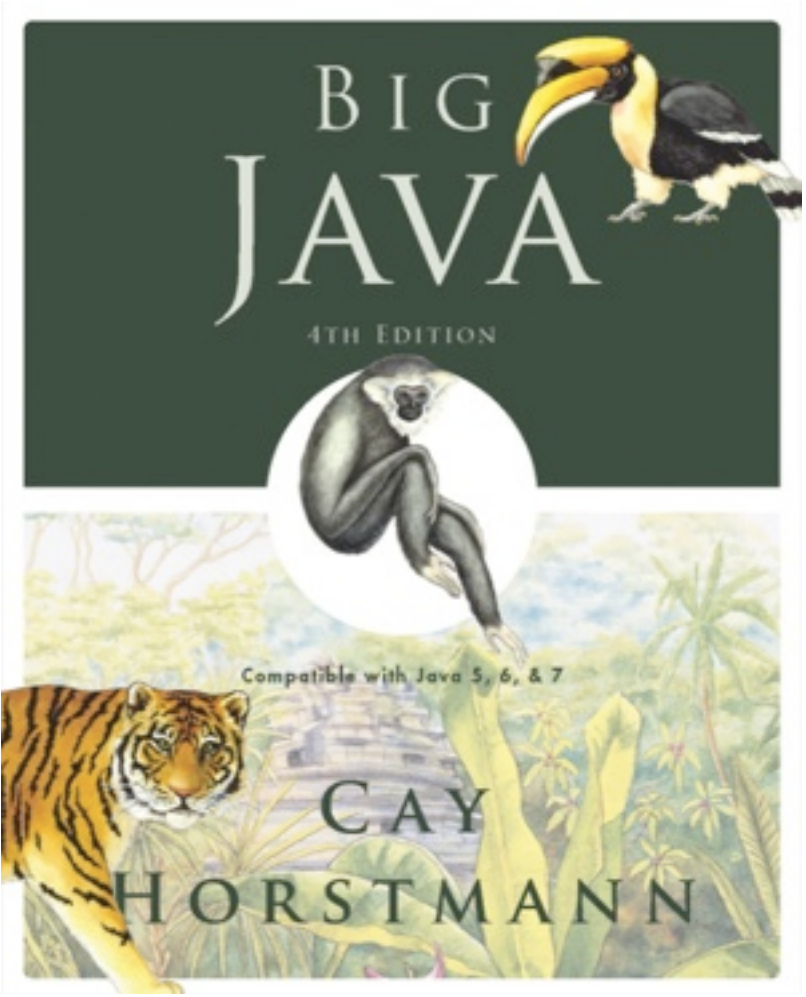


Chapter 4 – Fundamental Data Types

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`, `4.3E24`, `1E-14`

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

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
```

Big Numbers

```
BigInteger n = new BigInteger("1000000");
BigInteger r = n.multiply(n);
System.out.println(r); // Prints 1000000000000

BigDecimal d = new BigDecimal("4.35");
BigDecimal e = new BigDecimal("100");
BigDecimal f = d.multiply(e);
System.out.println(f); // Prints 435.00
```

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

```
public class Math
{
   . . .
   public static final double E = 2.7182818284590452354;
   public static final double PI = 3.14159265358979323846;
}

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

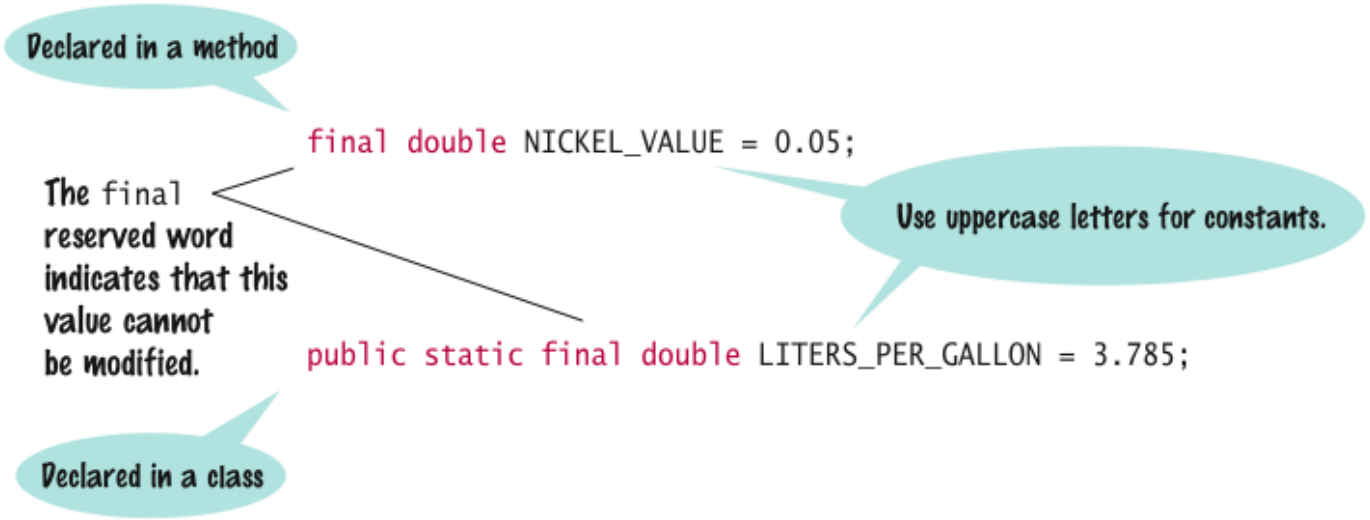
Syntax 4.1 Constant Definition

Syntax

Declared in a method: `final` *typeName variableName* = *expression*;

Declared in a class: *accessSpecifier* `static final` *typeName variableName* = *expression*;

Example



ch04/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

ch04/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

ch04/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   }
```

ch04/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 }
```

ch04/cashregister/CashRegisterTester.java (cont.)

Program Run:

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

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`

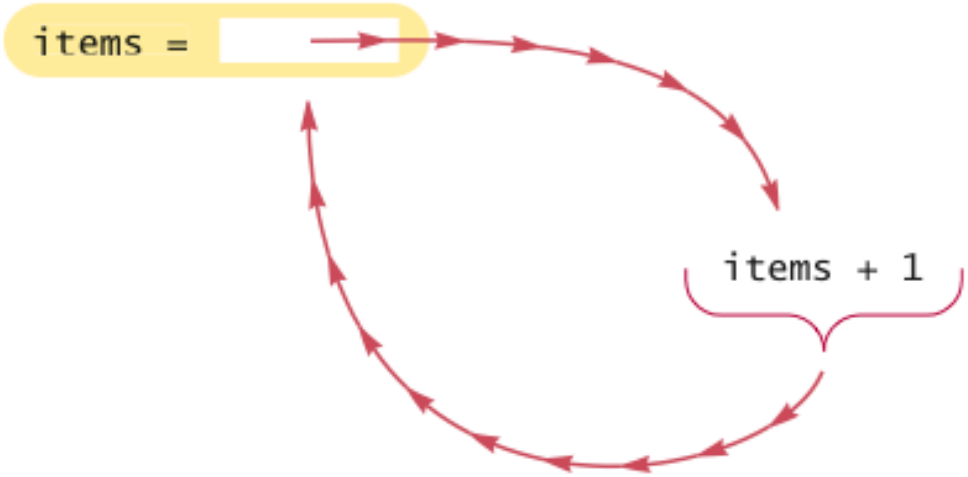


Figure 1 Incrementing a Variable

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`

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)`
- 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

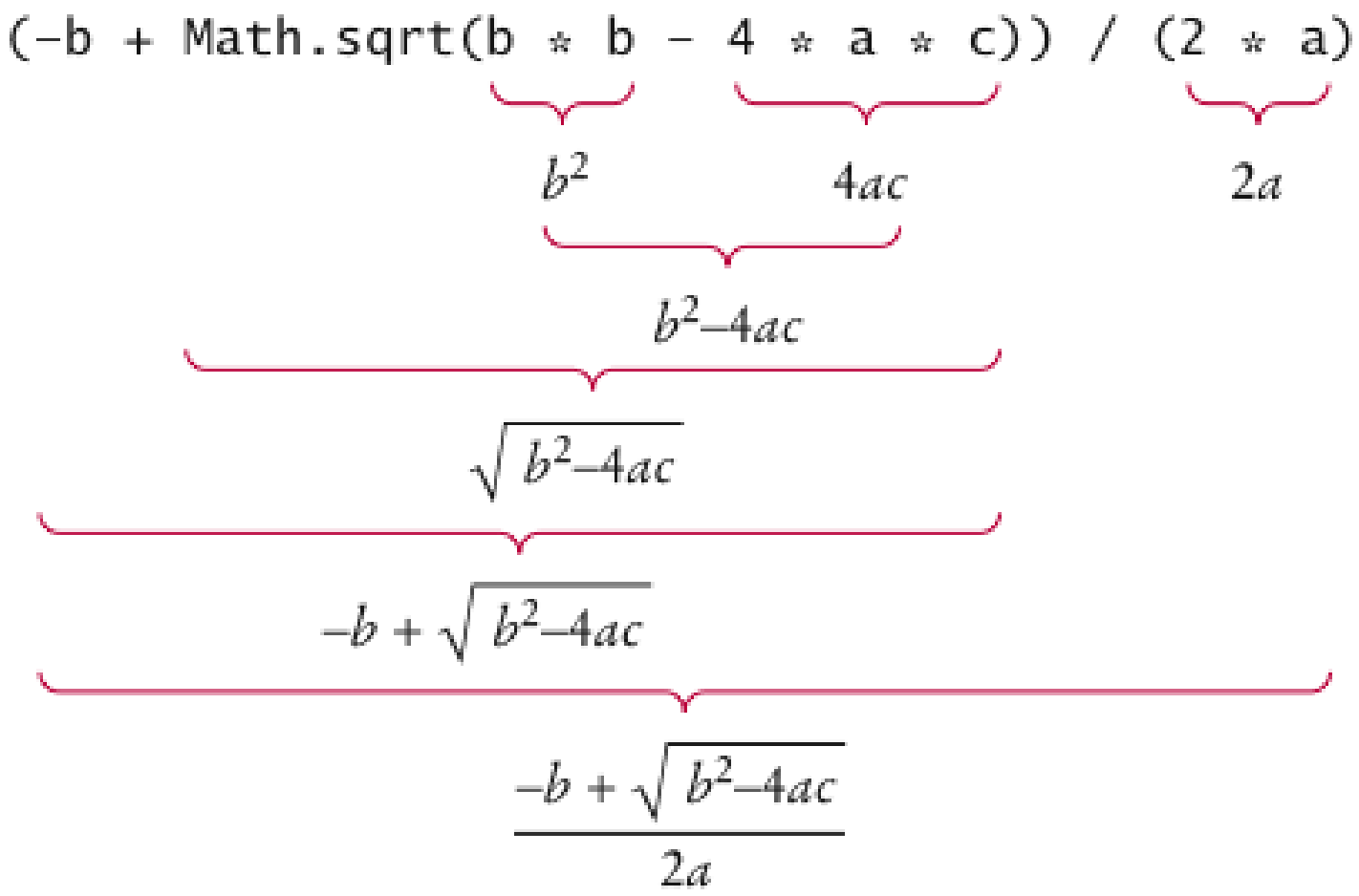


Figure 2 Analyzing an Expression

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;
```

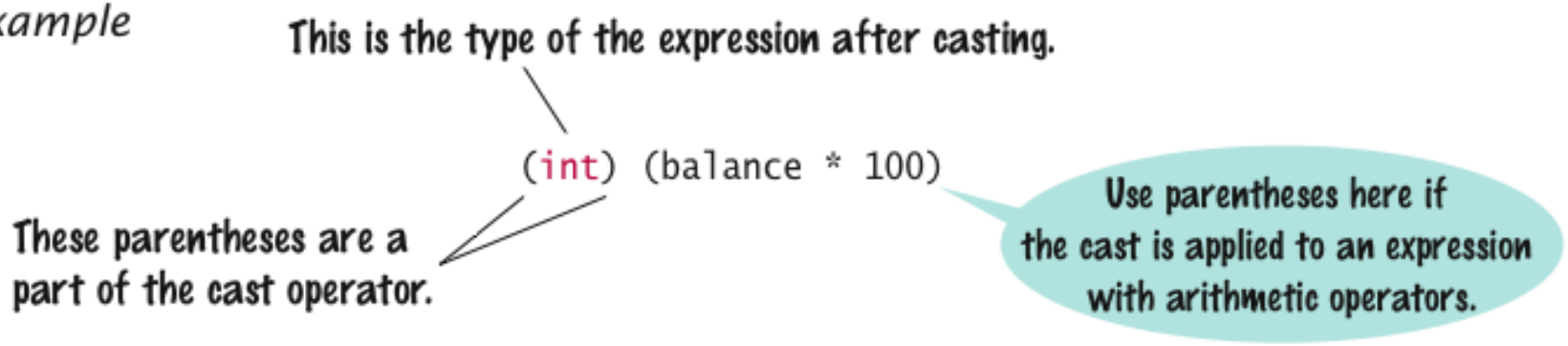
- `Math.round` converts a floating-point number to nearest integer:

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

Syntax 4.2 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.

Calling Static Methods

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

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

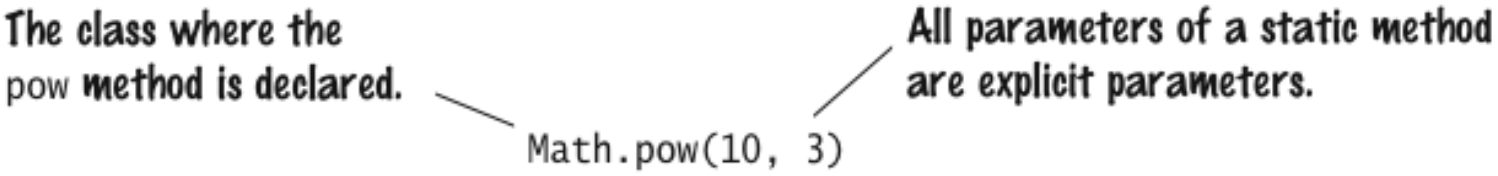
- Static methods are declared inside classes
- Naming convention: Classes start with an uppercase letter; objects start with a lowercase letter:

```
Math
System.out
```

Syntax 4.3 Static Method Call

Syntax *ClassName.methodName(parameters)*

Example



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); //str = "19"
double x = Double.parseDouble(x); //x = "3.95"
```

- Convert to string:

```
// int n = 7;

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

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

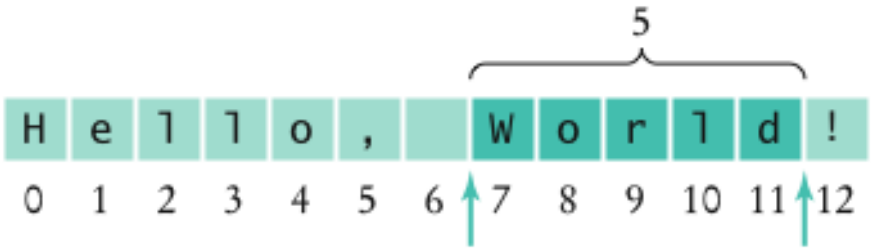


Figure 4 Extracting a Substring

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();
  ```
- `nextDouble` reads a `double`
- `nextLine` reads a line (until user hits Enter)
- `next` reads a word (until any white space)
  ```

ch04/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

ch04/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   }
```

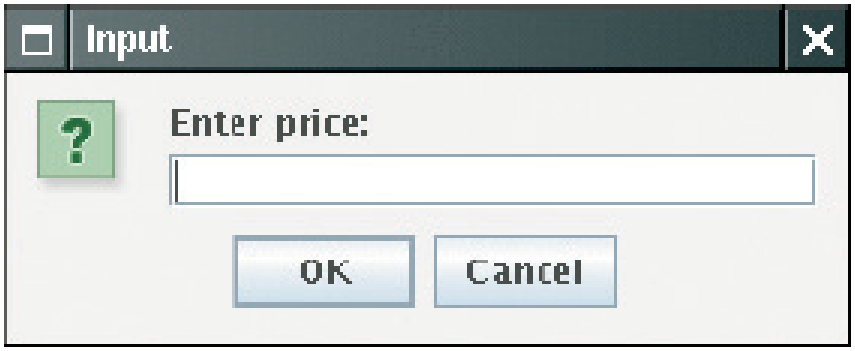
Continued

ch04/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
```

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`