

5~6. Basic Computation

[ECE20016/ITP20003] Java Programming

Agenda



- Variables and Expressions
- The Class String
- Keyboard and Screen I/O
- Documentation and Style

Variables



- **Variables** in a program are used to store data such as numbers and letters.
 - **Places** to store data.
 - They are implemented as **memory locations**.
 - Store a particular type of data.
 - The data stored by a variable is called its **value**.
 - A variable **must be declared** before it is used.

EggBasket

LISTING 2.1 A Simple Java Program

```
public class EggBasket
{
    public static void main(String[] args)
    {
        int numberOfBaskets, eggsPerBasket, totalEggs;
        numberOfBaskets = 10;
        eggsPerBasket = 6;
        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);
    }
}
```

Variable
declarations

Assignment statement

Sample Screen Output

```
If you have
6 eggs per basket and
10 baskets, then
the total number of eggs is 60
```

Naming and Declaring Variables

- Variable declaration:

Type Variable_1, Variable_2, ...;

Ex) int numberOfBaskets;
 int eggsPerBasket;
 int totalEggs;

- A variable's type determines what kinds of values it can hold (**int**, **double**, **char**, etc.).
- Choose names that are helpful such as *count* or *speed*, but not *c* or *s*.
- Variable declaration can be concatenated by comma operator
Ex) int numberOfBaskets, eggsPerBasket, totalEggs;

Storing Values to Variables

- Assignment

Variable = Expression;

- The "equal sign" is called the **assignment operator**.

Cf. Mathematical equal operator: '=='

- *Expression* can be one of followings

- Another variable
- A literal or constant (such as a number)
- Something more complicated which combines variables and literals using operators (such as + and -)

Ex) numberOfBaskets = 10;

eggsPerBasket = 6;

score = numberOfCards + handicap;

eggsPerBasket = eggsPerBasket - 2;

Data Types



- A **class type** is used for a class of **objects** (**data** + **methods**).
 - "Java is fun" is a value of class type **String**
→ Type for complex data
- A **primitive type** is used for **simple, non-decomposable values** such as an individual number or individual character.
 - **int**, **double**, and **char** are primitive types.
→ Type for simple data

Primitive Types

Type Name	Kind of Value	Memory Used	Range of Values
byte	Integer	1 byte	−128 to 127
short	Integer	2 bytes	−32,768 to 32,767
int	Integer	4 bytes	−2,147,483,648 to 2,147,483,647
long	Integer	8 bytes	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	Floating-point	4 bytes	$\pm 3.40282347 \times 10^{+38}$ to $\pm 1.40239846 \times 10^{-45}$
double	Floating-point	8 bytes	$\pm 1.79769313486231570 \times 10^{+308}$ to $\pm 4.94065645841246544 \times 10^{-324}$
char	Single character (Unicode)	2 bytes	All Unicode values from 0 to 65,535
boolean		1 bit	True or false

Ex) short day = 10;
float cost = 195.20;
char initial = 'i';
boolean flag = true;

Primitive Types



- Integer types (byte, short, int, and long)
 - Ex) 0, -1, 365, 124565465464467
 - int is most common
- Floating-point types (float and double)
 - Ex) 0.99, -22.8, 3.14159, 5.0
 - double is more common
 - Floating-point numbers often are only approximations since they are stored with a finite number of bits.
 - Ex) 1.0/3.0 is slightly less than 1/3
- Character type (char)
 - Ex) 'a' 'A' '#' ''
- Boolean type (boolean)
 - true* or *false*

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
 - Mainly for auto-generated names.
- The first character cannot be a digit.
- Java is **case sensitive**
Ex) stuff, Stuff, and STUFF are different identifiers.
- An identifier cannot be a keyword (or reserved word) used for special, predefined meanings in Java.
Ex) int, static, public, for, return, ...

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. (e.g. studentName)

Where to Declare Local 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 */
    ...
}
```

Initializing Variables



- A variable that has been declared, but not 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.
- To protect against an uninitialized variable (and to keep the compiler happy), assign a value at the time the variable is declared.

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

Simple Input



- Data can be entered from the keyboard using
`Scanner keyboard = new Scanner(System.in);`
- Followed, for example, by
`eggsPerBasket = keyboard.nextInt();`
 - Reads one int value from the keyboard and assigns it to eggsPerBasket.

Simple Input

LISTING 2.2 A Program with Keyboard Input

```
import java.util.Scanner;
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);
    }
}
```

Gets the Scanner class from the package (library) java.util

Sets up things so the program can accept keyboard input

Reads one whole number from the keyboard

Simple Screen Output



- Syntax

`System.out.println(<contents to display>);`

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

- Outputs the string literal "the count is "
- Followed by the current value of the variable count.

Constants

- Literal expressions such as 2, 3.7, or 'y' are called constants.
 - Numeral constants can be preceded by a + or – sign
- Floating-point constants can be written
 - With digits after a decimal point or
 - Using e notation.
 - Ex) $865000000.0 \rightarrow 8.65e8$ // denotes 8.65×10^8
 - $0.000483 \rightarrow 4.83e-4$ // denotes 4.83×10^{-4}
 - The number in front of the e does not need to contain a decimal point.
- Named constants using the 'final' modifier
`public static final Type Variable = Constant;`
Ex) `public static final double PI = 3.14159;`

Assignment Compatibilities



- Java is said to be strongly typed.
 - Ex) You can't assign a floating point value to a variable declared to store an integer.

- Sometimes conversions between numbers are possible.
 - Ex) `doubleVariable = 7;`

 - `byte --> short --> int --> long --> float --> double`
 - But not to a variable of any type further to the left.
 - `char --> int`

Type Casting



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

Ex)

```
double distance = 9.0;
```

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

- Illegal without (int)
- The value of (int)distance is 9,
 - Any nonzero value to the right of the decimal point is truncated rather than rounded.
- The value of distance, both before and after the cast, is 9.0.

Arithmetic Operators



- Arithmetic expressions can be formed using
 - Operators, such as +, -, *, and /
 - Operands, such as variables or numbers
- Type of expression
 - 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.
Ex) `int hoursWorked = 40;`
`double payRate = 8.25;`
→ `hoursWorked * payRate` is a double with a value of 500.0.

Arithmetic Operations



- Expressions with two or more operators can be viewed as a series of steps, each involving only two operands.
Ex) $\text{balance} + (\text{balance} * \text{rate})$
- The result is the rightmost type from the following list that occurs in the expression.
byte --> short --> int --> long --> float --> double

Arithmetic Operators

- Division operator

- When both operands are integer types, the result is truncated, not rounded.

Ex) $99/100$ has a value of 0.

- The mod (%) operator is used with operators of integer type to obtain the **remainder** after integer division.

Ex) $14 \% 4$ is equal to 2. // $14 = 4 * 3 + 2$.

- 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.

Parentheses and Precedence

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

Ex) $(\text{cost} + \text{tax}) * \text{discount}$
 $\text{cost} + (\text{tax} * \text{discount})$

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

Highest Precedence

First: the unary operators $+$, $-$, $!$, $++$, and $--$

Second: the binary arithmetic operators $*$, $/$, and $\%$

Third: the binary arithmetic operators $+$ and $-$

Lowest Precedence

Precedence Rules



- When binary operators have equal precedence
 - Left to right precedenceEx) $5 + 3 - 2$
- When unary operators have equal precedence.
 - Right to left precedenceEx) $- ++a$ $// ! ++a$ is NOT a valid Java expression

Making Code Clearer



- Even when parentheses are not needed, they can be used to make the code clearer.
 - `balance + (interestRate * balance)`
- Spaces also make code clearer
Ex) `balance + interestRate*balance`

Sample Expressions

Ordinary Math	Java (Preferred Form)	Java (Parenthesized)
$rate^2 + delta$	<code>rate * rate + delta</code>	<code>(rate * rate) + delta</code>
$2(salary + bonus)$	<code>2 * (salary + bonus)</code>	<code>2 * (salary + bonus)</code>
$\frac{1}{time + 3mass}$	<code>1 / (time + 3 * mass)</code>	<code>1 / (time + (3 * mass))</code>
$\frac{a - 7}{t + 9v}$	<code>(a - 7) / (t + 9 * v)</code>	<code>(a - 7) / (t + (9 * v))</code>

Specialized Assignment Operators



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

Ex) `amount = amount + 5;`

- can be written as

`amount += 5;`

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`

Increment and Decrement Operators



■ Equivalent operations

- `count++;`
- `++count;`
- `count = count + 1;`

- `count--;`
- `--count;`
- `count = count - 1;`

Increment and Decrement Operators in Expressions



Ex)

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

→ After executing, result has a value of 15 and m has a value of 5

Ex)

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

→ After executing, result has a value of 12 and m has a value of 5

Agenda



- Variables and Expressions
- **The Class String**
- Keyboard and Screen I/O
- Documentation and Style

The Class String



- A value of type String is a
 - Sequence of characters
 - Treated as a single item.

Ex) "Enter a whole number from 1 to 99."

String Constants and Variables

- Declaring

`String variable_name;`

Ex) `String greeting;`

`greeting = "Hello!";`

or

`String greeting = "Hello!";`

or

`String greeting = new String("Hello!");`

- Printing

`System.out.println(greeting);`

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.

Concatenating Strings and Integers



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

The answer is 42

String Methods



- An object of the String class stores data consisting of a sequence of characters.
- Objects have **methods** as well as **data**
 - Note: object = data + operations

The Method `length()`



- The `length()` method returns the number of characters in a particular String object.


```
String greeting = "Hello";  
int n = greeting.length();
```

- The method `length()` returns an int.

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

```
int count = command.length();  
System.out.println("Length is " + command.length());  
count = command.length() + 3;
```

String Indices



<i>Indices</i>	0	1	2	3	4	5	6	7	8	9	10	11
	J	a	v	a		i	s		f	u	n	.

- A position is referred to an index.
 - Positions start with 0, not 1.

Ex) The 'J' in "Java is fun." is in position 0

Ex) The 'f' in "Java is fun." is at index 8.

String Methods

`charAt` (*Index*)

Returns the character at *Index* in this string. Index numbers begin at 0.

`compareTo` (*A_String*)

Compares this string with *A_String* to see which string comes first in the lexicographic ordering. (Lexicographic ordering is the same as alphabetical ordering when both strings are either all uppercase letters or all lowercase letters.) Returns a negative integer if this string is first, returns zero if the two strings are equal, and returns a positive integer if *A_String* is first.

`concat` (*A_String*)

Returns a new string having the same characters as this string concatenated with the characters in *A_String*. You can use the \Downarrow operator instead of `concat`.

`equals` (*Other_String*)

Returns true if this string and *Other_String* are equal. Otherwise, returns false.

String Methods



`equalsIgnoreCase(Other_String)`

Behaves like the method `equals`, but considers uppercase and lowercase versions of a letter to be the same.

`indexOf(A_String)`

Returns the index of the first occurrence of the substring *A_String* within this string. Returns -1 if *A_String* is not found. Index numbers begin at 0.

`lastIndexOf(A_String)`

Returns the index of the last occurrence of the substring *A_String* within this string. Returns -1 if *A_String* is not found. Index numbers begin at 0.

String Methods



`length()`

Returns the length of this string.

`toLowerCase()`

Returns a new string having the same characters as this string, but with any uppercase letters converted to lowercase.

`toUpperCase()`

Returns a new string having the same characters as this string, but with any lowercase letters converted to uppercase.

String Methods

`replace(OldChar, NewChar)`

Returns a new string having the same characters as this string, but with each occurrence of *OldChar* replaced by *NewChar*.

`substring(Start)`

Returns a new string having the same characters as the substring that begins at index *Start* of this string through to the end of the string. Index numbers begin at 0.

`substring(Start, End)`

Returns a new string having the same characters as the substring that begins at index *Start* of this string through, but not including, index *End* of the string. Index numbers begin at 0.

`trim()`

Returns a new string having the same characters as this string, but with leading and trailing whitespace removed.

String Processing

```
public class StringDemo
{
    public static void main (String [] args)
    {
        String sentence = "Text processing is hard!";
        int position = sentence.indexOf ("hard");
        System.out.println (sentence);
        System.out.println ("012345678901234567890123");
        System.out.println ("The word \"hard\" starts at index " + position);
        sentence = sentence.substring (0, position) + "easy!";
        sentence = sentence.toUpperCase ();
        System.out.println ("The changed string is:");
        System.out.println (sentence);
    }
}
```

```
Text processing is hard!
012345678901234567890123
The word "hard" starts at index 19
The changed string is:
TEXT PROCESSING IS EASY!
```

Escape Characters



- How would you print the following string?
"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.");`

Escape Characters

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

\ " 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.

Examples

```
System.out.println("abc\\def");
```



abc\\def

```
System.out.println("new\\nline");
```



new
line

```
char singleQuote = '\\';
```

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



\

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).
Ex) Hangul, Chinese characters, Arabic characters, ...

Agenda



- Variables and Expressions
- The Class String
- **Keyboard and Screen I/O**
- Documentation and Style

Screen Output



- System.out is an object that is part of Java.
 - println() is one of the methods available to the System.out object.
- The concatenation operator (+) is useful when everything does not fit on one line.

Ex) `System.out.println("Lucky number = " + 13 +
"Secret number = " + number);`
- Do not break the line except immediately before or after the concatenation operator (+).

Screen Output



- Alternatively, use `print()`
`System.out.print("One, two,");`
`System.out.print(" buckle my shoe.");`
`System.out.println(" Three, four,");`
`System.out.println(" shut the door.");`
 - ending with a `println()`.
- Result
One, two, buckle my shoe. Three, four,
shut the door.

Keyboard Input



- Java 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.

Using the Scanner Class



- Near the beginning of your program, insert
`import java.util.Scanner;`
- 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();`

ScannerDemo (Listing 2.5)

```
import java.util.Scanner;
public class ScannerDemo
{
    public static void main (String [] args)
    {
        Scanner keyboard = new Scanner (System.in);
        System.out.println ("Enter two whole numbers");
        System.out.println ("separated by one or more spaces:");
        int n1, n2;
        n1 = keyboard.nextInt ();
        n2 = keyboard.nextInt ();
        System.out.println ("You entered " + n1 + " and " + n2);
        System.out.println ("Next enter two numbers.");
        System.out.println ("A decimal point is OK.");
        double d1, d2;
        d1 = keyboard.nextDouble ();
        d2 = keyboard.nextDouble ();
        System.out.println ("You entered " + d1 + " and " + d2);
        System.out.println ("Next enter two words:");
        String s1, s2;
        s1 = keyboard.next ();
        s2 = keyboard.next ();
        System.out.println ("You entered \"" + s1 + "\" and \"" + s2 + "\"");
        s1 = keyboard.nextLine (); //To get rid of '\n'
        System.out.println ("Next enter a line of text:");
        s1 = keyboard.nextLine ();
        System.out.println ("You entered: \"" + s1 + "\"");
    }
}
```

ScannerDemo (Listing 2.5)

Enter two whole numbers
separated by one or more spaces:

42 43

You entered 42 and 43
Next enter two numbers.
A decimal point is OK.

9.99 21

You entered 9.99 and 21.0
Next enter two words:

plastic spoons

You entered "plastic" and "spoons"
Next enter a line of text:

May the hair on your toes grow long and curly.

You entered "May the hair on your toes grow long and curly."

Some Scanner Class Methods

Scanner_Object_Name.next()

Returns the `String` value consisting of the next keyboard characters up to, but not including, the first delimiter character. The default delimiters are whitespace characters.

Scanner_Object_Name.nextLine()

Reads the rest of the current keyboard input line and returns the characters read as a value of type `String`. Note that the line terminator '`\n`' is read and discarded; it is not included in the string returned.

Scanner_Object_Name.nextInt()

Returns the next keyboard input as a value of type `int`.

Scanner_Object_Name.nextDouble()

Returns the next keyboard input as a value of type `double`.

Scanner_Object_Name.nextFloat()

Returns the next keyboard input as a value of type `float`.

Some Scanner Class Methods

Scanner_Object_Name.nextLong()

Returns the next keyboard input as a value of type `long`.

Scanner_Object_Name.nextByte()

Returns the next keyboard input as a value of type `byte`.

Scanner_Object_Name.nextShort()

Returns the next keyboard input as a value of type `short`.

Scanner_Object_Name.nextBoolean()

Returns the next keyboard input as a value of type `boolean`. The values of `true` and `false` are entered as the words *true* and *false*. Any combination of uppercase and lowercase letters is allowed in spelling *true* and *false*.

Scanner_Object_Name.useDelimiter(*Delimiter_Word*);

Makes the string *Delimiter_Word* the only delimiter used to separate input. Only the exact word will be a delimiter. In particular, blanks, line breaks, and other whitespace will no longer be delimiters unless they are a part of *Delimiter_Word*.

This is a simple case of the use of the `useDelimiter` method. There are many ways to set the delimiters to various combinations of characters and words, but we will not go into them in this book.

nextLine() Method Caution

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

Ex)

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

- Assume input shown
 - n is set to 42
 - but s1 is set to the empty string.

42

and don't you
forget it.

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 = "";`

Other Input Delimiters (optional)



- Almost any combination of characters and strings can be used to separate keyboard input.

Ex) Change the delimiter to "##"

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

- whitespace will no longer be a delimiter for keyboard2 input

Agenda



- Variables and Expressions
- The Class String
- Keyboard and Screen I/O
- **Documentation and Style**

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.

Meaningful Variable Names



- 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 variable names with lowercase letters.
 - Start class names with uppercase letters.

Comments



- Comments are written into a program as needed explain the program.
 - They are useful to the programmer, but they are ignored by the compiler.
- 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`

Comments



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

`/*`

This program should only
be used on alternate Thursdays,
except during leap years, when it should
only be used on alternate Tuesdays.

`*/`

Comments



- 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  
*/
```

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.

Indentation



- Indentation should communicate nesting clearly.
 - Proper indentation helps communicate to the human reader the nested structures of the program
- A good choice is **four spaces (or a tab) 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.

Using Named Constants

- To avoid confusion, always name constants (and variables).
 - `area = PI * radius * radius;`
 - is clearer than
 - `area = 3.14159 * radius * radius;`
- Place constants near the beginning of the program.
- Once the value of a constant is set (or changed by an editor), it can be used (or reflected) throughout the program.
 - Ex) `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 that another literal with the same value might be changed unintentionally.

Declaring Constants



- Syntax

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

Ex) 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.

Named Constants

```
import java.util.Scanner;
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 = PI * radius * radius;
        System.out.println ("A circle of radius " + radius + " inches");
        System.out.println ("has an area of " + area + " square inches.");
    }
}
```

```
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.
```