# 3

# Introduction to Classes and Objects

© 1992-2007 Pearson Education, Inc.  All rights reserved.

# Last time

- practice with Eclipse IDE (lab)
- program output
- program input
- Java’s primitive types (int, double, boolean, etc.)
- control statements (if-else, for loops, etc.)

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Objectives

- review: arithmetic and relational operators
- classes and objects in Java
- instance variables
- instance methods
- getters and setters

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Review: Arithmetic

- Arithmetic calculations used in most programs
  - Usage
    - `*` for multiplication
    - `/` for division
    - `%` for remainder
    - `+`, `-`
  - Integer division truncates remainder

    `7 / 5` evaluates to `1`
  - Remainder operator `%` returns the remainder

    `7 % 5` evaluates to `2`

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Arithmetic (Cont.)

- Operator precedence
  - Some arithmetic operators act before others (i.e., multiplication before addition)
  - Use parenthesis when needed
  - Example: Find the average of three variables `a`, `b` and `c`
    - Do not use: `a + b + c / 3`
    - Use: `( a + b + c ) / 3`

© 1992-2007 Pearson Education, Inc. All rights reserved.

| Operator(s) | Operation(s) | Order of evaluation (precedence) |
|---|---|---|
| * | Multiplication | Evaluated first. If there are several operators of this type, they are evaluated from left to right. |
| / | Division | |
| % | Remainder | |
| + | Addition | Evaluated next. If there are several operators of this type, they are evaluated from left to right. |
| - | Subtraction | |

Fig. 2.12 | **Precedence of arithmetic operators.**

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Review: Relational Operators

- `if` statement
  - Simple version in this section, more detail later

- Conditions in `if` statements can be either `true` or `false`

- Conditions can be formed using equality or relational operators (next slide)

© 1992-2007 Pearson Education, Inc. All rights reserved.

| Standard algebraic equality or relational operator | Java equality or relational operator | Sample Java condition | Meaning of Java condition |
|---|---|---|---|
| *Equality operators* | | | |
| = | == | x == y | x is equal to y |
| ≠ | != | x != y | x is not equal to y |
| *Relational operators* | | | |
| > | > | x > y | x is greater than y |
| < | < | x < y | x is less than y |
| ≥ | >= | x >= y | x is greater than or equal to y |
| ≤ | <= | x <= y | x is less than or equal to y |

Fig. 2.14 | **Equality and relational operators.**

© 1992-2007 Pearson Education, Inc. All rights reserved.

| Operators | Associativity | Type |
|---|---|---|
| * / % | left to right | multiplicative |
| + - | left to right | additive |
| < <= > >= | left to right | relational |
| == != | left to right | equality |
| = | right to left | assignment |

Fig. 2.16 | **Precedence and associativity of operations discussed.**

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Review: Memory Concepts

- Variables
  - Every variable has a name, a type, a size and a value
  - Name corresponds to location in memory
  - When new value is placed into a variable, replaces (and destroys) previous value
  - Reading variables from memory does not change them

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Primitive Types vs. Reference Types

- **Types in Java**
  - **Primitive**
    - `boolean`, `byte`, `char`, `short`, `int`, `long`, `float`, `double`
  - **Reference (sometimes called nonprimitive types)**
    - **Objects**
    - **Default value of** `null`

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Java primitive data types

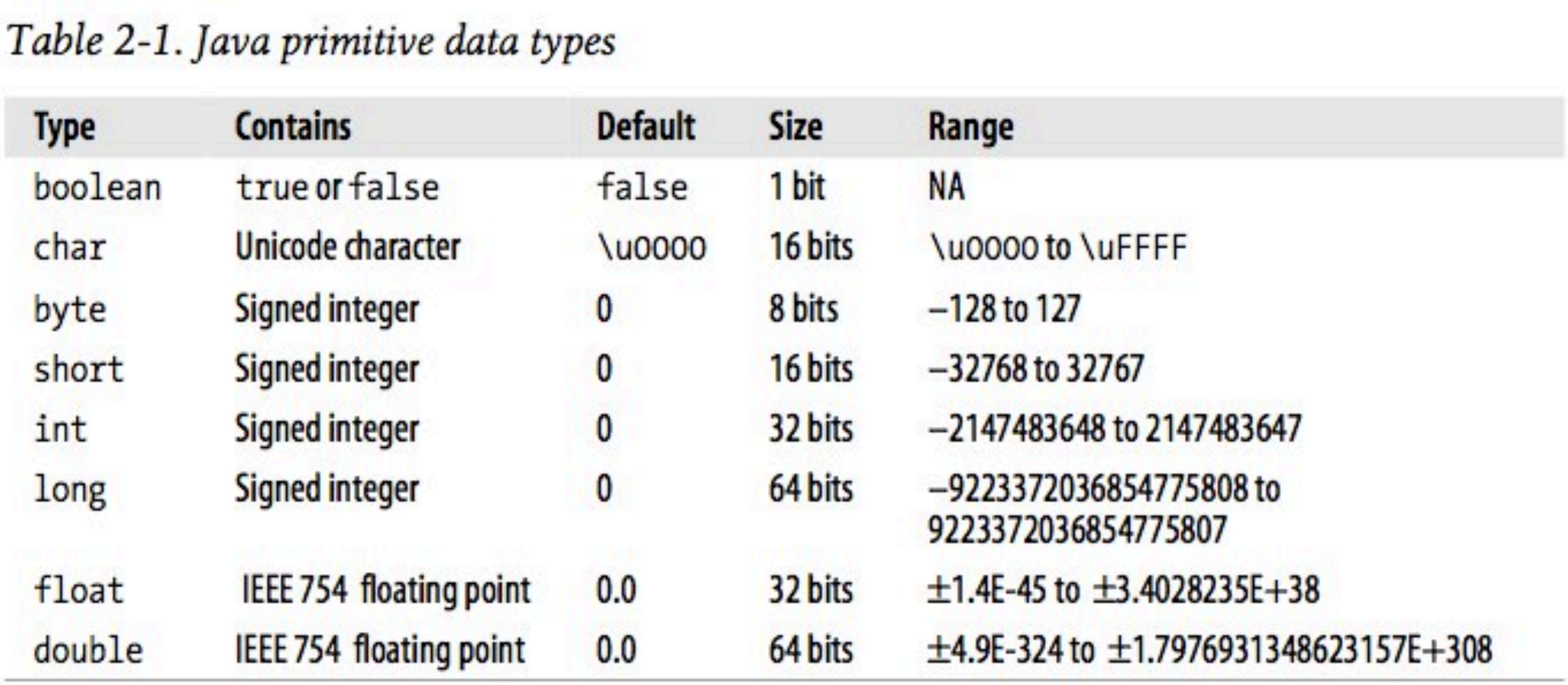


*Table 2-1. Java primitive data types*

| Type | Contains | Default | Size | Range |
|---|---|---|---|---|
| boolean | true or false | false | 1 bit | NA |
| char | Unicode character | \u0000 | 16 bits | \u0000 to \uFFFF |
| byte | Signed integer | 0 | 8 bits | –128 to 127 |
| short | Signed integer | 0 | 16 bits | –32768 to 32767 |
| int | Signed integer | 0 | 32 bits | –2147483648 to 2147483647 |
| long | Signed integer | 0 | 64 bits | –9223372036854775808 to 9223372036854775807 |
| float | IEEE 754 floating point | 0.0 | 32 bits | ±1.4E-45 to ±3.4028235E+38 |
| double | IEEE 754 floating point | 0.0 | 64 bits | ±4.9E-324 to ±1.7976931348623157E+308 |

© 1992-2007 Pearson Education, Inc.  All rights reserved.

## Outline

3.1 Introduction

3.2 Classes, Objects, Methods and Instance Variables

3.3 Declaring a Class with a Method and Instantiating an Object of a Class

3.4 Declaring a Method with a Parameter

3.5 Instance Variables, *set* Methods and *get* Methods

3.6 Primitive Types vs. Reference Types

3.7 Initializing Objects with Constructors

3.8 Floating-Point Numbers and Type `double`

3.9 (Optional) GUI and Graphics Case Study: Using Dialog Boxes

3.10 (Optional) Software Engineering Case Study: Identifying the Classes in a Requirements Document

3.11 Wrap-Up

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Java and OOP

- Object-oriented programming (OOP)
  - Technique to organize and package code
  - So we can reuse it and share it with others

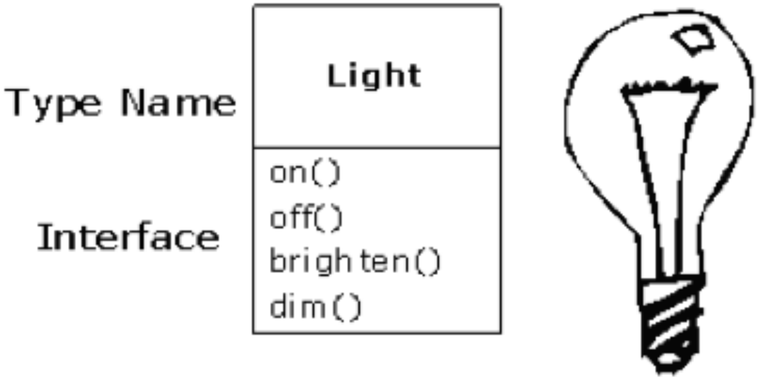
- Every piece of Java code is part of a class

- Every Java class is part of a package

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Data Abstraction and Encapsulation

- Classes and objects
- Data abstraction
  - Abstract data types (ADTs)
- Information hiding
  - Classes normally hide details of their implementation, only expose public interface to their clients



© 1992-2007 Pearson Education, Inc. All rights reserved.

# Classes, Objects, Methods and Instance Variables

- Classes contain one or more attributes
  - Specified by instance variables
  - Carried with the object as it is used

© 1992-2007 Pearson Education, Inc.  All rights reserved.

# Classes, Objects, Methods and Instance Variables

- **Class provides one or more methods**
- **Method represents function (task) in a program**
  - **Hides from its user the complex tasks that it performs**
  - **Method call tells method to perform its task**

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Initializing Objects with Constructors

- Constructors
  - Initialize an object of a class
  - Java requires a constructor for every class
  - Java will provide a default no-argument constructor if none is provided
  - Called when keyword `new` is followed by the class name and parentheses

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Keyword `public`

- keyword `public` is an access modifier
- Class declarations include:
  - Access modifier
  - Keyword `class`
  - Pair of left and right braces

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Method Declarations

- Method declarations
  - Keyword `public` indicates method is available to public
  - Keyword `void` indicates no return type
  - Access modifier, return type, name of method and parentheses comprise method header

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Declaring a Method with a Parameter

- **Method parameters**
  - **Additional information passed to a method**
  - **Supplied in the method call with arguments**
  - **Uses a comma-separated list**

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Classes and Objects

- Java is extensible
  - Programmers can create new classes
- Class instance creation expression
  - Keyword `new`
  - Then name of class to create and parentheses
- Calling a method
  - Object name, then dot separator ( **.** )
  - Then method name and parentheses

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Example: Cube class

© 1992-2007 Pearson Education, Inc.  All rights reserved.

# Notes on `import` declarations

- `java.lang` is implicitly imported into every program
- Default package
  - Contains classes compiled in the same directory
  - Implicitly imported into source code of other files in directory
- Imports unnecessary if fully-qualified names are used

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Good Programming Practice 8.2

**Avoid reinventing the wheel. Study the capabilities of the Java API.**

**If the API contains a class that meets your program' s requirements, use that class rather than create your own.**

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Packages

- How to package reusable software in Java
  - Design and implement `public` classes
  - Organize classes in properly named packages
  - Add a `package` declaration to the source-code file
  - Ship your package(s) to your program clients

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Package Declaration

- `package` declaration
  - must be the first executable statement in the file
  - `package` name often consist of your Internet domain name in reverse order followed by names for your code
    - example: `com.deitel.jhtp7.ch08`
  - `package` name is part of the fully qualified class name
    - Distinguishes between multiple classes with the same name belonging to different packages (name conflict)
  - class name without `package` name is the simple name

© 1992-2007 Pearson Education, Inc. All rights reserved.

# Import Packages

- Import the reusable class into a program
  - Single-type-import declaration
    - Imports a single class
    - Example: `import java.util.Random;`
  - Type-import-on-demand declaration
    - Imports all classes in a package
    - Example: `import java.util.*;`

© 1992-2007 Pearson Education, Inc. All rights reserved.