5. Variables

Java

Fall 2009

Instructor: Dr. Masoud Yaghini

Outline

- Types of Variables
- Naming
- Declaring Variables
- Primitive Data Types
- Default Values
- Literals
- Getting Input from the Console: Scanner Class
- References

Types of Variables

Types of Variables

- In the Java programming language, the terms "field" and "variable" are both used.

- Java actually has four kinds of variables:
  - **Instance Variables** (**Non-Static Fields**)
  - **Class Variables** (**Static Fields**)
  - **Local Variables**
  - **Parameters**

Instance Variables (Non-Static Fields)

- Objects store their individual state in **non-static fields**.
- Non-static fields are also known as **instance variables** because their values are unique to each instance of a class (to each object, in other words).
- Example:
  - The **speed** of one bicycle is independent from the **speed** of another bicycle.

Class Variables (Static Fields)

- A given class will only have one copy of each of its static fields / class variables and these will be shared among all the objects.
- Each class variable exists even if no objects of the class have been created.
- Use the word **static** to declare a static field.
- Example:
  - A field defining the number of gears for a particular kind of bicycle could be marked as static since conceptually the same number of gears will apply to all instances.
  - The code **static int numGears = 6;** would create such a static field.
  - The keyword **final** could be added, to indicate that the number of gears will never change.

Local Variables

- **Local variables** are available only within the method that declares them, never anywhere else
- The syntax for declaring a **local variable** is similar to declaring a field
- For example, **int count = 0;**

Parameters

- For the main method is public

  static void main(String[ ] args)

- The args variable is the parameter to this method.
- The parameters are always classified as "variables," not "fields."

Fields vs. Variables

- If we are talking about "fields in general" (excluding local variables and parameters), we may simply say "**fields**."
- If the discussion applies to "all of the above," we may simply say "**variables**.“
- If the context calls for a distinction, we will use specific terms (static field, local variable, etc.) as appropriate.

Naming

Naming

- Variable names are case sensitive.
  - which means that uppercase letters are different from lowercase letters
  - The variable X is therefore different from the variable x
  - and a rose is not a Rose is not a ROSE
- A variable's name can be any legal unlimited-length sequence of Unicode letters and digits

Naming

- **Beginning letter**
  - A variable's name can be beginning with a letter, the dollar sign, "$", or the underscore character, "_".
  - The **convention**, however, is to always begin your variable names **with a letter**
- **Illegal names**
  - The name you choose must not be a keyword or reserved word. See Appendix A, "Java Language Keywords"
  - White space is not permitted
  - They cannot start with a number

Naming

- **Use full words**
  - When choosing a name for your variables, use full words instead of cryptic abbreviations.
  - For example, fields named cadence, speed, and gear, are much more intuitive than abbreviated versions, such as s, c, and g.

Naming

- If the name you choose consists of only one word, spell that word in all lowercase letters.
  - Example: cadence, speed
- If it consists of more than one word, capitalize the first letter of each subsequent word.
  - Example: gearRatio, currentGear
- If your variable stores a constant value, capitalizing every letter and separating subsequent words with the underscore character
  - Example: static final int NUM_GEARS = 6;

Declaring Variables

Declaring Variables

- **Using variables**
  - Declaration
  - Assignment (initialization)
- Variable declarations consist of:
  - a **type**, and
  - a **variable name**
- Example: int gear = 1;
  - Doing so tells your program that:
    - a variable named "gear" exists,
    - holds numerical data, and
    - has an initial value of "1".

Declaring Variables

- A **variable's data type** determines the values it may contain, plus the operations that may be performed on it.
- You can string together variable names of the same type on one line:

```
int x, y, z;
```

- You can also give each variable an initial value when you declare it:

```
int x = 1, y = 20, z = 300;
```

Assigning Values to Variables

- Once a variable has been declared, you can assign a value to that variable by using the assignment operator =:

```
size = 14;

tooMuchCaffeine = true;
```

Primitive Data Types

Primitive Data Types

- A primitive type is predefined by the language and is named by a reserved keyword.
- Primitive values do not share state with other primitive values.
- The eight primitive data types:
  - **Integer types**: byte, short, int, long
  - **Real types**: float, double
  - **Logical type**: boolean
  - **Character type**: char

Primitive Data Types

- **byte**
  - 8 bits signed integer, -128 to 127
- **short**
  - 16 bits signed integer, -32,768 to 32,767
- **int**
  - 32 bits signed integer, -2,147,483,648 to 2,147,483,647
- **long**
  - 64 bits signed integer, -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Primitive Data Types

- **float**
  - single-precision 32-bit floating point
- **double**
  - double-precision 64-bit floating point
- **boolean**
  - has only two possible values: true and false.
  - Use this data type for simple flags that track true/false conditions.
- **char**
  - single 16-bit Unicode character.
  - It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

Primitive Data Types

- Note that all the primitive types are in **lowercase**
- Which type you choose for your variables depends on the range of values you expect that variable to hold

Character strings

- The Java programming language also provides special support for character strings via the java.lang.String class.
- Enclosing your character string within double quotes will automatically create a new String object;
  - for example, String s = "this is a string";
- The String class is not technically a primitive data type, but considering the special support given to it by the language

Default Values

Default Values of Fields

- **Fields** that are declared but not initialized will be set to a reasonable default by the compiler.
- Relying on such default values, however, is generally considered bad programming style.

Data Types and Their Default Values

Data Type	Default Value (for fields)
byte	`0`
short	`0`
int	`0`
long	`0L`
float	`0.0f`
double	`0.0d`
char	`'\u0000'`
`String` (or any object)	`null`
boolean	`false`

Default Values of Local Variables

- The compiler **never assigns** a default value to an **unassigned local variable**.
- If you cannot initialize your local variable where it is declared, make sure to assign it a value before you attempt to use it.
- your Java program will not compile if you try to use an **unassigned local variable**

Literals

Literals

- A **literal** is any number, text, or other information that directly represents a value.
- As shown below, it's possible to assign a literal to a variable of a primitive type:
  - boolean result = true;
  - char capitalC = 'C';
  - byte b = 100;
  - short s = 10000;
  - int i = 100000;
- true, C, 100, 10000, 100000 are literals.

Floating Point Literals

- The floating point types (float and double) can also be expressed using:
  - E or e (for scientific notation),
  - F or f (32-bit float literal), and
  - D or d (64-bit double literal; this is the default and by convention is omitted).
- Examples:
  - double d1 = 123.4;
  - double d2 = 1.234e2; // same value as d1,
  - float f1 = 123.4f;

Boolean Literals

- Boolean literals consist of the keywords **true** and **false**
- These keywords can be used anywhere you need a test or as the only possible values for boolean variables

char Literals

- Literals of types char may contain any Unicode (UTF-16) characters.
- Character literals are expressed by a single character surrounded by single quotation marks
  - `a', `#', `3', and so on
- The Java programming language also supports a few special escape sequences for char and String literals:

  \b (backspace), \t (tab), \n (line feed),
  \f (form feed), \r (carriage return), \" (double quote),
  \' (single quote), \\ (backslash).

String Literals

- A combination of characters is a string
- Strings in Java are instances of the class String
- Strings are not simply arrays of characters as they are in C or C++
- Because string objects are real objects in Java, they have methods that enable you to combine, test, and modify strings very easily
- String literals consist of a series of characters inside double quotation marks:
  - "Hi, I'm a string literal."

String Literals

- Strings can contain character constants such as double quote:
  - "Nested strings are \"strings inside of\" other strings“
- When you use a string literal in your Java program, Java automatically creates an instance of the class String for you with the value you give it

null Literal

- There's also a special null literal that can be used as a value for any reference type.
- null may be assigned to any variable, except variables of primitive types.
- Therefore, null is often used in programs as a marker to indicate that some object is **unavailable**.

Getting Input from the Console: Scanner Class

Getting Input from the Console

- You can obtain input from the console.
- Java uses System.out to refer to the standard output device, and System.in to the standard input device.
- By default the output device is the **console**, and the input device is the **keyboard**.

Getting Input from the Console

- To perform console output, you simply use the println method to display a primitive value or a string to the console.
- You can use the Scanner class to create an object to read input from System.in, as follows:

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

Getting Input from the Console

- A **Scanner object** contains the following methods for reading an input:
  - next(): reading a string. A string is delimited by spaces.
  - nextByte(): reading an integer of the byte type.
  - nextShort(): reading an integer of the short type.
  - nextInt(): reading an integer of the int type.
  - nextLong(): reading an integer of the long type.
  - nextFloat(): reading a number of the float type.
  - nextDouble(): reading a number of the double type.

Getting Input from the Console

- For example, the following statements prompt the user to enter a double value from the console.

```
System.out.print("Enter a double value: ");
Scanner scanner = new Scanner(System.in);
double d = scanner.nextDouble();
```

- Example:
  - TestScanner.java

Getting Input from the Console

- A great strength of Java is its rich set of predefined classes that programmers can reuse rather than "reinventing the wheel."
- These classes are grouped into **packages** named collections of classes.
- Collectively, **Java's packages** are referred to as the **Java class library**, or the **Java Application Programming Interface** (Java API).
- Programmers use import declarations to identify the predefined classes used in a Java program.

Getting Input from the Console

```
import java.util.Scanner; // program uses class Scanner
```

- The import declaration indicates that this example uses Java's predefined Scanner class (discussed shortly) from package java.util.
- Then the compiler attempts to ensure that you use class Scanner correctly.

Getting Input from the Console

System.out.print("Enter second integer: "); // prompt

- Prompts the user to input the second integer.

- uses System.out.print to display the message:
  - "Enter first integer:".
- This message is called a **prompt** because it directs the user to take a specific action.
- System is a class.
- Class System is part of package **java.lang**.

Getting Input from the Console

- Notice that class **System** is not imported with an import declaration at the beginning of the program.
- By default, package **java.lang** is imported in every Java program;
- **java.lang** is the only package in the Java API that does not require an import declaration.

References

References

- S. Zakhour, S. Hommel, J. Royal, I. Rabinovitch, T. Risser, M. Hoeber, **The Java Tutorial: A Short Course on the Basics**, 4th Edition, Prentice Hall, 2006. (Chapter 3)

The End