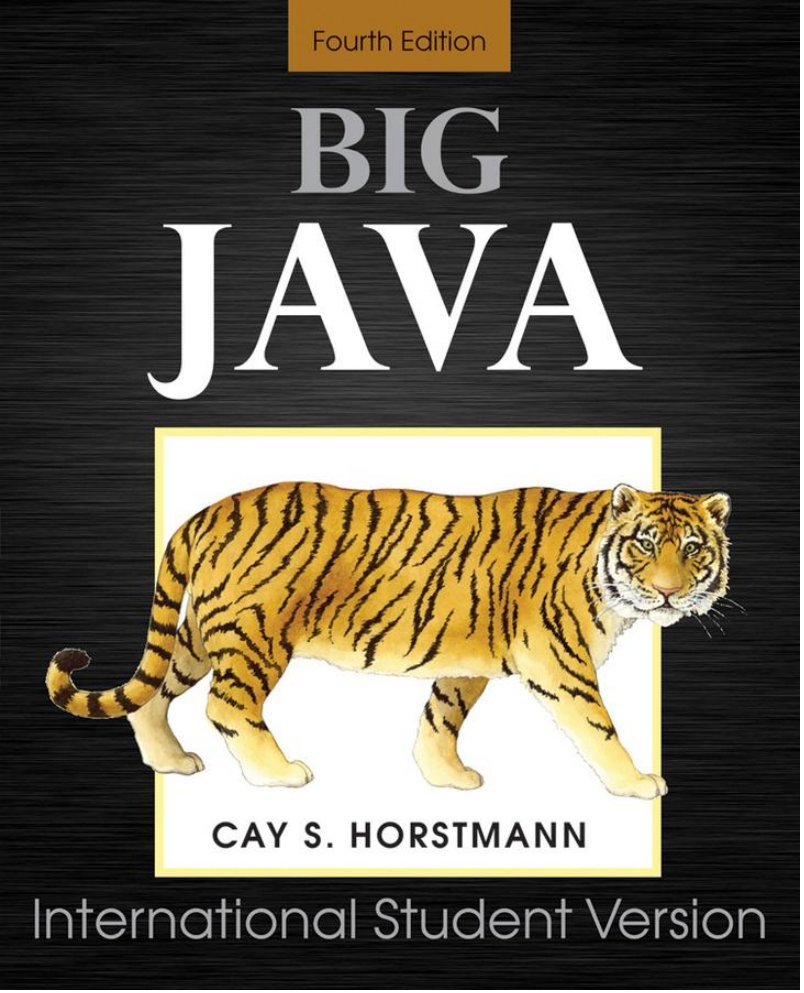


Chapter 3 – Fundamental Data Types

BFH, M. Hudritsch based on Big Java by C. Horstmann

Chapter Goals

- To understand integer and floating-point numbers
- To recognize the limitations of the numeric types
- To become aware of causes for overflow and roundoff errors
- To understand the proper use of constants
- To write arithmetic expressions in Java
- To use the `String` type to define and manipulate character strings
- To learn how to read program input and produce formatted output

Number Types

- `int`: integers, no fractional part:

```
1, -4, 0
```

- `double`: floating-point numbers (double precision):

```
0.5, -3.11111, 3.3E24, 1E-14
```

Floating Zahlen immer mit f am Schluss bezeichnen
0.5f, -3.1111f etc.

- A numeric computation overflows if the result falls outside the range for the number type:

```
int n = 1000000;
System.out.println(n * n); // prints -727379968
```

- Java: 8 primitive types, including four integer types and two floating point types

Primitive Types

Type	Description	Size
SIGNED `int`	The integer type, with range -2,147,483,648 . . . 2,147,483,647	4 bytes
SIGNED `byte`	The type describing a single byte, with range -128 . . . 127	1 byte
SIGNED `short`	The short integer type, with range -32768 . . . 32767	2 bytes
SIGNED `long`	The long integer type, with range -9,223,372,036,854,775,808 . . . 9,223,372,036,854,775,807	8 bytes
SIGNED `double`	The double-precision floating-point type, with a range of about $\pm 10^{308}$ and about 15 significant decimal digits	8 bytes
SIGNED `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 bit

Alloziert dann trotzdem mehr

Number Types: Floating-point Types

- Rounding errors occur when an exact conversion between numbers is not possible:

```
double f = 4.35;
System.out.println(100 * f); // prints 434.99999999999994
```

- Java: Illegal to assign a floating-point expression to an integer variable:

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

Self Check 3.2

Suppose you want to write a program that works with population data from various countries. Which Java data type should you use?

Answer: The world's most populous country, China, has about 1.2×10^9 inhabitants. Therefore, individual population counts could be held in an int. However, the world population is over 6×10^9. If you compute totals or averages of multiple countries, you can exceed the largest int value. Therefore, double is a better choice. You could also use long, but there is no benefit because the exact population of a country is not known at any point in time.

Self Check 3.3

Which of the following initializations are incorrect, and why

a. `int dollars = 100.0;` Verliert Wert

b. `double balance = 100;`

Answer: The first initialization is incorrect. The right hand side is a value of type `double`, and it is not legal to initialize an `int` variable with a `double` value. The second initialization is correct — an `int` value can always be converted to a `double`.

Constants: `final`

- A `final` variable is a 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

```
final double QUARTER_VALUE = 0.25;
final double DIME_VALUE = 0.1;
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;
```

Constants: `static final`

- If constant values are needed in several methods, declare them together with the instance fields of a class and tag them as `static` and `final`

Konstante abrufen, ohne Objekt zu erzeugen

- Give `static final` constants public access to enable other classes to use them

```
public class Math
{
   . . .
   public static final double E = 2.71828182845904523 54;
   public static final double PI = 3.14159265358979323846;
}

double circumference = Math.PI * diameter;
```

ch03/cashregister/CashRegister.java

```
 1  /**
 2     A cash register totals up sales and computes change due.
 3  */
 4  public class CashRegister
 5  {
 6     public static final double QUARTER_VALUE = 0.25;
 7     public static final double DIME_VALUE = 0.1;
 8     public static final double NICKEL_VALUE = 0.05;
 9     public static final double PENNY_VALUE = 0.01;
10
11     private double purchase;
12     private double payment;
13
14     /**
15        Constructs a cash register with no money in it.
16     */
17     public CashRegister()
18     {
19        purchase = 0;
20        payment = 0;
21     }
22
```

Continued

ch03/cashregister/CashRegister.java (cont.)

```
23      /**
24         Records the purchase price of an item.
25         @param amount the price of the purchased item
26      */
27      public void recordPurchase(double amount)
28      {
29         purchase = purchase + amount;
30      }
31
32      /**
33         Enters the payment received from the customer.
34         @param dollars the number of dollars in the payment
35         @param quarters the number of quarters in the payment
36         @param dimes the number of dimes in the payment
37         @param nickels the number of nickels in the payment
38         @param pennies the number of pennies in the payment
39      */
40      public void enterPayment(int dollars, int quarters,
41            int dimes, int nickels, int pennies)
42      {
43         payment = dollars + quarters * QUARTER_VALUE + dimes * DIME_VALUE
44               + nickels * NICKEL_VALUE + pennies * PENNY_VALUE;
45      }
46
```

Continued

ch03/cashregister/CashRegister.java (cont.)

```
47      /**
48         Computes the change due and resets the machine for the next customer.
49         @return the change due to the customer
50      */
51      public double giveChange()
52      {
53         double change = payment - purchase;
54         purchase = 0;
55         payment = 0;
56         return change;
57      }
58   }
```

ch03/cashregister/CashRegisterTester.java

```
1  /**
2     This class tests the CashRegister class.
3  */
4  public class CashRegisterTester
5  {
6     public static void main(String[] args)
7     {
8        CashRegister register = new CashRegister();
9
10       register.recordPurchase(0.75);
11       register.recordPurchase(1.50);
12       register.enterPayment(2, 0, 5, 0, 0);
13       System.out.print("Change: ");
14       System.out.println(register.giveChange());
15       System.out.println("Expected: 0.25");
16
17       register.recordPurchase(2.25);
18       register.recordPurchase(19.25);
19       register.enterPayment(23, 2, 0, 0, 0);
20       System.out.print("Change: ");
21       System.out.println(register.giveChange());
22       System.out.println("Expected: 2.0");
23    }
24 }
```

ch03/cashregister/CashRegisterTester.java (cont.)

Program Run:

```
Change: 0.25
Expected: 0.25
Change: 2.0
Expected: 2.0
```

Self Check 3.4

What is the difference between the following two statements?

```
final double CM_PER_INCH = 2.54;
```

and

```
public static final double CM_PER_INCH = 2.54;
```

Answer: The first definition is used inside a method, the second inside a class.

Self Check 3.5

What is wrong with the following statement sequence?

```
double diameter = . . .;
double circumference = 3.14 * diameter;
```

Answer:

1. You should use a named constant, not the "magic number" 3.13.
2. 3.14 is not an accurate representation of π.

Arithmetic Operators

- Four basic operators:
  - *addition:* `+`
  - *subtraction:* `-`
  - *multiplication:* `*`
  - *division:* `/`
- Parentheses control the order of subexpression computation:

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

- Multiplication and division bind more strongly than addition and subtraction:

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

Increment and Decrement

- `items++` is the same as `items = items + 1`
- `items--` subtracts `1` from `items`

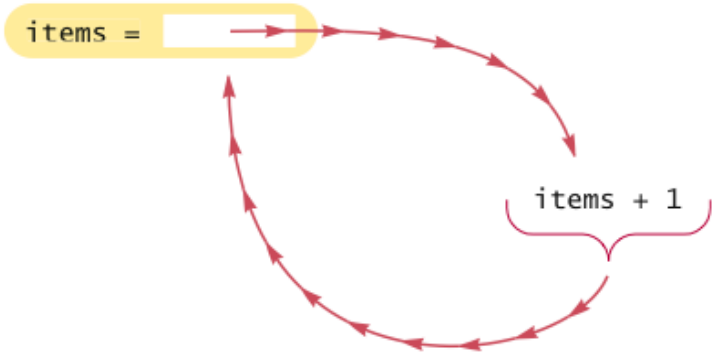


Figure 1 Incrementing a Variable

- `++items` does the increment before passing items to a func.
- `--items` does the decrement before passing items to a func.

Integer Division

- `/` is the division operator
- If both arguments are integers, the result is an integer. The remainder is discarded
- `7.0 / 4` yields `1.75`
  `7 / 4` yields `1`
- Get the remainder with `%` (pronounced "modulo")
  `7 % 4` is `3`

Wenn möglich, immer mit Multiplikation statt Division rechnen! Also statt
4 / 2
lieber
4.0 * 0.5

Integer Division

Example:

```
final int PENNIES_PER_NICKEL = 5;
final int PENNIES_PER_DIME = 10;
final int PENNIES_PER_QUARTER = 25;
final int PENNIES_PER_DOLLAR = 100;

// Compute total value in pennies
int total = dollars * PENNIES_PER_DOLLAR + quarters
   * PENNIES_PER_QUARTER + nickels * PENNIES_PER_NICKEL
   + dimes * PENNIES_PER_DIME + pennies;

// Use integer division to convert to dollars, cents
int dollars = total / PENNIES_PER_DOLLAR;
int cents = total % PENNIES_PER_DOLLAR;
```

Powers and Roots

- `Math` class: contains methods `sqrt` and `pow` to compute square roots and powers
- To compute x^n, you write `Math.pow(x, n)`
- However, to compute x^2 it is significantly more efficient simply to compute `x * x`
- To take the square root of a number, use `Math.sqrt`; for example, `Math.sqrt(x)`

Sehr aufwändig! Mehrmals überlegen, ob das wirklich notwendig ist. Leider in der Graphikprogrammierung sehr oft notwendig

- In Java,

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

can be represented as

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

Analyzing an Expression

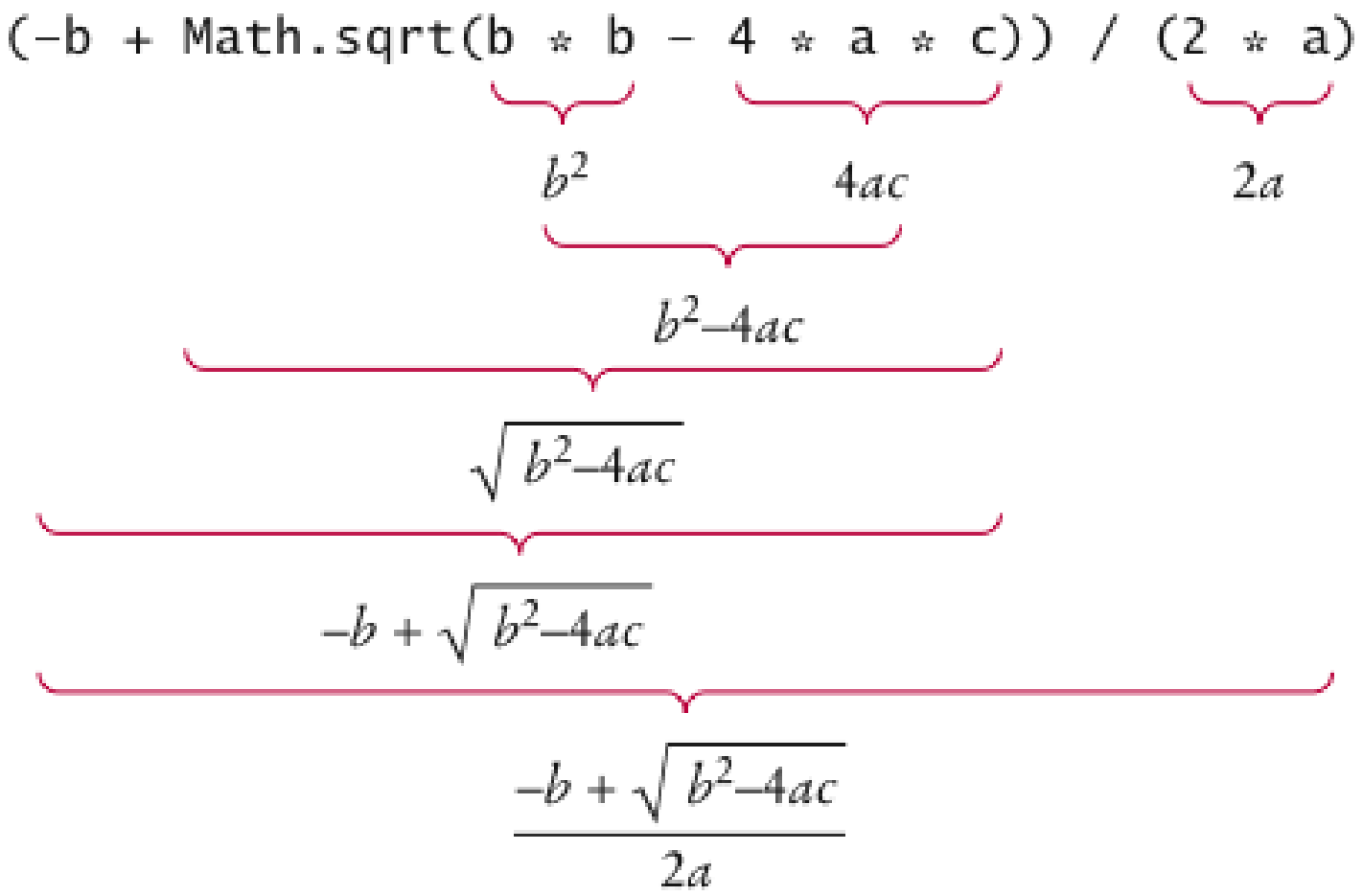


Figure 2 Analyzing an Expression

Mathematical Methods

Function	Returns
`Math.sqrt(x)`	square root
`Math.pow(x, y)`	power x^y
`Math.exp(x)`	e^x
`Math.log(x)`	natural log
`Math.sin(x)`, `Math.cos(x)`, `Math.tan(x)`	sine, cosine, tangent (x in radians)
`Math.round(x)`	closest integer to x
`Math.min(x, y)`, `Math.max(x, y)`	minimum, maximum

Cast and Round

- **Cast** converts a value to a different type:

```
double balance = total + tax;
int dollars = (int) balance;
```

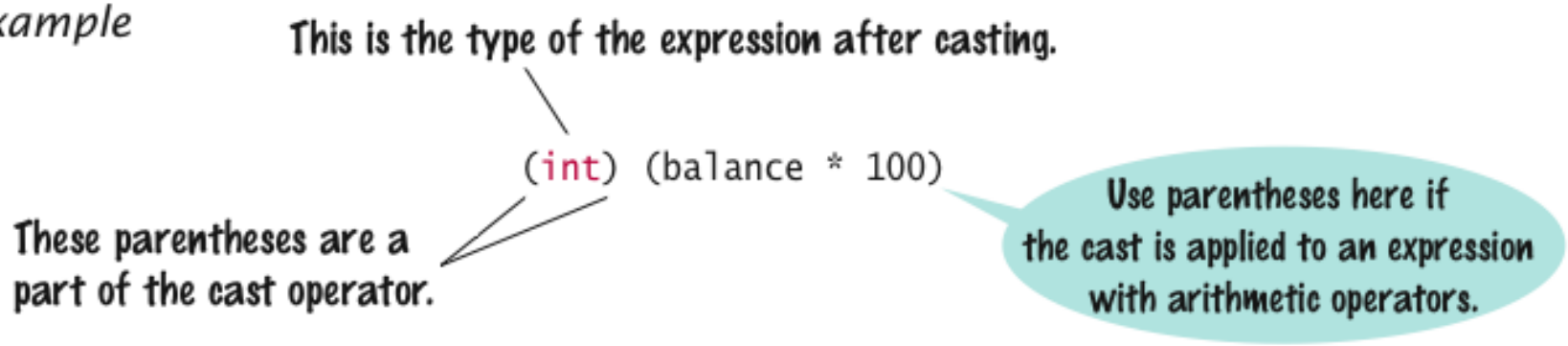
- `Math.round` converts a floating-point number to nearest integer:

```
long rounded = Math.round(balance);
// if balance is 13.75, then rounded is set to 14
```

Syntax 3.2 Cast

Syntax *(typeName) expression*

Example



Arithmetic Expressions

Table 3 Arithmetic Expressions

Mathematical Expression	Java Expression	Comments
$\frac{x + y}{2}$	`(x + y) / 2`	The parentheses are required; `x + y / 2` computes $x + \frac{y}{2}$.
$\frac{xy}{2}$	`x * y / 2`	Parentheses are not required; operators with the same precedence are evaluated left to right.
$\left(1 + \frac{r}{100}\right)^n$	`Math.pow(1 + r / 100, n)`	Complex formulas are "flattened" in Java.
$\sqrt{a^2 + b^2}$	`Math.sqrt(a * a + b * b)`	`a * a` is simpler than `Math.pow(a, 2)`.
$\frac{i + j + k}{3}$	`(i + j + k) / 3.0`	If i, j, and k are integers, using a denominator of 3.0 forces floating-point division.

Self Check 3.6

What is the value of `n` after the following sequence of statements?

```
n--;
n++;
n--;
```

Answer: One less than it was before.

Self Check 3.7

What is the value of `1729 / 100`? Of `1729 % 100`?

Answer: `17` and `29`

Self Check 3.8

Why doesn't the following statement compute the average of `s1`, `s2`, and `s3`?

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

Answer: Only `s3` is divided by `3`. To get the correct result, use parentheses. Moreover, if `s1`, `s2`, and `s3` are integers, you must divide by `3.0` to avoid integer division:

```
(s1 + s2 + s3) / 3.0
```

Self Check 3.9

What is the value of `Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2))` in mathematical notation?

Answer:

$$\sqrt{x^2 + y^2}$$

Self Check 3.10

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

Answer: When the fractional part of `x` is ≥ 0.5

Calling Static Methods

- A `static` method does not operate on an object

```
double x = 4;
double root = x.sqrt(); // Error
```

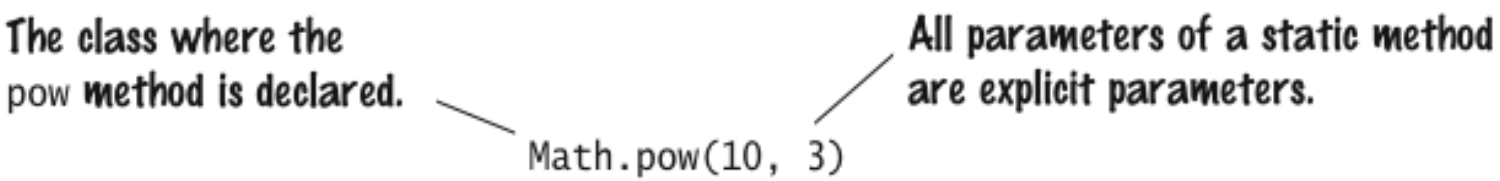
- Static methods are declared inside classes
- Naming convention: Classes start with an uppercase letter; objects start with a lowercase letter:

```
Math
System.out
```

Syntax 3.3 Static Method Call

Syntax *ClassName.methodName(parameters)*

Example



Self Check 3.12

Why can't you call `x.pow(y)` to compute x^y?

Answer: `x` is a number, not an object, and you cannot invoke methods on numbers.

Self Check 3.13

Is the call `System.out.println(4)` a static method call?

Answer: No – the `println` method is called on the object `System.out`.

Packages von Language und System müssen wir nicht selber integrieren. Deshalb ist System.out ein Objekt und println dessen Methode. Und weil das Objekt zuerst erstellt wird, ist es keine statische Methode. ...?

The `String` Class

- A string is a sequence of characters
- Strings are objects of the `String` class
- A string *literal* is a sequence of characters enclosed in double quotation marks:

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

- String *length* is the number of characters in the String
  - *Example:* `"Harry".length()` *is 5*
- Empty string: `""`

Concatenation

- Use the + operator:

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

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

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

Concatenation in Print Statements

- Useful to reduce the number of `System.out.print` instructions:

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

versus

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

Converting between Strings and Numbers

- Convert to number:

```
int n = Integer.parseInt(str);
double x = Double.parseDouble(x);
```

- Convert to string:

```
String str = "" + n;
str = Integer.toString(n);
```

- Achtung:

```
double d = Double.parseDouble(" 1000,2E12")); // Error
int i = Integer.parseInt("123‘456‘789"));     // Error
```

Kommas in den Strings erst in Punkte wandeln und die Apostrophe ganz löschen.
Erst dann parsen!

Substrings

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

- Supply start and “past the end” position
- First position is at `0`

Figure 3 String Positions

Substrings

- `String sub2 = greeting.substring(7, 12); // sub2 is "World"`
- Substring length is “past the end” - start

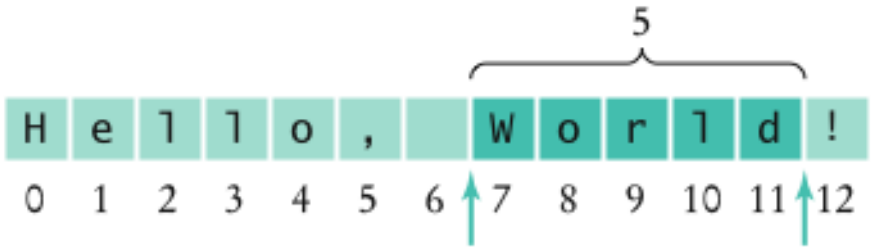


Figure 4 Extracting a Substring

German Keyboard

A German Keyboard

Thai Alphabet

The Thai Alphabet

Chinese Ideographs

Chinese Ideographs

Reading Input

- `System.in` has minimal set of features — it can only read one byte at a time
- In Java 5.0, Scanner class was added to read keyboard input in a convenient manner
- ```
  Scanner in = new Scanner(System.in);
  System.out.print("Enter quantity:");
  int quantity = in.nextInt();
  ```

  Konsolenartiges Einlesen der Eingaben
- `nextDouble` reads a `double`
- `nextLine` reads a line (until user hits Enter)
- `next` reads a word (until any white space)
  ```

ch03/cashregister/CashRegisterSimulator.java

```
 1  import java.util.Scanner;
 2
 3  /**
 4     This program simulates a transaction in which a user pays for an item
 5     and receives change.
 6  */
 7  public class CashRegisterSimulator
 8  {
 9     public static void main(String[] args)
10     {
11        Scanner in = new Scanner(System.in);
12
13        CashRegister register = new CashRegister();
14
15        System.out.print("Enter price: ");
16        double price = in.nextDouble();
17        register.recordPurchase(price);
18
19        System.out.print("Enter dollars: ");
20        int dollars = in.nextInt();
```

Continued

ch03/cashregister/CashRegisterSimulator.java (cont.)

```
21          System.out.print("Enter quarters: ");
22          int quarters = in.nextInt();
23          System.out.print("Enter dimes: ");
24          int dimes = in.nextInt();
25          System.out.print("Enter nickels: ");
26          int nickels = in.nextInt();
27          System.out.print("Enter pennies: ");
28          int pennies = in.nextInt();
29          register.enterPayment(dollars, quarters, dimes, nickels, pennies);
30
31          System.out.print("Your change: ");
32          System.out.println(register.giveChange());
33       }
34    }
```

Continued

ch03/cashregister/CashRegisterSimulator.java (cont.)

Program Run:

```
Enter price: 7.55
Enter dollars: 10
Enter quarters: 2
Enter dimes: 1
Enter nickels: 0
Enter pennies: 0
Your change: is 3.05
```

Self Check 3.16

Why can't input be read directly from `System.in`?

Answer: The class only has a method to read a single byte. It would be very tedious to form characters, strings, and numbers from those bytes.

Self Check 3.17

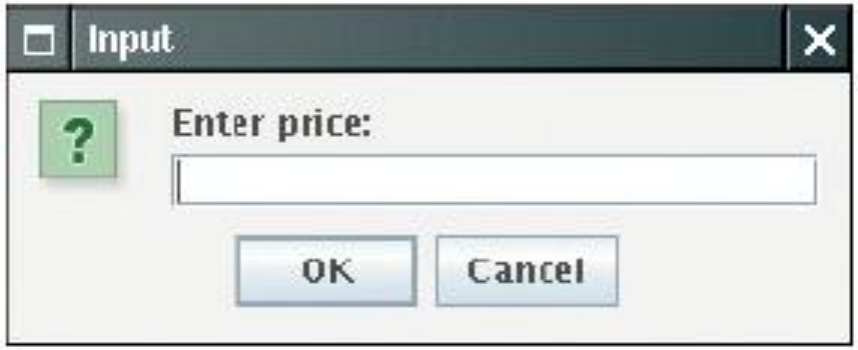
Suppose `in` is a `Scanner` object that reads from `System.in`, and your program calls

```
String name = in.next();
```

What is the value of name if the user enters `John Q. Public`?

Answer: The value is `"John"`. The `next` method reads the next *word*.

Reading Input From a Dialog Box



An Input Dialog Box

Reading Input From a Dialog Box

- `String input = JOptionPane.showInputDialog(`*`prompt`*`)`
- Convert strings to numbers if necessary:

```
int count = Integer.parseInt(input);
```

- Conversion throws an exception if user doesn't supply a number — see Chapter 11
- Add `System.exit(0)` to the `main` method of any program that uses `JOptionPane`

Exercises chapter 3

Let n be an integer and x a floating-point number. Explain the difference between

```
n = (int) x;
```

and

```
n = (int) Math.round(x);
```

Kommastellen werden abgeschnitten

Kommastellen werden auf die volle Stelle auf/abgerundet

Let n be an integer and x a floating-point number. Explain the difference between

```
n = (int) (x + 0.5);
```

and

```
n = (int) Math.round(x);
```

Zuerst wird 0.5 dazu addiert. Zweck: Die Zahl wird aufgerundet, falls notwendig
2.3 + 0.5 = 2.8 -> 2
2.5 + 0.5 = 3 -> 3

Mit der Rundenmethode wird die Logik dahinter ausgeführt

Be negativen Zahlen wird der Shortcut im Gegensatz zu Math.round() falsch rechnen

Exercises chapter 3

Your job is to transform numbers 1, 2, 3, . . ., 12 into the corresponding month names January, February, March, . . ., December. Implement a class Month whose constructor parameter is the month number and whose getName method returns the month name. *Hint:* Make a very long string "January February March . . . ", in which you add spaces such that each month name has the same length. Then use substring to extract the month you want.

Exercises chapter 3

What are the values of the following expressions? In each line, assume that

```
double x = 2.5;
double y = -1.5;
int m = 18;
int n = 4;
```

a. `x + n * y - (x + n) * y`

b. `m / n + m % n`

c. `5 * x - n / 5`

d. `Math.sqrt(Math.sqrt(n))`

e. `(int) Math.round(x)`

f. `(int) Math.round(x) + (int) Math.round(y)`

g. `1 - (1 - (1 - (1 - (1 - n))))`

Exercises chapter 3

- Write a class to compute the date of Easter Sunday. Easter Sunday is the first Sunday after the first full moon of spring. Use this algorithm, invented by the mathematician Carl Friedrich Gauss in 1800:

  1. Let `y` be the year (such as 1800 or 2001).
  2. Divide `y` by `19` and call the remainder `a`. Ignore the quotient.
  3. Divide `y` by `100` to get a quotient `b` and a remainder `c`.
  4. Divide `b` by `4` to get a quotient `d` and a remainder `e`.
  5. Divide `8 * b + 13` by `25` to get a quotient `g`. Ignore the remainder.
  6. Divide `19 * a + b - d - g + 15` by `30` to get a remainder `h`. Ignore the quotient.
  7. Divide `c` by `4` to get a quotient `j` and a remainder `k`.
  8. Divide `a + 11 * h` by `319` to get a quotient `m`. Ignore the remainder.
  9. Divide `2 * e + 2 * j - k - h + m + 32` by `7` to get a remainder `r`. Ignore the quotient.
  10. Divide `h - m + r + 90` by `25` to get a quotient `n`. Ignore the remainder.
  11. Divide `h - m + r + n + 19` by `32` to get a remainder `p`. Ignore the quotient.

  Then Easter falls on day `p` of month `n`. For example, if `y` is `2001`:

```
a = 6                g = 6              r = 6
b = 20               h = 18             n = 4
c = 1                j = 0, k = 1       p = 15
d = 5, e = 0         m = 0
```

  Therefore, in 2001, Easter Sunday fell on April 15. Write a class `Easter` with methods `getEasterSundayMonth` and `getEasterSundayDay`.

Exercises chapter 3

Schreiben Sie eine Klasse, die metrische Zahlen in km, m, cm und mm in imperiale (englische) Einheiten mile, yard, feet und inch und umgekehrt umwandeln kann.

Fliesskommazahlen müssen mit , oder . als Dezimalseparator eingegeben werden können.

Googlen Sie, wie die Nachkommastellen von imperialen Einheiten ausgegeben werden.