COMP-248
Object Oriented Programming I

Lecture 02:

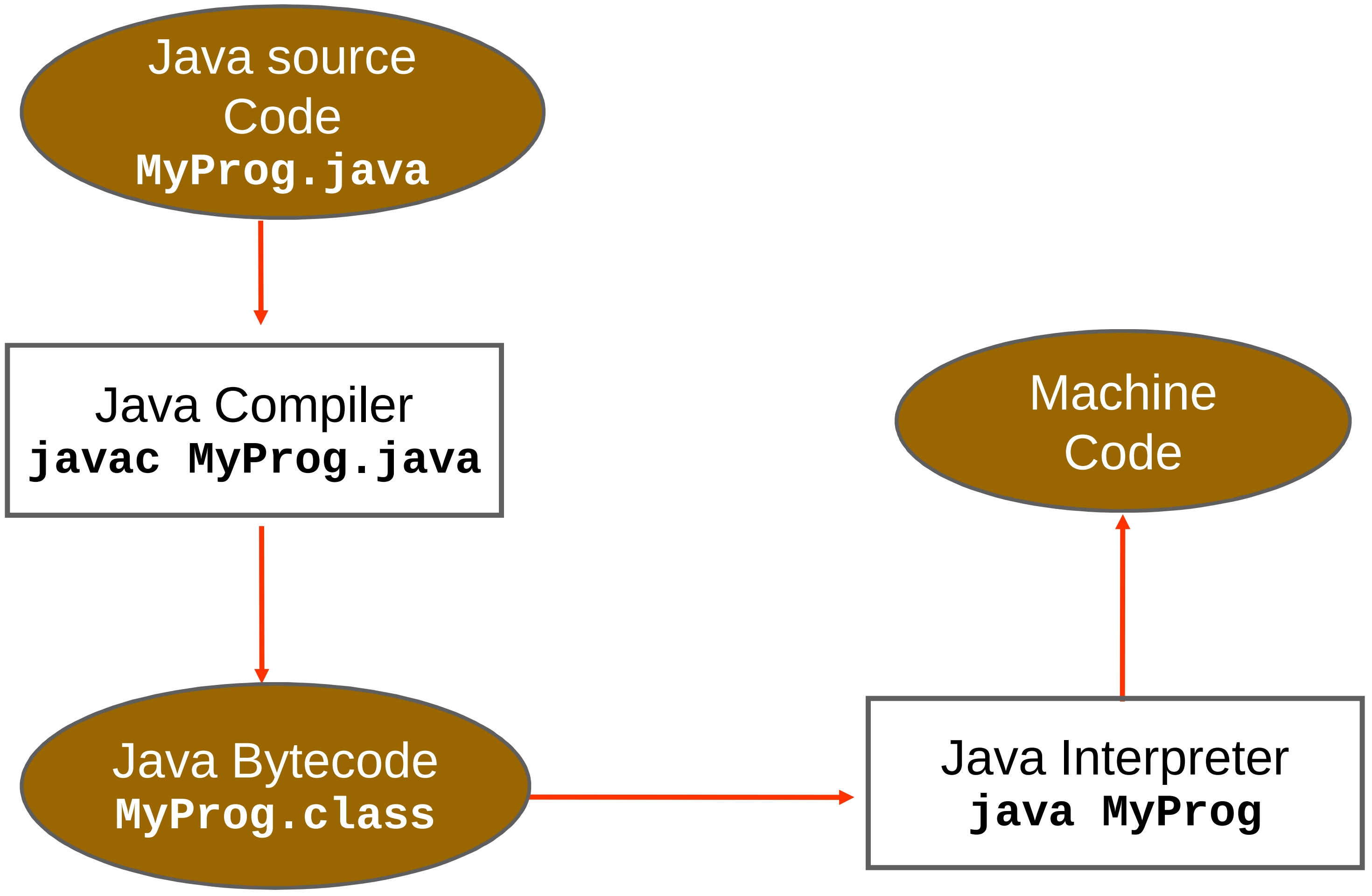
Java Fundamentals

Emad Shihab, PhD

Parts of the slides are taken from Prof. L. Kosseim

Last class:

Java Translation



Java source Code MyProg.java
Java Compiler javac MyProg.java
Java Bytecode MyProg.class
Java Interpreter java MyProg
Machine Code

Last class:

Java Program Structure

```
//  comments about the class
public class MyProgram
{

   //  comments about the method
   public static void main (String[] args)
   {

   }

}
```

method header

method body

MyProgram.java

Last Class:
Three types of errors

- Compile-time (syntax) errors
- Run-time errors
- Logical (semantic

Today, we will see:

- Java Fundamentals
  - **Comments**
  - Identifiers
  - Indentation
  - Primitive Types
  - Variables
  - Output & Input
  - Assignment
  - Arithmetic Expressions
  - (later) More Assignment Operators
  - (later) Assignment Compatibility
  - Strings

1- Comments (p. 49)

- also called *inline* documentation
- used to explain the purpose of the program and describe processing steps (the algorithm)
- do not affect how a program works (are ignored by the compiler)
- can take 3 forms:

```
// this comment runs to the end of the line
```

Line

```
/*   this comment runs to the terminating
     symbol, even across line breaks             */
```

Block

```
/** this is a javadoc comment   */
```

javadoc

When to comment

- Very difficult to determine, however, there are a few rules of thumb:
  - When you know your code needs to be explained
  - You know others are not/less experienced with what you did
  - You did something bad/risky

- **Note**: make your comments worthwhile (don’t over crowd the code with useless comments)

- Always strive for self documentation ☺

In this chapter, we will see…

1. Comments
2. **Identifiers**
3. Indentation
4. Primitive Types
5. Variables
6. Output & Input
7. Assignment
8. Arithmetic Expressions
9. (later) More Assignment Operators
10. (later) Assignment Compatibility
11. Strings

Template of a simple Java program

```
//************************************************************
// comments on the program (authors, purpose, …)
//************************************************************

public class SomeIdentifier
{
   //--------------------------------------------------------
   //  comments on the main method
   //--------------------------------------------------------
   public static void main (String[] args)
   {
      // declarations of variables and constants
      // statements of the main method
   }
}
```

SomeIdentifier.java

2- Identifiers (p. 13)

- are the words a programmer uses in a program to **name variables, classes, methods**, …

- Rules to create an identifier:
  - can be made up of: letters, digits, the underscore character (_), and the dollar sign ($)
  - no limit on length
  - **cannot** begin with a digit
  - **cannot** be a *reserved word*

Just checking...

Which of the following is <u>not</u> a valid identifier?

a) abc

b) ABC

c) Abc

d) Ab.C

e) a bc

Just checking...

Which of the following is not a valid identifier?

a) a_bc%

b) A$BC

c) _Abc

d) 1AbC

e) $abc

Guidelines for Identifiers

- Give a significant name!
- Avoid '$' (since it is reserved for special purposes)
- by convention:

  class names --> use title case

  ex: **`MyClass, Lincoln`**

  constants --> use upper case

  ex: **`MAXIMUM`**

  variables, methods, ... --> start with lowercase

  `ex: aVar, a_var`

- Avoid Predefined identifiers:

  Although they can be redefined, it is confusing and dangerous

  `System     String     println`

- Remember: Java is case sensitive!

52 Java reserved words

`abstract`	`else`	`interface`	`super`
`assert`	`extends`	`long`	`switch`
`boolean`	`false`	`native`	`synchronized`
`break`	`final`	`new`	`this`
`byte`	`finally`	`null`	`throw`
`case`	`float`	`package`	`throws`
`catch`	`for`	`private`	`transient`
`char`	`goto`	`protected`	`true`
`class`	`if`	`public`	`try`
`const`	`implements`	`return`	`void`
`continue`	`import`	`short`	`volatile`
`default`	`instanceof`	`static`	`while`
`do`	`int`	`strictfp`	
`double`			

Examples

Identifier	Correct or not?
`GST`	
`PriceBeforeTax`	
`Student_3`	
`student#3`	
`Shipping&HandlingFee`	
`Class`	
`___123`	
`the account`	
`1floor`	

In this chapter, we will see…

1. Comments
2. Identifiers
3. **Indentation**
4. Primitive Types
5. Variables
6. Output & Input
7. Assignment
8. Arithmetic Expressions
9. (later) More Assignment Operators
10. (later) Assignment Compatibility
11. Strings

3- Indentation (p. 50)

- Spaces, blank lines, and tabs are called white space
- White space is used to separate words and symbols in a program
- Extra white space is ignored
- Programs should be formatted to enhance readability, using consistent indentation

Example 1 of bad indentation

```
//********************************************************************
//  Lincoln2.java
//  Demonstrates a poorly formatted, though valid, program.
//********************************************************************

public class Lincoln2{public static void main(String[]args){
System.out.println("A quote by Abraham Lincoln:");
System.out.println("Whatever you are, be a good one.");}}
```

Example 2 of bad indentation

```
//********************************************************************
//  Lincoln3.java
//  Demonstrates another valid program that is poorly formatted.
//********************************************************************
          public       class
     Lincoln3
   {
                 public
   static
        void
  main
        (
String
            []
    args                       )
  {
  System.out.println        (
"A quote by Abraham Lincoln:"          )
  ;        System.out.println
            (
       "Whatever you are, be a good one."
      )
  ;
}
          }
```

Example 3 of good indentation

```
//********************************************************************
//  Lincoln3.java
//  Demonstrates a properly formatted program.
//********************************************************************

public class Lincoln3
{
   public static void main(String[]args)
   {
       System.out.println("A quote by Abraham Lincoln:");
       System.out.println("Whatever you are, be a good one.");
   }
}
```

In this chapter, we will see…

4- Primitive Types (p. 16)

- 8 primitive data types in Java

  Numeric

  4 types to represent integers (ex. 3, -5):

  ```
  byte, short, int, long
  ```

  2 types to represent floating point numbers (ex. 3.5):

  ```
  float, double
  ```

  Characters (ex. **'a'**)

  ```
  char
  ```

  Boolean values (true/false)

  ```
  boolean
  ```

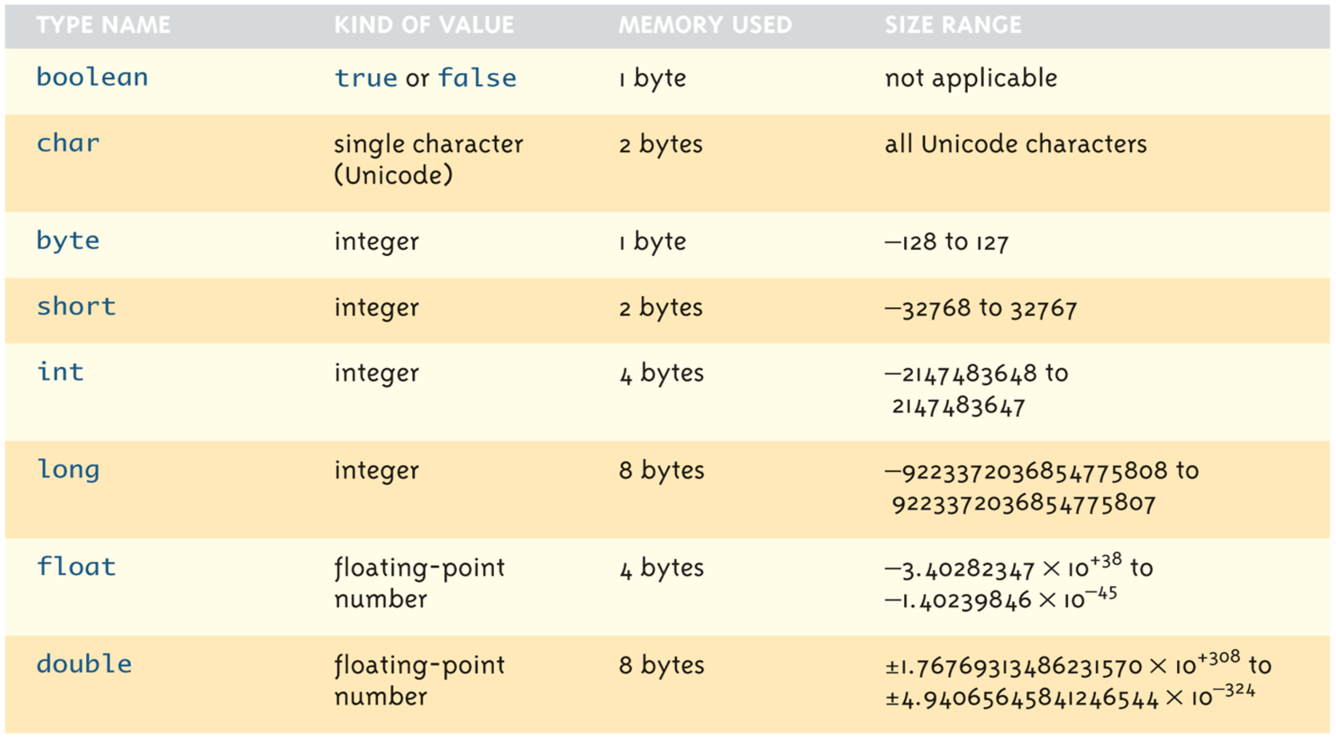
Numerical Types

- The difference between:

  `byte, short, int, long` *AND* **`float, double`**

  is their size (so the values they can store)

Display 1.2 **Primitive Types**



TYPE NAME	KIND OF VALUE	MEMORY USED	SIZE RANGE
`boolean`	`true` or `false`	1 byte	not applicable
`char`	single character (Unicode)	2 bytes	all Unicode characters
`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	$-3.40282347 \times 10^{+38}$ to $-1.40239846 \times 10^{-45}$
`double`	floating-point number	8 bytes	$\pm 1.76769313486231570 \times 10^{+308}$ to $\pm 4.94065645841246544 \times 10^{-324}$

⚠ Round-Off Errors in Floating-Point Numbers

- Floating point numbers are only approximate quantities
  - Mathematically, the floating-point number 1.0/3.0 is equal to 0.3333333. . .
  - A computer may store 1.0/3.0 as something like 0.3333333333

Characters

- A char stores a single character
- delimited by single quotes:

  'a'   'X'   '7'   '$'   ','   '\n'

- characters are ordered according to a character set and each character corresponds to a unique number code

- Java uses the Unicode character set

  16 bits per character, so 65,536 possible characters

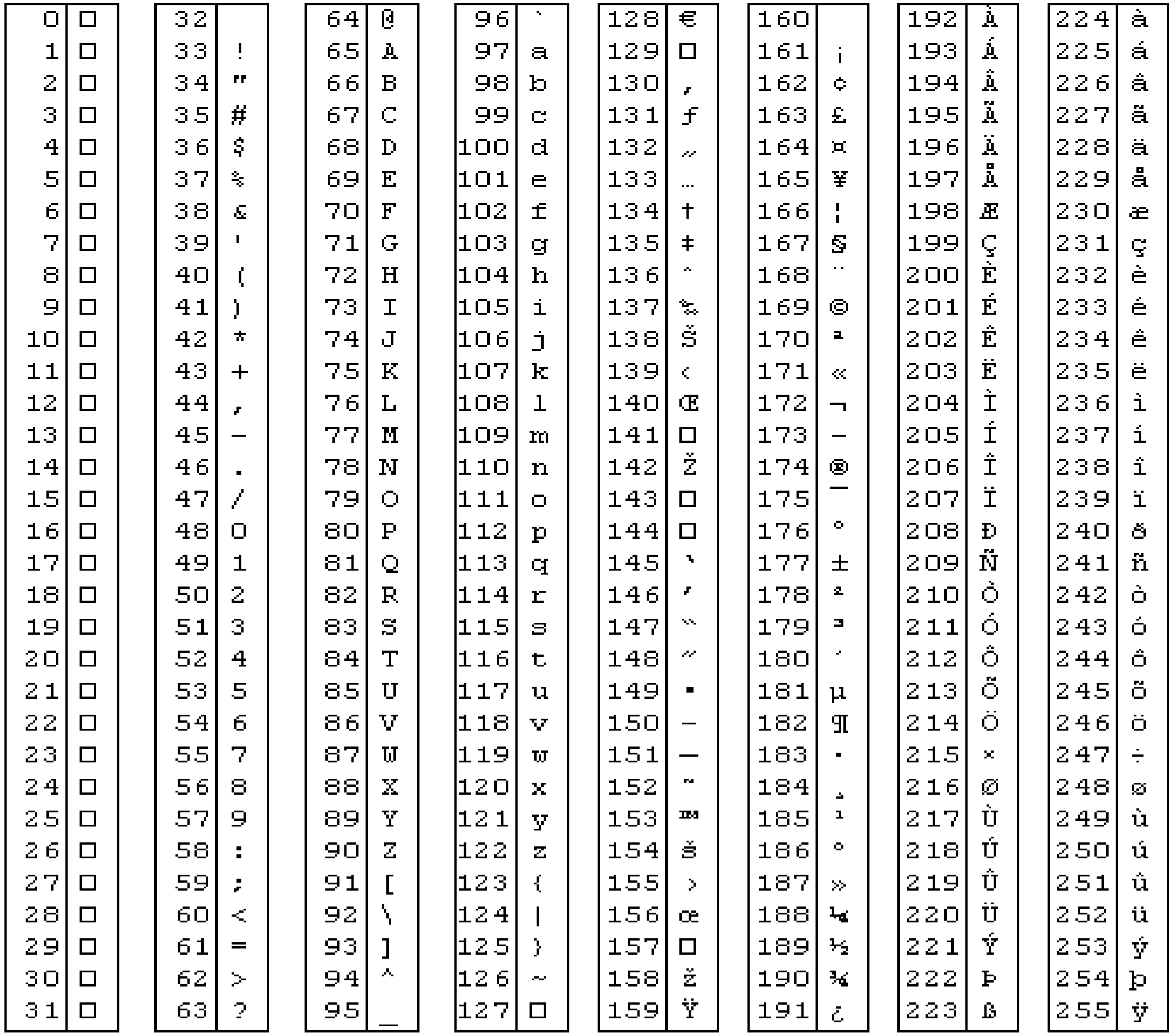
  Unicode is an international character set, containing symbols and characters from languages with different alphabets

Characters

- The **ASCII character set** is older and smaller than **Unicode**, but is still popular
- The ASCII characters are a subset of the Unicode character set, including:

uppercase letters	’A’, ’B’, ’C’, …
lowercase letters	’a’, ’b’, ’c’, …
punctuation	’. ’, ’; ’, …
digits	’0’, ’1’, ’2’, …
special symbols	’&’, ’|’, ’\’, …
control characters	’\n’, ’\t’, …

Part of the Unicode Character



0	□	32		64	@	96	`	128	€	160		192	À	224	à
1	□	33	!	65	A	97	a	129	□	161	¡	193	Á	225	á
2	□	34	"	66	B	98	b	130	‚	162	¢	194	Â	226	â
3	□	35	#	67	C	99	c	131	ƒ	163	£	195	Ã	227	ã
4	□	36	$	68	D	100	d	132	„	164	¤	196	Ä	228	ä
5	□	37	%	69	E	101	e	133	…	165	¥	197	Å	229	å
6	□	38	&	70	F	102	f	134	†	166	¦	198	Æ	230	æ
7	□	39	'	71	G	103	g	135	‡	167	§	199	Ç	231	ç
8	□	40	(	72	H	104	h	136	ˆ	168	¨	200	È	232	è
9	□	41	)	73	I	105	i	137	‰	169	©	201	É	233	é
10	□	42	*	74	J	106	j	138	Š	170	ª	202	Ê	234	ê
11	□	43	+	75	K	107	k	139	‹	171	«	203	Ë	235	ë
12	□	44	,	76	L	108	l	140	Œ	172	¬	204	Ì	236	ì
13	□	45	-	77	M	109	m	141	□	173	-	205	Í	237	í
14	□	46	.	78	N	110	n	142	Ž	174	®	206	Î	238	î
15	□	47	/	79	O	111	o	143	□	175	¯	207	Ï	239	ï
16	□	48	0	80	P	112	p	144	□	176	°	208	Ð	240	ð
17	□	49	1	81	Q	113	q	145	‘	177	±	209	Ñ	241	ñ
18	□	50	2	82	R	114	r	146	’	178	²	210	Ò	242	ò
19	□	51	3	83	S	115	s	147	“	179	³	211	Ó	243	ó
20	□	52	4	84	T	116	t	148	”	180	´	212	Ô	244	ô
21	□	53	5	85	U	117	u	149	•	181	µ	213	Õ	245	õ
22	□	54	6	86	V	118	v	150	–	182	¶	214	Ö	246	ö
23	□	55	7	87	W	119	w	151	—	183	·	215	×	247	÷
24	□	56	8	88	X	120	x	152	˜	184	¸	216	Ø	248	ø
25	□	57	9	89	Y	121	y	153	™	185	¹	217	Ù	249	ù
26	□	58	:	90	Z	122	z	154	š	186	º	218	Ú	250	ú
27	□	59	;	91	[	123	{	155	›	187	»	219	Û	251	û
28	□	60	<	92	\	124	|	156	œ	188	¼	220	Ü	252	ü
29	□	61	=	93	]	125	}	157	□	189	½	221	Ý	253	ý
30	□	62	>	94	^	126	~	158	ž	190	¾	222	Þ	254	þ
31	□	63	?	95	_	127	□	159	Ÿ	191	¿	223	ß	255	ÿ

Booleans

- A boolean value represents a true or false expression

- The reserved words **`true`** and **`false`** are the only valid values for a boolean type

⚠ NOT… 0 and 1

In this chapter, we will see…

1. Comments
2. Identifiers
3. Indentation
4. Primitive Types
5. **Variables**
6. Output & Input
7. Assignment
8. Arithmetic Expressions
9. (later) More Assignment Operators
10. (later) Assignment Compatibility
11. Strings

5- Variables (p. 15)

- a name for a location in memory
- used to store information (ex. price, size, …)
- must be declared before it is used
    - indicate the variable's **name**
    - indicate the **type** of information it will contain
    - declaration can be anywhere in the program (but before its first use)

data type

variable name

```
int total;
```

```
int count, temp, result;
```

Multiple variables can be created in one declaration

Tip: Initialize Variables

- A variable that has been declared but has not yet been given a value is said to be *uninitialized*

- In certain cases an uninitialized variable is given a default value
  - It is best not to rely on this
  - Explicitly initialized variables have the added benefit of improving program clarity

Initialization at declaration

- A variable can be given an initial value in the declaration

```
boolean isChild = false;
int base = 32, max = 149;
```

- When a variable is used in a program, its **current value is used**

Example

```
//*******************************************************************
//  PianoKeys.java
//
//  Demonstrates the declaration and initialization of an integer variable.
//*******************************************************************

public class PianoKeys
{
   //  Prints the number of keys on a piano.
   public static void main (String[] args)
   {
      int keys = 88;

      System.out.println("A piano has" + keys + "keys.");
   }
}
```

filename??

???

```
A piano has 88 keys.
A piano has88keys.
```

Output

Constants (p. 21)

- Similar to a variable but can only hold one value while the program is active
- The compiler will issue an error if you try to change the value of a constant during execution
- Use the **`final`** modifier

```
final int MIN_AGE = 18;
```

- Constants:
  - give names to otherwise unclear literal values
  - facilitate updates of values used throughout a program
  - prevent inadvertent attempts to change a value

In this chapter, we will see…

1. Comments
2. Identifiers
3. Indentation
4. Primitive Types
5. Variables
6. **Output & Input**
7. Assignment
8. Arithmetic Expressions
9. (later) More Assignment Operators
10. (later) Assignment Compatibility
11. Strings

6- Output & Input (chap. 2)

`System.out.print`

Displays what is in parenthesis

`System.out.println`

Displays what is in parenthesis
Advances to the next line

Examples:

```
System.out.print("hello");
System.out.print("you");

System.out.println("hello");
System.out.println("you");

System.out.println();

int price = 50;
System.out.print(price);

char initial = 'L';
System.out.println(initial);
```

```
helloyouhello
you

50L
```

Output

Multiple output

```
System.out.println("hello" + "you");
double price = 9.99;
int nbItems = 5;
System.out.println("total = " + price*nbItems + "$");
```

???

```
helloyou
total = 49.95$
```

Output

⚠ **`print`** and **`println`**, + is the concatenation…

you need parenthesis for the + to be addition

```
int x = 1, y = 2;
System.out.println("x+y="+x+y);
System.out.println("x+y="+(x+y));
```

???

```
x+y=12
x+y=3
```

Output

⚠ cannot *cut* a string over several lines

```
System.out.println("this is a
                   long string"); // error!
System.out.println("this is a" +
                   "long string"); // ok
```

Escape sequences (p. 42)

```
System.out.println ("I said "Hi" to her.");
```

```
???
```
Output

To print a double quote character

- Use an ***escape sequence***
  - sequence is a series of characters that represents a special character
  - begins with a backslash character (\)
  - considered as 1 single character

```
System.out.println ("I said \"Hi\" to her.");
```

```
???  I said "Hi" to her.
```
Output

Escape sequences

- Some Java escape sequences:

Escape Sequence	Meaning
`\b`	backspace
`\t`	tab
`\n`	newline
`\"`	double quote
`\'`	single quote
`\\`	backslash

Just checking...

- What will the following statement output?

```
System.out.print("one\ntwo\nthree\n");
```

```
a)one two three
b)one\ntwo\nthree\n
c)"one\ntwo\nthree\n“
d) one
   two
   three
e)onetwothree
```

Just checking...

- What statement will result in the following output?

```
Read the file "c:\windows\readme.txt"
```

```
System.out.print
a) ("Read the file "c:\windows\readme.txt");
b) ("Read the file "c:\windows\readme.txt"");
c) ("Read the file "c:\\windows\\readme.txt");
d) ("Read the file \"c:\\windows\\readme.txt\"");
e) ("Read the file \"c:\windows\readme.txt\"");
```

Console Input (p. 76)

since Java 5.0, use the **`Scanner`** class

the keyboard is represented by the **`System.in`** object

```
import java.util.Scanner;

public class MyProgram
{
   public static void main (String[] args)
   {
      Scanner myKeyboard = new Scanner(System.in);
      …
      String name = myKeyboard.next();
      int age = myKeyboard.nextInt();
      …
```

1. Create an object of class Scanner
2. Reads one **word** from the keyboard
3. Reads an integer from the keyboard

To read from a Scanner

to read *tokens*, use a *nextSomething()* method

```
nextBoolean(),
nextByte,
nextInt(),
nextFloat(),
nextDouble(),
next(),
nextLine()
…
```

tokens are delimited by whitespaces
(ie blank spaces, tabs, and line breaks)

Note: no **`nextChar()`**

```
import java.util.Scanner;
…
Scanner myKeyboard = new Scanner(System.in);
System.out.println("Your name:");
String name = myKeyboard.next();
System.out.println("Welcome " + name + " Enter your age:");
int age = myKeyboard.nextInt();
```

Example: ScannerDemo.java

```
//**********************************************************
// Author: W. Savitch (modified by L. Kosseim)
//
//  This program demonstrates how to read tokens from
//  the console with the Scanner class
//**********************************************************

import java.util.Scanner; // we need to import this class

public class ScannerDemo
{
   public static void main(String[] args)
   {
      // let's declare our scanner
      Scanner keyboard = new Scanner(System.in);
```

Example: ScannerDemo.java

```
 // let's ask the user for some input
 System.out.println("Enter the number of pods followed by");
 System.out.println("the number of peas in a pod:");

 // let's read the user input
 int numberOfPods = keyboard.nextInt();
 int peasPerPod = keyboard.nextInt();

 // let’s do some calculations
 int totalNumberOfPeas = numberOfPods*peasPerPod;

// let's display some output
 System.out.print(numberOfPods + " pods and ");
 System.out.println(peasPerPod + " peas per pod.");
 System.out.println("The total number of peas = " + totalNumberOfPeas);
}
}
```

A note on `readLine`

`nextLine` reads the remainder of a line of text starting where the last reading left off

This can cause problems when combining it with different methods for reading from the keyboard such as `nextInt`

ex:

```
Scanner keyboard = new Scanner(System.in);
int n = keyboard.nextInt();
String s1 = keyboard.nextLine();
String s2 = keyboard.nextLine();
```

input:

```
2
Heads are better than
1 head.
```

what are the values of `n`, `s1`, and `s2`?

need an extra invocation of `nextLine` to get rid of the end of line character after the 2

Next class

1. Comments
2. Identifiers
3. Indentation
4. Primitive Types
5. Variables
6. Output & Input
7. **Assignment**
8. Arithmetic Expressions
9. (later) More Assignment Operators
10. (later) Assignment Compatibility
11. Strings