Java Fundamentals, Compartmentalization

"Operators, Classes & Objects, Methods, Control Structures, Classes namespaces, Visibility and Accessibility"

Advanced Programming

Shakirullah Waseeb
shakir.waseeb@gmail.com

Nangarhar University

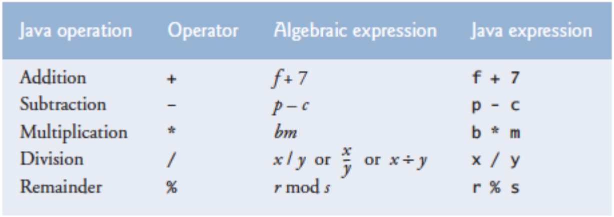
March 7, 2018

Agenda

Agenda

Arithmetic Operators

- Operators used for arithmetic calculations



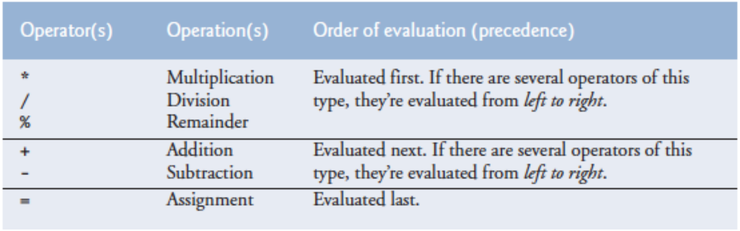
Java operation	Operator	Algebraic expression	Java expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

[1]

Figure: arithmetic operators

Arithmetic Operators Precedence

- Precedence of arithmetic operators



Operator(s)	Operation(s)	Order of evaluation (precedence)
* / %	Multiplication Division Remainder	Evaluated first. If there are several operators of this type, they're evaluated from *left to right*.
+ -	Addition Subtraction	Evaluated next. If there are several operators of this type, they're evaluated from *left to right*.
=	Assignment	Evaluated last.

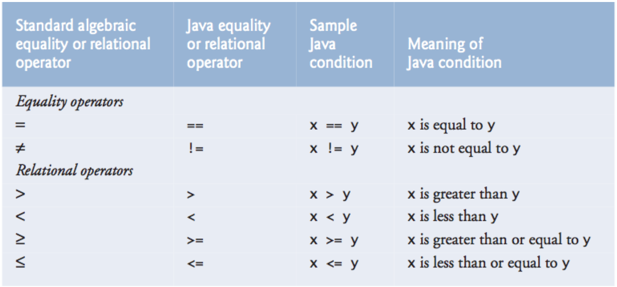
[1]

Figure: precedence of arithmetic operators

Agenda

Equality and relational operators

- condition is an expression that can be true or false
- e.g conditional expression in *if* selection statement, which make decision on condition's value
- Conditions in *if* statements can be formed using *equality* ($==, !=$) or *relational* ($>, <, >=, <=$) operators



Standard algebraic equality or relational operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
Equality operators			
=	`==`	`x == y`	x is equal to y
≠	`!=`	`x != y`	x is not equal to y
Relational operators			
>	`>`	`x > y`	x is greater than y
<	`<`	`x < y`	x is less than y
≥	`>=`	`x >= y`	x is greater than or equal to y
≤	`<=`	`x <= y`	x is less than or equal to y

[1]

Figure: precedence of arithmetic operators

Agenda

Class Declaration and Definition

- classes declared with *public* keyword must be:
  saved in a separate file
  file name must be same with class name

A simple class declaration example

```
public class SimpleClass {
            public void dispMessage(String str){
                System.out.println(str);
                }
            }
```

Save it as SimpleClass.java and compile it as javac SimpleClass.java

Agenda

Instantiation and Execution

- Every java application has a class that contain a *main* method, where application starts its execution
- Some programmers refer to such class as a driver class
- Example program containing *main* method

Example program

```
public class SimpleClassApp {
            public static void main (String args[]){
                  SimpleClass sc = new SimpleClass();
                  sc.dispMessage("A message from application");
                  }
            }
```

Save it as SimpleClassApp.java, compile it using command javac SimpleClassApp.java, and finally run it as java SimpleClassApp.

Agenda

Sequential, Selectional, and Repetition structures

- Control structure:
  sequential execution; execute in the order in which program is written
  transfer of control; specify which instruction to execute next
- Sequence Structure
  - normal execution of program instruction in the order they are written
- Selection Structure
  - single selection statements (*if* statement)
  - double selection statements (*if .. else* statement)
  - multiple selection statements (*switch* statement)
- Repetition Structure
  - also called looping statements
  - looping continuation condition
  - *while*, *do while*, and *for*

Agenda

Break and Continue statements

- *break* statement:
  when executed in a *while, for, do...while or switch*, causes immediate exit from that statement
  typically use to escape early from a loop or to skip the remainder of a *switch*
- *continue* statement:
  when executed in a *while, for or do...while*, skips the remaining statements in the loop body and proceeds with the *next iteration* of the loop
  while and do...while: immediately test loop-continuation
  for: increment expression executes, then loop-continuation is tested

Introduction

- Compartmentalization (dividing into groups and categories) of class name space
- To avoid class name collision
- Mechanisms for partitioning the class name space into more manageable chunks
- Naming and visibility control mechanism

Class packaging in Java

- Java uses ***package*** to compartmentalize class name space
- Class can be defined inside a **package** that are not accessible outside the package
- Even class members can be defined are only exposed to other members of the same package
- Allows classes to have intimate knowledge of each other, but not expose that knowledge to external world
- A java source file can contain any (or all) of the following four internal parts:
  - A single **package** statement (optional)
  - Any number of **import** statements (optional)
  - A single **public** class declaration (required)
  - Any number of classes **private** to the package (optional)

Agenda

Defining a Package

- Quite easy:simply include a **package** statement as the first statement in a Java source file
- any classes declared in this file will belong to specified package
- **package** statement defines a name space in which classes are stored
- if package statement is omitted, class names are put into default package, having no name
- general form of **package** statement:
  package *pkgname*
- Java uses file system directories to store packages
- More than one file can include the same **package** statement
- packages hierarchy can be created using a period
  package *pkg1[.pkg2[.pkg3]]*
  package java.awt.image;

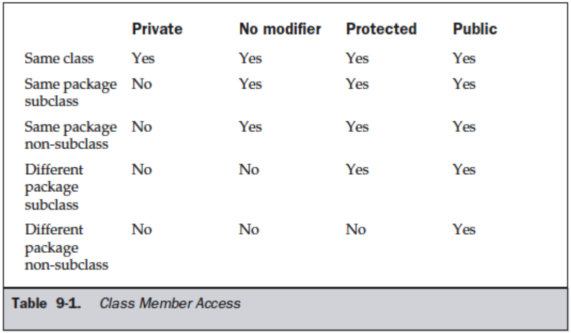
Importing Packages

- Use **import** statement to bring certain classes, or entire package, into visibility
- In Java **import** statements occur immediately following the **package** statement (if it exists) and before any class definition
- General form of **import** statement:
- **import** *pkg1[.pkg2[.classname—*]]*
- **import** *java.util.Date*
- **import** *java.lang.**

Agenda

Visibility of class members

- Java addresses four categories of visibility for class members:
  - Subclasses in the same package
  - Non-subclasses in the same package
  - Subclasses in different packages
  - Classes that are neither in the same package nor subclasses



	Private	**No modifier**	**Protected**	**Public**
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Table 9-1. *Class Member Access*

[2]

Your Turn: Time to hear from you!

[1]

[1] https://fensafitters.files.wordpress.com/2013/07/3d095.jpg

References

P.J. Deitel, H.M. Deitel
Java How to program, 10th Edition .
Prentice Hall, 2015.

Herbert Schildt
The complete reference Java2, 5th Edition .
McGraw-Hill/Osborne, 2002.