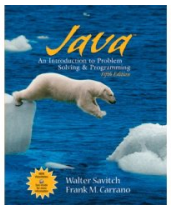
2. Primitive Types, Strings, and Console I/O

Prof. Dr. Harald Gall
Institut für Informatik
Universität Zürich
http://seal.ifi.uzh.ch



University of Zurich
Department of Informatics

s.e.a.l.

Objectives

- become familiar with Java primitive types (numbers, characters, etc.)
- learn about assignment statements and expressions
- learn about strings
- become familiar with classes, methods, and objects

Objectives, cont.

- learn about simple keyboard input and screen output
- learn about windows-based input and output using the `JOptionPane` class

© 2005 W. Savitch, Pearson Prentice Hall

Outline

- Primitive Types and Expressions
- The Class `String`
- Keyboard and Screen I/O
- Documentation and Style

© 2005 W. Savitch, Pearson Prentice Hall

Prerequisite

- familiarity with the notions of *class*, *method*, and *object*

© 2005 W. Savitch, Pearson Prentice Hall

Primitive Types and Expressions: Outline

Variables
Java Identifiers
Primitive Types
Assignment Statements
Specialized Assignment Operators
Simple Screen Output
Simple Input

Primitive Types and Expressions: Outline, cont.

Number Constants
Assignment Compatibilities
Type Casting
Arithmetic Operations
Parentheses and Precedence Rules
Increment and Decrement Operators

Variables and Values

- *Variables* store data such as numbers and letters
  - They are places to store data
  - They are implemented as memory locations
- The data stored by a variable is called its *value*
  - The value is stored in the memory location
- Its value can be changed

© 2005 W. Savitch, Pearson Prentice Hall

Variables and Values, cont.

- Example
  - `class EggBasket`

© 2005 W. Savitch, Pearson Prentice Hall

Variables and Values, cont.

- variables

```
numberOfBaskets
eggsPerBasket
totalEggs
```

- assigning values

```
eggsPerBasket = 6;
eggsPerBasket = eggsPerBasket - 2;
```


© 2005 W. Savitch, Pearson Prentice Hall

Naming and Declaring Variables

- Choose names that are helpful such as `count` or `speed`, but not `c` or `s`
- When you declare a variable, you provide its name and type

```
int numberOfBaskets,eggsPerBasket;
```

- A variable's type determines what kinds of values it can hold (`int`, `double`, `char`, etc.)
- A variable must be declared before it is used

© 2005 W. Savitch, Pearson Prentice Hall

Syntax and Examples

- syntax

type variable_1, variable_2, …;

(*variable_1* is a generic variable called a *syntactic variable*)

- examples

```
int styleChoice, numberOfChecks;
double balance, interestRate;
char jointOrIndividual;
```


© 2005 W. Savitch, Pearson Prentice Hall

Types in Java

- A *class type* is used for a class of objects and has both data and methods.
  - "`Frankie goes to Hollywood`" is a value of class type `String`
- A *primitive type* is used for simple, nondecomposable values such as an individual number or individual character.
  - `int`, `double`, and `char` are primitive types.

© 2005 W. Savitch, Pearson Prentice Hall

Naming Conventions

- Class types begin with an uppercase letter (e.g. `String`).
- Primitive types begin with a lowercase letter (e.g. `int`).
- Variables of both class and primitive types begin with a lowercase letters (e.g. `myName`, `myBalance`).
  - Multiword names are "punctuated" using uppercase letters (= CamelCase).

© 2005 W. Savitch, Pearson Prentice Hall

Where to Declare Variables

- Declare a variable
  - just before it is used or
  - at the beginning of the section of your program that is enclosed in `{}`.

```
public static void main(String[] args) {
  /* declare variables here */
  int numberOfBaskets, eggsPerBasket, totalEggs;
```


© 2005 W. Savitch, Pearson Prentice Hall

Java Identifiers

- An *identifier* is a name, such as the name of a variable
- Identifiers may contain only
  - letters
  - digits (0 through 9)
  - the underscore character (_)
  - and the dollar sign symbol ($) which has a special meaning

  but the first character cannot be a digit

© 2005 W. Savitch, Pearson Prentice Hall

Java Identifiers, cont.

- identifiers may not contain any spaces, dots (`.`), asterisks (`*`), or other characters:

  `7-11` `netscape.com` `util.*` (not allowed)

- Identifiers can be arbitrarily long
- Since Java is *case sensitive*, `stuff`, `Stuff`, and `STUFF` are different identifiers

© 2005 W. Savitch, Pearson Prentice Hall

Grammatik / EBNF

- Grammatik: Regeln zur exakten Definition einer "korrekten" Schreibweise
  - Missverständnisse ausschließen: Grammatik formal verfassen
  - Kein Interpretationsspielraum: Text ist entweder "richtig" oder "falsch"
- Gegensatz zu natürlichen Sprachen:
  - keine formale Grammatik, keine exakte Abgrenzung richtig/falsch
- Populäre Schreibweise: "Extended Backus-Naur Form" (EBNF)
  - Liste von **Produktionen** = Ersetzungsregeln
  - Jede Produktion:
    - Linke Seite = Platzhalter, Variable, **Nichtterminal**
    - Rechte Seite = Folge von Symbolen, durch die die linke Seite ersetzt werden kann
  - Symbol: Nichtterminal oder **Terminal**, letzteres kann nicht mehr ersetzt werden

© 2005 W. Savitch, Pearson Prentice Hall

Metasymbole der EBNF

⇒	trennt linke und rechte Seite
(...)	gruppiert Symbolfolgen
[...]	Option, geklammerte Symbole dürfen auch weggelassen werden
*	beliebige Wiederholung (auch null-mal)
+	ein- oder mehrmalige Wiederholung
|	trennt Alternativen

Beispiel: Grammatik für ganzzahlige Numerale:

sign ⇒ "+" | "-"
digit ⇒ "0" | ... | "9"
numeral ⇒ [sign] digit+

© 2005 W. Savitch, Pearson Prentice Hall

Keywords or Reserved Words

- Words such as `if` are called *keywords* or *reserved words* and have special, predefined meanings
- Keywords cannot be used as identifiers
- See Appendix 1 for a complete list of Java keywords
- other keywords: `int, public, class`

© 2005 W. Savitch, Pearson Prentice Hall

Primitive Types

- four integer types: `byte`, `short`, `int`, and `long`
  - `int` is most common
- two floating-point types: `float` and `double`
  - `double` is more common
- one character type: `char`
- one boolean type: `boolean`

© 2005 W. Savitch, Pearson Prentice Hall

Primitive Types, cont.

Type Name	Kind of Value	Memory Used	Size Range
`byte`	*integer*	*1 byte*	–128 to 127
`short`	*integer*	*2 bytes*	–32768 to 32767
`int`	*integer*	*4 bytes*	–2147483648 *to* 2147483647
`long`	*integer*	*8 bytes*	–9223372036854775808 *to* 9223372036854775807
`float`	*floating-point number*	*4 bytes*	$\pm 3.40282347 \times 10^{+38}$ *to* $\pm 1.40239846 \times 10^{-45}$
`double`	*floating-point number*	*8 bytes*	$\pm 1.76769313486231570 \times 10^{+308}$ *to* $\pm 4.94065645841246544 \times 10^{-324}$
`char`	*single character (Unicode)*	*2 bytes*	*all Unicode characters*
`boolean`	`true` *or* `false`	*1 bit*	*not applicable*

Display 2.2
Primitive Types

© 2005 W. Savitch, Pearson Prentice Hall

Examples of Primitive Values

- integer types

```
0  -1  365  12000
```

- floating-point types

```
0.99  -22.8  3.14159 5.0
```

- character type

```
'a'  'A'  '#'  ' '
```

- boolean type

```
true  false
```


© 2005 W. Savitch, Pearson Prentice Hall

Assignment Statements

- An assignment statement is used to assign a value to a variable.

```
answer = 42;
```

- The "equal sign" is called the *assignment operator.*
- We say:
  "The variable named `answer` is assigned a value of 42," or more simply, "`answer` is assigned 42."

© 2005 W. Savitch, Pearson Prentice Hall

Assignment Statements, cont.

- Syntax
  - `variable = expression`
- EBNF
  - *assignment ⇒ identifier "=" expression ";"*
  - where expression can be another variable, a literal or constant (such as a number), or something more complicated which combines variables and literals using operators (such as + and -)

© 2005 W. Savitch, Pearson Prentice Hall

Assignment Examples

```
amount = 3.99;
firstInitial = 'W';
score = numberOfCards + handicap;
eggsPerBasket = eggsPerBasket - 2;
```


© 2005 W. Savitch, Pearson Prentice Hall

Assignment Evaluation

- The expression on the right-hand side of the assignment operator (=) is evaluated first.
- The result is used to set the value of the variable on the left-hand side of the assignment operator.

```
score = numberOfCards + handicap;
eggsPerBasket = eggsPerBasket - 2;
```


© 2005 W. Savitch, Pearson Prentice Hall

Specialized Assignment Operators

- Assignment operators can be combined with arithmetic operators (including `-`, `*`, `/`, and `%`).

```
amount = amount + 5;
```

can be written as

```
amount += 5;
```

yielding the same results.

© 2005 W. Savitch, Pearson Prentice Hall

Simple Screen Output

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

outputs the string literal "`The count is` " followed by the current value of the variable `count`.

© 2005 W. Savitch, Pearson Prentice Hall

Simple Input

- Sometimes the data needed for a computation are obtained from the user at run time.
- Keyboard input requires

```
import java.util.*
```

at the beginning of the file.

© 2005 W. Savitch, Pearson Prentice Hall

Simple Input, cont.

- Data can be entered from the keyboard using

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

followed, for example, by

```
eggsPerBasket = keyboard.nextInt();
```

which reads one `int` value from the keyboard and assigns it to `eggsPerBasket`

© 2005 W. Savitch, Pearson Prentice Hall

Simple Input, cont.

- class EggBasket2

```
import java.util.*;
public class EggBasket2
{
    public static void main(String[] args)
    {
        int numberOfBaskets, eggsPerBasket, totalEggs;
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter the number of eggs in each basket:");
        eggsPerBasket = keyboard.nextInt();
        System.out.println("Enter the number of baskets:");
        numberOfBaskets = keyboard.nextInt();

        totalEggs = numberOfBaskets * eggsPerBasket;

        System.out.println("If you have");
        System.out.println(eggsPerBasket + " eggs per basket and");
        System.out.println(numberOfBaskets + " baskets, then");
        System.out.println("the total number of eggs is " + totalEggs);

        System.out.println("Now we take two eggs out of each basket.");

        eggsPerBasket = eggsPerBasket - 2;
        totalEggs = numberOfBaskets * eggsPerBasket;

        System.out.println("You now have");
        System.out.println(eggsPerBasket + " eggs per basket and");
        System.out.println(numberOfBaskets + " baskets.");
        System.out.println("The new total number of eggs is "
                                        + totalEggs);
    }
}
```

Sample Screen Dialog

```
Enter the number of eggs in each basket:
6
Enter the number of baskets:
10
If you have
6 eggs per basket and
10 baskets, then
the total number of eggs is 60
Now we take two eggs out of each basket.
You now have
4 eggs per basket and
10 baskets.
The new total number of eggs is 40
```

Display 2.5
Program with Keyboard Input

© 2005 W. Savitch, Pearson Prentice Hall

Number Constants

- Literal expressions such as `2`, `3.7`, or '`y`' are called *constants*
- Integer constants can be preceded by a `+` or `-` sign, but cannot contain commas
- Floating-point constants can be written
  - with digits after a decimal point or
  - using *e notation*, also called *scientific notation* or *floating-point notation*
- Examples
  - `865000000.0` can be written as `8.65e8`
  - `0.000483` can be written as `4.83e-4`
- The number in front of the `e` does not need to contain a decimal point

© 2005 W. Savitch, Pearson Prentice Hall

Assignment Compatibilities

- Java is said to be *strongly typed*
  - You can't, for example, assign a floating point value to a variable declared to store an integer.
- Sometimes conversions between numbers are possible.

```
double doubleVariable;
doubleVariable = 7;
```

is possible even if `doubleVariable` is of type `double`, for example.

© 2005 W. Savitch, Pearson Prentice Hall

Assignment Compatibilities cont.

- A value of one type can be assigned to a variable of any type further to the right

```
byte --> short --> int --> long
--> float --> double
```

but not to a variable of any type further to the left.

- You can assign a value of type `char` to a variable of type `int`

© 2005 W. Savitch, Pearson Prentice Hall

Floatingpoint

- "Gleitkommazahlen", "Fliesskommazahlen"
- Bezeichnung mit reserviertem Wort `"double"`, gleichberechtigt zu `"int"`

fpnumeral ⇒ [sign] digit+ "." digit [exponent] [doublesuffix]*
fpnumeral ⇒ [sign] "." digit+ [exponent] [doublesuffix]
fpnumeral ⇒ [sign] digit+ exponent [doublesuffix]
fpnumeral ⇒ [sign] digit+ doublesuffix
exponent ⇒ ("E" | "e") [sign] digit+
doublesuffix ⇒ "D" | "d"

- Typ einer Variable bei der Definition festgelegt:

```
int i;
double d;
final double pi = 3.1415926
```


© 2005 W. Savitch, Pearson Prentice Hall

`int` vs. `double`

- Floatingpoint-Arithmetik rechnerisch viel genauer, wozu noch ganzzahlige Arithmetik?
- `int`-Arithmetik wichtig weil...
  - `int` ist schneller
  - `double` braucht mehr Platz
  - `int` immer exakt, **`double`** nicht
    Beispiel:
    - `(1.0/x)*x - 1.0` liefert nicht immer null
- `int` wenn möglich
- `double` wenn es die Aufgabe erfordert

© 2005 W. Savitch, Pearson Prentice Hall

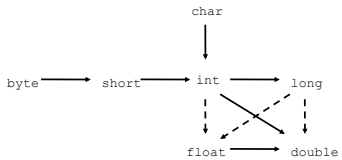
Implizite Typkonversion int→double

- Zwei Operanden gleichen Typs:
  - Operandentyp = Ergebnistyp
- Gemischte Operandentypen:
  - double ist Ergebnistyp
    - `1 + 2 → 3` `(int)`
    - `1.0 + 2 → 3.0` `(double)`
    - `1 + 2.0 → 3.0` `(double)`
    - `1.0 + 2.0 → 3.0` `(double)`

• Automatische Typumwandlung int→double:
"implizite Typkonversion"
Keine implizite Typkonversionen double→int

© 2005 W. Savitch, Pearson Prentice Hall

Legale Konvertierungen



→ ... Konvertierung ohne Informationsverlust
⇢ ... Konvertierung mit möglichem Informationsverlust

© 2005 W. Savitch, Pearson Prentice Hall

Implizite Konvertierungen

- Werden 2 Werte durch einen binären Operator verknüpft:
  - Ist einer der Operanden
    - ein `double`, so wird der andere zu `double` konvertiert,
    - ein `float`, so wird der andere zu `float` konvertiert,
    - ein `long`, so wird der andere zu `long` konvertiert
    - anderenfalls beide zu `int` konvertiert werden
  - bevor die Operation ausgeführt wird (*implicit type conversion*)

© 2005 W. Savitch, Pearson Prentice Hall

`float` and `double` Literale

- Scientific notation:
  - `98.6, 986e-1, 0.986e2, 9.86e1`
- Um `float` von `double` zu unterscheiden, muss der Literal ein "f" am Schluss stehen haben
  - `3.14159f`
- Das gleiche gilt für `double Werte. Um sie von int Werten zu unterscheiden hängt man ein "d" an.`
  - `98d`

© 2005 W. Savitch, Pearson Prentice Hall

Benutzung von `float` und `double`

```
int j = 12223334444;
float x = 122233334444.0f;

System.out.println("j = " + j);
System.out.println("x = " + x);
j = j + 1;
x = x + 1.0;
System.out.println("j = " + j);
System.out.println("x = " + x);
```

Output:

```
j = 1222333444
x = 1.22233344E9
j = 1222333445
x = 1.22233344E9
```


© 2005 W. Savitch, Pearson Prentice Hall

Implementation

```
public double convertFeetToMeters(double feet) {
      return feet * 0.3048;
   }
```

```
public static final double METERSPERFOOT = 0.3048;
```

- Besser:

```
      return feet * METERSPERFOOT;
```


© 2005 W. Savitch, Pearson Prentice Hall

Type Casting

- A *type cast* temporarily changes the value of a variable from the declared type to some other type.
- For example,

```
double distance;
distance = 9.0;
int points;
points = (int)distance;
```

(illegal without `(int)`)

© 2005 W. Savitch, Pearson Prentice Hall

Type Casting, cont.

- The value of `(int)distance` is `9`, but the value of `distance`, both before and after the cast, is `9.0`.
- Any nonzero value to the right of the decimal point is *truncated* rather than *rounded.*

```
double distance;
distance = 9.99;
int points;
points = (int)distance;
```


© 2005 W. Savitch, Pearson Prentice Hall

Initializing Variables

- A variable that has been declared, but no yet given a value is said to be *uninitialized*.
- Uninitialized class variables have the value `null`.
- Uninitialized primitive variables may have a default value.
- It's good practice not to rely on a default value.

© 2005 W. Savitch, Pearson Prentice Hall

Initializing Variables, cont.

- To protect against an uninitialized variable (and to keep the compiler happy), assign a value at the time the variable is declared.
- Examples:

```
int count = 0;
char grade = 'A';
```


© 2005 W. Savitch, Pearson Prentice Hall

Initializing Variables, cont.

- Syntax

```
type variable_1 = expression_1, variable_2 =
expression_2, …;
```

- Example

```
int number = 0, increment = 1;
```


© 2005 W. Savitch, Pearson Prentice Hall

Imprecision in Floating-Point Numbers

- Floating-point numbers often are only approximations since they are stored with a finite number of bits.
- Hence `1.0/3.0` is slight less than 1/3.
- `1.0/3.0 + 1.0/3.0 + 1.0/3.0` is less than 1.

© 2005 W. Savitch, Pearson Prentice Hall

Arithmetic Operations

- Arithmetic expressions can be formed using the `+`, `-`, `*`, and `/` operators together with variables or numbers referred to as *operands*.
  - When both operands are of the same type, the result is of that type.
  - When one of the operands is a floating-point type and the other is an integer, the result is a floating point type.

© 2005 W. Savitch, Pearson Prentice Hall

Arithmetic Operations, cont.

- Example

  If `hoursWorked` is an `int` to which the value `40` has been assigned, and `payRate` is a double to which `8.25` has been assigned

  ```
  hoursWorked * payRate
  ```

  is a `double` with a value of `330.0`

© 2005 W. Savitch, Pearson Prentice Hall

Arithmetic Operations, cont.

- Expressions with two or more operators can be viewed as a series of steps, each involving only two operands.
  - The result of one step produces one of the operands to be used in the next step.
- example

```
balance + (balance * rate)
```


© 2005 W. Savitch, Pearson Prentice Hall

Arithmetic Operations, cont.

- if at least one of the operands is a floating-point type and the rest are integers, the result will be a floating point type.
- The result is the rightmost type from the following list that occurs in the expression.

```
byte --> short --> int --> long
--> float --> double
```


© 2005 W. Savitch, Pearson Prentice Hall

The Division Operator

- The division operator (`/`) behaves as expected if one of the operands is a floating-point type.
- When both operands are integer types, the result is truncated, not rounded.
  - Hence, `99/100` has a value of `0`.

© 2005 W. Savitch, Pearson Prentice Hall

The `mod` Operator

- The `mod` (`%`) operator is used with operators of integer type to obtain the remainder after integer division
- 14 divided by 4 is 3 *with a remainder of 2*
  - Hence, `14 % 4` is equal to `2`
- The mod operator has many uses, including
  - determining if an integer is odd or even
  - determining if one integer is evenly divisible by another integer

© 2005 W. Savitch, Pearson Prentice Hall

Parentheses and Precedence

- Parentheses can communicate the order in which arithmetic operations are performed
- examples:

```
(cost + tax) * discount
 cost + (tax * discount)
```

- Without parentheses, an expressions is evaluated according to the *rules of precedence*.

© 2005 W. Savitch, Pearson Prentice Hall

Precedence Rules

Highest Precedence

First: the unary operators: +, –, ++, – –, and !
Second: the binary arithmetic operators: *, /, and %
Third: the binary arithmetic operators: + and –

Lowest Precedence

Display 2.4

Precedence Rules

© 2005 W. Savitch, Pearson Prentice Hall

Precedence Rules, cont.

- The *binary* arithmetic operators `*`, `/`, and `%`, have *lower precedence* than the *unary* operators `+`, `-`, `++`, `--`, and `!`, but have *higher precedence* than the binary arithmetic operators `+` and `-`.
- When binary operators have equal precedence, the operator on the left acts before the operator(s) on the right.

© 2005 W. Savitch, Pearson Prentice Hall

Precedence Rules, cont.

- When unary operators have equal precedence, the operator on the right acts before the operation(s) on the left.
- Even when parentheses are not needed, they can be used to make the code clearer.

```
balance + (interestRate * balance)
```

- Spaces also make code clearer

```
balance + interestRate*balance
```

but spaces do not dictate precedence.

© 2005 W. Savitch, Pearson Prentice Hall

Sample Expressions

Ordinary Mathematical Expression	Java Expression (Preferred Form)	Equivalent Fully Parenthesized Java Expression
$rate^2 + delta$	`rate*rate + delta`	`(rate*rate) + delta`
$2(salary + bonus)$	`2*(salary + bonus)`	`2*(salary + bonus)`
$\frac{1}{time + 3\ mass}$	`1/(time + 3*mass)`	`1/(time + (3*mass))`
$\frac{a - 7}{t + 9v}$	`(a - 7)/(t + 9*v)`	`(a - 7)/(t + (9*v))`

Display 2.5
Arithmetic Expressions in Java

© 2005 W. Savitch, Pearson Prentice Hall

Case Study: Vending Machine Change

- requirements
  - The user enters an amount between 1 cent and 99 cents.
  - The program determines a combination of coins equal to that amount.
  - For example, 55 cents can be two quarters and one nickel.

© 2005 W. Savitch, Pearson Prentice Hall

Case Study, cont.

- sample dialog

```
Enter a whole number from 1 to 99.
```
The machine will determine a combination of coins.

```
87 cents in coins:
  3 quarters
  1 dime
  0 nickels
  2 pennies
```


© 2005 W. Savitch, Pearson Prentice Hall

Case Study, cont.

- variables needed

```
int amount, quarters, dimes, nickels, pennies;
```


© 2005 W. Savitch, Pearson Prentice Hall

Case Study, cont.

- Algorithm - first version:

1. *Read the amount.*
2. *Find the maximum number of quarters in the amount.*
3. *Subtract the value of the quarters from the amount.*
4. *Repeat the last two steps for dimes, nickels, and pennies.*
5. *Print the original amount and the quantities of each coin.*

© 2005 W. Savitch, Pearson Prentice Hall

Case Study,cont.

- The algorithm doesn't work properly, because the original amount is changed by the intermediate steps.
  - The original value of `amount` is lost.
- Change the list of variables

```
int amount, originalAmount, quarters, dimes,
nickles, pennies;
```

- and update the algorithm

© 2005 W. Savitch, Pearson Prentice Hall

Case Study, cont.

1. Read the amount.
2. Make a copy of the amount.
3. Find the maximum number of quarters in the amount.
4. Subtract the value of the quarters from the amount.
5. Repeat the last two steps for dimes, nickels, and pennies.
6. Print the original amount and the quantities of each coin.

© 2005 W. Savitch, Pearson Prentice Hall

Case Study, cont.

- Write Java code that *implements* the algorithm written in pseudo code

© 2005 W. Savitch, Pearson Prentice Hall

Case Study, cont.

- How do we determine the number of quarters (or dimes, nickels, or pennies) in an amount?
- There are 2 quarters in 55 cents, but there are also 2 quarters in 65 cents.
- That's because

```
55 / 2 = 2 and 65 / 25 = 2
```


© 2005 W. Savitch, Pearson Prentice Hall

Case Study, cont.

- How do we determine the remaining amount?
- The remaining amount can be determined using the mod operator

```
55 % 25 = 5 and 65 % 25 = 15
```

- and similarly for dimes and nickels
- Pennies are simply `amount % 5`

© 2005 W. Savitch, Pearson Prentice Hall

Case Study, cont.

- `class ChangeMaker`

```
import java.util.*;

public class ChangeMaker
{
    public static void main(String[] args)
    {
        int amount, originalAmount,
            quarters, dimes, nickels, pennies;

        System.out.println("Enter a whole number from 1 to 99.");
        System.out.println("I will output a combination of coins");
        System.out.println("that equals that amount of change.");

        Scanner keyboard = new Scanner(System.in);
        amount = keyboard.nextInt();

        originalAmount = amount;
        quarters = amount/25;
        amount = amount%25;
        dimes = amount/10;
        amount = amount%10;
        nickels = amount/5;
        amount = amount%5;
        pennies = amount;

        System.out.println(originalAmount
                + " cents in coins can be given as:");
        System.out.println(quarters + " quarters");
        System.out.println(dimes + " dimes");
        System.out.println(nickels + " nickels and");
        System.out.println(pennies + " pennies");
    }
}
```

Sample Screen Dialog

```
Enter a whole number from 1 to 99.
I will output a combination of coins
that equals that amount of change.
87
87 cents in coins can be given as:
3 quarters
1 dimes
0 nickels and
2 pennies
```

Display 2.6
Change-Making Program

© 2005 W. Savitch, Pearson Prentice Hall

Case Study, cont.

- The program should be tested with several different amounts.
- Test with values that give zero values for each possible coin denomination.
- Test with amounts close to
  - extreme values such as 0, 1, 98 and 99
  - coin denominations, such as 24, 25, and 26.

© 2005 W. Savitch, Pearson Prentice Hall

Increment (and Decrement) Operators

- used to increase (or decrease) the value of a variable by 1
- easy to use, important to recognize
- the increment operator

  `count++` or `++count`
- the decrement operator

  `count--` or `--count`

© 2005 W. Savitch, Pearson Prentice Hall

Increment (and Decrement) Operators

- equivalent operations
  - `count++;`
  - `++count;`
  - `count = count + 1;`
  - `count--;`
  - `--count;`
  - `count = count - 1;`

© 2005 W. Savitch, Pearson Prentice Hall

Increment (and Decrement) Operators in Expressions

- after executing

```
int m = 4;
int result = 3 * (++m)
```

`result` has a value of `15` and `m` has value `5`

- after executing

```
int m = 4;
int result = 3 * (m++)
```

result has a value of `12` and `m` has value `5`

© 2005 W. Savitch, Pearson Prentice Hall

The Class String

- We've used constants of type `String` already.
  "`Enter a whole number from 1 to 99.`"
- A value of type `String` is a sequence of characters treated as a single item.

© 2005 W. Savitch, Pearson Prentice Hall

Declaring and Printing Strings

- declaring

```
String greeting;
greeting = “Hello!”;
```

- or

```
String greeting = “Hello!”;
```

- or

```
String greeting = new String(“Hello!”);
```

- printing

```
System.out.println(greeting);
```


© 2005 W. Savitch, Pearson Prentice Hall

Concatenation of Strings

- Two strings are *concatenated* using the + operator.

```
String greeting = “Hello”;
String sentence;
sentence = greeting + “ officer”;
System.out.println(sentence);
```

- Any number of strings can be concatenated using the + operator.

© 2005 W. Savitch, Pearson Prentice Hall

Concatenating Strings and Integers

```
String solution;
solution = “The answer is“ + 42;
System.out.println (solution);

> The answer is 42
```


© 2005 W. Savitch, Pearson Prentice Hall

Classes

- A *class* is a type used to produce objects.
- An *object* is an entity that stores data and can take actions defined by *methods.*
- An object of the `String` class stores data consisting of a sequence of characters.
- The `length()` method returns the number of characters in a particular `String` object.

```
int howMany = solution.length()
```

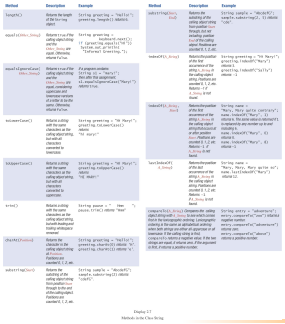

© 2005 W. Savitch, Pearson Prentice Hall

Objects, Methods, and Data

- Objects within a class
  - have the same methods
  - have the same kind(s) of data but the data can have different values.
- Primitive types have values, but no methods.

© 2005 W. Savitch, Pearson Prentice Hall

String Methods



© 2005 W. Savitch, Pearson Prentice Hall

The Method length()

- The method `length()` returns an `int`.
- You can use a call to method `length()` anywhere an `int` can be used.

```
int count = solution.length();
System.out.println(solution.length());
count = solution.length() + 3;
```


© 2005 W. Savitch, Pearson Prentice Hall

Positions in a String

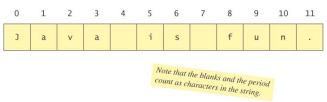
- positions start with 0, not 1.
  - The '`J`' in "`Java is fun.`" is in position 0

© 2005 W. Savitch, Pearson Prentice Hall

Positions in a `String`, cont.

- A position is referred to an an *index*.
  - The '`f`' in "`Java is fun.`" is at index 8.

The twelve characters in the string "`Java is fun.`" have indices 0 through 11. The index of each character is shown above it.



Note that the blanks and the period count as characters in the string.

Display 2.8
String Indices

© 2005 W. Savitch, Pearson Prentice Hall

(Not) Changing `String` Objects

- No methods allow you to change the value of a `String` object.
- But you can change the value of a String variable.

```
String pause = "   Hmm   ";
pause = pause.trim();
pause = pause + "mmm!";
pause = "Ahhh";
```

value of pause

Hmm

Hmm

Hmmmmm

Ahhh

© 2005 W. Savitch, Pearson Prentice Hall

Using the `String` Class

- `class StringDemo`

```
public class StringDemo
{
    public static void main(String[] args)
    {
        String sentence = "Text processing is hard!";
        int position;
        position = sentence.indexOf("hard");
        System.out.println(sentence);
        System.out.println("01234567890123456789012345");
        System.out.println("The word \"hard\" starts at index "
                           + position);
        sentence = sentence.substring(0, position) + "easy!";
        System.out.println("The changed string is:");
        System.out.println(sentence);
    }
}
```

Sample Screen Dialog

```
Text processing is hard!
01234567890123456789012345
The word "hard" starts at index 19
The changed string is:
Text processing is easy!
```

The meaning of "\" is discussed in the subsection titled "Escape Characters."

Display 2.9
Using the String Class

© 2005 W. Savitch, Pearson Prentice Hall

Escape Characters

- How would you print

  `"Java" refers to a language.?`
- The compiler needs to be told that the quotation marks (") do not signal the start or end of a string, but instead are to be printed.

```
System.out.println(
"\"Java\" refers to a language.");
```


© 2005 W. Savitch, Pearson Prentice Hall

Escape Characters

`\"`	Double quote.
`\'`	Single quote.
`\\`	Backslash.
`\n`	New line. Go to the beginning of the next line.
`\r`	Carriage return. Go to the beginning of the current line.
`\t`	Tab. Add whitespace up to the next tab stop.

Display 2.10

Escape Characters

- Each escape sequence is a single character even though it is written with two symbols.

© 2005 W. Savitch, Pearson Prentice Hall

Examples

```
System.out.println("abc\\def");
abc\def
System.out.println("new\nline");
new
line
char singleQuote = '\'';
System.out.println(singleQuote);
'
```


© 2005 W. Savitch, Pearson Prentice Hall

The Unicode Character Set

- Most programming languages use the *ASCII* character set.
- Java uses the *Unicode* character set which includes the ASCII character set.
- The Unicode character set includes characters from many different alphabets (but you probably won't use them).

© 2005 W. Savitch, Pearson Prentice Hall

Keyboard and Screen I/O: Outline

Screen Output
Keyboard Input

Screen Output

- We've seen several examples of screen output already.
- `System.out` is an object that is part of Java.
- `println()` is one of the methods available to the `System.out` object.

© 2005 W. Savitch, Pearson Prentice Hall

Screen Output, cont.

- The concatenation operator (+) is useful when everything does not fit on one line.

```
System.out.println("When everything " +
  "does not fit on one line, use the" +
  " concatenation operator ('+')");
```

  - Do not break the line except immediately before or after the concatenation operator (+).

© 2005 W. Savitch, Pearson Prentice Hall

Screen Output, cont.

- Alternatively, use `print()`

```
System.out.print("When everything ");
System.out.print("does not fit on ");
System.out.print("one line, use the ");
System.out.print("\"print\" ");
System.out.println("statement");
```

ending with a `println()`.

© 2005 W. Savitch, Pearson Prentice Hall

Screen Output, cont.

- syntax

```
System.out.println(output_1 + output_2 + … +
  output_n);
```

- example

```
System.out.println (1967 + " " + "Oldsmobile" + " " +
  442);
```

```
1967 Oldsmobile 442
```


© 2005 W. Savitch, Pearson Prentice Hall

Keyboard Input

- Java 5.0 has reasonable facilities for handling keyboard input.
- These facilities are provided by the `Scanner` class in the `java.util` package.
  - A *package* is a library of classes.

© 2005 W. Savitch, Pearson Prentice Hall

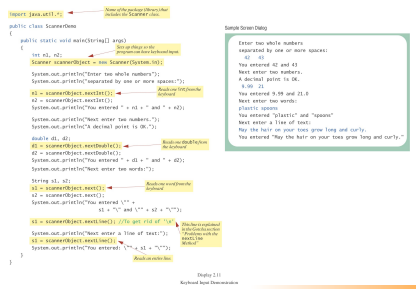
Using the Scanner Class

- Near the beginning of your program, insert
  `import java.util.*`
- Create an object of the `Scanner` class
  ```
  Scanner keyboard =
      new Scanner (System.in)
  ```
- Read data (an `int` or a `double`, for example)
  ```
  int n1 = keyboard.nextInt();
  double d1 = keyboard.nextDouble();
  ```


© 2005 W. Savitch, Pearson Prentice Hall

Keyboard Input Demonstration

- `class ScannerDemo`



© 2005 W. Savitch, Pearson Prentice Hall

Some Scanner Class Methods

- Syntax

  Int_Variable = Object_Name.nextInt();
  Double_Variable = Object_Name.nextDouble();
  String_Variable = Object_Name.next();
  String_Variable = Object_Name.nextLine();

© 2005 W. Savitch, Pearson Prentice Hall

Scanner Class Methods

- examples

```
int count = keyboard.nextInt();
double distance = keyboard.nextDouble();
String word = keyboard.next();
String wholeLine = keyboard.nextLine();
```

- Remember to prompt the user for input, e.g.

```
System.out.print("Enter an integer: ");
```


© 2005 W. Savitch, Pearson Prentice Hall

`nextLine()` Method Caution

- The `nextLine()` method reads the remainder of the current line, even if it is empty.

© 2005 W. Savitch, Pearson Prentice Hall

`nextLine()` Method Caution, cont.

- example

```
int n;
String s1, s2;
n = keyboard.nextInt();
s1 = keyboard.nextLine();
s2 = keyboard.nextLine();
5440
or bust
```

`n` is set to `5440`

but `s1` is set to the empty string.

© 2005 W. Savitch, Pearson Prentice Hall

The Empty String

- A string can have any number of characters, including zero.
- The string with zero characters is called the *empty* string.
- The empty string is useful and can be created in many ways including

```
String s3 = "";
```


(optional) Other Input Delimiters

- Almost any combination of characters and strings can be used to separate keyboard input.
- to change the delimiter to "##"

```
keyboard2.useDelimiter("##");
```

  - whitespace will no longer be a delimiter for `keyboard2` input

(optional) Other Input Delimiters, cont.

- `class DelimitersDemo`

```
import java.util.*;
public class DelimitersDemo
{
    public static void main(String[] args)
    {
        Scanner keyboard1 = new Scanner(System.in);
        Scanner keyboard2 = new Scanner(System.in);
        keyboard2.useDelimiter("##");
        //The delimiters for keyboard1 are the whitespace characters.
        //The only delimiter for keyboard2 is ##.
        String s1, s2;
        System.out.println("Enter a line of text with two words:");
        s1 = keyboard1.next();
        s2 = keyboard1.next();
        System.out.println("the two words are \"" + s1
                           + "\" and \"" + s2 + "\"");

        System.out.println("Enter a line of text with two words");
        System.out.println("delimited by ##:");
        s1 = keyboard2.next();
        s2 = keyboard2.next();
        System.out.println("the two words are \"" + s1
                           + "\" and \"" + s2 + "\"");
    }
}
```

keyboard1.use/
keyboard2.use
delimiters.

Sample Screen Dialog

```
Enter a line of text with two words:
Funny wo##rd##
The two words are "Funny" and "wo##rd##"
Enter a line of text with two words
delimited by ##:
Funny wo##rd##
The two words are "Funny wo" and "rd"
```

Display 2.13
Changing Delimiters (Optional)

Documentation and Style: Outline

- Meaningful Names
- Self-Documentation and Comments
- Indentation
- Named Constants

© 2005 W. Savitch, Pearson Prentice Hall

Documentation and Style

- Most programs are modified over time to respond to new requirements.
- Programs which are easy to read and understand are easy to modify.
- Even if it will be used only once, you have to read it in order to debug it .

© 2005 W. Savitch, Pearson Prentice Hall

Meaningful Names for Variables

- A variable's name should suggest its use.
- Observe conventions in choosing names for variables.
  - Use only letters and digits.
  - "Punctuate" using uppercase letters at word boundaries (e.g. `taxRate`).
  - Start variables with lowercase letters.
  - Start class names with uppercase letters.

© 2005 W. Savitch, Pearson Prentice Hall

Documentation and Comments

- The best programs are self-documenting.
  - clean style
  - well-chosen names
- Comments are written into a program as needed explain the program.
  - They are useful to the programmer, but they are ignored by the compiler.

© 2005 W. Savitch, Pearson Prentice Hall

Comments

- A comment can begin with //.
  - Everything after these symbols and to the end of the line is treated as a comment and is ignored by the compiler.

```
double radius; //in centimeters
```


© 2005 W. Savitch, Pearson Prentice Hall

Comments, cont.

- A comment can begin with /* and end with */
  - Everything between these symbols is treated as a comment and is ignored by the compiler.

```
/* the simplex method is used to
   calculate the answer*/
```


© 2005 W. Savitch, Pearson Prentice Hall

Comments, cont.

- A *javadoc* comment, begins with /** and ends with */.
  - It can be extracted automatically from Java software.

```
/** method change requires the number of coins to be
  nonnegative */
```


© 2005 W. Savitch, Pearson Prentice Hall

When to Use Comments

- Begin each program file with an explanatory comment
  - what the program does
  - the name of the author
  - contact information for the author
  - date of the last modification.
- Provide only those comments which the expected reader of the program file will need in order to understand it.

© 2005 W. Savitch, Pearson Prentice Hall

Comments Example

- `class CircleCalculation`

```
import java.util.*;
/**
Program to determine area of a circle.
Author: Jane Q. Programmer.
E-mail Address: janeqhomemachine.etc.etc.
Programming Assignment 2.
Last Changed: October 7, 2006.
*/

public class CircleCalculation
{
    public static void main(String[] args)
    {
        double radius; //in inches
        double area; //in square inches
        Scanner keyboard = new Scanner(System.in);

        System.out.println(
                "Enter the radius of a circle in inches:");
        radius = keyboard.nextDouble();
        area = 3.14159 * radius * radius;
        System.out.println("A circle of radius " + radius + " inches");
        System.out.println("Has an area of " + area + " square inches.");
    }
}
```

This can be after the key comment if you prefer.

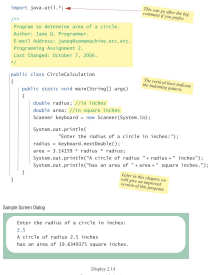
The vertical lines indicate the indentation pattern.

Enter the data changes you will give an improved version of this program.

Sample Screen Dialog

```
Enter the radius of a circle in inches:
2.5
A circle of radius 2.5 inches
has an area of 18.6349375 square inches.
```

Display 2.14
Comments and Indenting



© 2005 W. Savitch, Pearson Prentice Hall

Indentation

- Indentation should communicate nesting clearly.
- I good choice is four spaces for each level of indentation.
- Indentation should be consistent.
- Indentation should be used for second and subsequent lines of statements which do not fit on a single line.

© 2005 W. Savitch, Pearson Prentice Hall

Indentation, cont.

- Indentation does not change the behavior of the program.
- Improper indentation can miscommunicate the behavior of the program.

© 2005 W. Savitch, Pearson Prentice Hall

Named Constants

- To avoid confusion, always name constants (and variables).

```
circumference = PI * radius;
```

is clearer than

```
circumference = 3.14159 * 6.023;
```

- Place constants near the beginning of the program.

© 2005 W. Savitch, Pearson Prentice Hall

Named Constants, cont.

- Once the value of a constant is set (or changed by an editor), it can be used (or reflected) throughout the program.

```
public static final double INTEREST_RATE = 6.65;
```

- If a literal (such as 6.65) is used instead, every occurrence must be changed, with the risk than another literal with the same value might be changed unintentionally.

© 2005 W. Savitch, Pearson Prentice Hall

Declaring Constants

- syntax

```
public static final Variable_Type = Constant;
```

- examples

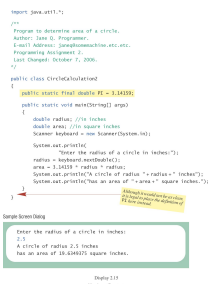
```
public static final double PI = 3.14159;
public static final String MOTTO = "The customer is
  always right.";
```

  - By convention, uppercase letters are used for constants.

© 2005 W. Savitch, Pearson Prentice Hall

Named Constants

- class CircleCalculation2



```
import java.util.*;
/**
 Program to determine area of a circle.
 Author: Jane Q. Programmer.
 E-mail Address: jane@somemachine.etc.etc.
 Programming Assignment 2.
 Last Changed: October 7, 2006.
*/
public class CircleCalculation2
{
    public static final double PI = 3.14159;

    public static void main(String[] args)
    {
        double radius; //in inches
        double area; //in square inches
        Scanner keyboard = new Scanner(System.in);

        System.out.println(
                "Enter the radius of a circle in inches:");
        radius = keyboard.nextDouble();
        area = 3.14159 * radius * radius;
        System.out.println("A circle of radius " + radius +" inches");
        System.out.println("has an area of " + area +" square inches.");
    }
}
```

Although it is conventional to put all letters in the name of a constant in uppercase, the compiler will allow you to use lowercase letters in the name of a constant.

Sample Screen Dialog

```
Enter the radius of a circle in inches:
2.5
A circle of radius 2.5 inches
has an area of 19.6349375 square inches.
```

Display 2.13
Naming a Constant

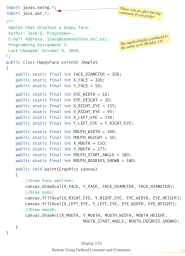
© 2005 W. Savitch, Pearson Prentice Hall

(optional) Graphics Supplement: Outline

- Style Rules Applied to a Graphics Applet
- `JOptionPane`
- Inputting Numeric Types
- Multi-Line Output Windows

Style Rules Applied to a Graphics Applet

- `class HappyFace`

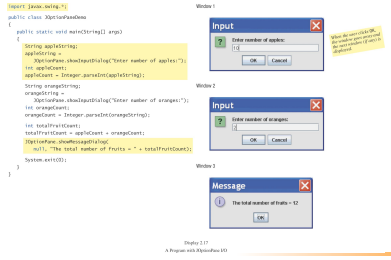


Style Rules Applied to a Graphics Applet, cont.

- Named constants makes it easier to find values.
- Comments and named constants make changing the code much easier.
- Named constants protect against changing the wrong value.

JOptionPane

- class JOptionPaneDemo



JOptionPane, cont.

- `JOptionPane` can be used to construct windows that interact with the user.
- The `JOptionPane` class is imported by

```
import javax.swing.*;
```

- The `JOptionPane` class produces windows for obtaining input or displaying output.

JOptionPane, cont.

- Use `showInputDialog()` for input .
- Only string values can be input.
- To convert an input value from a string to an integer use the `parseInt()` method from the `Integer` class, use

```
appleCount = Integer.parseInt(appleString);
```

JOptionPane, cont.

- Output is displayed using the `showMessageDialog` method.

```
JOptionPane.showMessageDialog(null, "The total number
  of fruits = " + totalFruitCount);
```


© 2005 W. Savitch, Pearson Prentice Hall

JOptionPane, cont.

- syntax
  - input

```
String_Variable = JOptionPane.showInputDialogue
(String_Expression);
```

  - output

```
JOptionPane.showMessageDialog(null,
String_Expression);
```

- System.exit(0) ends the program.

© 2005 W. Savitch, Pearson Prentice Hall

JOptionPane Cautions

- If the input is not in the correct format, the program will *crash.*
- If you omit the last line (`System.exit(0)`), the program will not end, even when the OK button in the output window is clicked.
- Always label any output.

© 2005 W. Savitch, Pearson Prentice Hall

Inputting Numeric Types

- `JOptionPane.showInput Dialog` can be used to input any of the numeric types.
  - Simply convert the input string to the appropriate numeric type.

Type Name	Method for Converting
byte	Byte.parseByte(*String_To_Convert*)
short	Short.parseShort(*String_To_Convert*)
int	Integer.parseInt(*String_To_Convert*)
long	Long.parseLong(*String_To_Convert*)
float	Float.parseFloat(*String_To_Convert*)
double	Double.parseDouble(*String_To_Convert*)

To convert a value of type `String` to a value of the type given in the first column, use the method given in the second column. Each of the methods in the second column returns a value of the type given in the first column. The *String_To_Convert* must be a correct string representation of a value of the type given in the first column. For example, to convert to an `int`, the *String_To_Convert* must be a whole number (in the range of the type `int`) that is written in the usual way without any decimal point.

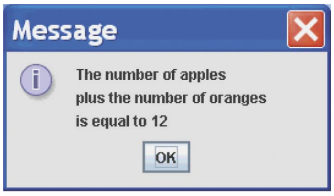
Display 2.18
Methods for Converting Strings to Numbers

Multi-Line Output Windows

- To output multiple lines using the method `JOptionPane.showMessage Dialog`, insert the new line character '`\n`' into the string used as the second argument.

```
OptionPane.showMessageDialog(null,
  "The number of apples\n" +
  "plus the number of oranges\n" +
  "is equal to " + totalFruit);
```

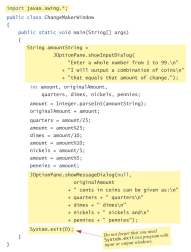

Multi-Line Output Windows, cont.



Display 2.19
A Multiline Output Window

Programming Example

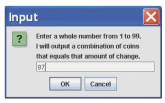
- `class ChangeMakerWindow`



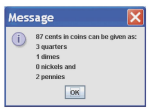
Display 2.20
Change Program with I/O Windows

Programming Example, cont.

Input Window



Output Window



Display 2.20
Change Program with I/O Windows

Summary

- You have become familiar with Java primitive types (numbers, characters, etc.).
- You have learned about assignment statements and expressions.
- You have learned about stings.
- You have become familiar with classes, methods, and objects.

Summary, cont.

- You have learned about simple keyboard input and screen output.
- (optional) You have learned about windows-based input and output using the JOptionPane class.

© 2005 W. Savitch, Pearson Prentice Hall