

Topic 4: Standard Java data types

1. Java classes, objects, and methods
2. Numeric data types
3. Character and string data
4. File stream I/O

Objectives

10. Declare numeric variables and assign them expressions as values
11. Manipulate character strings in Java
12. Read and write a sequential file

1. Java classes, objects, and methods

- A *class* is an abstract specification for *objects*, which are data items
- *Example*: “students” is (informally) a class, while a particular student is represented as an object
- Classes have *members* that are methods (operations) or *properties* (data items, instance variables)
- Methods are invoked by writing an object’s or class’s name, a dot, and the method name, followed by parameter(s) in parentheses

A Java program defines a class

- Program must define one *public class* and may define any number of private classes
- Public class defines the *application*
- Each class may define one or more *method* (subprogram) and *attribute* (data item)
- A public class must define a method called *main*, which executes automatically
- A method definition contains executable statements

Java packages

- I/O and many other features are not part of the Java language
- They are made part of a Java program by using code *packages*
- The packages are *imported* to the .java file using the *import* command at the top of the file:

```
import java.awt;
```

Output objects and method calls

- Example: `System.out.println("Hello");`
- *System* is a predefined class
- *out* is an output stream object, a member of the *System* class
- *println* is a method of the class of *out*
- “Hello” is a parameter to *println*

Classes and methods for input/output

- Standard Java classes: *Scanner* and *System*
- Methods (subprograms) *print* and *println* cause output of their parameter values

```
System.out.print("Sum");  
System.out.print("Sum="+y);
```

- Values of different types may be concatenated with “+” operator
- Input requires creation of a *Scanner* object and call to a *next...* method

```
Scanner in = new Scanner(System.in);  
x = in.nextInt();
```

Some standard Java packages

- *java.lang* (automatically imported)
 - *System*
 - *String*
- *java.awt*: Abstract Windowing Toolkit, graphics classes
- *java.applet*: classes for Web applets

The standard library has thousands of classes. The Application Programming Interface documentation (java.sun.com/javase/6/docs/app) explains each class and how to use it in Java programs

2. Numeric data types

- *Data type*: a category that defines the storage and meaning of a pattern of bits
- Java stores two kinds of numeric values: *integer* and *floating-point*
- *int* is a signed 32-bit integer type, with range of values $-2G \dots +2G$ (32 bits: $2^{32} = 4G$)
- After *int* $x = 9$;, x is an expression of type *int*
- Other integer types: *byte* (8 bits), *short* (16 bits), *long* (64 bits)
- Floating-point types: *float*, *double*

Standard numeric types

Type	Storage (bits)	Range
<i>byte</i>	8	$-128 \dots 127$
<i>short</i>	16	$-32,768 \dots 32,767$
<i>int</i>	32	$-2.1 \times 10^9 \dots 2.1 \times 10^9$
<i>long</i>	64	$-2^{63} \dots 2^{63}$
<i>float</i>	32	$-3.4 \times 10^{38} \dots 3.4 \times 10^{38}$
<i>double</i>	64	$-1.8 \times 10^{308} \dots 1.8 \times 10^{308}$

[cf textbook, p. 48]

int-double type conversions

```
// Convert float value to int
double f = 10.5;
int i = (int)f;
System.out.println(f+" converted to int is "+i);

// Find floor of the value:
double floor = Math.floor(f);
System.out.println("Floor of "+f+" is "+floor);

// Find ceiling of the value:
double ceil = Math.ceil(f);
System.out.println("Ceiling of "+f+" is "+ceil);

// Round off the value:
double round = Math.round(f);
System.out.println(f+" rounded is "+round);

[ num.java ]
```

Arithmetic operators

- *Binary operators have 2 operands:*

+ - * / %

(Note: division of integers produces an *int* value; division by 0 is undefined)

- % modulo or remainder operator; e.g.:

- 7 % 3 = 1

- The clock time 3 hours after 11:00 is

(11 + 3) % 12

- The clock time *h* hours after time *t* is

(*t* + *h*) % 12, except a result of 0 should change to 12

- *Unary negation:* -2 -y -(5 + 9)

Math notation and Java

Math

Java

$$\frac{a+1}{b-2}$$

`(a+1) / (b-2)`

$$a^2$$

`a * a`

or `Math.pow(a, 2)`

c modulo d

`c % d`

$$\sqrt{2}$$

`Math.sqrt(2)`

Extended assignment operators

`x += y` means `x = x + y`

`x -= y` means `x = x - y`

`x *= y` means `x = x * y`

`x /= y` means `x = x / y`

`x %= y` means `x = x % y`

`++x` or `x++` mean `x = x + 1`

`--x` or `x--` mean `x = x - 1`

Preincrement and predecrement operators change value before evaluation; post operators change value after evaluation

Operator precedence

- Parenthesized operations come first
- Unary minus has high precedence
- Multiplication and division precede addition and subtraction
- Operations of same precedence proceed left to right

- *Examples:*

```

8 - 2 + 5      8 - ( 2 + 5 )
3 * 2 + 4      3 * ( 2 + 4 )  3 + 2 * 4
3 + 6 / 2      ( 3 + 6 ) / 2  -2 + 3
1 + 3 % 2      25 % 5 * 2

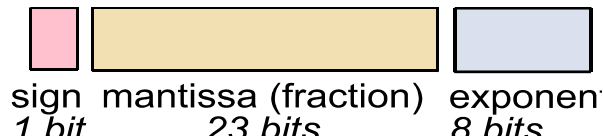
```

Overflow and rounding errors

- When a value is assigned that exceeds the capacity of a variable, *overflow* occurs and incorrect value is stored
- Results of floating-point division are often stored with representational error
- Floating-point currency values are inappropriate for financial computations
- Overflow and representational error may be avoided by use of Java math library methods *Math.BigInteger* and *Math.BigDecimal*

Data types *float* and *double*

- May store fractional values of great range;
float 32 bits, *double* 64 bits
- *Storage*: sign bit, fraction, exponent, based on scientific-notation concept
- Floating-point storage may entail representational error
- The numeral floats because the exponent part compensates for a shift to eliminate 0's on left



Numeric type conversions

- *Automatic*: occurs by promotion or truncation
- *Examples*:
`int y = 4.3; out.print(y); // 4`
`double z = 2;`
- *Type casts* coerce types.
E.g., `(double(1) / 2)` yields 0.5
`out.print((int) 4.2);` outputs 4
- *Danger*: overflow (watch warnings)
- Use *Math.round* to assign floating-point values to integer variables
- *Example*: `num.java`

Methods in the *Math* package

- *Example*, returning square root:
`Math.sqrt(4)`
- *Math* is a class (type), not an object
- Other static library methods in *Math*:
 - `pow(a,b)` returns a^b
 - `sin(x)`, `cos(x)` etc. trig functions
 - `log(x)` natural log
 - `round`, `ceil`, `floor` closest int, ceiling, floor
 - `max(a,b)`, `min` largest, smallest of two

Random-number generator

- Chance makes games more interesting
- Java class *Random* generates a stream of *integers* or *doubles*
- *Example*, simulating roll of die:
`Random gen = new Random();`
`boolean is_heads = (gen.nextInt(2) > 1)`
- `nextInt(n)` returns a random integer in $(0..n]$
- `nextDouble(1)` returns a *double* in $(0..1]$

3. Character and string data

- *char* is a standard primitive type
- *String* is a structured type defined by the standard library package
- *String* objects are sequences of characters in memory
- *Files* are stream objects; *streams* are sequences of characters going to output devices or coming from input devices

Standard data type *char*

- *char* data items are 16 bits, representing up to 65,536 symbols in Unicode
- The ASCII table maps a character set to integers 0...127 (7 bits)
- Character literals use single quotes
- Escape sequences express special characters: newline '\n', tab '\t', backspace '\b', single quote '\'', null character '\0', backslash '\\'
- To declare a variable that stores *one* character:
char c;

Java strings

- String literals use double quotes:
`String stu_id = "ab12";`
- Concatenation operator '+':
`String greeting = "Hi"+stu_id;`
- Concatenation converts all types to *String*, allowing grouping of output items of different types in *println* calls
- Methods of *String* class: *length()*, *parseInt()*, *substring()*, *charAt()*
- See example code

String declaration, input, output

```
// Set up for input:
Scanner cin = new Scanner(System.in);
String name = new String();

// Input:
System.out.print("Please enter your name: ");
name = cin.next();

// Display:
System.out.println("Hello "+name);

[str.java]
```

String methods

Expression	Value
<code>String s1 = toupper("abcd")</code>	ABCD
<code>char c = s1.charAt(2);</code>	C
<code>String s2 = s1.substring(2,2)</code>	CD
<code>int i = s1.indexOf('B')</code>	1

Other methods:

```
length(), compareTo(t),
compareToIgnoreCase(t), equals(t),
equalsIgnoreCase(t), startsWith(t),
startsWith(t, i), endsWith(t),
contains(cs), indexOf(t), indexOf(t, i),
indexOf(c), indexOf(c, i), charAt(i),
s.substring(i), substring(i, j),
toLowerCase(), toUpperCase(), trim(),
replace(c1, c2), replace(cs2, cs3)
```

Initializing variables

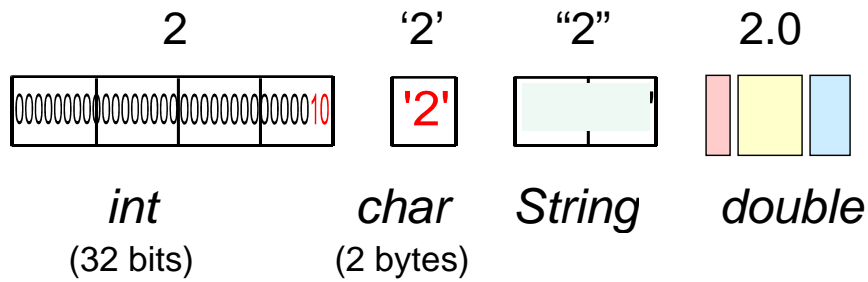
- The value of a variable is may be undefined until it is given a value explicitly
- Use of variable with undefined value is error:

```
int x;
System.out.print(x); // error
```

- Safest practice is to initialize all variables in declaration

```
int x = 2;
double sum = 0.0;
char c = ' ';
String s = "Hello";
```

Storage for four standard types



27

Formatting output

- *System.out.printf()* enables specification of width, precision, and type of output elements
- *Example:*

```
System.out.printf("%-6s%5.2f%n", "Tax:", tot);
```

displays a *String* and a *float* value, with the string left-aligned across width of six, and the *float* of width five and precision to two decimal places
- `%n` denotes end-of-line character
- Other type specifiers: `%d` (decimal), `%x` (hexadecimal), `%o` (octal), `%e` (exponential),

Formatted strings

- *String.format(fmt_spec, parm_list)* returns a string formatted using
 - %s string
 - %n end of line
 - %8d decimal integer right-flush across 8 columns
 - %8.2f floating-point with two digits precision
 - %-8s left-flush across 8 columns

4. File stream I/O

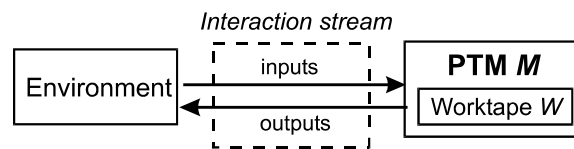
- A *file stream* is a sequence of characters moving to or from a storage device
- Java standard file manipulation classes: *Scanner, FileReader, PrintWriter*
- If a file cannot be opened, a *FileNotFoundException* is thrown at runtime, possibly generating an error message
- *Exceptions* are covered later in this course

Stream input/output

- A *stream* is a sequence of characters moving from or to a device



- Stored data occupies finite space
- destination/sources for streams: keyboard, screen, ports, disk files
- A user-controlled I/O loop may never end



Java stream input

- *System* package defines object *System.in* with *nextInt* input method
- *Scanner* class enables creation of input objects using *new* operator:

```
Scanner in = new Scanner(System.in);  
x = in.nextInt();
```
- *Scanner* object *in* reads next integer from (keyboard) input stream
- *Scanner* methods: *nextChar*, *nextInt*, *nextDouble*, *nextLine*, *next*

Inputting numbers and strings

```
double d = in.nextDouble();
String s = in.next();
    // reads until white-space char
String s = in.nextLine();
    // reads until end of line
```

File stream input

- To open file for input:

```
FileReader reader = new FileReader("x.txt");
Scanner fin = new Scanner(reader);
```
- The *FileReader* class defines a sequential text file stream and its constructor opens the named file
- To read integer from input file (see keyboard input):

```
x = fin.nextInt();
```
- To close input file:

```
fin.close();
```
- Any *Scanner* method usable for keyboard input is also valid for file input
- *Example file:* `update.java`

File stream output

- To open file for output:
`PrintWriter out = new PrintWriter("x.txt");`
- *Warning:* Opening a text file for output in this way erases any data previously stored under this name
- To write to file:
`out.println("Hello");`
- To close output file:
`out.close();`
- Any *System.out* method usable for screen output is also valid for file output
- *Example program:* `update.java`

File errors

- Attempting to open a file to read generates a *runtime error* (exception) if file is not found
- Error is represented as an *exception*
- Any method, such as *main*, that opens a file to read should be defined with “throws `FileNotFoundException`” in header
- Other file errors include attempting to read past end of file, attempting to read item of wrong data type

Objects vs. object references

- When a variable is declared with a class as its type, it refers to an *object* or is a *reference*
- If *Purchase* is a class, then
`Purchase p = new Purchase();`
declares an *object*
- If *p* is a *Purchase*, then
`Purchase q = p;`
declares a *reference* to *p*
- References store *addresses* of objects, not *copies*

Null objects

- For class *C*, the following is acceptable:
`C x = null;`
- In that case, *x* is a null object reference, denoting no object
- Later the variable may be given a non-null value:
`x = new C();`
- Object references may be tested for nullness:
`if (x == null)...`

References

Cay Horstmann. *Big Java*, 3rd ed. Wiley, 2008.

D. Keil. Java file input/output. Classroom handout, 2008.

_____. Reading and displaying file records. Classroom handout, 2008.