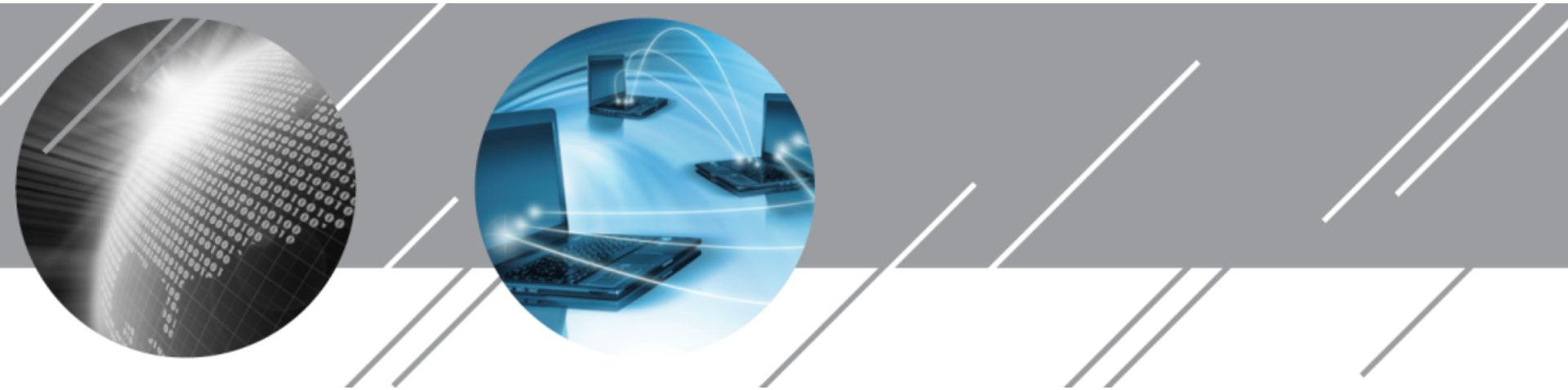
Object Oriented Programming

Introduction to Java

Ch. 2. Basic Computation

Primitive Types, Strings and Console I/O



Dept. of AI. Software, Gachon University
Ahyoung Choi

Short Review : Class

Concept of Class and Object

- "Class" refers to a blueprint. It defines the variables and methods the objects support

- "Object" is an instance of a class. Each object has a class which defines its data and behavior

Class Members

- **A class can have three kinds of members:**
  - ***fields***: data variables which determine the status of the class or an object
  - ***methods***: executable code of the class built from statements. It allows us to manipulate/change the status of an object or access the value of the data member
  - (***nested classes and nested interfaces)***

Sample Class

Sample class

```
class Pencil {
    public String color = "white";
    public int length;
    public float diameter;
    public static long nextID = 0;

    public void setColor (String newColor) {
          color = newColor;
    }
}
```

Pencil.java

```
public class Pencil {
 public String color = "white";
 public int length;
 public float diameter;
 private float price;

    public static long nextID = 0;

    public void setPrice (float newPrice) {
          price = newPrice;
    }

    public void printPrice () {
          System.out.println(price);
 }
```

CreatePencil.java

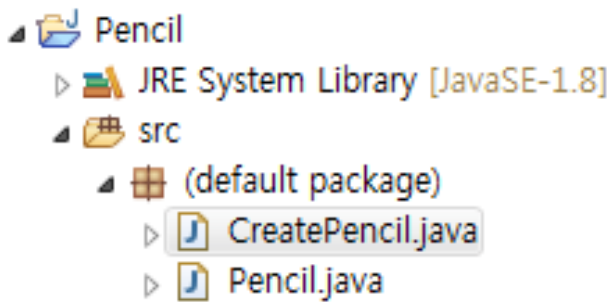
```
public class CreatePencil {
 public static void main(String args[]){
        Pencil p1 = new Pencil();
        p1.setPrice(200);
        p1.printPrice();
    }
}
```

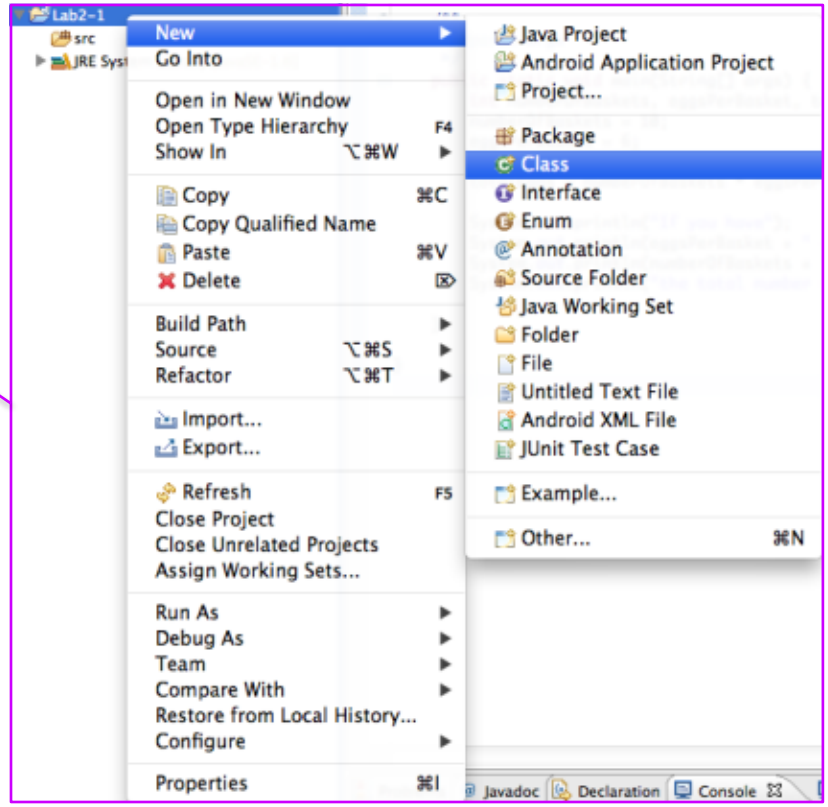
```
%> javac Pencil.java
%> javac CreatePencil.java
%> java CreatePencil
```

Lab: Pencil

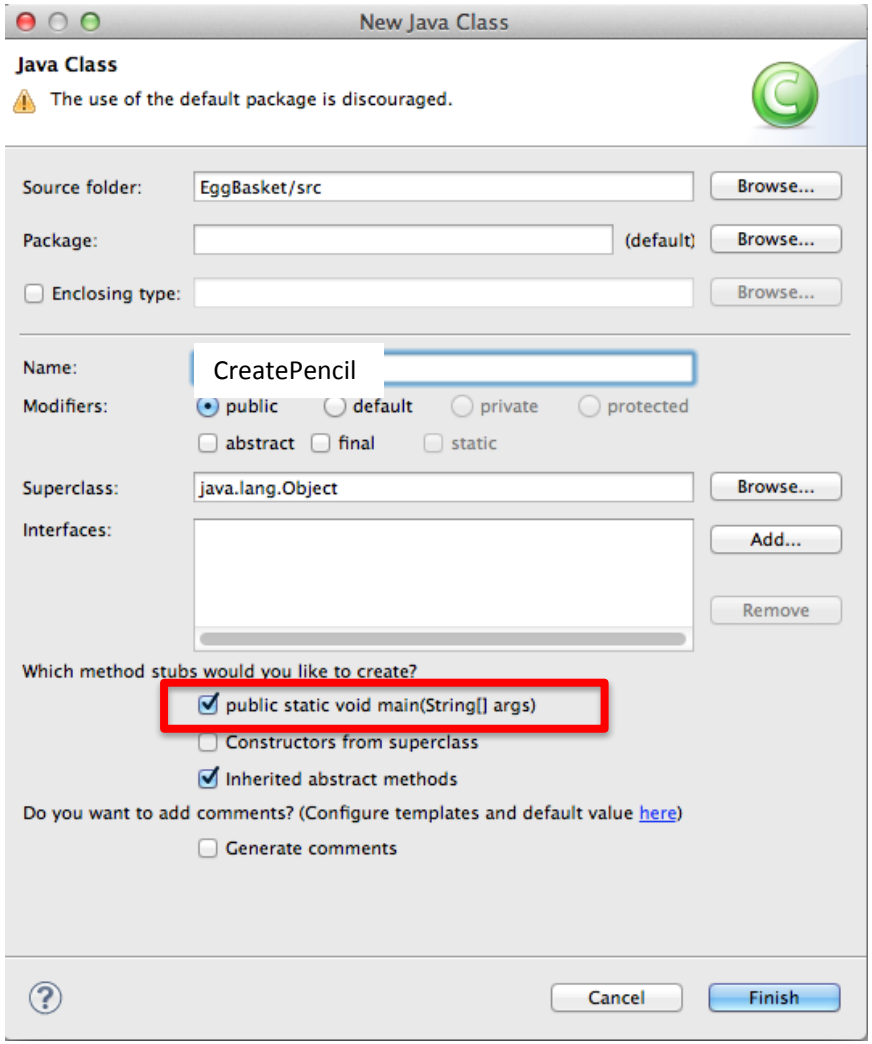
- **New Java Project**
  - Eclipse [File]-[New]->[Java Project]
  - Name : Lab2-0

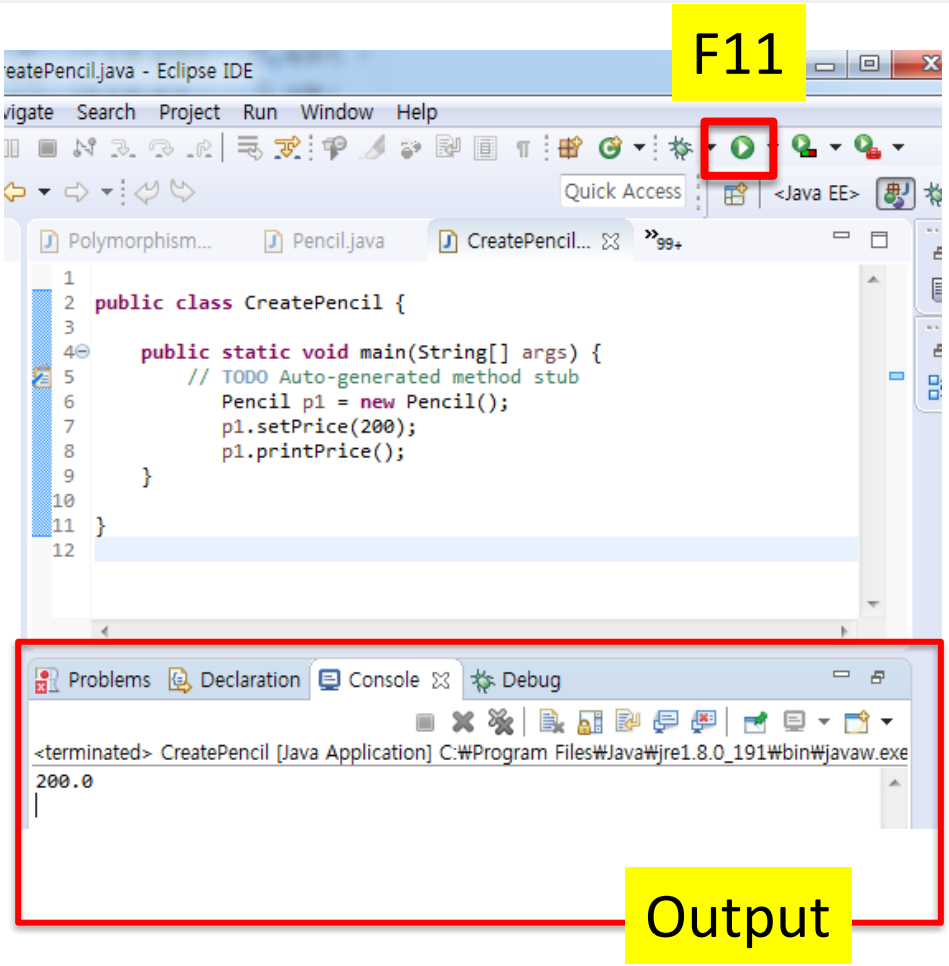
- **Add** Class





Lab: Pencil


New Java Class
Java Class
The use of the default package is discouraged.
Source folder: EggBasket/src
Browse...
Package:
(default)
Enclosing type:
Name: CreatePencil
Modifiers: public default private protected
abstract final static
Superclass: java.lang.Object
Interfaces:
Add...
Remove
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
Generate comments
Cancel
Finish


F11
eatePencil.java - Eclipse IDE
Search Project Run Window Help
Quick Access
Polymorphism...
Pencil.java
CreatePencil...
public class CreatePencil {
public static void main(String[] args) {
// TODO Auto-generated method stub
Pencil p1 = new Pencil();
p1.setPrice(200);
p1.printPrice();
}
}
Problems
Declaration
Console
Debug
<terminated> CreatePencil [Java Application] C:₩Program Files₩Java₩jre1.8.0_191₩bin₩javaw.exe
200.0
Output

Outline

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

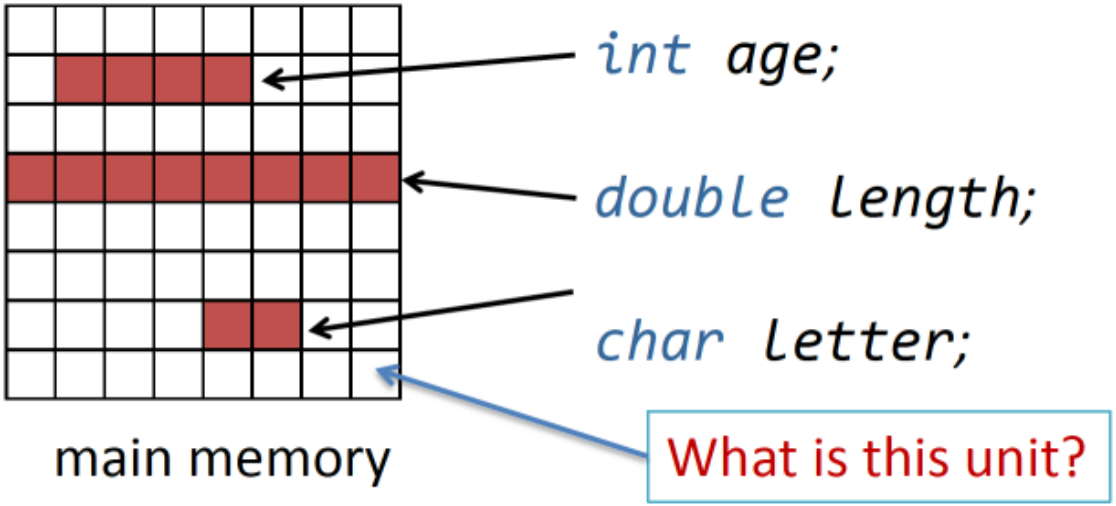
2.1 Variables and Expressions

Variables

- Used to store data in program
  - *Variables* store data such as numbers and letters.
    - Think of them as places to store data.
    - Correspond to memory locations.
- The data stored by a variable is called its *value.*
  - The value is stored in the memory location.
  - Can be changed throughout program

How to use variables

- **Declare** a variable
- **Assign** a value to the variable
- **Change** the value of the variable



When declaring a variable, a certain amount of memory is assigned/allocated based on the declared primitive type

Variable declaration

- A variable must be declared before it is used
- Choose names that are helpful
  - such as `count` or `speed`, but not `c` or `s`.
- You provide its **type** and **name**.

```
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.

Data Types

- A *primitive type* is used for simple, nondecomposable values such as an individual number or individual character.
  - `int`, `double`, and `char` are primitive types.

- A *class type* is used for a class of objects and has both data and methods.
  - `"Java is fun"` is a value of class type `String`

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	–9,223,372,036,8547,75,808 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

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`

Variable names

Exp1) No special meaning for it
int a$ ➔ possible
Exp2) javac uses $ in some automatically-generated variable names: for example, this$0 et al are used for the implicit this references from the inner classes to their outer classes.

e.g.) public enum Type { WALKING, RUNNING, TRACKING, HIKING }

- May contain only
  - Letters
  - Digits (0 through 9)
  - Underscore character (_): *Constant or enum (fixed set of constant)*
  - Dollar sign symbol ($)
- Cannot have a digit as its first character
- Case-sensitive
  - i.e., *stuff*, *Stuff*, and *stuFF* are all different
- Identifiers may not contain any spaces, dots (.), asterisks (*), or other characters:

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

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., number**O**f**E**ggs, price**P**er**B**asket

Keywords or Reserved Words

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

Where to Declare Variables

- Declare a variable
  - 1. Just before it is used

    or
  - 2. At the beginning of the section of your program that is enclosed in `{ }`.

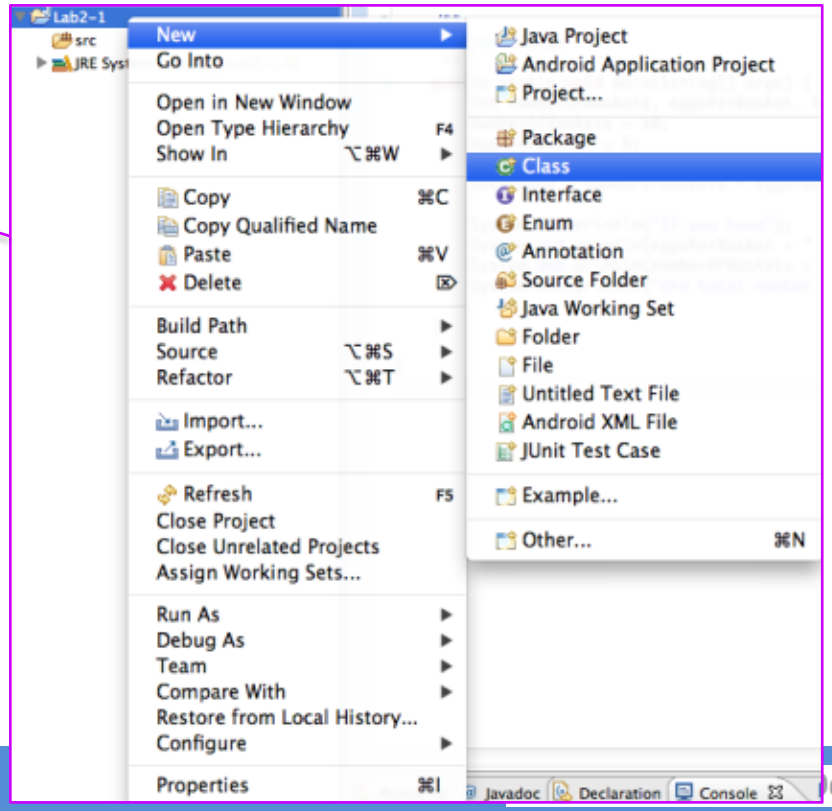
```
public static void main(String[] args)
{ /* declare variables here */
    . . .
}
```

Initializing Variables

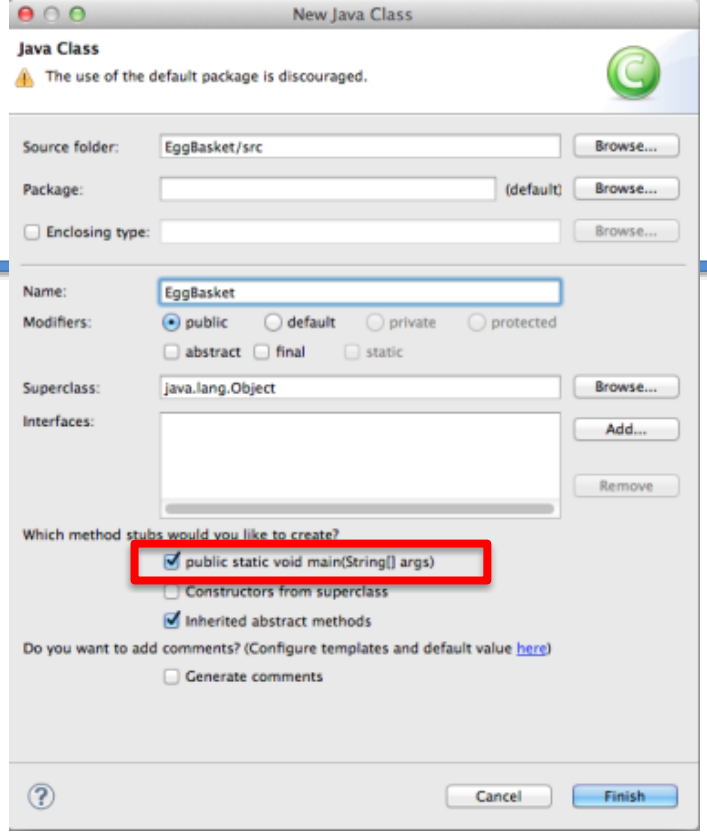
- A variable that has been declared, but no yet given a value is said to be *uninitialized.*
- Uninitialized class variables have the value `null`.
- Uninitialized primitive variables may have a default value.
- Examples:
  - int count = 0; char grade = 'A';
  - String flightNumber = “KE101”;

Lab: Variables

- View sample program listing 2.1
  - **New Java Project**
    - **Eclipse [File]-[New]->[Java Project]**
    - **Name : Lab2-1**
  - **Add** Class



Lab: Variables



- View sample program listing 2.1
  - `Class EggBasket`

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

Lab: Execute / Run!

Eclipse File Edit Run Source Navigate Search Project Refactor Window Help

Java – EggBasket/src/EggBasket.java – Eclipse SDK – /

Package Explorer

- EggBasket
  - src
    - (default package)
      - EggBasket.java
  - JRE System Library [JavaSE-1.6]

Run

```
public class EggBasket {

    /**
     * @param args
     */
    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);

    }

}
```

Change the variable values and check the output

Variables and Values

- Variables (int)

```
numberOfBaskets
eggsPerBasket
totalEggs
```

- Assigning values

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

Assignment Statements

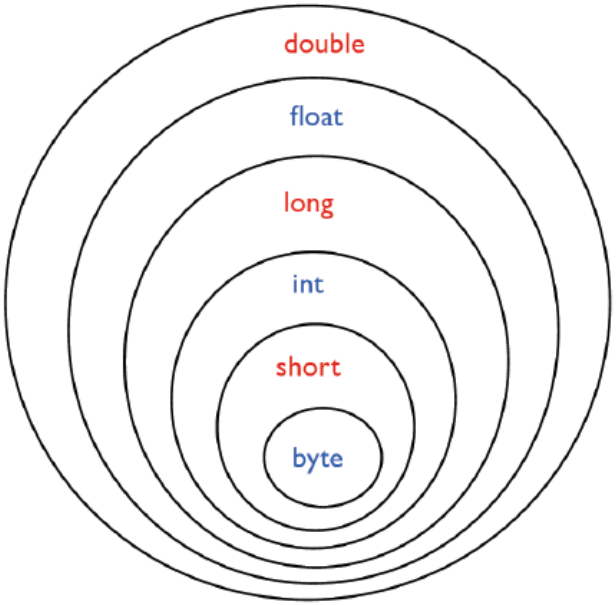
- Assignment evaluation
  - The expression on the right-hand side of the assignment operator (=) is evaluated first
  - The result is used to set the value of the variable on the left-hand side of the assignment operator
  - Examples:
    - score = numberOfCards + handicap;
    - eggsPerBasket = eggsPerBasket - 2;

Assignment Compatibilities

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

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

- Some examples
  - myShort = myInt; (wrong!)
  - myByte = myLong; (wrong!)
  - myFloat = myByte; (Right)
  - myLong = myInt; (Right)



`long`	Integer	8 bytes	–9,223,372,036,8547,75,808 to 9,223,372,036,854,775,807
`float`	Floating-point	4 bytes	$\pm 3.40282347 \times 10^{+38}$ to $\pm 1.40239846 \times 10^{-45}$

Assignment compatibility

- Java is said to be *strongly typed*
  - Conversions between numbers are possible, **but you can't assign a floating point value to an integer variable**
  - double Variable = 7;
  - int Variable = 3.5; (✗)
- How to solve this? **Type casting**
  - You can ask the computer to change the type of values which are against the compatibility
  - myInt = 9.8d; (✗)
  - myInt = (int)9.8d; (ok)

Simple Input

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

  ```
  import java.util.Scanner
  ```

  at the beginning of the file.

Simple Input

- 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`.

Simple Screen Output

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

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

Constants

- `2, 3.7,` or `'y'` are called *constants*.
- Integer constants can be preceded by a + or − sign.
- Example
  - **5** is an integer constant (Default type)
  - 5L is a long constant
  - **5.0** is a double constant (Default type)
  - 5.0f is a float constant

Named Constants

- Java provides mechanism to …
  - Define a variable
  - Initialize it
  - Fix the value so it cannot be changed

```
public static final Type Variable = Constant;
```

  - Example

```
public static final double PI = 3.14159;
```

e Notation

- Floating-point constants can be written
  - With digits after a decimal point or
  - Using *e notation.*
- e notation is 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`

Arithmetic Operators

- Arithmetic expressions can be formed using the `+`, `-`, `*`, and `/` operators together with variables or numbers referred to as *operands*.
- If any operand is of float-point type, so is the result
  - hoursWorked * payRate

    // 40(int) * 8.25 (double) ➔ 500.0(double)
- Expressions with two or more operators can be viewed as a series of steps
  - balance + (balance * rate)

Division and modulo operator

- Division operator (`/`) behaves as expected if one of the operands is a floating-point type.
  - e.g, `99/100` has a value of `0`.
    - When both operands are integer types, the result is truncated, not rounded.
- The `mod` (`%`) operator is used with operators of integer type to obtain the remainder after integer division.
  - e.g. `14 % 4` is equal to `2`
  - Typical usage
    - Determining if an integer is odd or even
    - Determining if one integer is evenly divisible by another integer.

Increment/Decrement Operators

- To increase (or decrease) the value of a variable by 1
  - count++ *or* ++count // increment operator
  - count-- *or* --count // decrement operator
- After executing
  - int m = 4;
    int result = 3 * (++m);
  - *result* has a value of 15 and *m* has a value of 5
- After executing
  - int m = 4;
    int result = 3 * (m++);
  - *result* has a value of 12 and *m* has a value of 5

Specialized Assignment Operators

- Assignment operators can be combined with arithmetic operators (including -, *, /, and %).
- Two assignments below yield the same result
  - amount = amount + 5;
  - amount += 5;

Parentheses() and Precedence

- Parentheses determine the order in which arithmetic operations are performed
- Examples:
  - (cost + tax) * discount
  - cost + (tax * discount)
- Without parentheses, an expression is evaluated according to the rules of precedence

Precedence Rules

Highest Precedence

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

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

Third: the binary arithmetic operators + and –

Lowest Precedence

- When binary operators have equal precedence, the operator on the left acts before the operator(s) on the right.

Sample Expressions

- Figure 2.3 Some Arithmetic Expressions in Java

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

Practice 2.1

- Ex2_1a. Write a following program.
  - Read a four-digit integer, such as 2017
  - Display one digit per line, e.g., 2, 0, 1, and 7 in each line
- Ex2_1b. Write a following program.
  - Read a temperature in Fahrenheit
  - Compute a temperature in Celsius and print it
  - C = 5 (F - 32) / 9

2.2 The CLASS String

Strings

- A value of type String is a sequence of characters
  - E.g., “Hello out there.”
- No primitive type for strings in Java
  - Instead, Java provides a class called String

Class `String`

- We've used constants of type `String` already.
- Declaration
  - `String` greeting;
    greeting = "Hello!";
  - `String` greeting = "Hello!";
  - `String` greeting = new String("Hello!");
- Print
  - System.out.println("Hello!");
  - 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.
  - System.out.println("Good " + "morning, " + "Vietnam!");
- When concatenating with the values of other types, they are first converted to String values
  - int result = 42;
    System.out.println("The answer is " + result);

`String` Indices

- **FIGURE 2.4 String Indices**

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

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

- Positions start with 0, not 1.
  - The 'J' in "Java is fun." is in position 0
- A position is referred to an index.
  - The 'f' in "Java is fun." is at index 8.

String methods

- How to obtain the length of a string ?

```
String greeting = "Hello";
```

```
System.out.println( strlen(greeting)) ??
```

String methods

- An object of the String class stores data consisting of a sequence of characters.
- Each object has methods as well as data
- The length() method returns the number of characters in a particular String object.

object

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

method

Method `length()`

- Returns an **`int`** value indicating the number of characters in a String object
- Can be used anywhere an **`int`** can be used

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

`String` Methods

`charAt` (*Index*) Returns the character at *Index* in this string. Index numbers begin at 0.	String a = "Hello Java"; char k = a.charAt(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.	String a = “apple"; String b=“bear”; System.out.println(a.comparTo(b));
`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 + operator instead of `concat`.	String result = a.concat(b);
`equals`(*Other_String*) Returns true if this string and *Other_String* are equal. Otherwise, returns false.	String a = "Hello"; String b=“hello”; System.out.println(a.equals(b)); System.out.println(a.eaualsIgnoreCase(b));
`equalsIgnoreCase`(*Other_String*) Behaves like the method `equals`, but considers uppercase and lowercase versi 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.	String a = "Hello Java"; System.out.println(a.indexOf("Java"));
`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 a = “2018-03-12"; int index1= a.indexOf(“-"); int index2= a.lastIndexOf(“-");

`String` Methods

`length()` Returns the length of this string.	`String a = "Hello Java";` `int num = a.length();`

`toLowerCase()`
Returns a new string having the same characters as this string, but with any u letters converted to lowercase.
	`String a = “hello";`
	`String b= a.toUpperCase();`
	`String c=a.substring(0,1).toUpperCase();`

`toUpperCase()` Returns a new string having the same characters as this string, but with any lowercase letters converted to uppercase.	
`replace(`*OldChar*`,` *NewChar*`)` Returns a new string having the same characters as this string, but with each occurrence of *OldChar* replaced by *NewChar*.	`String a = “hello";` `String b = a.replace(“e”,”a”);`

`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.
	`String a = “hello";`
	`String b = a.substring(0,3);`
	`String c = a.substring(1);`

`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 a = “  hello  ";` `String b = a.trim();`

Escape Characters

- How would you print

  `"Java" refers to a language.` ?

  – We would write:

    - System.out.println(""Java" refers to a language."); (✗)
    - System.out.println("\"Java\" refers to a language."); (ok)

  – Compiler is told that quotation marks (") do not signal the start or end of a string, but instead are to be printed

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.

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

Examples

```
abc\def
```

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

```
new
line
```

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

```
'
```

```
char singleQuote = '\'';
System.out.println(singleQuote)
  ;
```

Lab: String processing

- Make a program to get the following result

Screen Output

```
Text processing is hard!
0123456789012345678901 23
The word "hard" starts at index 19
The changed string is:
TEXT PROCESSING IS EASY!
```

hard 문자를 easy로 대체 후 대문자로 출력

Lab: String processing

The meaning of \" is discussed in the section entitled "Escape Characters."

```
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("0123456789012345678901 23");
        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);
    }
}
```

Screen Output

```
Text processing is hard!
0123456789012345678901 23
The word "hard" starts at index 19
The changed string is:
TEXT PROCESSING IS EASY!
```

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.
- Unicode vs ASCII
  - ASCII defines 128 characters, uses 7 bits to represent a character.
  - Unicode defines (less than) 2^{21}characters

ASCII table

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Unicode table

TABLE OF SPECIAL CHARACTERS - UNICODE & ISO-8859-1

The decimal digits xxx used to create special characters, as well as accented characters in West European languages.
For the following characters, the digits for decimal Unicode and ISO 8859-1 are identical.

Char	Code		Char	Code		Char	Code		Char	Code		Char	Code		Char	Code		Char	Code		Char	Code
	160		¡	161		¢	162		£	163		¤	164		¥	165		¦	166		§	167
¨	168		©	169		ª	170		«	171		¬	172			173		®	174		¯	175
°	176		±	177		²	178		³	179		´	180		µ	181		¶	182		·	183
¸	184		¹	185		º	186		»	187		¼	188		½	189		¾	190		¿	191
À	192		-	193		Â	194		Ã	195		Ä	196		Å	197		Æ	198		Ç	199
È	200		É	201		Ê	202		Ë	203		Ì	204		Í	205		Î	206		Ï	207
Ð	208		Ñ	209		Ò	210		Ó	211		Ô	212		Õ	213		Ö	214		×	215
Ø	216		Ù	217		Ú	218		Û	219		Ü	220		Ý	221		Þ	222		ß	223
à	224		á	225		â	226		ã	227		ä	228		å	229		æ	230		ç	231
è	232		é	233		ê	234		ë	235		ì	236		í	237		î	238		ï	239
ð	240		ñ	241		ò	242		ó	243		ô	244		õ	245		ö	246		÷	247
ø	248		ù	249		ú	250		û	251		ü	252		ý	253		þ	254		ÿ	255

Examples: To generate the Copyright symbol ©, Type © or hold down the [ALT] key and type: 0169

©2002 B. C. Biega . All rights reserved.

Please return to http://www.biega.com

String memory size?

- Memory usage of Java objects
  - Object header: 8 bytes of housekeeping data recording an object's class, ID and status flags
  - Memory for primitive fields: e.g. int 4 byte
  - Memory for reference fields: 4 byte each
  - Padding up to multiply 8
- E.g. an object with a two long fields, three int fields and a boolean (40 byte)
  - 8 bytes for the header;
  - 16 bytes for the 2 longs (8 byte each);
  - 12 bytes for the 3 ints (4 byte each);
  - 1 byte for the boolean;
  - a further 3 bytes of padding (to make 40, a multiple of 8)

String memory size?

- Memory usage of String objects
  - *"String"* consists of more than one object
  - *"Char"* take up two bytes
  - A String contains the following:
    - a char array containing the actual characters;
    - int for the cached calculation of the hash code.
    - (not in Java 8) ~~offset into the array at which the string starts;~~
    - (not in Java 8) ~~the length of the string;~~
- E.g. Empty char array: 40 byte
  - 8 bytes: Object header
  - 4 byte: Memory for reference fields
  - 12 byte: three int field (hash code)
  - 12 byte: char array(assume 3) +length (int, 4 byte)
  - 4 byte: padding
- Minimum memory usage of a Java String
  - 8 * (int) ((((no chars) * 2) + 45) / 8)

2.3 Keyboard and Screen I/O

System class

- A static variable is common to all the instances (or objects) of the class because it is a class level variable.
- Only a single copy of static variable is created and shared among all the instances of the class.

- Facilities provided by System
  - Standard output
  - Error output streams
  - Standard input and access to externally defined properties and environment variables.
  - A means of loading files and libraries
- It cannot be instantiated ➔ defined with "static"

Field definition	Explanation
static PrintStream err	This is the "standard" error output stream.
static InputStream in	This is the "standard" input stream.
static PrintStream out	This is the "standard" output stream.

Screen Output

- We've seen several examples of screen output.
  - `System.out` is an object that is part of Java.
  - `println()` is one of the methods of the `System.out` object.
- Concatenation operator (+) is useful
  - System.out.println("Lucky number = " + 13 + " Secret number = " + number);
- Alternatively, use `print()`

```
System.out.print("One, two,");
System.out.print(" buckle my shoe.");
System.out.println(" shut the door.");
```

ending with a `println()`.

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.

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();
```

Scanner method

Scannner_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.

Scannner_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.

Scannner_Object_Name`.nextInt()`
Returns the next keyboard input as a value of type `int`.

Scannner_Object_Name`.nextDouble()`
Returns the next keyboard input as a value of type `double`.

Scannner_Object_Name`.nextFloat()`
Returns the next keyboard input as a value of type `float`.

Scanner method

Scannner_Object_Name`.nextLong()`
Returns the next keyboard input as a value of type `long`.

Scannner_Object_Name`.nextByte()`
Returns the next keyboard input as a value of type `byte`.

Scannner_Object_Name`.nextShort()`
Returns the next keyboard input as a value of type `short`.

Scannner_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*.

Scannner_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.

Scanner methods

- Scanner methods
  - nextDouble() – reads one double value from keyboard
    - double d1 = keyboard.nextDouble();
  - next() – reads one word from keyboard
    - String s1 = keyboard.next();
  - nextLine() – reads an entire line
    - reads the remainder of the current line, even if it is empty.
    - String s2 = keyboard.nextLine();

Lab: Keyboard Input Demonstration

- View sample program

  `class 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."
```

LISTING 2.5 A Demonstration of Keyboard Input *(part 1 of 2)*

```java
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 + "\"");
    }
}
```

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

Sets things up so the program can accept keyboard input

Reads one `int` *value from the keyboard*

Reads one `double` *value from the keyboard*

Reads one word from the keyboard

This line is explained in the next Gotcha section.

Reads an entire line

Practice 2.2

- Ex2_2. Write a following program
  - Read a line of text
  - Move the first word to the end and capitalize the first character, and then print it
  - E.g., “Java is the language” → “Is the language Java”

2.4 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.
- The best programs are self-documenting.
  - Clean style
  - Well-chosen names

Meaningful Variable Names

- 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.

Comments

- Written into a program as needed for self-explanation and ignored by the compiler.
- A single line comment starts with //
  - double radius; // in centimeters
- A multi-line comment begins with /* and end with */
  - /* This program should only
    be used on alternate Thursdays */
- A *javadoc* comment begins with /** and ends with */
  - Extracted automatically from Java software
  - /** method change requires the number of coins
    to be non-negative */

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 for each level of indentation. (You simply use [TAB] key.)
  - Indentation does not change the behavior of the program.

Using Named Constants

- Once the value of a constant is set,

it can be used throughout the program

```
area = PI * radius * radius;
```

is clearer than

```
area = 3.14159 * radius * radius;
```

- Place constants near the beginning of the program.

```
public static final double INTEREST_RATE = 6.65;
public static final String MOTTO = "The customer
is always right.";
```

- By convention, uppercase letters are used for constants.

static? (고정된, 정적인)

- Get memory only once in the class area at the time of class loading
- Use to refer to the common property of all objects

final?

- Define an entity that can only be assigned once.

Lab: Named Constants

- View sample program

  `class CircleCalculation2,` listing 2.8

```
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.
```

```java
/**
 Program to compute area of a circle.
 Author: Jane Q. Programmer.
 E-mail Address: janeq@somemachine.etc.etc.
 Programming Assignment 2.
 Last Changed: October 7, 2008.
*/

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.");
    }
}
```

Although it would not be as clear, it is legal to place the definition of PI *here instead.*

Practice 2.3

- Ex2_3a. Write a following program
  - Read the price of an item in cents: a multiple of 5 between 25 and 100, i.e., 25, 30, …, 95, or 100
  - Assume you paid **a dollar(100 cents)**, and print the number of quarter (25cents), dime (10 cents), and nickel (5 cents) coins for the change
  - E.g., for an item of 45 cents, the change is 55 cents, which is given by 2 quarters, 0 dimes, and 1 nickels

Assignment

- Read chapter 3.1-3.3