Some Java Fundamentals

Chapter 2

Chapter Contents

Chapter Objectives

- Observe Java primitive types and their literals
- Explain Java syntax rules
- Contrast primitive types and reference types
- Study variables and constants
- Investigate internal representation of primitive types

Chapter Objectives

- Observe the structure and declaration of classes
- Discover need for import statements
- Note how to use methods
- Study Java API organization
- Look at designing and building simple GUI applications

2.1 Example: A Payroll Program

- Computerize the calculation of employee wages.
  - Employees are paid a fixed hourly rate
  - They can work any number of hours
  - No overtime is paid
- Use object-oriented design
  - Describe behavior
  - Identify objects
  - Identify operations
  - Organize objects & operations in an algorithm

Behavior

- Display on the screen a prompt for …
  - hours worked
  - hourly rate
- Enter values via keyboard
- Compute wages
- Display calculations with descriptive label

Objects

Description of object	Type	kind	Name
the program	??	??	??
screen	`Screen`	variable	`theScreen`
prompt for hrs and rate	`String`	constant	none
number hrs worked	`double`	variable	`hoursWorked`
hourly pay rate	`double`	variable	`hourlyRate`
keyboard	`Keyboard`	variable	`theKeyboard`
wages	`double`	variable	`wages`
descriptive label	`String`	constant	none

Operations

- Display strings (prompts) on screen
- Read numbers for hours and rate (restrict to non negatives)
- Compute wages
- Display real value (wages) and a string on screen

Algorithm

1. Construct **`theScreen`** and **`theKeyboard`**
2. Ask **`theScreen`** to display prompt for hours
3. Ask **`theKeyboard`** to read value and store in **`hoursWorked`**
4. Ask **`theScreen`** to display prompt for rate
5. Ask **`theKeyboard`** to read value and store in **`hourlyRate`**
6. Compute **`wages = hoursWorked x hourlyRate`**
7. Ask **`theScreen`** to display **`wages`** and descriptive label

Coding, Testing, Maintenance

- Note Figure 2.1
  - Code
  - Sample runs
- Maintenance
  - Enhance to include overtime wages
  - Display output using $999.99 style format
- Note revision Figure 2.2

2.2 Types, Variables, and Constants

- Types of objects must be declared before they are used
- Declaration of variables requires a certain syntax
- In declaration, the name of a variable is associated with a type

Types

- `void`
  - denotes the absence of any type
- `String [ ]`
  - in general, a sequence of characters
- `Keyboard, Screen`
  - associated to the Input and Output (I/O) devices normally used
- `double`
  - associated with real (numbers with fractions) values

Primitive Types

- `byte`, `short`, `int`, and `long`
  - for integer values of various sizes
- `float` and `double`
  - for real (rational) values of differing accuracy
- `boolean`
  - for logical (true/false) values
- `char`
  - for individual characters

Reference Types

- Built of other types
  - Example: `String`, `Screen`, `Keyboard`
- Also considered “class types”
- Reference types
  - begin with uppercase letter
  - not known to Java compiler, must be explained
- Contrast primitive types
  - begin with lower case letter
  - are known to Java compiler

Literals – Examples

- Integers
  - `4  19  -5  0  1000`
- Doubles
  - `3.14  0.0  -16.123`
- Strings
  - `"Hi Mom"  "Enter the number : "`
- Character
  - `'A'  'X'  '9'  '$'  '\n'`
- Boolean
  - `true false`

Identifiers

- Names given to variables, objects, methods
- Must not be a Java keyword
  - See Appendix B for list of keywords
- May begin with a letter or the underline character _
- Followed by any number of characters, digits, or _ (note, no blanks)
- Identifiers should be well chosen
  - use complete words (even phrases)
  - this helps program documentation

Conventions for Identifiers

- Classes
  - Names given in lowercase except for first letter of each word in the name
- Variables
  - Same as classes, except first letter is lowercase
- Constants
  - All caps with _ between words
- Methods
  - like variable names but followed by parentheses

Declaration Statements

- Purpose is to provide compiler with meaning of an identifier
- Accomplished in declaration statement
- Some declarations (classes and methods) are provided and must be imported

```
import ann.easyio.*;
```

- Variables to store values must be declared
  - they can be initialized at time of declaration
  - initialized with a literal or even with keyboard input
  - if not explicitly initialized, the default initial value is zero

Values Held by Variables

- Primitive-type variables
  - store a value of the specified type (int, double)
- Reference-type variables
  - store an address of memory location where value is stored
  - thought of as a handle for the object that actually stores the values

Variable Declaration Syntax

- Syntax:
  `type  variable_name;`
  or
  `type variable_name = expression;`
- Note
  - `type` must be known to the compiler
  - `variable_name` must be a valid identifier
  - `expression` is evaluated and assigned to `variable_name` location
  - In the first form, a default value is given (`0`, `false`, or `null`, depending on `type`)

Constants

- Value of object cannot be changed
  - for oft used math values such as PI
  - for values which will not change for a given program
  - improve readability of program
  - facilitate program maintenance
- Declaration syntax:

```
final type CONSTANT_NAME = expression;
```

  - `final` is a Java keyword, makes a constant
  - `type` must be known by compiler
  - `CONSTANT_NAME` must be valid identifier
  - `expression` evaluated
  - should be placed at beginning of class or method

Part of the Picture: Data Representation

How literals of the primitive types are represented and stored in memory.

Representing Integers

- Binary digits used to represent base 10 numbers
  $58_{ten} = 111010_{two}$
  - The 1s and 0s are stored as binary digits in specified number of bits (32 shown in text)
- Negative numbers often stored in “two's complement” representation
  - Invert values, switch 1s for 0s and 0s for 1s
  - Leading bit specifies the sign (0 for +, 1 for –)
- If a number is too large for the number of bits allocated, the condition is overflow

Representing Reals

- Consider
  $22.625_{10} = 10110.101_2 = 1.0110101_2 \times 2^4$
  - The 1.0110101 is stored as the "mantissa"
  - The 4 is stored as the exponent or "characteristic"
- IEEE format
  - Leftmost bit is sign for mantissa
  - 8 bits for exponent
  - Rightmost 23 bits store mantissa
- Problems include
  - Overflow – number too large for exponent
  - Underflow – number too small for exponent
  - Roundoff error – conversion between decimal & binary

Representing Characters

- A numeric code is assigned to each symbol to be represented
- ASCII uses 8 bits
  - Very common for programming languages
  - Limited to 128 characters
- Unicode uses 16 bits
  - newer, used by Java
  - Allows 65,536 different symbols

Representing Booleans

- Only two possible values
  - `true` and `false`
- Only need two possible numbers, `0` and `1`
- Single bit is all that is needed

2.3 Some Basic Program Features

- Comments and documentation
- Classes
- Importing packages
- Using Methods

Comments and Opening Documentation

- Opening documentation should include:
  - description of what program does
  - input needed, resulting output
  - special techniques, algorithms used
  - instructions for use of program
  - Name of programmer, date, modification history
- Opening documentation is multiline
  - between `/*` `*/` character pairs
- Inline comments
  - following `//` double slashes
- Comments ignored by compiler

Classes

- Classes built for real world objects that cannot be represented using available types
- A class is an “extension” of Java
- Definition of class: “a group or category of things that have a set of attributes in common.”
- In programming: a pattern, blueprint, or template for modeling real world objects which have similar attributes

Class Declaration

- Syntax:

```
class className extends existingClassName
{
// Attributes (variables & constants)
// and behaviors (methods)
}
```

- Where
  - ***className*** is the name of a new reference type
  - ***existingClassName*** is any class name known to the compiler
- **{** and **}** mark the boundaries of the declaration

Purpose of Class Declaration

- Creates a new type that the compiler can use to create objects
- This new type inherits all attributes and behaviors of ***existingClassName***
- Note:
  - ***Object*** is often used for ***existingClassName***
  - in this case the **extends** object may be omitted

Importing Packages

- Related classes grouped together into a container called a “package”
  - program specifies where to find a desired class
- Fully-qualified name

  `package_name1.ClassName` or
  `package_name1.package_name2.ClassName`
- By using the `import package_name1` the prefixes using the dot notation can be omitted
- Syntax

  `import package_name.* ;` or
  `import package_name.ClassName;`
  - where `ClassName` is any class stored with `package_name`

Using Methods

- Call, invoke, or send a message to the method of an existing object

  `theScreen.print(" … ");`

  - `theScreen` is the object
  - `print( )` is the method being called
- Syntax of the call:
  - the name of the object
  - the dot '.'
  - the name of the method
  - arguments

Value Returning Methods

- Some methods return a value
- Programmer must also do something with the value to be returned
  - assign the value to a variable

```
variable_name = objectName.methodName(arguments);
```

  - send the value to another method as the parameter

2.4 Java Documentation – API

- Note the sample programs so far …
  - For several tasks, we found a Java method to solve it
  - Other times the programmer writes the class and methods required
- Java designers have provided over 1600 classes
  - Called the Java Application Programmer's Interface or API
  - Each class provides variety of useful methods
  - Classes grouped into packages

API Documentation

- Finding needed package or class
- Hypertext-based documentation system, accessible on World Wide Web
- First page of web site has 3 frames
  - Alphabetical list of packages
  - Alphabetical list of classes
  - A “main” frame that initially lists the Java packages

Web Based Documentation

- Clicking on the name of the package in the “main” frame produces a list of the classes in that package
- Click on name of a class displays information about that class
  - List of fields (variables, constants)
  - List of methods for the class
- Click on a method for a detailed description of the methods

2.5 Introduction to GUIs: A GUI Greeter

- Problem Scenario
  Write a program with graphical user interface that
  - displays a window with prompt for name box to enter name
  - OK and Cancel buttons
  - User enters name, clicks OK
  - Second window gives greeting, uses name, displays a button for terminating program

Objects

Description of Object	Type	Kind	Name
the program	??	??	`GUIgreeter`
window for prompt	input dialog		
prompt for user's name	`String`	constant	
window for greeting	message dialog		
user's name	`String`	varying	`name`
personalized greeting	`String`	varying	

Operations

- Display a window containing a prompt and a text box
- Read a String from the window's text box
- Hide the window
- Display second window with personalized greeting
- Terminate program

Coding in Java

- Note source code in Figure 2.3 Application GUIGreeter
- Note run of program
  - Window for prompt and input
  - Window for Greeting
- Note improved version, Figure 2.4

Input Dialog

- Input dialogs are GUI widgets
  - used to get text input from user
  - Example
    `showInputDialog(prompt);`
  - `prompt` can be
    - a string
    - a graphic image
    - another Java Object

Message Dialog

- A GUI widget for displaying information
- Example

```
showMessageDialog(null, message, title, messageKind);
```

- Message kind
  - can be: error, information, warning, question, or plain
  - used by interface manager to display proper icon