0 is 0.`$\underbrace{\texttt{3333333333333333}}_{\text{16 digits}}$

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

displays `1.0F / 3.0F is 0.`$\underbrace{\texttt{33333334}}_{\text{7 digits}}$

Exercise

Do programming question 1.7 (at the end of chapter 1) in your textbook.

Scientific Notation

Floating-point literals can also be specified in scientific notation, for example, 1.23456e+2, same as 1.23456e2, is equivalent to 123.456, and 1.23456e-2 is equivalent to 0.0123456. E (or e) represents an exponent and it can be either in lowercase or uppercase.

- float has 23 bits reserved for the mantissa, 8 bits for exponent and 1 bit for sign
- double has 52 bits reserved for the mantissa, 11 bits for exponent and 1 bit for sign

- How are the floating point numbers represented?
  - In Exponential notation
  - So a number like 123.45 would be represented like this: 1.2345E2
  - The E would represent the exponent or in this case E2 would mean $* 10^2$
  - $1.2345*10^2 = 123.45$
  - $0.0012 = 1.2E\text{-}3 = 1.2*10^{-3}$

- So what is the mantissa?
  – It is the portion before the E
- What is the exponent?
  – It is the portion after the E

- Eg. 1.2345E3
  – Mantissa = 1.2345
  – Exponent = 3

- What about if we try to store a value that is bigger than the maximum?
  - It will overflow.
  - **int** value = 2147483647 + 1;
    - //value will actually be -2147483648
- What if a floating point number is too close to 0?
  - Underflow
  - Java approximates it to 0

Arithmetic Expressions

$$\boxed{\frac{3+4x}{5}-\frac{10(y-5)(a+b+c)}{x}+9(\frac{4}{x}+\frac{9+x}{y})}$$

Can be translated in java to this:

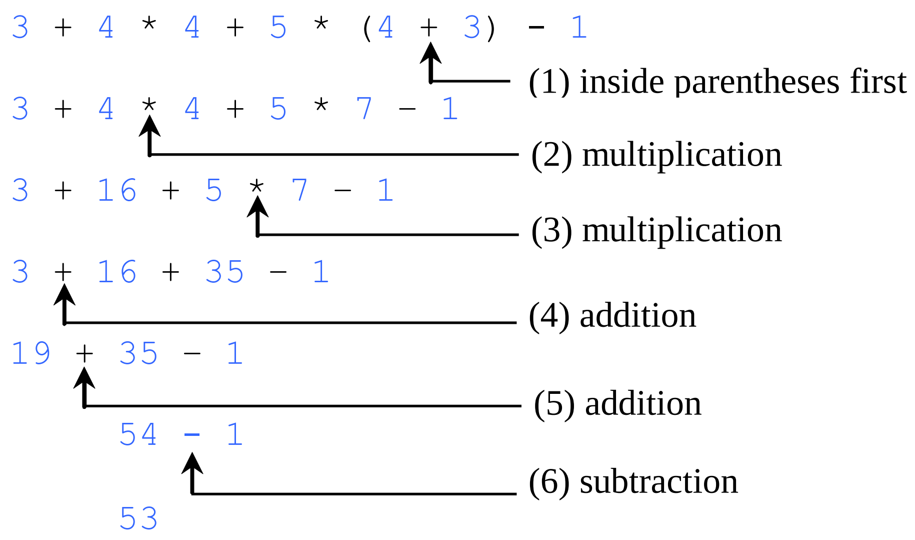
(3+4*x)/5 – 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)

Order of precedence

- Java uses the same order of precedence as in normal arithmetic
- Inside (),
- *, /, %,
- +, -

How to Evaluate an Expression

Though Java has its own way to evaluate an expression behind the scene, the result of a Java expression and its corresponding arithmetic expression are the same. Therefore, you can safely apply the arithmetic rule for evaluating a Java expression.



Problem: Converting Temperatures

Write a program that converts a Fahrenheit degree to Celsius using the formula:

$$celsius = (\tfrac{5}{9})(fahrenheit - 32)$$

Note: you have to write
celsius = (5.0 / 9) * (fahrenheit – 32)

FahrenheitToCelsius Run

Problem: Keeping Two Digits After Decimal Points

Write a program that displays the sales tax with two digits after the decimal point.

SalesTax Run

Common Errors and Pitfalls

- Common Error 1: Undeclared/Uninitialized Variables and Unused Variables
- Common Error 2: Integer Overflow
- Common Error 3: Round-off Errors
- Common Error 4: Unintended Integer Division
- Common Error 5: Redundant Input Objects

- Common Pitfall 1: Redundant Input Objects

Common Error 1: Undeclared/Uninitialized Variables and Unused Variables

double interestRate = **0.05**;

double interest = interestrate * **45**;

Common Error 2: Integer Overflow

```
int value = 2147483647 + 1;
// value will actually be -2147483648
```

Common Error 3: Round-off Errors

System.out.println(**1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1**);

System.out.println(**1.0 - 0.9**);

Common Error 4: Unintended Integer Division

```java
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2;
System.out.println(average);
```

(a)

```java
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2.0;
System.out.println(average);
```

(b)