Principles of Computer Science I

Prof. Nadeem Abdul Hamid
CSC 120 – Fall 2005
Lecture Unit 4 - Data Types

Lecture Outline

- Integer and floating-point numbers
- Limitations of numeric types
- Use of constants
- Arithmetic expressions
- Working with character strings
- User input
- Formatted output

Number Types in Java

- Every value (piece of data) is either
  - Object reference
  - Primitive data type
- Primitive (fundamental) data types
  - Six are for numbers – 4 for integers; 2 for f.p.
- Each number type has different range
  - Depends on number of bits used to represent number

Type	Description	Size
`int`	The integer type, with range –2,147,483,648 . . . 2,147,483,647	4 bytes
`byte`	The type describing a single byte, with range –128 . . . 127	1 byte
`short`	The short integer type, with range –32768 . . . 32767	2 bytes
`long`	The long integer type, with range –9,223,372,036,854,775,808 . . . –9,223,372,036,854,775,807	8 bytes
`double`	The double-precision floating-point type, with a range of about $\pm 10^{308}$ and about 15 significant decimal digits	8 bytes
`float`	The single-precision floating-point type, with a range of about $\pm 10^{38}$ and about 7 significant decimal digits	4 bytes
`char`	The character type, representing code units in the Unicode encoding scheme	2 bytes
`boolean`	The type with the two truth values false and true	1 byte

Possible Computation Errors

- Overflow

```
int n = 1000000;
System.out.println(n * n);
```

  - Use `BigInteger` class to avoid
- Rounding

```
double f = 4.35;
System.out.println(100*f);
```

  - Use `BigDecimal` class to avoid
- To keep code simple, in this class we will just use primitive types
  - For real-world programs, be careful! — *e.g.* do not use floating point types for financial computations

Converting Between Types

- OK to assign integer value to f.p. variable

```
int dollars = 100;
double balance = dollars;
```

- Opposite direction is error:

```
double balance = 13.75;
int dollars = balance;
```

  - May lose *precision*
- Use a *cast* to explicitly convert a value to a different type

```
int dollars = (int) balance;
```

  - Tells compiler that you agree to possible information loss
- To round to nearest whole number, use Math.round

```
long rounded = Math.round(balance);
```

Syntax: Cast

```
(typeName) expression;
```

Example:
```
(int) (balance * 100)
```

Purpose:
To convert an expression to a different type (may result in information loss with primitive types)

When does the case `(long) x` yield a different result from the call `Math.round(x)` ?

Constants

- Values that do not change
  - Often have special significance in a computation

```
payment = dollars + quarters * 0.25 + dimes * 0.10
        + nickels * 0.05 + pennies * 0.01;
```

```
// Clearer version of computation
double quarterValue = 0.25;
double dimeValue = 0.10;
double nickelValue = 0.05;
double pennyValue = 0.01;

payment = dollars + quarters * quarterValue + dimes * dimeValue
        + nickels * nickelValue + pennies * pennyValue;
```


`final` Variables

```
// Version of computation using named constants
final double QUARTER_VALUE = 0.25;
final double DIME_VALUE = 0.10;
final double NICKEL_VALUE = 0.05;
final double PENNY_VALUE = 0.01;

payment = dollars + quarters * QUARTER_VALUE + dimes * DIME_VALUE
        + nickels * NICKEL_VALUE + pennies * PENNY_VALUE;
```


Named Constants

- A **`final`** variable is a (*named*) *constant*
  - Once its value has been set, it cannot be changed
- Named constants make programs easier to read and maintain
- Convention: use all-uppercase names for constants

Class Constants

- If constant values are needed by several methods, declare them together with the instance fields of a class and tag them as `static` and `final`
- Give `static final` constants `public` access to enable other code to use them

```
public class CashRegister {
    ...

    // Constants
    public static final double QUARTER_VALUE = 0.25;
    public static final double DIME_VALUE = 0.10;
    ...
}
```


Syntax: Constant Definition

In a method:

```
final typeName varName = expression;
```

In a class:

```
accessSpec static final typeName varName = expression;
```

Example:
(see previous slides)

Purpose:
To define a named constant in a method or a class

Enhancing CashRegister

```
/**
     Enters the payment received from the customer.
     @param dollars the number of dollars in the payment
     @param quarters the number of quarters in the payment
     @param dimes the number of dimes in the payment
     @param nickels the number of nickels in the payment
     @param pennies the number of pennies in the payment
  */
  public void enterPayment(int dollars, int quarters,
        int dimes, int nickels, int pennies)
```

CashRegister.java
CashRegisterTester.java

Programming Tips: Constants and Variables

- **Do not** use 'magic numbers'

```
h = 31 * h + ch;
```

- *vs.*

```
final int HASH_MULTIPLIER = 31;
h = HASH_MULTIPLIER * h + ch;
```

- **Do** use descriptive variable names

```
payment = d + q * QV + di * DIV + n * NV + p * PV;
```

- *vs.*

```
payment = dollars + quarters * QUARTER_VALUE + dimes * DIME_VALUE
        + nickels * NICKEL_VALUE + pennies * PENNY_VALUE;
```


Assignment

- Assignment operator: =
  - Does not indicate equality of any type
  - Left hand side: variable name
  - Right hand side: single value or expression
- `items = items + 1;`
  - Computes value of `items + 1`
  - Places result back into `items` variable
- `items++;`
  - Increments value of `items` variable
- `items--;`
  - Decrements value of `items` variable

Assignment Shortcut Operators

- Can combine arithmetic operators +-/*% with assignment

```
balance += amount;
```

- has same effect as

```
balance = balance + amount;
```

- `items *= 2;` **<===>** `items = items * 2;`

Arithmetic Operators

- **+** (addition) **-** (subtraction) ***** (multiplication)
- Two kinds of division **/**
  - 'Normal' – if at least one of numbers is f.p.
  - 'Integer' – if both numbers are integers, result is an integer and remainder is discarded

    `7.0 / 4` yields `1.75`

    `7 / 4` yields `1`
- **%** (modulo) operator
  - Computes the remainder of a division

    `7 % 4` yields `3`

Using the Modulo Operator

- Typical use

```
int numberPennies = 435;
int dollars = numberPennies / 100;
int cents = numberPennies % 100;
```

- Try Exercise R4.13

The Math Class

- Contains a collection of mathematical methods, like `sqrt` (square root) and `pow` (power)
  - See Table 2, page 120, Chapter 4

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

19

Integer Division: Common Error

```
int s1 = 5; // Score of test 1
int s2 = 6; // Score of test 2
int s3 = 3; // Score of test 3
double average = (s1 + s2 + s3) / 3; // Error!!!
System.out.println(average);
```

- Solutions:

```
double total = s1 + s2 + s3;
double average = total / 3;
```

  - or

```
double average = (s1 + s2 + s3) / 3.0;
```

20

Roundoff Errors

```
double f = 4.35;
int n = (int) ( 100 * f );
System.out.println( n );  // !!!
```

- Remedy: try using `Math.round` method

21

Programming Tips: Expressions

- Use white space around operators to increase human readability

```
x1=(-b+Math.sqrt(b*b-4*a*c))/(2*a);
x1 = ( -b + Math.sqrt(b * b - 4 * a * c) ) / (2 * a);
```

- Factor out common code blocks
  - More efficient
  - Less possibility of typos

```
x1 = (-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a);
x2 = (-b - Math.sqrt(b * b - 4 * a * c)) / (2 * a);
```

vs.

```
double root = Math.sqrt(b * b - 4 * a * c);
x1 = (-b + root) / (2 * a);
x2 = (-b - root) / (2 * a);
```

22

Using `static` Methods

- A **`static`** method does not operate on an object
- Static methods are defined inside classes
  - Called using name of the class
  - May have explicit parameters

```
Math.sqrt( 9.0 )
```

- Recall naming conventions
  - Class names start with uppercase letter
  - Method, object names start with lowercase

23

Strings

- A string is a sequence of characters
  - Represented in Java by the **`String`** class
- String constants: enclosed in quotation marks

```
"Hello, World!"
```

- Length can be computed using `length` method
- Empty string `""` has length 0

24

Concatenation

- Use the + operator to put strings together to form a longer string

```
String name = "Dave";
String message = "Hello, " + name;
// message is "Hello, Dave"
```

- If one argument of + operator is a string, the other is also converted to a string

```
String a = "Agent";
int n = 7;
String bond = a + n; // bond is Agent7
```

25

Concatenation in Print Statements

- Useful to reduce the number of `System.out.print` method calls

```
System.out.print( "The total is " );
System.out.println( total );
```

```
System.out.println( "The total is " + total );
```

26

Converting Strings to Numbers

- To convert a `String` value, like `"19"`, into an `int` (integer) value, use

```
String input = "19";
...
int count = Integer.parseInt( input );
```

- To convert to floating point, use the `Double.parseDouble` method
- If string contains non-numeric characters, 'exception' (error) occurs

27

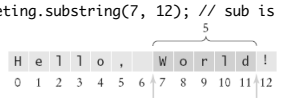
Substrings

- Extract part of string using `substring` method

```
String substring(int begin, int pastEnd)
```

- String position numbers start with 0 (zero)

```
String greeting = "Hello, World!";
String sub = greeting.substring(0, 5); // sub is "Hello"
String sub2 = greeting.substring(7, 12); // sub is "World"
```



28

Alternate Version of `substring`

- Using only one parameter, returns characters from start position to end of string

```
String tail = greeting.substring(7);
```

29

Escape Sequences

- Used to include special characters in a string
- Preceded by \ (backslash) – called the *escape character*
  - \" - quotation marks
  - \' - single quote
  - \n - newline
  - \\ - backslash
- How would you display these lines of text using a single string?

```
He said, "The secret file
is 'c:\secret.txt'."
```

30

char Data Type

- Holds code value for a character
- Every character in the alphabet has a given numeric value in the Unicode encoding scheme (Appendix B)
- Use single quotes for character constants

```
char first = 'H';
char newline = '\n';
```


chars and Strings

- Strings in Java are sequences of Unicode characters
- `charAt` method returns the character at a given position in the string (starting from 0)

```
String greeting = "Hello, World!";
char ch = greeting.charAt( 0 );  // ch is 'H'
```

- Unicode system allows representation of international alphabets (see Advanced Topic 4.5, Random Fact 4.2)

Understanding Data Types

- What's the difference between the following values in Java?
  - 9
  - 9.0
  - "9"
  - '9'

Understanding Compiler Error Messages

```
// Test class full of errors
public class Test {

    public static void main(String[] args) {
        String s = "Hello there";
        char ch = 'abc';  // syntax (compile-time) error
        char p = s.charAt( 100 );
        String t = s.substring( -4 );

        int i = 4 / 0;
    }
}
```

```
$ javac Test.java
Test.java:6: unclosed character literal
        char ch = 'abc';
                  ^
Test.java:6: unclosed character literal
        char ch = 'abc';
                     ^
2 errors
```


Understand Exceptions

```
// Test class full of errors
public class Test {

    public static void main(String[] args) {
        String s = "Hello there";
        // char ch = 'abc';  // syntax (compile-time) error
        char p = s.charAt( 100 );
        String t = s.substring( -4 );

        int i = 4 / 0;
    }
}
```

- Look for
  - Name/type of exception
  - Line number of occurrence

```
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of
range: 100
        at java.lang.String.charAt(String.java:444)
        at Test.main(Test.java:7)
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of
range: -4
        at java.lang.String.substring(String.java:1438)
        at java.lang.String.substring(String.java:1411)
        at Test.main(Test.java:8)
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Test.main(Test.java:10)
```


Keyboard Input

- `System.in` – object corresponding to keyboard input stream
  - Very primitive - reads byte at a time
- For more convenient user input, use the `Scanner` class (new to Java 5.0)

```
Scanner in = new Scanner(System.in);
System.out.print("Enter quantity: ");
int quantity = in.nextInt();
```

'Input prompt'

Scanner Methods

- `nextInt()`
- `nextDouble()`
- `nextWord()`
  - Returns the next word input as a `String` object
  - End of the word is indicated by *whitespace*: space/end of line/tab
- `nextLine()`
  - Returns next entire line of input as a `String`

Input from a Dialog Box

- If not using `Scanner` (Java version prior to 5.0), easy way to get user input is create pop-up window
  - (Advanced Topic 4.7)

```
import javax.swing.JOptionPane;

public class Test {
    public static void main(String[] args) {

        String input = JOptionPane.showInputDialog( "Enter price:" );
        double price = Double.parseDouble( input );
        System.out.println( "You entered: " + price );

        System.exit(0);
    }
}
```

Needed to force program to exit

Formatted Output

```
double total = 3.50;
final double TAX_RATE = 8.5;  // Tax rate in percent
double tax = total * TAX_RATE / 100; // tax is 0.2975
System.out.println( "Total: " + total );
System.out.println( "Tax:   " + tax );
```

Output:

```
Total: 3.5
Tax:   0.2975
```

```
System.out.printf( "Total: %5.2f%n", total );
System.out.printf( "Tax:   %5.2f%n", tax );
```

Output:

```
Total: 3.50
Tax:   0.30
```


Using the `printf` Method

```
System.out.printf( "Total: %5.2f%n", total );
```

Format string

Format specifiers

Other parameters - values filled into corresponding fields of the format string

Format Specifiers

Basic format code: `%f`

Format type

- d — decimal integer
- x — hexadecimal integer
- o — octal integer
- f — fixed floating-point
- e — exponential f.p.
- g — general f.p.
  - (uses shorter of e/f)
- s — string
- n — platform-independent line end

Format code options: `%5.2f`

Width *- the number of spaces in which to fit the value (adds blank spaces if necessary)*

Precision *- the number of digits after decimal point*

Format Flags

- Immediately follow the % character
  - `-` (hyphen) — left justification
  - `0` (zero) — show leading zeroes (in numbers)
  - `+` (plus) — show plus sign for positive numbers
  - `(` — enclose negative numbers in parentheses
  - `,` (comma) — show decimal separators
  - `^` — convert letters to uppercase

String `format` Method

- `printf` is a method of the `PrintStream` class
  - `System.out` is a `PrintStream` object
- The `String` class has a (static) `format` method similar to `printf`
  - Returns a string instead of producing output

```
String message = String.format( "Total:%5.2f", total );
```

  - sets message to the value "Total: 3.50"

43