Lập trình hướng đối tượng

Biên soạn: Nguyễn Thị Tuyết Hải

Email: tuyethai@ptithcm.edu.vn

Liên lạc với giảng viên

Email: tuyethai@ptithcm.edu.vn

Tiêu đề (Subject): phải bắt đầu với **[2024-1 LTHĐT]**

Email không ghi rõ tiêu đề trên sẽ được bộ lọc tự động xoá

Đề cương môn học

Lý thuyết: 30

Bài tập, thực hành: 15

Chuyên cần (10%) Điểm danh ngẫu nhiên trong lớp, vắng 1 lần trừ 2 điểm

Bài tập (20%) Đồ án, 3 sinh viên / nhóm

Giữa kỳ (20%) Thi trắc nghiệm

Cuối kỳ (50%): Thi thực hành

Điều kiện thi cuối kỳ: điểm giữa kỳ >= 4

References

1. Core Java Volume I–Fundamentals 10th Edition, Cay S. Horstmann

Outline

Chapter 1: An introduction to Java

1.1 Java as a Programming Platform

- a pleasant syntax and comprehensible semantics
- a high-quality execution environment
- a vast library

1.2 The Java “White Paper” Buzzwords

1. Simple
2. Object-Oriented
3. Distributed
4. Robust
5. Secure
6. Architecture-Neutral
7. Portable
8. Interpreted
9. High-Performance
10. Multithreaded
11. Dynamic

1.2 The Java “White Paper” Buzzwords

1. Simple: a **cleaned-up** version of C++ syntax, **stand-alone** run on small machines

2. Object-Oriented: a programming technique focuses on the **data-objects**, and on the **interfaces** to those objects

3. Distributed: an extensive library of routines for dealing with TCP/IP protocols

4. Robust: more reliable by **early checking for possible problems, runtime checking**

5. Secure: Java browser plug-ins **no longer trust** remote code **unless** it is **digitally signed** and users **have agreed** to its execution.

1.2 The Java “White Paper” Buzzwords

6. Architecture-Neutral: The compiled code is **executable on many processors**, given the presence of the **Java runtime system**

7. Portable: portable interfaces, platform-independent

8. Interpreted: Java interpreter can execute bytecodes directly on any machine

9. High-Performance: the bytecodes can be translated at runtime into machine code for the particular CPU

10. Multithreaded

11. Dynamic: easily adapt, libraries can freely add new methods and instance variables

A Short History of Java

Version	Year	New Features	# classes, interfaces
1.0	1996	The language itself	211
1.1	1997	Inner classes	477
1.2	1998	The strictfp modifier	1,524
1.3	2000	None	1,840
1.4	2002	Assertions	2,723
5.0	2004	Generic classes, “for each” loop, varargs, autoboxing, metadata, enumerations, static import	3,279
6	2006	None	3,793
7	2011	Switch with strings, diamond operator, binary literals, exception handling enhancements	4,024
8	2014	Lambda expressions, interfaces with default methods, stream and date/time libraries	4,240
9	2017	Modules, miscellaneous language and library enhancements	6,005

Chapter 2: The Java Programming Environment

2.1.1 Downloading the JDK

1. Download: https://www.oracle.com/java/technologies/javase-downloads.html

Name	Acronym	Explanation
Java Development Kit	JDK	The software for programmers who want to **write** Java programs
Java Runtime Environment	JRE	The software for consumers who want to **run** Java programs
Server JRE		The software for running Java programs on servers
Standard Edition	SE	The Java platform for use on desktops and simple server applications
Enterprise Edition	EE	The Java platform for complex server applications
Micro Edition	ME	The Java platform for use on small devices
JavaFX		An alternate toolkit for graphical user interfaces that is included with certain Java SE distributions prior to Java 11
OpenJDK		A free and open source implementation of Java SE

2.1.2 Setting up the JDK

- Windows: launch the setup program, remove spaces in the path name
    - add the **jdk** directory to the executable path
    - https://docs.oracle.com/cd/E19182-01/820-7851/inst_cli_jdk_javahome_t/
    - https://docs.oracle.com/cd/E19182-01/821-0917/inst_jdk_javahome_t/index.html
- Linux: simply uncompress the .tar.gz file to a location of your choice
    - add a line such as the following to the end of your **~/.bashrc** or **~/.bash_profile** file: **export PATH=jdk/bin:$PATH**
- Mac: run the installer
  e.g.: /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/

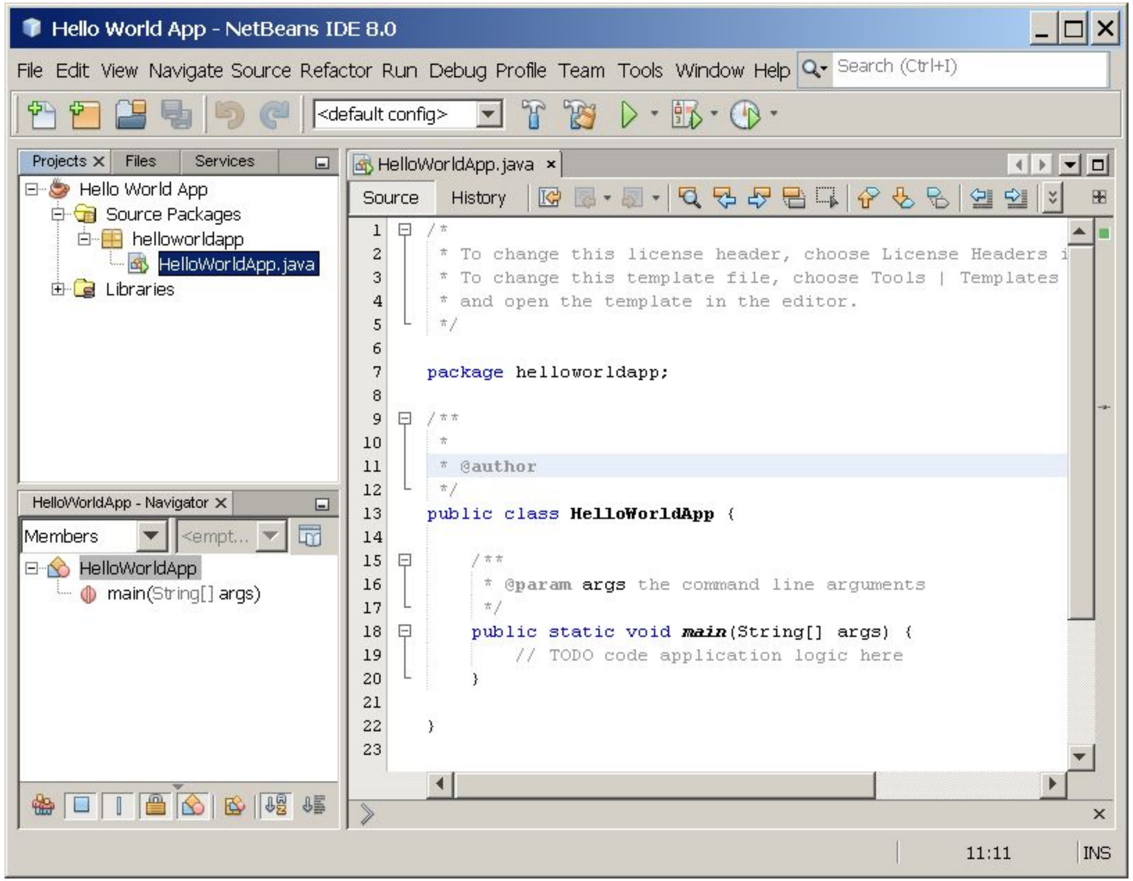
** The installation directory is denoted as **jdk***

2.2 Using the Command-Line Tools

Compile: **javac** welcome.java

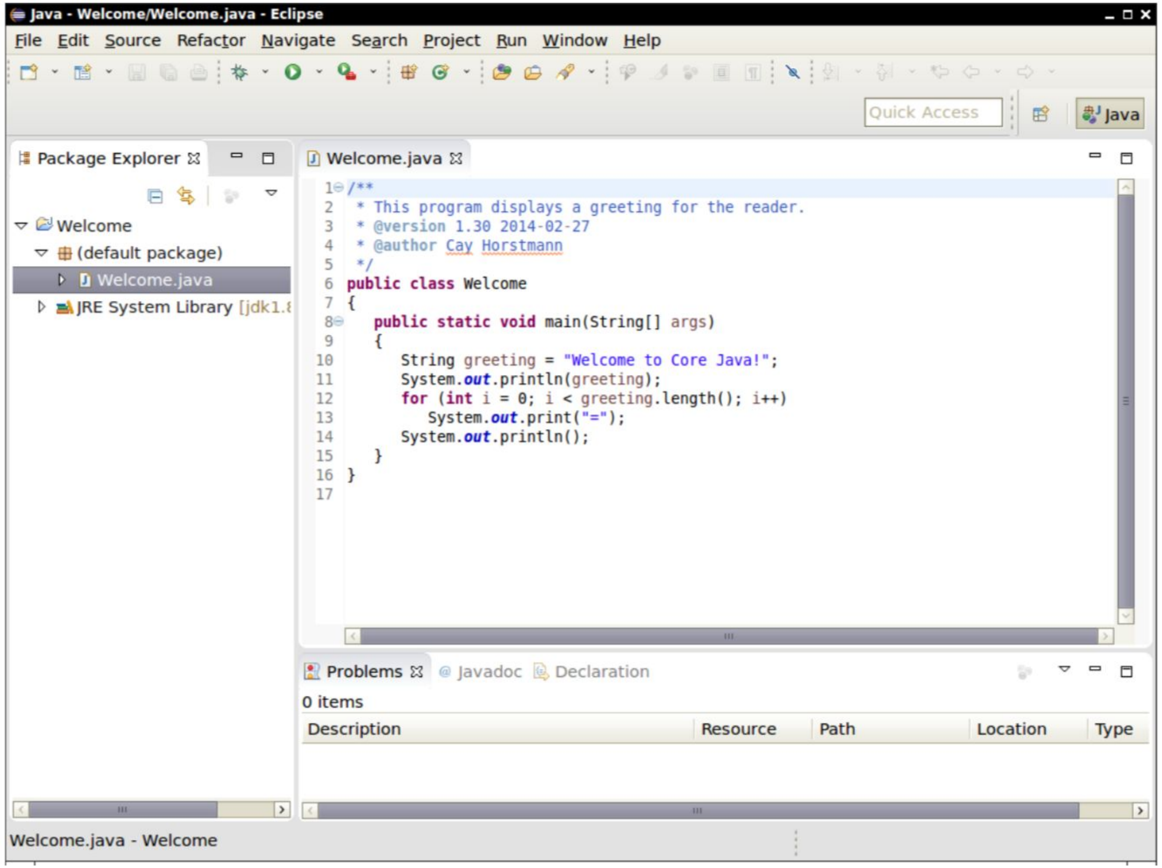
Run: **java** welcome

2.3 Using an Integrated Development Environment



```
/*
 * To change this license header, choose License Headers i
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

package helloworldapp;

/**
 *
 * @author
 */
public class HelloWorldApp {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }

}
```

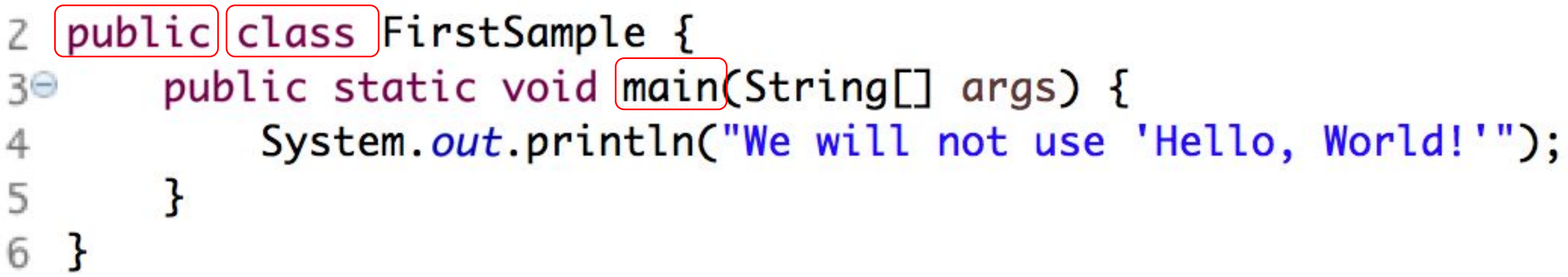
2.3 Using an Integrated Development Environment



```java
/**
 * This program displays a greeting for the reader.
 * @version 1.30 2014-02-27
 * @author Cay Horstmann
 */
public class Welcome
{
   public static void main(String[] args)
   {
      String greeting = "Welcome to Core Java!";
      System.out.println(greeting);
      for (int i = 0; i < greeting.length(); i++)
         System.out.print("=");
      System.out.println();
   }
}
```

Chapter 3: Fundamental Programming Structures in Java

3.1 A Simple Java Program

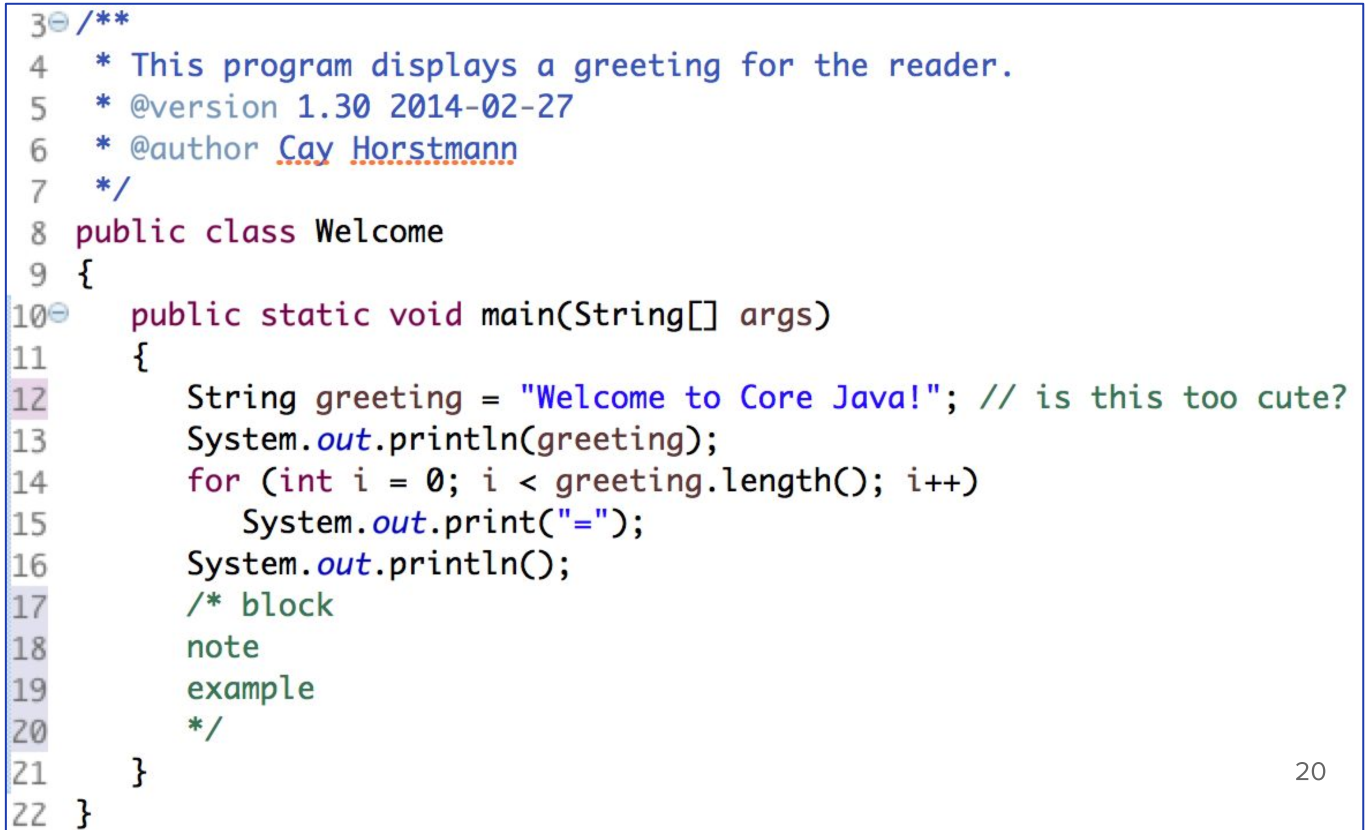


```
public class FirstSample {
    public static void main(String[] args) {
        System.out.println("We will not use 'Hello, World!'");
    }
}
```

- case sensitive
- public: access modifier - control the level of access
- class: everything in a Java program lives inside a class.
- FirstSample: class name
- file name of the source code: same as the name of the public class
- Java application must have a main method

3.2 Comments

- Not show up in the executable program
  - // each line
  - /* */: block
  - /** */: generate documentation



```java
/**
 * This program displays a greeting for the reader.
 * @version 1.30 2014-02-27
 * @author Cay Horstmann
 */
public class Welcome
{
   public static void main(String[] args)
   {
      String greeting = "Welcome to Core Java!"; // is this too cute?
      System.out.println(greeting);
      for (int i = 0; i < greeting.length(); i++)
         System.out.print("=");
      System.out.println();
      /* block
      note
      example
      */
   }
}
```

3.3 Data Types

- strongly typed language: a variable + a type
- 8 primitive types:
    - four integer types
    - two floating-point number types
    - character type
    - a boolean type

3.3.1 Integer Types

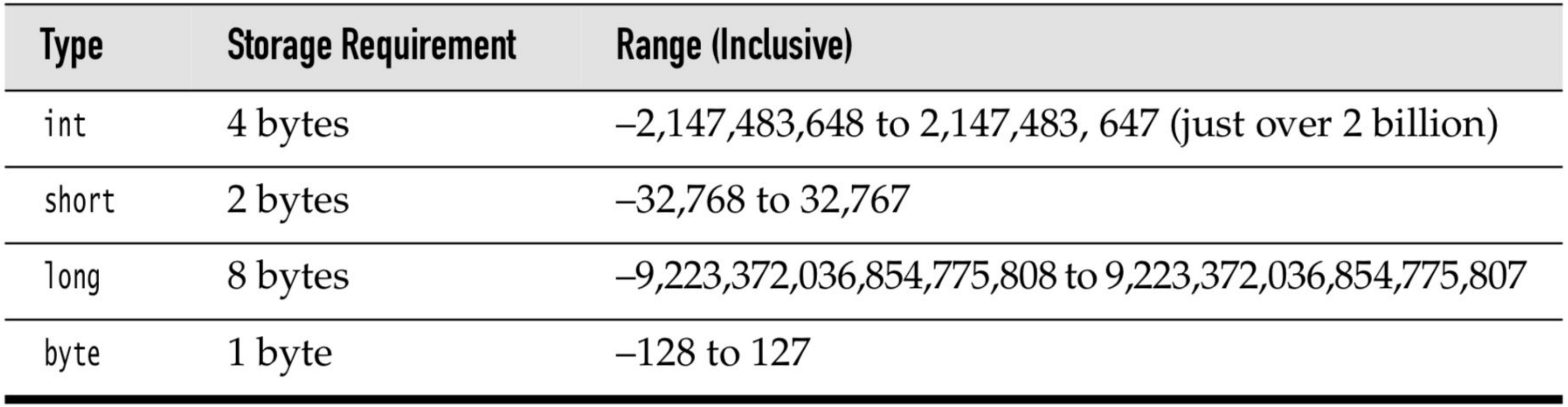


Table 3.1 Java Integer Types

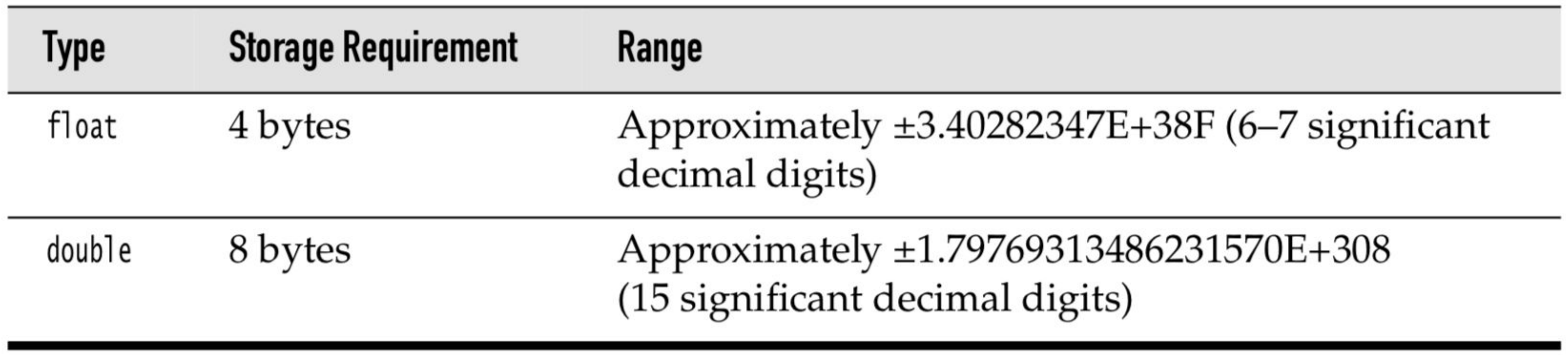
Type	Storage Requirement	Range (Inclusive)
`int`	4 bytes	–2,147,483,648 to 2,147,483, 647 (just over 2 billion)
`short`	2 bytes	–32,768 to 32,767
`long`	8 bytes	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`byte`	1 byte	–128 to 127

3.3.1 Integer Types

- Long integer: a suffix L or l, e.g., 4000000000L
- Hexadecimal number: a prefix 0x or 0X, e.g., 0xCAFE is 51966
- Octal numbers: a prefix 0, e.g., 010 is 8
- Binary numbers: a prefix 0b or 0B, e.g., 0b1001 is 9

3.3.2 Floating-Point Types

Table 3.2 Floating-Point Types



Type	Storage Requirement	Range
`float`	4 bytes	Approximately ±3.40282347E+38F (6–7 significant decimal digits)
`double`	8 bytes	Approximately ±1.79769313486231570E+308 (15 significant decimal digits)

Three special floating-point values

- Positive infinity
- Negative infinity
- NaN (not a number)

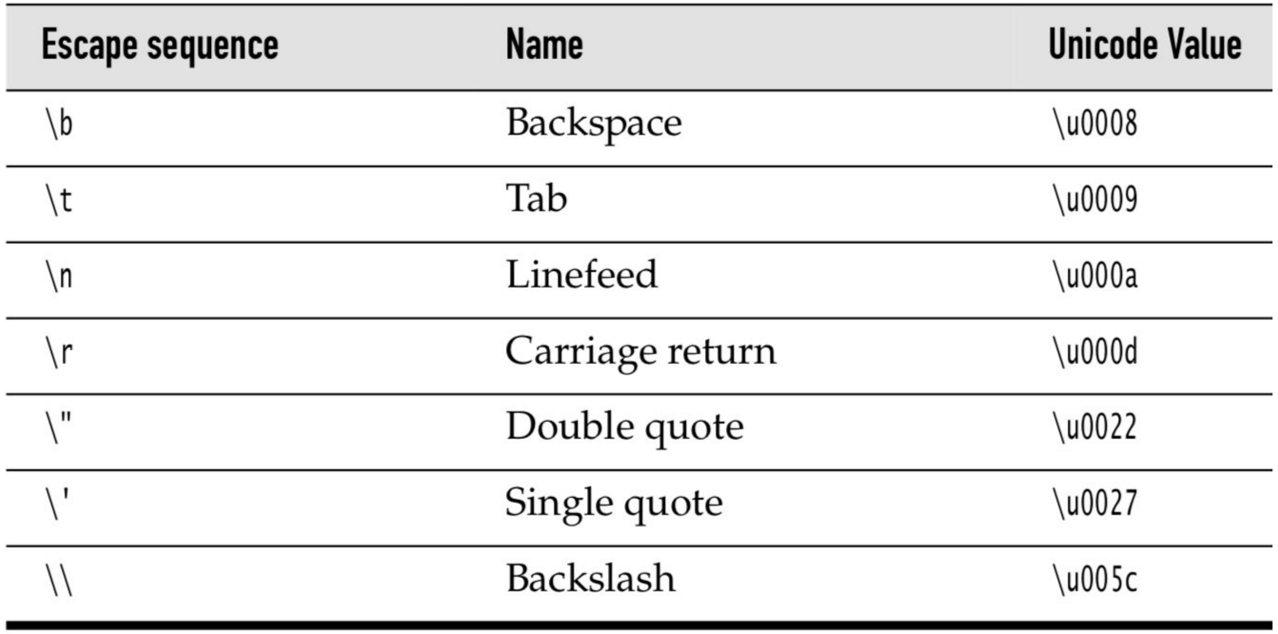
```
double inf = Double.POSITIVE_INFINITY;
System.out.println(inf + 5);
System.out.println(inf - inf); // same as Double.NaN
System.out.println(inf * -1); // same as Double.NEGATIVE_INFINITY
```

3.3.3 The char Type

Values of type **char**: \u0000 to \uFFFF

(*) char 'A' vs. string "A"

Table 3.3 Escape Sequences for Special Characters



Escape sequence	Name	Unicode Value
\b	Backspace	\u0008
\t	Tab	\u0009
\n	Linefeed	\u000a
\r	Carriage return	\u000d
\"	Double quote	\u0022
\'	Single quote	\u0027
\\	Backslash	\u005c

3.3.4 Unicode and the char Type

The char type describes a code unit in the UTF-16 encoding.

Not use the char type unless you are actually manipulating UTF-16 code units.

3.3.5 The boolean Type

- false, true
- cannot convert between integers and boolean values

3.4 Variables

- Every variable has a type
- A variable name:
  - case sensitive
  - begin with a letter
  - a sequence of letters or digits
- Common letters: 'A'–'Z', 'a'–'z', '_', '$', ...
- Common digits: '0'–'9' …
- Should not:
  - not use a reserved word
  - not use ‘$’, it is intended for names that are generated by the Java compiler and other tools.

```
double salary;

int vacationDays;

long earthPopulation;

boolean done;
```

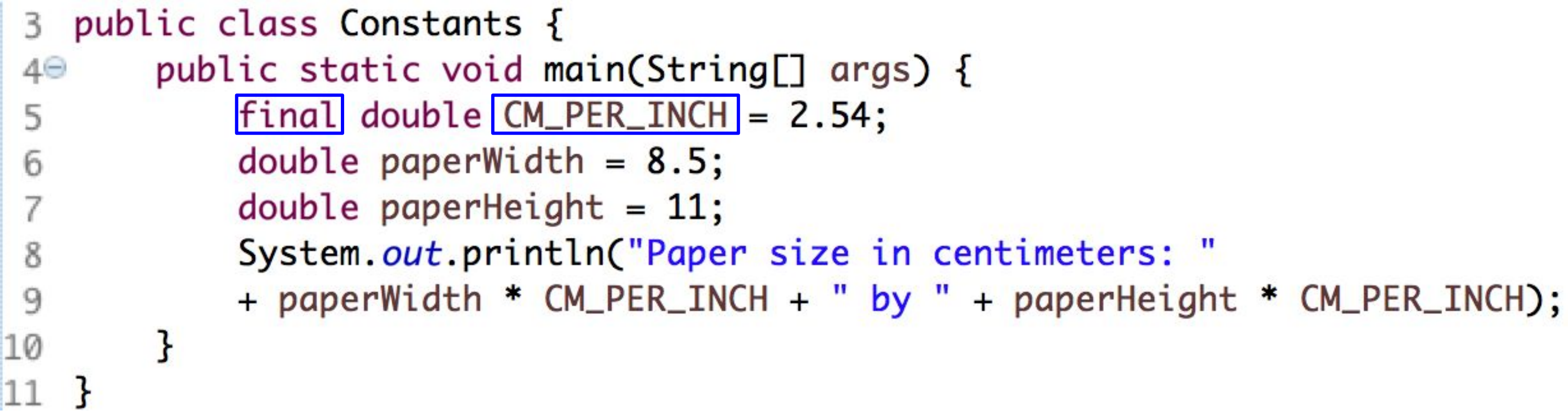
3.4.1 Initializing Variables

- Declare -> initialize -> use

```
int vacationDays;

System.out.println(vacationDays);
// ERROR--variable not initialized


int vacationDays;

vacationDays = 12;

System.out.println(vacationDays);


int vacationDays = 12;
```

3.4.2 Constants

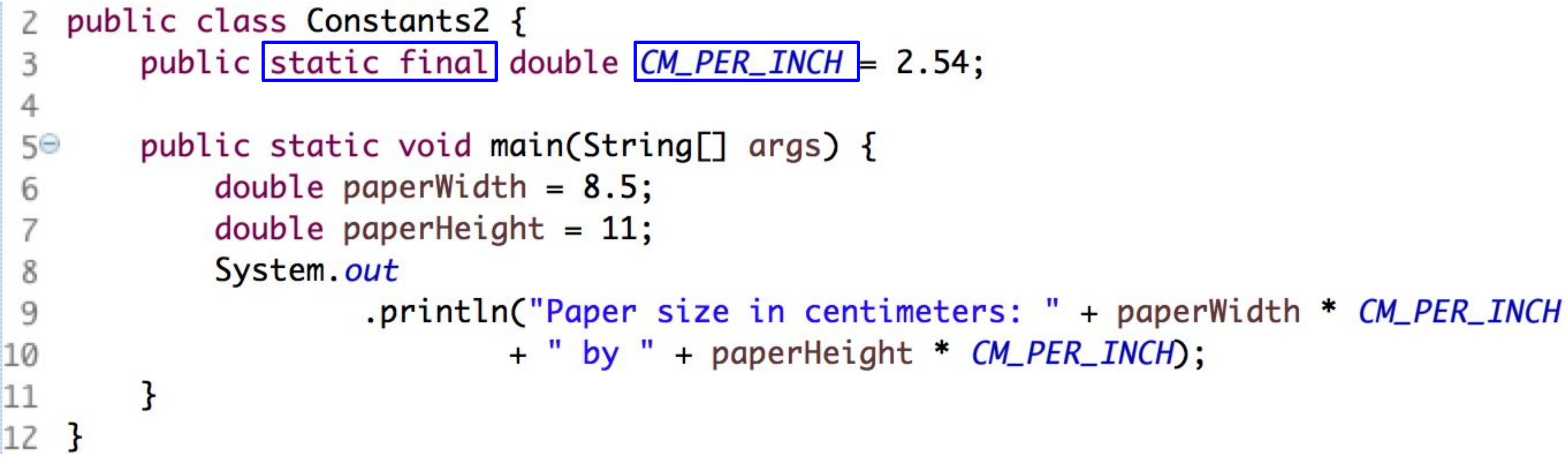
- Keyword: **final**
- Name: all uppercase



```
public class Constants {
    public static void main(String[] args) {
        final double CM_PER_INCH = 2.54;
        double paperWidth = 8.5;
        double paperHeight = 11;
        System.out.println("Paper size in centimeters: "
        + paperWidth * CM_PER_INCH + " by " + paperHeight * CM_PER_INCH);
    }
}
```

3.4.2 Constants

- Class constants
- Keyword: **static final**

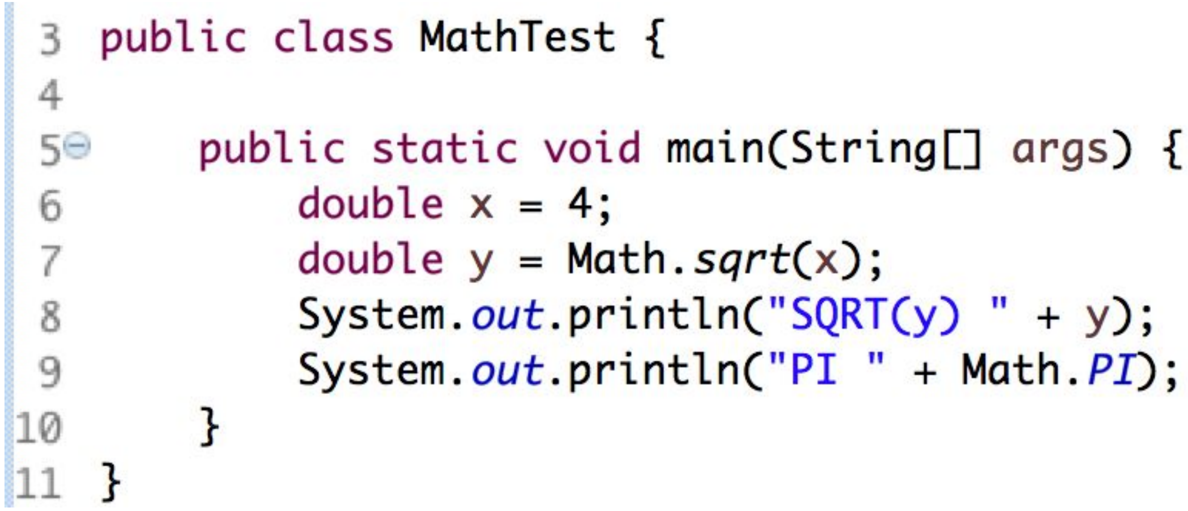


```
public class Constants2 {
    public static final double CM_PER_INCH = 2.54;

    public static void main(String[] args) {
        double paperWidth = 8.5;
        double paperHeight = 11;
        System.out
                .println("Paper size in centimeters: " + paperWidth * CM_PER_INCH
                        + " by " + paperHeight * CM_PER_INCH);
    }
}
```

3.5 Operators

3.5.1 Mathematical Functions and Constants

- The **Math** class contains a lot of mathematical functions and constants



```java
public class MathTest {

    public static void main(String[] args) {
        double x = 4;
        double y = Math.sqrt(x);
        System.out.println("SQRT(y) " + y);
        System.out.println("PI " + Math.PI);
    }
}
```

3.5.2 Conversions between Numeric Types

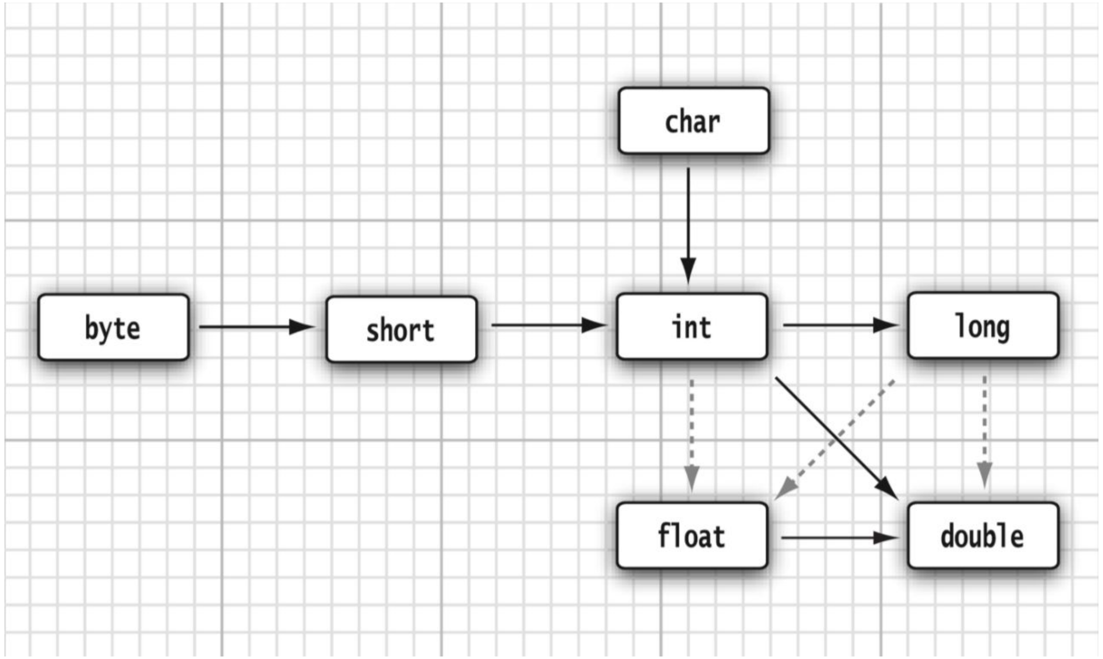


Figure 3.1 Legal conversions between numeric types (page 59):
solid arrows - conversions without information loss,
dotted arrows - conversions may lose precision

3.5.2 Conversions between Numeric Types, Casts

Loss information when converting int into float

```
int n = 123456789;
float f = n; // f is 1.23456792E8
double d = n; // d is 1.23456789E8
```

Numeric conversions are possible, but information may be lost.

```
double x = 9.997;
int nx = (int) x; // 9
```

3.5.4 Combining Assignment with Operators

A convenient shortcut for using binary operators in an assignment.

For example,

```
x+=4;
```

is equivalent to

```
x=x+4;
```

3.5.5 Increment and Decrement Operators

- ++n, n++ adds 1 to the current value of the variable n
- --n, n-- subtracts 1 from it

```
int m = 7;

int n = 7;

int a = 2 * ++m; //now a is 16, m is 8

int b = 2 * n++; //now b is 14, n is 8
```

3.5.6 Relational and boolean Operators

- == equality
- != inequality
- < less than
- > greater than
- <= less than or equal
- >= greater than or equal
- &&: "and" operator
- ||: "or" operator
- the ternary ?: operator

condition ? expression1 : expression2

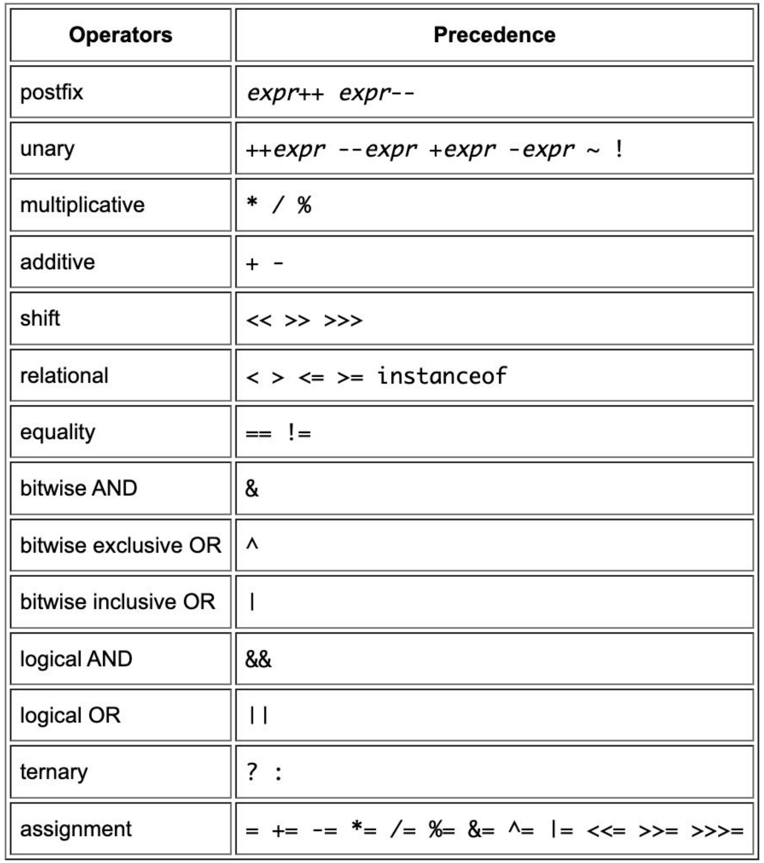
e.g. x < y ? x : y

3.5.7 Bitwise Operators

- & ("and")
- | ("or")
- ^ ("xor")
- ~ ("not")
- >> and << : shift a bit pattern to the right or left.

3.5.8 Parentheses and Operator Hierarchy

If no parentheses, operations are performed in the hierarchical order.



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	= += -= *= /= %= &= ^= |= <<= >>= >>>=

3.5.9 Enumerated Types

A variable should only hold a restricted set of values -> enumerated type

E.g.

```
enum Size { SMALL, MEDIUM, LARGE, EXTRA_LARGE };

Size s = Size.MEDIUM;
```

Quizzes on operators

Q1. If p=10, then the output of the expression c= p++ *7 is ???

Quizzes on operators

Q2. Find the output value of the following codes:

int a = 10;

int b = ***a-- + --a * 10***;

System.out.println("a " + a + "b " + b);

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	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Quizzes on operators

Q3. Find the output value of the following codes:

```
int a = 15, b= 14;
boolean x = (a-- >= ++b)? true:false;
System.out.println("x " + x);
```

3.6 Strings

Each string belongs to String class

E.g.

```
String e = ""; // an empty string

String greeting = "Hello";
```

3.6 Strings

- Substrings
- Concatenation:
  - string + string;
  - string + other
  - multiple strings

```
String greeting = "Hello";
String s = greeting.substring(0, 3);

String expletive = "Expletive";
String PG13 = "deleted";
String message = expletive + PG13;

int age = 13;
String rating = "PG" + age;

String all = String.join(" / ", "S", "M", "L", "XL");
// all is the string "S / M / L / XL"
```

3.6 Strings

- Strings are immutable: cannot change a character in an existing string

```
String greeting = "Hello";

# greeting -> “Help!”
greeting[3]=”p”;
greeting[4]=”!”;

# greeting -> “Help!”
greeting = greeting.substring(0, 3) + "p!";
```

3.6 Strings

- Testing Strings for Equality
  - check value: equals, equalsIgnoreCase
  - check location: ==, same location -> same value
    - only string literals are shared location
    - strings as the result of operations like + or substring are not shared the same location

```
String greeting = "Hello";

if (greeting.equals("Hello world")) ...
if (greeting.equalsIgnoreCase("Hello world")) ...

if (greeting == "Hello") . . .
// probably true
if (greeting.substring(0, 3) == "Hel") . . .
// probably false
```

3.6 Strings

- Empty and Null Strings
  - The empty string "" is a string of length 0.
  - Null strings

```
if (str.length() == 0) …
if (str.equals("")) …
if (str == null)
```

- Code Points and Code Units
  - the char data type is a code unit for representing Unicode code points
  - most commonly used Unicode characters: a single code unit.
  - some characters require 2 code units.
  - the number of code units required for a given string
  - the number of code points

```
String greeting = "𝕆 is the set of octonions";
int n = greeting.length(); // is 26.

int cpCount = greeting.codePointCount(0,
greeting.length()); // is 25.
```

3.6 Strings

- The String API: String class has more than 50 methods, page 72
- Reading the Online API Documentation: https://docs.oracle.com/javase/8/docs/s/

3.6 Strings

Building Strings

- construct an empty string builder
- add parts to string
- call the toString method when finishing building the string
- … page 78

```
StringBuilder builder = new StringBuilder();

builder.append(ch); // appends a single character
builder.append(str); // appends a string

String completedString = builder.toString();
```

Quizzes on String

Q1. What will be output of below statements?

```
String s = "Java String Quiz";
System.out.println(s.substring(5,10));
```

Quizzes on String

Q2. What will be output of below statements?

```
String s1 = "Cat";
String s2 = "Cat";
String s3 = new String("Cat");
System.out.println("a " + (s1 == s2));
System.out.println("b " + (s2 == s3));
System.out.println("c " + (s1.substring(0,2) == s2.substring(0, 2)));
```

Quizzes on String

Q3. What will be output of below statements?

```
String x = "abc";
String y = "abc";
x = x + y;
System.out.println(x);
```

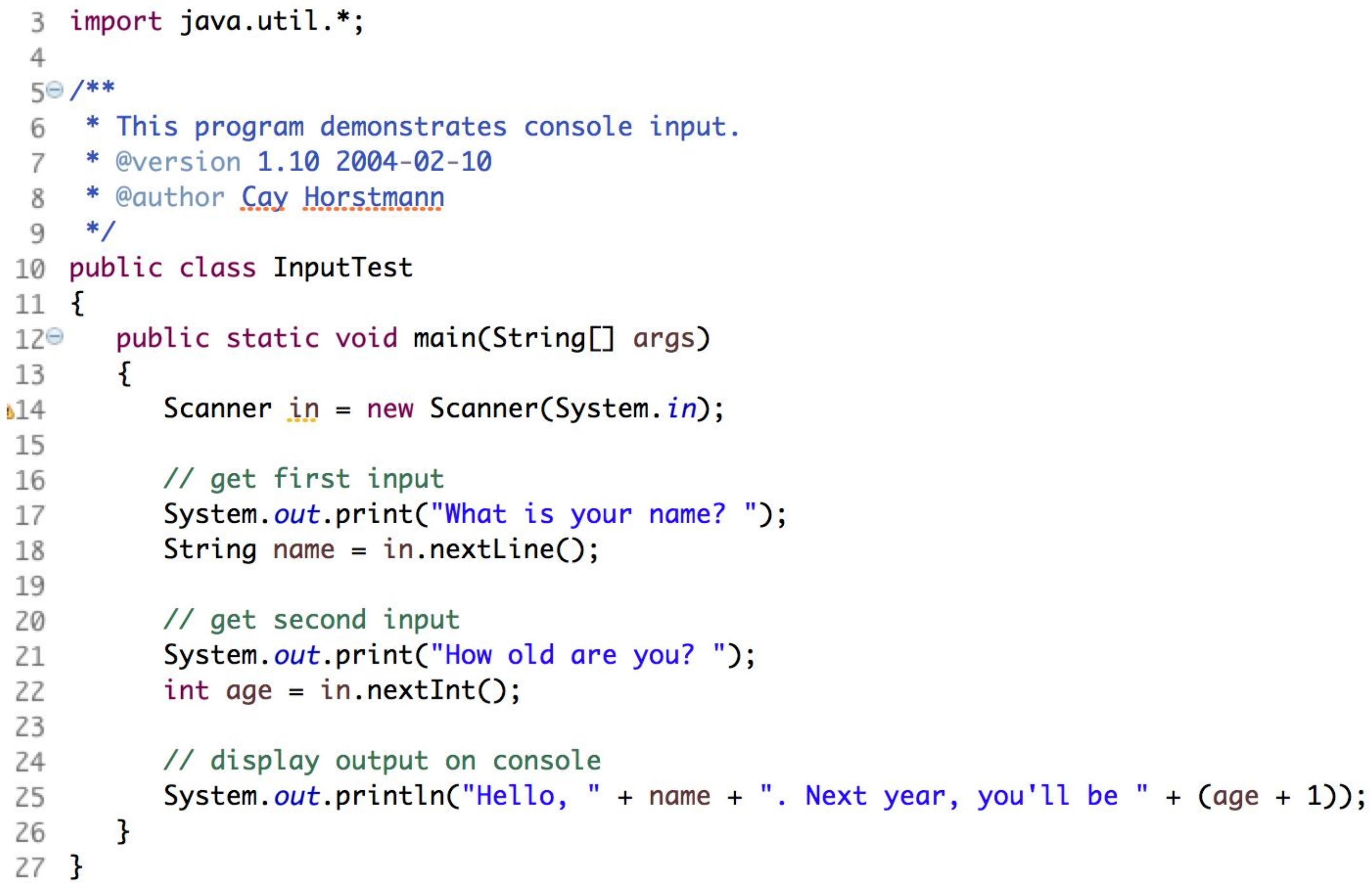
3.7 Input and Output

3.7.1 Reading Input

Reading plain text from a console (page 80)

```
import java.util.*;
Scanner in = new Scanner(System.in);
System.out.print("What is your name? ");
// the input might contain spaces
String name = in.nextLine();
// read a single word
String firstName = in.next();
System.out.print("How old are you? ");
int age = in.nextInt();
```

3.7.1 Reading Input



```
import java.util.*;

/**
 * This program demonstrates console input.
 * @version 1.10 2004-02-10
 * @author Cay Horstmann
 */
public class InputTest
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      // get first input
      System.out.print("What is your name? ");
      String name = in.nextLine();

      // get second input
      System.out.print("How old are you? ");
      int age = in.nextInt();

      // display output on console
      System.out.println("Hello, " + name + ". Next year, you'll be " + (age + 1));
   }
}
```

3.7.2 Formatting Output

```
double x = 10000.0 / 3;
System.out.println(x);
//3333.3333333333335
System.out.printf("%10.2f", x);
//    3333.33


String name = "Minh";
int age = 10;
String message = String.format("Hello, %s. Next year, you'll be %d",
name, age);
System.out.println(message);
```

3.7.2 Formatting Output

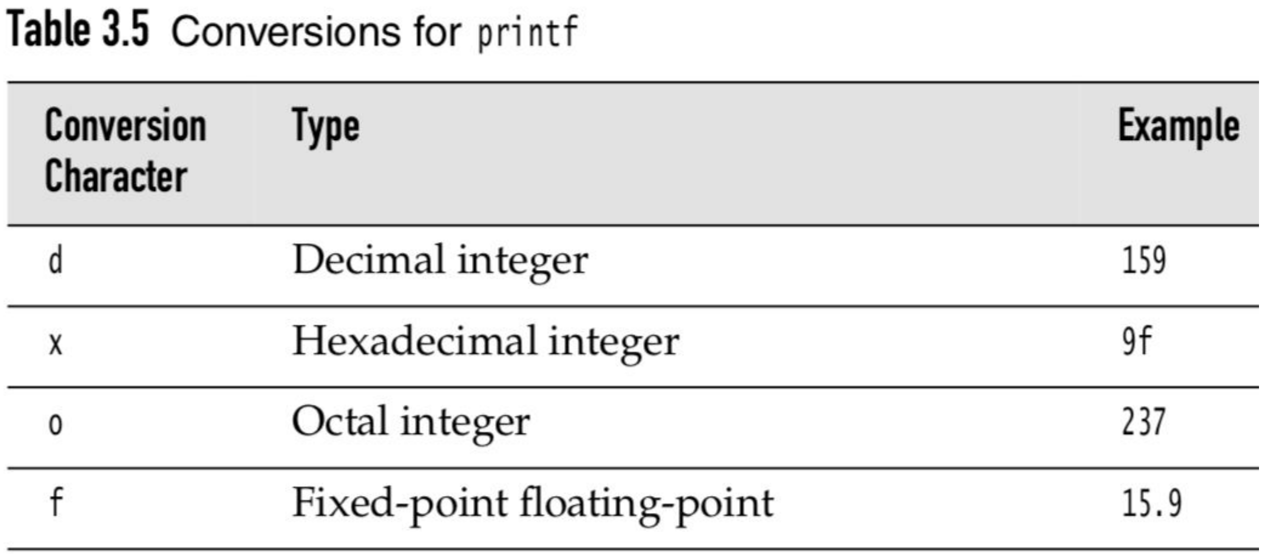


Table 3.5 Conversions for `printf`

Conversion Character	Type	Example
`d`	Decimal integer	`159`
`x`	Hexadecimal integer	`9f`
`o`	Octal integer	`237`
`f`	Fixed-point floating-point	`15.9`

3.7.3 File Input and Output

How to append text to file ?

```
String fileName = "data/example.txt";
File fileObj = new File(fileName);
try {
    FileWriter myWriter = new FileWriter(fileObj);
    myWriter.write("Test writing file\n");
    myWriter.write("Finish\n");
    myWriter.close();
  } catch (IOException e) {
    System.out.println("An error occurred.");
    e.printStackTrace();
  }
```

3.7.3 File Input and Output

```
// read from file, file path should be relative path
String fileName = "data/example.txt";
File fileObj = new File(fileName);
Scanner myReader;
try {
    myReader = new Scanner(fileObj);
    while (myReader.hasNextLine()) {
        String data = myReader.nextLine();
        System.out.println(data);
    }
    myReader.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

File path:

- Command line: the current directory of the command shell
- IDE: find the directory location

```
String dir =
System.getProperty("user.dir");
System.out.println(dir);
```

3.8 Control Flow

3.8.1 Block Scope:

- A block consists of a number of Java statements, surrounded by a pair of braces.
- Cannot redefine a variable inside a nested block

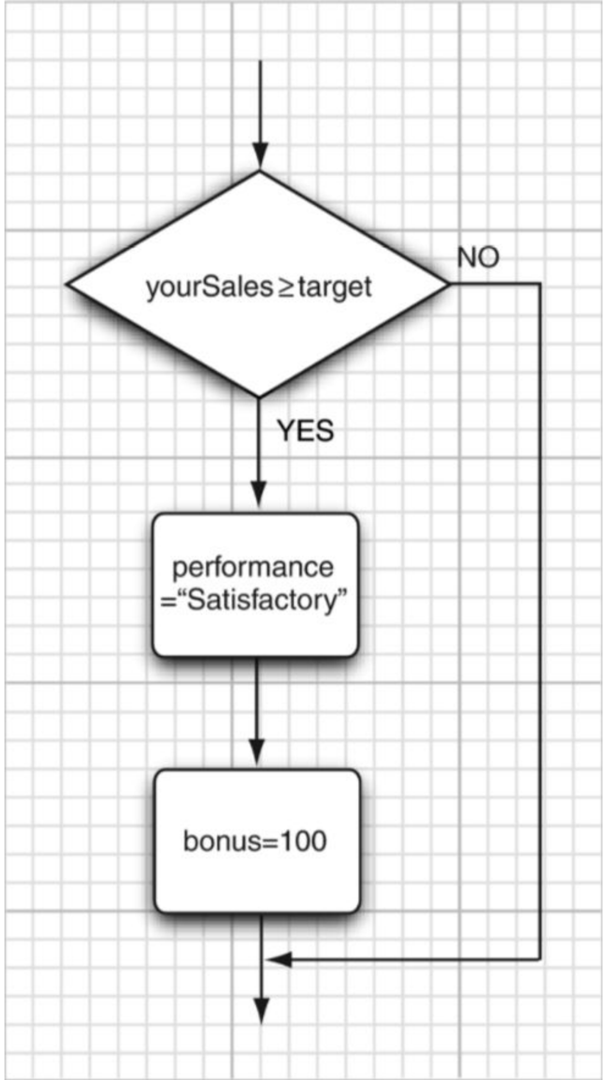
```
public static void main(String[] args) {
    int n; ...  {
        int k; …
        int n; // Error--can't redefine n in inner block
    } // k is only defined up to here
}
```

3.8 Control Flow

3.8.2 Conditional Statements

if (condition) statement

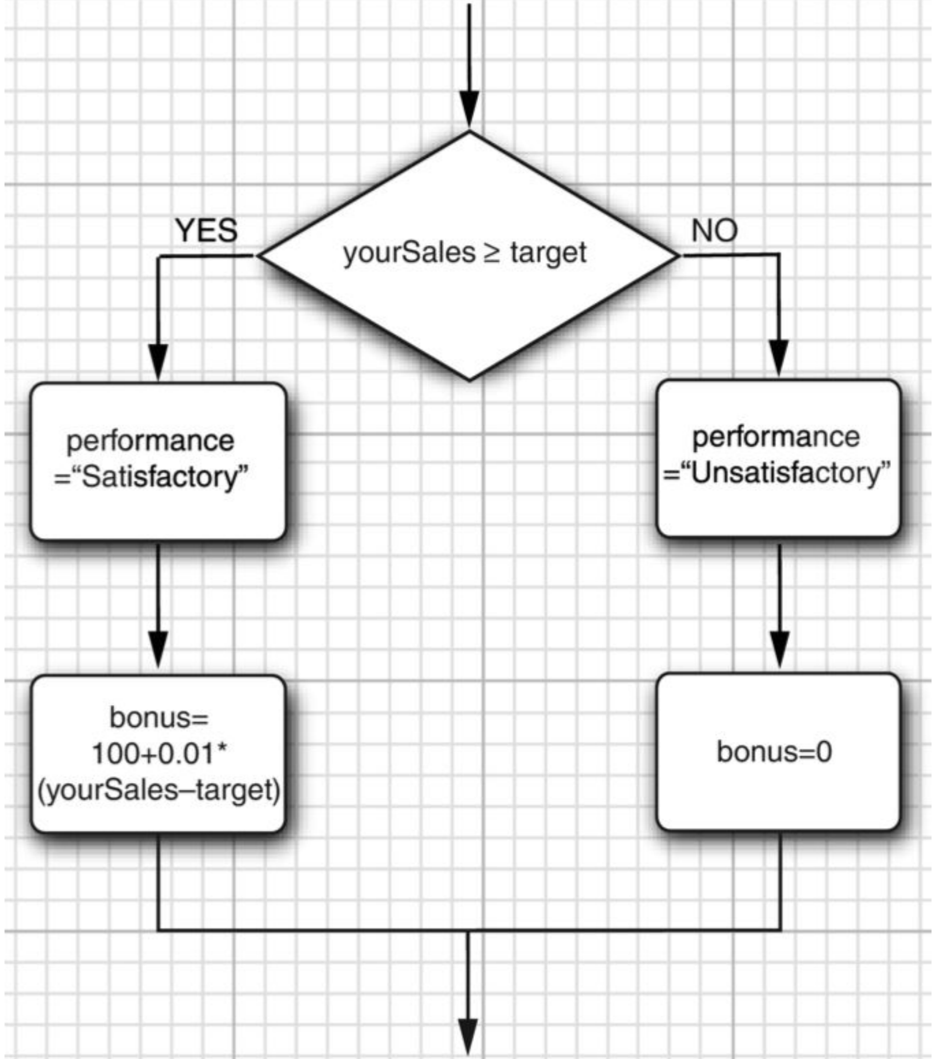
```
if (yourSales >= target) {
        performance = "Satisfactory";
        bonus = 100;
}
```



3.8 Control Flow

if (condition) statement1 else statement2

```
if (yourSales >= target) {
        performance = "Satisfactory";
        bonus = 100 + 0.01 * (yourSales - target);
}
else {
        performance = "Unsatisfactory";
        bonus = 0;
}
```

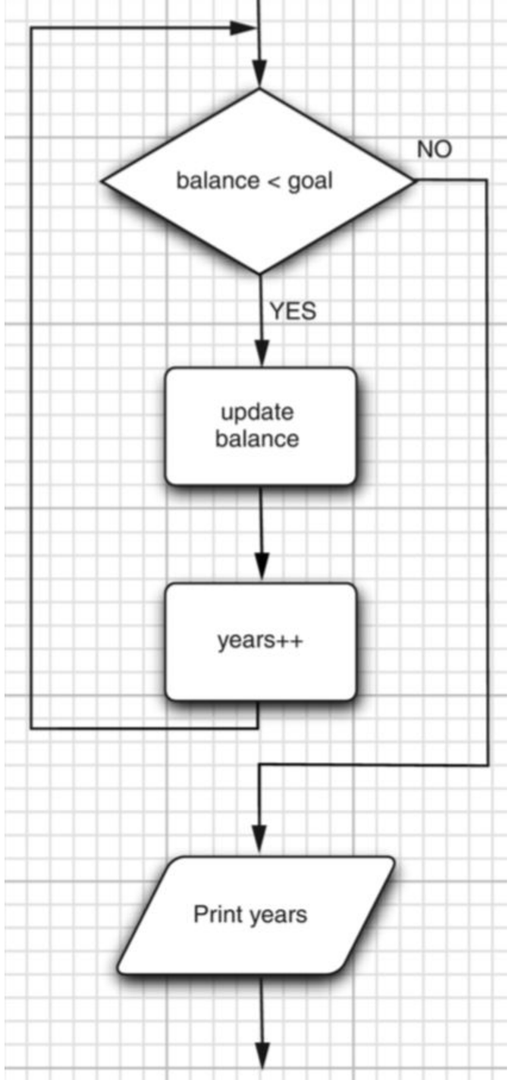

yourSales ≥ target
YES
NO
performance ="Satisfactory"
performance ="Unsatisfactory"
bonus= 100+0.01* (yourSales–target)
bonus=0

3.8 Control Flow

3.8.3 Loops

while (condition) statement

```
double balance = 1000;
double payment = 500;
double goal = 2000;
double interestRate = 20;
int years = 0;
while (balance < goal) {
        balance += payment;
        balance += balance * interestRate / 100;
        years++;
}
System.out.println(years + " years.");
```

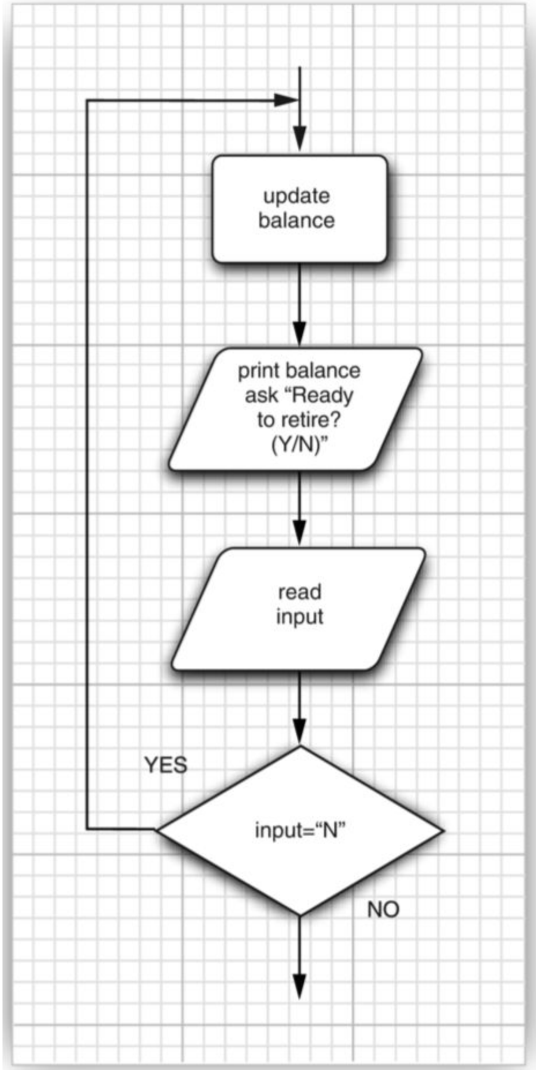


3.8 Control Flow

3.8.3 Loops

do statement while (condition)

```
do {
    balance += payment;
    balance += balance * interestRate / 100;
    years++;
}while (balance < goal);
System.out.println(years + " years.");
```

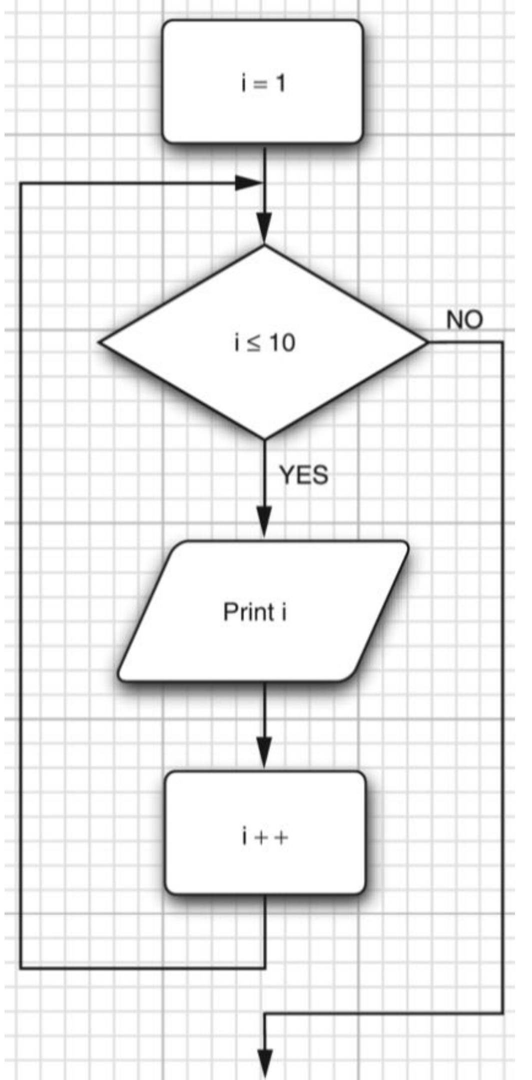


3.8 Control Flow

3.8.4 Determinate Loops

- counter initialization
- condition
- update the counter

```
for (int i = 1, j=2; i <= 10; i++, j++)
        System.out.println(i);

int i = 1, j=2;
while(i <= 10){
        System.out.println(i);
        i++;
        j++;
}
```



3.8 Control Flow

3.8.4 Determinate Loops

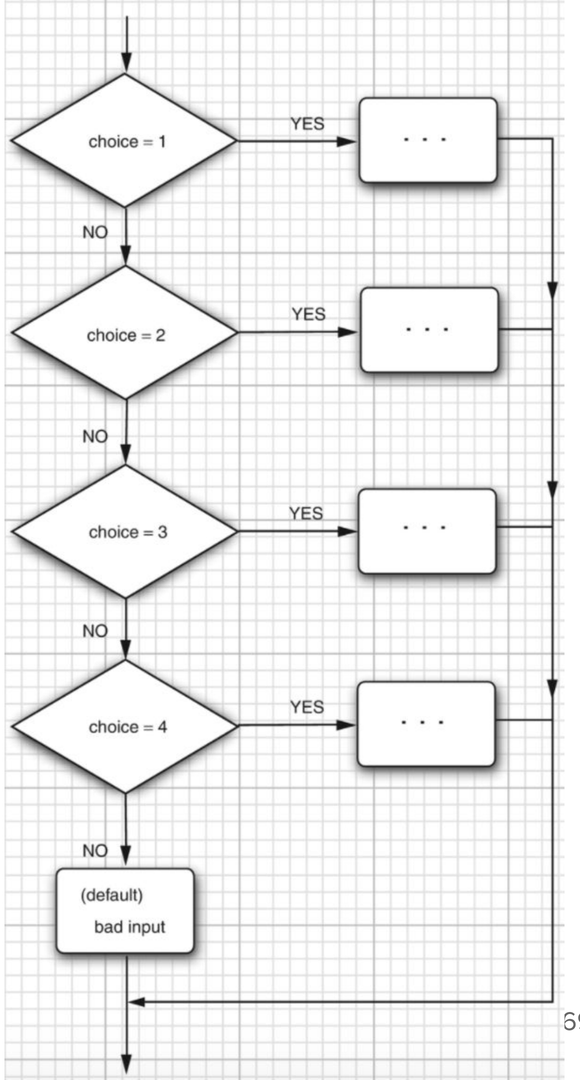
$$\frac{n \times (n-1) \times (n-2) \times \cdots \times (n-k+1)}{1 \times 2 \times 3 \times 4 \times \cdots \times k}$$

3.8 Control Flow

3.8.5 Multiple Selections- The switch Statement

```
switch (choice){
        case 1:
                . . .
                break;
        case 2:
                ...
                break;

        default:
                // bad input ...
                break;
}
```

```
if(choice==1){
        Lenh 1;
} else if (choice==2){
        Lenh 2;
} else if(choice==3){
        Lenh 3;
} else if(choice==4){
        Lenh 4;
} else {
        Bad input;
}
```



3.8 Control Flow

3.8.5 Multiple Selections—The switch Statement

A **case** label can be

- a constant expression of type char, byte, short, or int
- an enumerated constant
- a string literal

3.8 Control Flow

3.8.6 Statements That Break Control Flow

- break: breaks the regular flow of control
- continue: jumps immediately to the loop header, skipping the remainder of the current iteration

3.8 Control Flow

```
// code 1
while (years <= 10) {
     balance += payment;
     double interest = balance * interestRate / 100; balance += interest;
     years++; // 1, 2
     if (balance >= goal) break;
     System.out.println(years + " years.");
}
System.out.println(years + " years.");

// code 2
while (years <= 10) {
     balance += payment;
     double interest = balance * interestRate / 100; balance += interest;
     years++; // 1 2 3 4 5 6 7 8 9 10 11
     if (balance >= goal) continue;
     System.out.println(years + " years."); // 1 years
}
System.out.println(years + " years."); // 11 years
```

```
double balance = 1000;
double payment = 500;
double goal = 2000;
double interestRate = 20;
int years = 0;
```

3.8 Control Flow

3.8.6 Statements That Break Control Flow

```
Scanner in = new Scanner(System.in);
while (sum < goal){
        System.out.print("Enter a number: ");
        n = in.nextInt();
        if (n < 0)
                continue;
        sum += n; // not executed if n < 0
}
```

```
for (count = 1; count <= 100; count++) {
        System.out.print("Enter a number, -1 to quit: ");
        n = in.nextInt();
        if (n < 0)
                continue;
        sum += n; // not executed if n < 0
}
```

3.9 Big Numbers

- If the precision of the basic integer and floating-point types is not sufficient, you can use **BigInteger** and **BigDecimal**.
- Gets a big integer whose value equals x.

```
BigInteger a = BigInteger.valueOf(100);
```

- E.g. You need to pick 60 numbers (k=60) out of a possible 490 numbers (n=490), what is your odds of winning.

Code: v1ch03.BigIntegerTest

$$\frac{n \times (n-1) \times (n-2) \times \cdots \times (n-k+1)}{1 \times 2 \times 3 \times 4 \times \cdots \times k}$$

3.10 Arrays

- An array stores a collection of values of the same type,
- Once created, an array cannot be changed its size; if you want, use an arraylist
- Access each individual value through an integer index.
- Declare an array a of integers
- Declare and initialize
- Fill the elements in an array
- Find the number of elements of an array

```
int[] a;

// [2, 4, 6, 8]

// 0   1   2  3

int[] a = new int[100];

for (int i = 0; i < 100; i++)
        a[i] = i;

for (int i = 0; i < a.length; i++)

        System.out.println(a[i]);
```

3.10 Arrays

3.10.1 The “for each” Loop

```
for (variable : collection)
    statement
```

```
// ["2", "4", "6", "8"]

for (String element : a)
    System.out.println(element);

for (int i = 0; i < a.length; i++)
    System.out.println(a[i]);
```

3.10 Arrays

3.10.2 Array Initializers and Anonymous Arrays

- Initialize an array: *int[] smallPrimes = { 2, 3, 5, 7, 11, 13 };*
- Initialize an anonymous array: *new int[] { 17, 19, 23, 29, 31, 37 }*

3.10 Arrays

3.10.3 Array Copying

- Simple copy: both variables refer to the same array
- Copy all values of one array into a new array, **copyOf** in Arrays class
- Mutable

int[] smallPrimes = { 2, 3, 5, 7, 11, 13 };

int[] luckyNumbers = smallPrimes;
luckyNumbers[5] = 12;

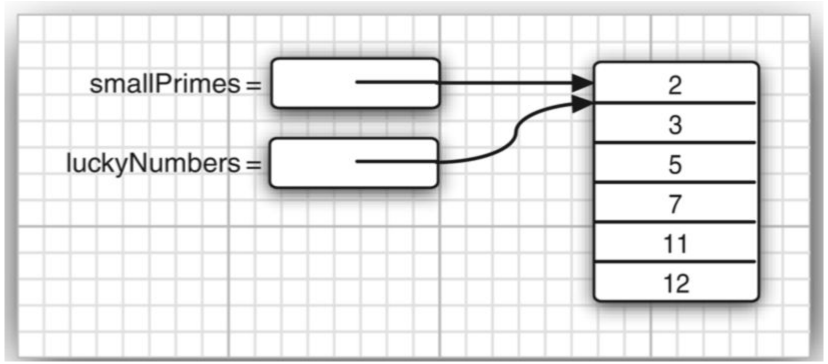
int[] copiedLuckyNumbers = Arrays.copyOf(luckyNumbers, luckyNumbers.length);

luckyNumbers == smallPrimes ? true

copiedLuckyNumbers==luckyNumbers? False

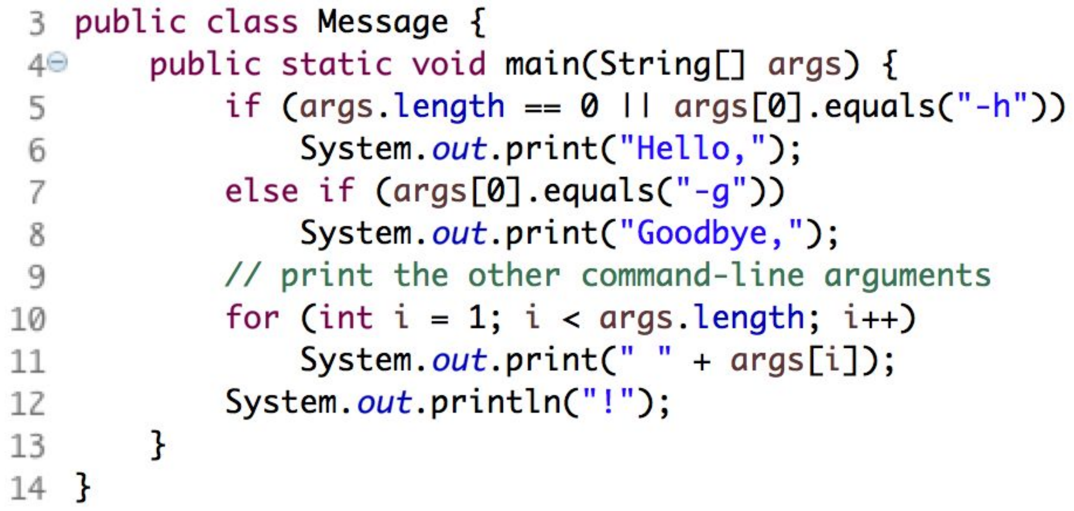
copiedLuckyNumbers[3] = 111;

luckyNumbers[3]?



3.10 Arrays

3.10.4 Command-Line Parameters



```
public class Message {
    public static void main(String[] args) {
        if (args.length == 0 || args[0].equals("-h"))
            System.out.print("Hello,");
        else if (args[0].equals("-g"))
            System.out.print("Goodbye,");
        // print the other command-line arguments
        for (int i = 1; i < args.length; i++)
            System.out.print(" " + args[i]);
        System.out.println("!");
    }
}
```

java Message -g cruel world

Goodbye, cruel world!

3.10 Arrays

3.10.5 Array Sorting

```
int[] a = new int[10000];
...
Arrays.sort(a) // QuickSort
```

E.g., v1ch03.LotteryDrawing

3.10 Arrays

3.10.6 Multidimensional Arrays

- Multidimensional arrays use more than one index to access array elements.
- Declare a multidimensional array and initialize it

```
double[][] balances;
balances = new double[NYEARS][NRATES];
```

- If you know the array elements, you can use a shorthand notation

```
int[][] magicSquare = {
{16, 3, 2, 13}, {5, 10, 11, 8}, {9, 6, 7, 12}, {4, 15, 14, 1}
};
```

- Access individual elements by two pairs of brackets: *balances[i][j]*

3.10 Arrays

3.10.6 Multidimensional Arrays

```
double[][] balances;
balances = new double[NYEARS][NRATES];
```

- Initialize the first row of the array with the initial balance

```
for (int i = 0; i < balances.length; i++) {
        for (int j = 0; j < balances[i].length; j++) {
                double oldBalance = balances[i - 1][j];
                double interest =  oldBalance *  interestRate;
                balances[i][j] = oldBalance + interest;
        }
}
```

- Print out a list of the elements:

```
        System.out.println(Arrays.deepToString(a));
```

- E.g. code v1ch03.CompoundInterest

3.10 Arrays

3.10.7 Ragged Arrays

- “ragged” arrays have different rows have different lengths.
- E.g., a triangular array
  Code: v1ch03.LotteryArray,