ECE25100: OBJECT ORIENTED PROGRAMMING

Note 2 _ Programming Basics in JAVA

Instructor: Xiaoli Yang

Primitive Data Types

- Data is a term given to information which is used by the computer. It can be entered into a program or computed internally.
- In JAVA, there are certain primitive types of data. The primitives differ in the amount of space that they use up.
- The type must be specified for all data stored. You should use the data type that suits your needs but does not waste memory by using more bytes than is necessary.

Primitive Data Types

Type	Data Range	Storage Size
byte	-2^7 (-128) to 2^7-1 (127) Default value: 0	8-bit signed two's complement integer
short	-2^{15} (-32768) to $2^{15}-1$ (32767) Default value: 0	16-bit signed two's complement integer
int	-2^{31} (-2147483648) to $2^{31}-1$ (2147483647) Default value: 0	32-bit signed two's complement integer
long	-2^{63} to $2^{63}-1$ (i.e., -9223372036854775808 to 9223372036854775807) Default value: 0L	64-bit signed two's complement integer
float	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38 Default value: 0.0f	32-bit IEEE 754 floating point
double	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308 Default value: 0.0d	64-bit IEEE 754 floating point
boolean	Two possible values: true and false Default value: false	1-bit
char	Minimum value is '\u0000' (or 0) Maximum value is '\uffff' (or 65,535 inclusive)	16-bit Unicode character

Variable

- A variable is like a box which can store some information for later use.
- We can put something in it and "get it" later as we need to. In fact, we don't really take things out of the box, we simply look at what is in there or replace the contents later on.
- Variable names should start with lower case and multiple word names should have every word capitalized (except the first). Do not use underscores.
- A variable name must be unique and it is case-sensitive. It cannot be a reserved word.

Reserved Words

- The followings are reserved by the JAVA language and are listed below:

abstract	default	goto	new	synchronized
boolean	do	if	package	this
break	double	implements	private	throw
byte	else	import	protected	throws
case	extends	instance of	public	transient
catch	false	int	return	true
char	final	interface	short	try
class	finally	long	static	void
const	float	native	super	volatile
continue	for	null	switch	while

Static Constant

- For static constant names, all uppercase letters are used with underscore characters as well and a value is assigned during the declaration:

```
LAST_ACCOUNT_NUMBER = 0;
INTEREST_RATE = 0.04;

PI = 3.14159265;
```

Global and Local Variable

- In Java, there is no global variables.
- A local variable is a "temporary" variable that we use in our programs. It:
  - holds a value of a specific type of primitive or object
  - is usable ONLY within the method that it is defined
- When do we need to use a local variable ?
  - to store intermediate results in an extensive computation
  - whenever we need to use a value more than once in a computation
  - to simplify steps in a piece of code.
  - it can help prevent code duplication

Declaring Variables

- In JAVA, all variables must be declared with:
  - a type (either a primitive, class or interface)
  - a name
  - Declaration is as follows: <type> <name>;
  - Examples:

```
int x;          // Declare x to be an integer variable;
double radius;  // Declare radius to be a double variable;
char a;         // Declare a to be a character variable;
```

Assignment Statements

- An assignment operator is used to give a value to the variable (=):
  - it binds a variable to an object (or primitive data type).
  - when using the variable afterwards, it is as if the object itself is used.
  - it is analogous to putting something into the box
  - Examples

```
x = 1;          // Assign 1 to x;
radius = 1.0;   // Assign 1.0 to radius;
a = 'A';        // Assign 'A' to a;
```

Declaring and Initializing in One Step

- `int x = 1;`
- `double d = 1.4;`

Variable Value

- Once assigned a value, then the variable name can be used in any JAVA coding expression.
  When evaluated the current value of the variable is substituted for that variable name.
- The value remains the same as long as it is not changed with the assignment operator (=).

Variable (Cont.): Blocks and Variable Scope

- In Java, we can actually group a bunch of statements into what is called a block by using the brace characters { }
- Variables defined within a block are not usable outside of the block:

```
int tax = 8;
float price = 23.45;
{
        double total = price * (1+ tax/100.0);
}
System.out.println("I owe $" + total);
```

Variable (Cont.): Blocks and Variable Scope

- The block therefore defines the scope (or visibility) of a variable. We cannot access the total variable here since it was declared within the block.

- The above code is for demonstration purposes only. This idea propagates throughout JAVA in that every variable MUST be defined within a block (i.e., between the braces { } characters). Hence, all variables are undefined outside those blocks.

More Variable Definition Examples

- Here are examples of variable declarations:

```
int      days;     // reserve space for a variable named days
days = 365;        // assign value of 365 to variable named days
double  weight;   // reserve space for a variable named weight
weight = 165.2;    // assign value of 165.2 to variable named weight
char     sex;      // reserve space for a variable named sex
sex = 'M';         // assign value of 'M' to variable named sex
boolean hungry;   // reserve space for a variable named hungry
hungry = false;
```

- Here are some examples of declaring and assigning objects to reference variables:

```
Date     birthday = new Date();
Scanner  keyboard = new Scanner(System.in);
```

Mathematical Operators:
Numeric Operators

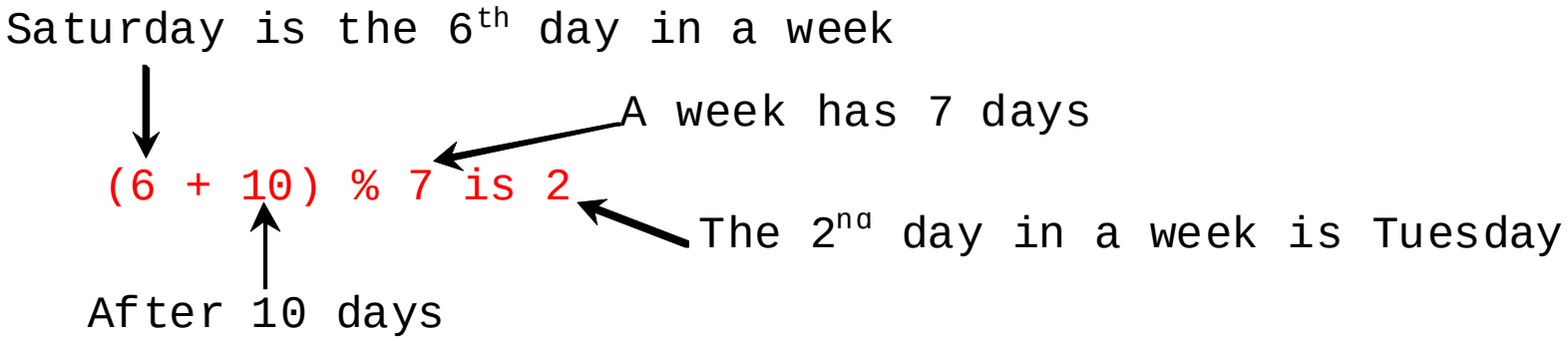
Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 – 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

Mathematical Operators (Cont.): Integer Division

- +, -, *, /, and %
- 5 / 2 yields an integer 2.
- 5.0 / 2 yields a double value 2.5
- 5 % 2 yields 1 (the remainder of the division)

Mathematical Operators (Cont.): Remainder Operator

- Remainder is very useful in programming. For example, an even number % 2 is always 0 and an odd number % 2 is always 1.
- Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days?



Mathematical Operators (Cont.):
Shortcut Assignment Operators

Operator	Example	Equivalent
+=	i += 8	i = i + 8
-=	f -= 8.0	f = f - 8.0
*=	i *= 8	i = i * 8
/=	i /= 8	i = i / 8
%=	i %= 8	i = i % 8

Mathematical Operators (Cont.): Increment and Decrement Operators

Operator	Name	Description
++var	preincrement	The expression (++var) increments var by 1 and evaluates to the new value in var after the increment.
var++	postincrement	The expression (var++) evaluates to the original value in var and increments var by 1.
--var	predecrement	The expression (--var) decrements var 1 and evaluates to the new value in var after the decrement.
var--	postdecrement	The expression (var--) evaluates to the original value in var and decrements var by1.

Mathematical Operators (Cont.): Increment and Decrement Operators

```
int i = 10;
int newNum = 10*i++;
```

Same effect as →

```
int i=10;
int newNum = 10*i;
i = i+1;
```

```
int i = 10;
int newNum = 10*(++i);
```

Same effect as →

```
int i=10;
i = i+1;
int newNum = 10*i;
```

Mathematical Operators (Cont.): Increment and Decrement Operators

- Using increment and decrement operators makes expressions short, but it also makes them complex and difficult to read.
- Avoid using these operators in expressions that modify multiple variables, or the same variable for multiple times such as this:

```
int k = ++i + i
```

Mathematical Operators (Cont.): Operator Precedence

- There are many more types of operators. Here is a table showing the operators in JAVA (some which we have not yet discussed) and their precedence.
- The topmost elements of the table have higher precedence and are therefore evaluated first (in a left to right fashion). In the table <expr> represents any java expression. If however, you are writing code that depends highly on this table, then it is likely that your code is too complex.
- We can always add parentheses (round brackets) to the expression to force a different ordering. Expressions in round brackets are evaluated first (left to right):

Mathematical Operators (Cont.): Operator Precedence

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Mathematical Operators (Cont.): Bitwise Operators

- JAVA also provides bitwise operators for integers and booleans:
  - ~ bitwise complement (prefix unary operator)
  - & bitwise and
  - | bitwise or
  - ^ bitwise exclusive-or
  - << shift bits left, filling in with zeros
  - >> shift bits right, filling in with sign bit
  - >>> shift bits right, filling in with zeros
- To understand how these work, you must understand how the numbers are stored as bits in the computer.

Variable Definition Examples

- Example 1: Print different types of variables _ SystemPrintTest.java
- Example 2: Define variables _ SuperCalculatorTester.java

Logical Operators

- In addition to the mathematical operators, there are also logical operators which can be used for standard comparisons between numbers:

  `<` less than

  `<=` less than or equal to

  `==` equal to

  `!=` not equal to

  `>=` greater than or equal to

  `>` greater than

- Examples:

```
5 < 7                 // returns true
23.321 >= 54.1        // returns false
'g' != 'G'            // returns true
'g' == 'g'            // returns true
```

Logical Operators (Cont.)

- The type of result returned from these comparisons is a boolean (i.e., true or false).
  Operators that operate on booleans are:

  ! not (prefix)

  && conditional and

  || conditional or

- Examples:

```
!(5 < 7)                    // returns false
('a' == 'a') && (5 < 7)     // returns true
('G' == 'a') && (5 < 7)     // returns false
('G' == 'a') || (5 < 7)     // returns true
('G' == 'a') || (5 >= 7)    // returns false
```

Logical Operators (Cont.)

- The && and || operators are short circuit. That means, during a && computation, if the left side is false, then there is no need to check the right side since the result of the entire expression will always be false.
- Similarly, if the left side is `true` during a || operation, then `true` is returned and the right side is not evaluated. Unfortunately, we cannot get a good feel for these operators until we discuss the if statement (later).
- Often, the Boolean operators are used with an IF statement, which is required often to make a decision in your program.

Benefits of Methods

- Write a method once and reuse it anywhere.
- Information hiding. Hide the implementation from the user.
- Reduce complexity.

The `Math` Class

- Class constants:
  - `PI`
  - `E`
- Class methods:
  - Trigonometric Methods
  - Exponent Methods
  - Rounding Methods
  - min, max, abs, and random Methods

A few of the Common Math Functions

- Trigonometric:

```
Math.sin(0)          // returns 0.0 which is a double
Math.cos(0)          // returns 1.0
Math.tan(0.5)        // returns 0.5463024898437905
```

- Conversion and Rounding:

```
Math.round(6.6)      // returns 7
Math.round(6.3)      // returns 6
Math.ceil(9.2)       // returns 10
Math.ceil(-9.8)      // returns -9
Math.floor(9.2)      // returns 9
Math.floor(-9.8)     // returns -10
Math.abs(-7.8)       // returns 7.8
Math.abs(7.8)        // returns 7.8
```

A few of the Common Math Functions

- Powers and Exponents:

```
Math.sqrt(144)                    // returns 12.0
Math.pow(5,2)                     // returns 25.0
Math.exp(2)                       // returns 7.38905609893065
Math.log(7.38905609893065)        // returns 2.0
```

- Comparison:

```
Math.max(560, 289)   // returns 560
Math.min(560, 289)   // returns 289
```

- Generation of a Random Number:

```
Math.random()       // returns a double >=0.0 and <1.0
```

The random Method

Generates a random double value greater than or equal to 0.0 and less than 1.0 (0 <= Math.random() < 1.0).

Examples:

(int)(Math.random() * 10) → Returns a random integer between 0 and 9.

50 + (int)(Math.random() * 50) → Returns a random integer between 50 and 99.

In general,

a + Math.random() * b → Returns a random number between a and a + b, excluding a + b.

Type Casting

Implicit casting

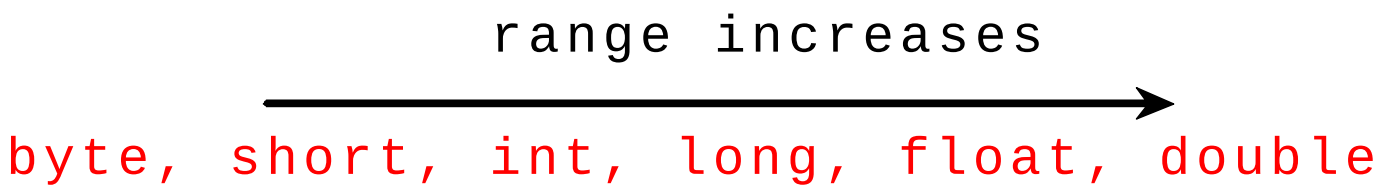
double d = 3; (type widening)

Explicit casting

int i = (int)3.0; (type narrowing)

int i = (int)3.9; (Fraction part is truncated)

What is wrong? int x = 5 / 2.0;



JDK 1.5 Feature

Formatting Numbers Using printf

- How can we format numbers when printing to the console ?
  - What if we want to show $2.00 as output ? What does this code do: System.out.println("$" + 2.00);
  - It actually only outputs $2.0.
- We can display the appropriate number of decimal places by making use of the printf method. Here is the format for the printf:
  - System.out.printf(formatString, value1, value2, ..., valueN);

Formatting Numbers Using printf (Cont.)

- For example, the following code will display the $2.00 that we want:
  - System.out.printf("Two dollars is displayed as $%4.2f", 2.00);
  - Here the formatString has a fixed set of characters: "Two dollars is displayed as $" which is then followed by a floating point value of 2.00.
  - The portion of the formatString "%4.2f" indicates that the value to be displayed should be a floating point number with a precision of 2 decimal places and a width of 4 characters in total.

Formatting Numbers Using printf (Cont.)

- The general formatString allows you to supply as many arguments as you wish. For example the following code:

```
float total = 2.00f;
System.out.printf("Total=$%4.2f + $%4.2fTAX = $%4.2f", total, total*0.06, total*1.06);
```

Will produce

```
Total=$2.00 + $0.12TAX = $2.12
```

- You MUST make sure however, that you have at least the required number of values for each format type expected.

Formatting Numbers Using printf (Cont.)

- For example, the following code will produce a MissingFormatArgumentException since one of the arguments (i.e., values) is missing:

```
float total = 2.00f;
System.out.printf("Total=$%4.2f + $%4.2fTAX = $%4.2f", total, total*0.06);
```

- If you supply more values than what is necessary, JAVA will ignore the last few that it does not need. This may cause your output to be different than what you expected:

```
System.out.printf("Total=$%4.2f plus TAX = $%4.2f", total, total*0.06, total*1.06);
```

will produce:

```
Total=$2.00 plus TAX = $0.12
```

- Also, you should be careful not to miss-match types, otherwise an IllegalFormatConversionException may occur as in the following code:

```
System.out.printf("Total=$%4.2f plus TAX = $%d", total, total*1.06);
```

Formatting Numbers Using printf (Cont.)

- You may also mix and match different format types in the same printf line. There are a few other format types that may be used in the formatString:

Type	Description of What It Displays	Example Output
%d	A general integer	4096
%x	An integer in lowercase hexadecimal	ff
%X	An integer in uppercase hexadecimal	FF
%o	An integer in octal	377
%f	A floating point number with a fixed number of spaces	83.43
%e	An exponential floating point number	7.869877e-03
%g	A general floating point number with a fixed number of significant digits	0.008
%s	A string as given	“Hello”
%S	A string in uppercse	“HELLO”
%n	A platform-independent line end	<CR><LF>
%b	A boolean in lowercase	true
%B	A boolean in uppercase	FALSE

Formatting Numbers Using printf (Cont.)

- In addition to these format types, there are various format flags that can be added after the % sign:

Format Flag	Description of What It Does	Example Output
-	Numbers are to be left justified	2378.348 followed by any necessary spaces
0	Leading zeros should be shown	000244.87
+	Plus sign should be shown if positive numbers	+67.34
(	Enclose number in round brackets if negative	(439.67)
,	Show decimal group separators	2,347,892.99

- Example 3: How to use Printf : PrintfTester
- Example 4: How to use Printf flag : PrintfFlagTester

Literals and Enumerated Types : Escape Sequences for Special Characters

Description	*Escape Sequence*	*Unicode*
Backspace	`\b`	`\u0008`
Tab	`\t`	`\u0009`
Linefeed	`\n`	`\u000A`
Carriage return	`\r`	`\u000D`
Backslash	`\\`	`\u005C`
Single Quote	`\'`	`\u0027`
Double Quote	`\"`	`\u0022`

References

- Carleton University COMP 1005 and 1006:
  http://www.scs.carleton.ca/~lanthier/teaching/COMP1405/Notes/
  http://www.scs.carleton.ca/~lanthier/teaching/COMP1406/Notes/
- Armstrong Atlantic State University
  http://www.cs.armstrong.edu/liang/intro10e/
- Oracle Java tutorial:
  http://docs.oracle.com/javase/tutorial/
- MIT opencourseware
  https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/