Chapter 3

Flow of Control

- 3.1 The if-else Statements
- 3.2 The Type Boolean
  - » Boolean data type and expressions
- 3.3 The Switch Statements
- 3.4 Graphics Supplement

Objective (this chapter)

- Use the Java Branching statements – if-else and switch – in a program
- compare values of a primitive type
- compare objects such as strings
- Use the primitive data type boolean
- Use simple enumerations in a program
- Use Color in a graphics programs
- Use the class JOptionPane to create a dialog box for a year-or-no question.

What is "Flow of Control"?

- Flow of Control
  - » <u>the execution order</u> of instructions in a program
- All programs can be written with three control flow elements:
  1. [redacted] - just go to the next instruction
  2. [redacted] - a choice of at least two
     - – either go to the next instruction
     - – or jump to some other instruction
  3. [redacted] - repeat a block of code at the end of the loop
     - – either go back and repeat the block of code
     - – or continue with the next instruction after the block

Java Flow Control Statements

[redacted]

- the default
- Java automatically executes the next instruction unless you use a branching statement

[redacted]

- if
- if-else
- if-else if-else if- … - else
- switch

[redacted]

- while
- do-while
- for

3.1 The if-else STATEMENTS

- If
- If-else statement
- Boolean expression
- Nested statement and compound statement
- Multibranch if-else statement
- Switch statement

Java **`if`** Statement

- Simple selection
- Do the next statement if test is true or skip it if false
- Syntax:

```
if ([redacted])
    Action if true;   //execute only if true
next action;   //always executed
```

- Note the indentation for [redacted] (not compile or execution correctness)

if Example

```
if(eggsPerBasket < 12)
  //begin body of the if statement
  System.out.println("Less than a dozen eggs per basket");
  //end body of the if statement
totalEggs = numberOfEggs * eggsPerBasket;
System.out.println("You have a total of
                           + totalEggs + " eggs.");
```

- The body of the if statement is conditionally executed
- Statements after the body of the if statement always execute

Multiple Statements

- `Action if true` can be either a single Java statement or a set of statements enclosed in braces { }.
- A set of statements in braces is called a [redacted] and can be used anywhere a single statement can be used.

All statements between braces are controlled by `if`

```
if(eggsPerBasket < 12)
{  //begin body of the if statement
   System.out.println("Less than a dozen ...");
   costPerBasket = 1.1 * costPerBasket
}   //end body of the if statement

totalEggs = numberOfEggs * eggsPerBasket;
System.out.println("You have a total of "
            + totalEggs + " eggs.");
```

Two-way Selection: **`if-else`**

- Select either one of two options
- Either do Action1 or Action2, depending on test value
- Syntax:

```
if (Boolean_Expression)
{
   Action1 //execute only if true
}
else
{
  Action2//execute only if false
}
Action3//always executed
```

`if-else` Examples

- Example with single-statement blocks:

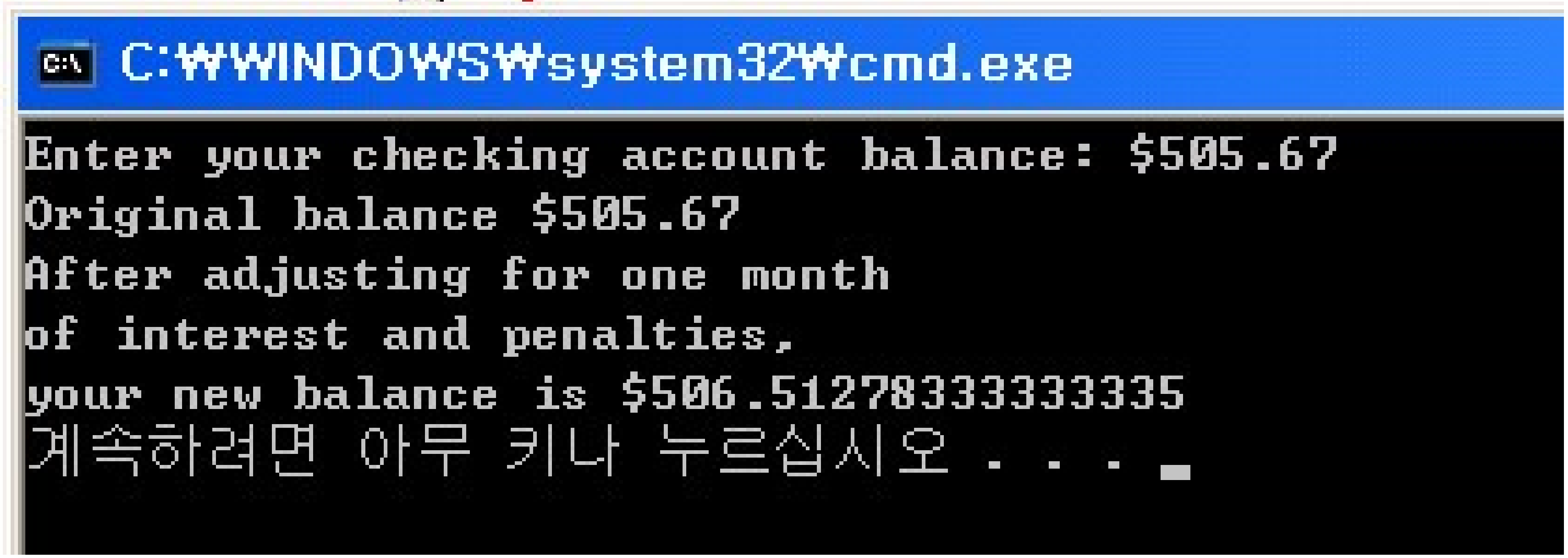
```
if(time < limit)
   System.out.println(“You made it.”);
else
   System.out.println(“You missed the deadline.”);
```

- Example with compound statements:

```
if(time < limit)
{
   System.out.println(“You made it.”);
   bonus = 100;
}
else
{
   System.out.println(“You missed the deadline.”);
   bonus = 0;
}
```

Listing 3.1

- Listing 3.1 A Program Using if-else
  - » BankBalance.java



```
Enter your checking account balance: $505.67
Original balance $505.67
After adjusting for one month
of interest and penalties,
your new balance is $506.5127833333335
계속하려면 아무 키나 누르십시오 . . .
```

```
// Listing 3.1 A Program Using if-else

import java.util.Scanner;

public class BankBalance
{
   public static final double OVERDRAWN_PENALTY = 8.00;
   public static final double INTEREST_RATE = 0.02;//2% annually

   public static void main(String[] args)
   {
      double balance;

      System.out.print("Enter your checking account balance: $");
      Scanner keyboard =  new Scanner(System.in);
      balance = keyboard.nextDouble( );
      System.out.println("Original balance $" + balance);

      if (balance >= 0)
         balance = balance + (INTEREST_RATE * balance)/12;
      else
         balance = balance - OVERDRAWN_PENALTY;

      System.out.println("After adjusting for one month");
      System.out.println("of interest and penalties,");
      System.out.println("your new balance is $" + balance);
   }
}
```

Omitting the `else` Part

- If the `else` part is omitted and the expression after the `if` is false, no action occurs.
- syntax

```
if (Boolean_Expression)
    Statement
```

- example

```
if (weight > ideal)
    caloriesPerDay -= 500;
```

Definition of Boolean Values

- Branching: there is more than one choice for the next instruction
- Which branch is taken depends on a test condition which evaluates to either true or false
- In general:
  if test is true then do this, otherwise it is false, do something else

Introduction to Boolean Expressions

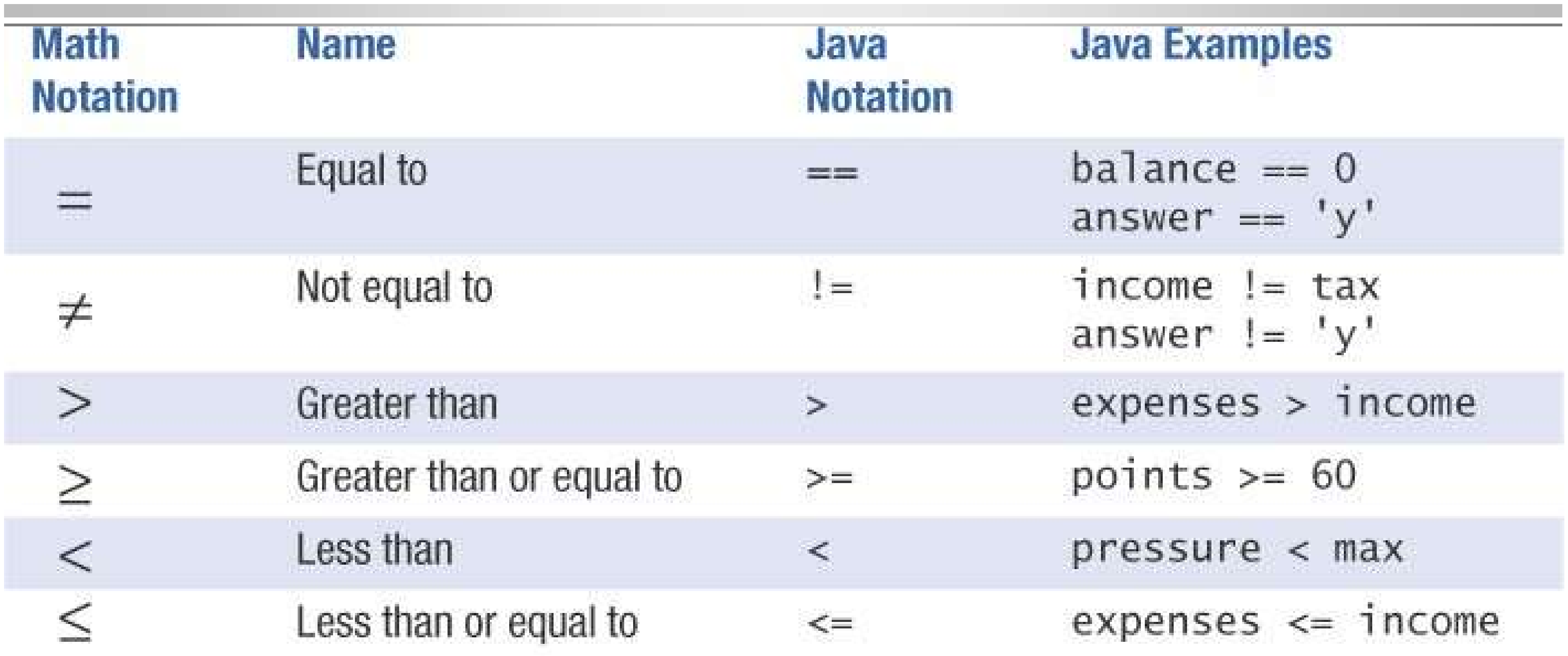
- *boolean* variables (or expressions)
  - » Variables (or expressions) that are either true or false
- the value of a boolean variable (or expression)
  - » `true` or `false`
- `boolean` is [redacted] type in Java
- examples

```
time < limit
balance <= 0
```

Boolean Expressions

- Boolean expressions
  - » test conditions (questions) that are <u>either true or false</u>
  - » Often two values are compared
- For example:
  Is A greater than B?
  Is A equal to B?
  Is A less than or equal to B?
  etc.
- A and B can be any data type (or class), but they <u>should be the [redacted] data type</u> (or class)

Java Comparison Operators



Math Notation	Name	Java Notation	Java Examples

$=$	Equal to	`==`	`balance == 0`
			`answer == 'y'`

$\neq$	Not equal to	`!=`	`income != tax`
			`answer != 'y'`

$>$	Greater than	`>`	`expenses > income`
$\geq$	Greater than or equal to	`>=`	`points >= 60`
$<$	Less than	`<`	`pressure < max`
$\leq$	Less than or equal to	`<=`	`expenses <= income`

Display 3.2

Java Comparison Operators

Compound Boolean Expressions

- Use `&&` to AND two or more conditions
  - » Expression will be true if both parts are true.
  - » `A && B` will be true if both `A` and `B` are true
- Use `||` to OR two or more conditions
  - » Expression will be true if either part is true, or if both parts are true.
  - » `A || B` will be true if either `A` or `B` is true, or if both `A` and `B` are true.

Compound Boolean Expressions

- Boolean expressions can be combined using the "and" (&&) operator.
- example

  ```
  if ((score > 0) && (score <= 100))
  ...
  ```

- if (0 < score <= 100)
  - » ??

Compound Boolean Expressions, cont.

- syntax

  (*Sub_Expression_1*) && (*Sub_Expression_2*)
- Parentheses often are used to enhance [redacted]
- The larger expression is true only when both of the smaller expressions are true.

Compound Boolean Expressions, cont

- Boolean expressions can be combined using the “or” (||) operator.
- example

```
if ((quantity > 5) || (cost < 10))
...
```

- syntax

  (*Sub_Expression_1*) || (*Sub_Expression_2*)

Compound Boolean Expressions

- Example: write a test to see if B is either 0 or between the values of B and C

```
(B == 0) || (A <= B && B < C)
```

- In this example the parentheses are not required but are added for clarity
  - » See text (and later slides) for Precedence rules
  - » Note [redacted] rules in text (and later in slides)
  - » [redacted] of a boolean expression
    - – Use a single `&` for AND and a single `|` for OR
    - – to avoid short-circuit evaluation and force

Negating a Boolean Expression

- A boolean expression can be negated using the “not” (!) operator.
- syntax

  !(Boolean_Expression)
- example

  (a || b) && !(a && b)

  which is the *exclusive or*

Using `== (Primitive Type)`

- == is appropriate
  - » for determining if [redacted] have the same value.

  if (a == 3) where a is [redacted]
- == is **not** appropriate
  - » for determining if [redacted] are equal. Use < and some appropriate tolerance

  if (abs(b - c) < epsilon)
  where b, c, and epsilon are

Using ==, cont (between objects).

- == is not appropriate for determining if two objects have the same value.
  - » if (s1 == s2), where s1 and s2 refer to strings, determines only if s1 and s2 refer the a common [redacted]
  - » If s1 and s2 refer to strings with identical sequences of characters, but stored in different memory locations, (s1 == s2) is [redacted]

Using ==, cont.

- To test the equality of objects of class String, use method equals.

  [redacted]

  or

  [redacted]

- To test for equality ignoring case, use method equalsIgnoreCase.

  ("Hello".equalsIgnoreCase("hello"))

`equals` and `equalsIgnoreCase`

- syntax

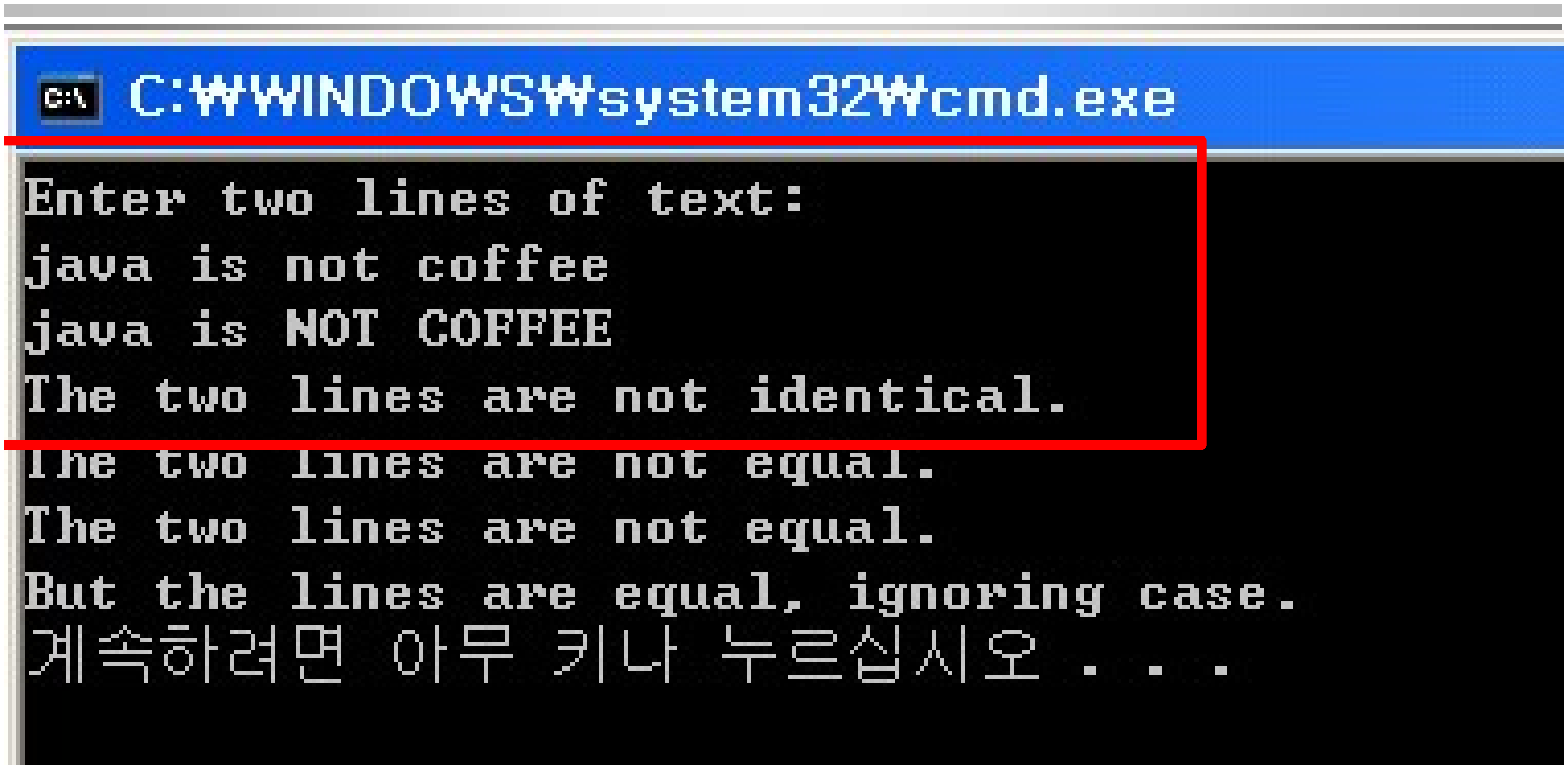
  String`.equals(`*Other_String*`)`

  String`.equalsIgnoreCase(`*Other_String*`)`

Listing 3.2

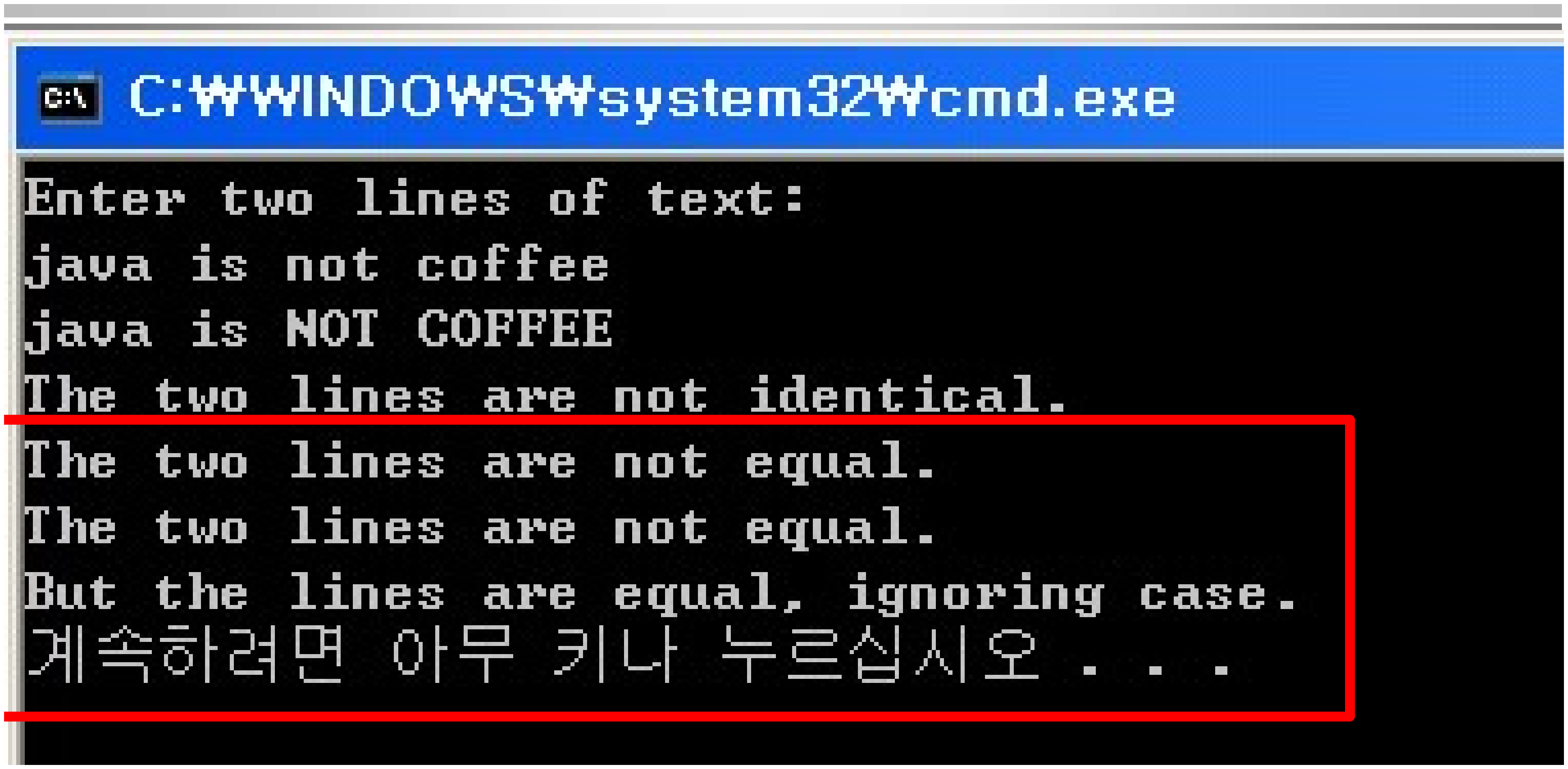
- Listing 3.2 Testing Strings for Equality
  - » StringEqualityDemo.java

```
//Listing 3.2 Testing Strings for Equality

public class StringEqualityDemo
{
   public static void main(String[] args)
   {
      String s1, s2;
      System.out.println("Enter two lines of text:");
      Scanner keyboard = new Scanner(System.in);
      s1 = keyboard.nextLine( ); // java is not coffee
      s2 = keyboard.nextLine( ); // java is NOT COFFEE(???)

      if (s1 == s2)
         System.out.println("The two lines are identical");
      else
         System.out.println("The two lines are not identical.");
```



```
Enter two lines of text:
java is not coffee
java is NOT COFFEE
The two lines are not identical.
The two lines are not equal.
The two lines are not equal.
But the lines are equal, ignoring case.
계속하려면 아무 키나 누르십시오 . . .
```

```
        if (s1.equals(s2))
            System.out.println("The two lines are equal.");
        else
            System.out.println("The two lines are not equal.");

        if (s2.equals(s1))
            System.out.println("The two lines are equal.");
        else
            System.out.println("The two lines are not equal.");

        if (s1.equalsIgnoreCase(s2))
            System.out.println(
                        "But the lines are equal, ignoring case.");
        else
            System.out.println(
                        "Lines are not equal even ignoring case.");
    }
}
```

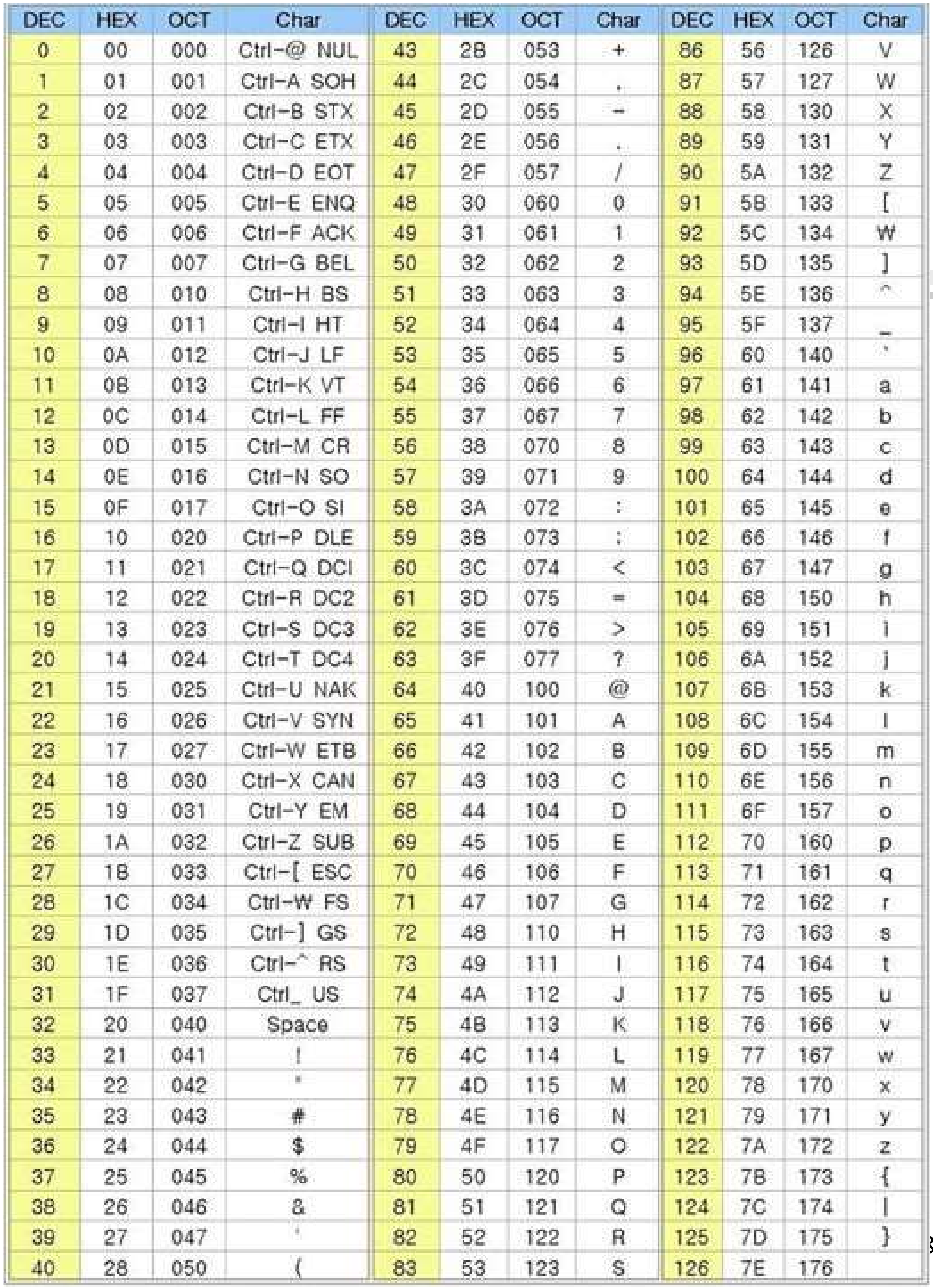


```
Enter two lines of text:
java is not coffee
java is NOT COFFEE
The two lines are not identical.
The two lines are not equal.
The two lines are not equal.
But the lines are equal, ignoring case.
계속하려면 아무 키나 누르십시오 . . .
```

Alphabetical Ordering

- Use [redacted] method of String class
- Uses ASCII lexicographic ordering where all uppercase letters come before all lowercase letters
  - » For example capital `'Z'` comes before small `'a'`
  - » Convert strings to all uppercase (or all lowercase) to avoid problems
- `s1.compareTo(s2)`
  - » returns a [redacted] value if s1 comes before s2
  - » returns zero if the two strings are equal
  - » returns a [redacted] value if s2 comes before s1

```
// Assume s1 and s2 are two string variables
// that have been given string values.
String upperS1 = s1.toUpperCase();
String upperS2 = s2.toUpperCase();
if (upperS1.compareTo(upperS2) < 0)
        System.out.println(s1 + " precedes " + s2);
```



DEC	HEX	OCT	Char	DEC	HEX	OCT	Char	DEC	HEX	OCT	Char
0	00	000	Ctrl-@ NUL	43	2B	053	+	86	56	126	V
1	01	001	Ctrl-A SOH	44	2C	054	,	87	57	127	W
2	02	002	Ctrl-B STX	45	2D	055	-	88	58	130	X
3	03	003	Ctrl-C ETX	46	2E	056	.	89	59	131	Y
4	04	004	Ctrl-D EOT	47	2F	057	/	90	5A	132	Z
5	05	005	Ctrl-E ENQ	48	30	060	0	91	5B	133	[
6	06	006	Ctrl-F ACK	49	31	061	1	92	5C	134	₩
7	07	007	Ctrl-G BEL	50	32	062	2	93	5D	135	]
8	08	010	Ctrl-H BS	51	33	063	3	94	5E	136	^
9	09	011	Ctrl-I HT	52	34	064	4	95	5F	137	_
10	0A	012	Ctrl-J LF	53	35	065	5	96	60	140	`
11	0B	013	Ctrl-K VT	54	36	066	6	97	61	141	a
12	0C	014	Ctrl-L FF	55	37	067	7	98	62	142	b
13	0D	015	Ctrl-M CR	56	38	070	8	99	63	143	c
14	0E	016	Ctrl-N SO	57	39	071	9	100	64	144	d
15	0F	017	Ctrl-O SI	58	3A	072	:	101	65	145	e
16	10	020	Ctrl-P DLE	59	3B	073	;	102	66	146	f
17	11	021	Ctrl-Q DCI	60	3C	074	<	103	67	147	g
18	12	022	Ctrl-R DC2	61	3D	075	=	104	68	150	h
19	13	023	Ctrl-S DC3	62	3E	076	>	105	69	151	i
20	14	024	Ctrl-T DC4	63	3F	077	?	106	6A	152	j
21	15	025	Ctrl-U NAK	64	40	100	@	107	6B	153	k
22	16	026	Ctrl-V SYN	65	41	101	A	108	6C	154	l
23	17	027	Ctrl-W ETB	66	42	102	B	109	6D	155	m
24	18	030	Ctrl-X CAN	67	43	103	C	110	6E	156	n
25	19	031	Ctrl-Y EM	68	44	104	D	111	6F	157	o
26	1A	032	Ctrl-Z SUB	69	45	105	E	112	70	160	p
27	1B	033	Ctrl-[ESC	70	46	106	F	113	71	161	q
28	1C	034	Ctrl-₩ FS	71	47	107	G	114	72	162	r
29	1D	035	Ctrl-] GS	72	48	110	H	115	73	163	s
30	1E	036	Ctrl-^ RS	73	49	111	I	116	74	164	t
31	1F	037	Ctrl_ US	74	4A	112	J	117	75	165	u
32	20	040	Space	75	4B	113	K	118	76	166	v
33	21	041	!	76	4C	114	L	119	77	167	w
34	22	042	"	77	4D	115	M	120	78	170	x
35	23	043	#	78	4E	116	N	121	79	171	y
36	24	044	$	79	4F	117	O	122	7A	172	z
37	25	045	%	80	50	120	P	123	7B	173	{
38	26	046	&	81	51	121	Q	124	7C	174	|
39	27	047	'	82	52	122	R	125	7D	175	}
40	28	050	(	83	53	123	S	126	7E	176	

Lexicographic Order

- Lexicographic order is similar to alphabetical order, but is it based on the order of the characters in the [redacted] (and Unicode) character set.
  - » Ex1)
    All the digits come before all the letters
  - » Ex 2)
    All the [redacted]etters come before all the [redacted] letters

.

Lexicographic Order, cont.

- Strings consisting of alphabetical characters can be compared using method compareTo and method toUpperCase or method toLowerCase.

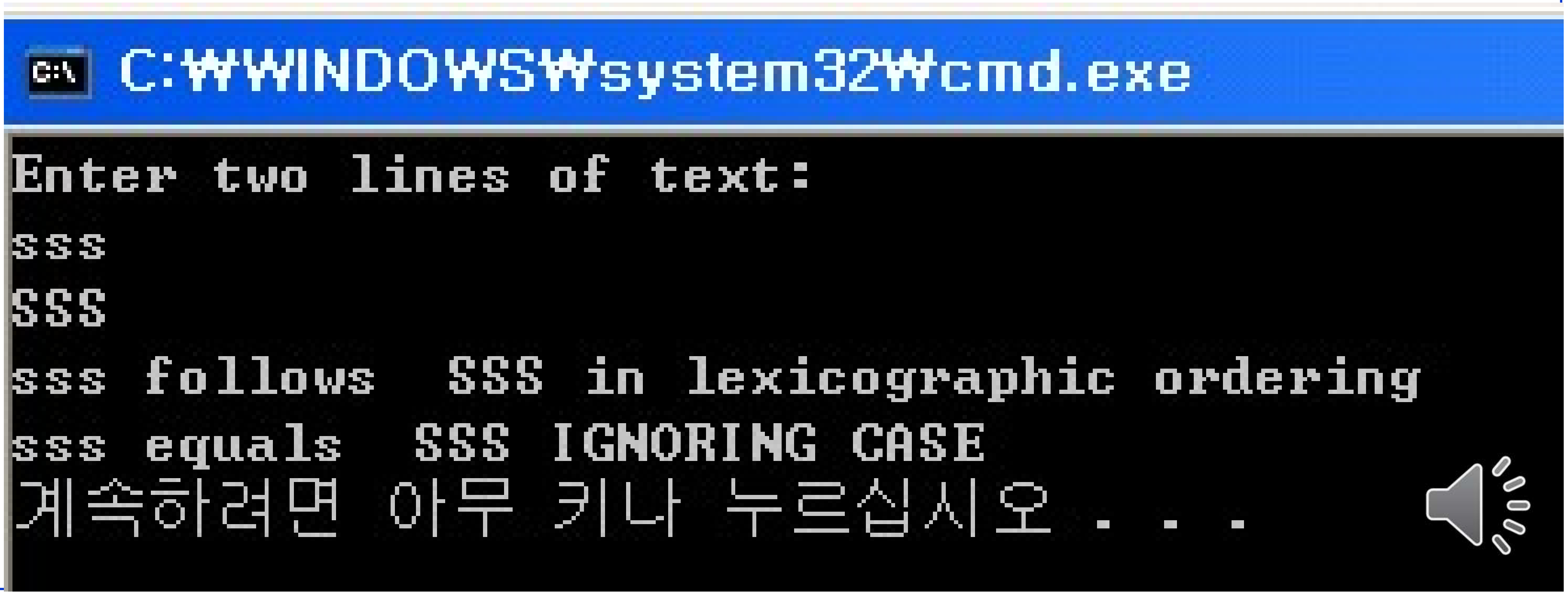
```
String s1 = “Hello”;
String lowerS1 = s1.toLowerCase();
String s2 = “hello”;
if (s1.compareTo(s2)) == 0
    System.out.println(“Equal!”);
```

Method `compareTo`

- syntax

  String_1.compareTo(*String_2*)
- Method compareTo returns
  - a negative number if *String_1* precedes *String_2*
  - zero if the two strings are equal
  - a positive number of *String_2* precedes *String_1.*

```
// Testing Alphabetical Order

public class AlphabeticalOrderDemo
{
    public static void main(String[] args)
    {
        String s1, s2;

        System.out.println("Enter two lines of text:");

        Scanner keyboard = new Scanner (System.in);
        s1 = keyboard.next( );   //sss ??
        s2 = keyboard.next( );   //SSS ??

        if (s1.compareTo(s2)<0)
            System.out.println( s1+" precedes "+s2+" in lexicographic
ordering");
        else if (s1.compareTo(s2)>0)
            System.out.println( s1+" follows  "+s2+" in lexicographic
ordering");
                          else //s1.compareTo(s2) == 0
             System.out.println( s1+" equals  "+s2);
            String upperS1=s1.toUpperCase();
            String upperS2=s2.toUpperCase();
            if (upperS1.compareTo(upperS2)<0)
                        System.out.println(s1+" precedes "+upperS2+" in 
lexicographic ordering");
            else if (upperS1.compareTo(upperS2)>0)
                        System.out.println(s1+" follows  "+upperS2+" in 
lexicographic ordering");
            else //s1.compareTo(s2) == 0
                        System.out.println( s1+" equals  "+upperS2+" IGNORING 
CASE");
          }
}
```



```
Enter two lines of text:
sss
SSS
sss follows  SSS in lexicographic ordering
sss equals  SSS IGNORING CASE
계속하려면 아무 키나 누르십시오 . . .
```

Nested `if` Statements

- One `if` statement can have another `if` statement inside it.
- These are called [redacted] ***`if` statements***.
- Inner statements are indented more than outer statements.

```
if (balance >= 0)
    if (RATE >= 0)
        balance = balance + (RATE * balance)/12;
    else
        System.out.println("Cannot have negative rate");
else
    balance = balance - OVERDRAWN_PENALTY;
```

The inner statement will be skipped entirely if `balance >= 0` is false.

outer statement

inner statement

Matching else and **`if`**

-

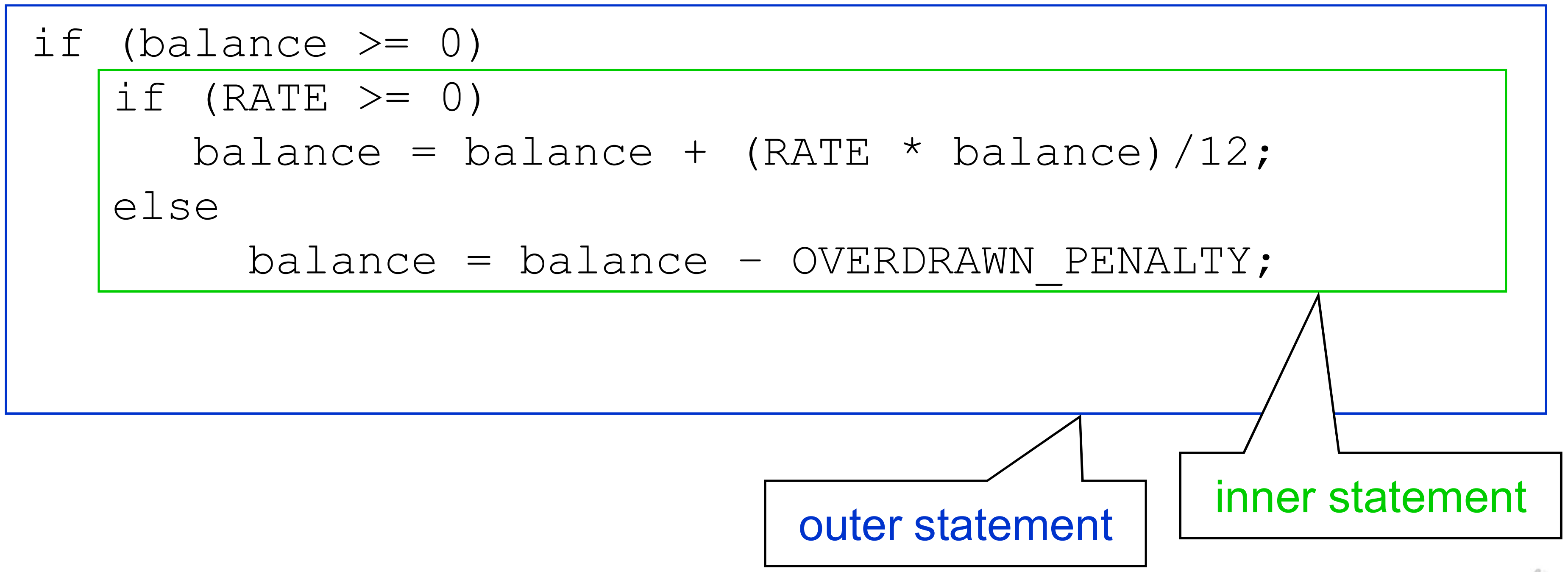
```
if (balance >= 0)
   if (RATE >= 0)
      balance = balance + (RATE * balance)/12;


else
   balance = balance - OVERDRAWN_PENALTY;
```

outer statement

inner statement

Matching else and `if`

•



```
if (balance >= 0)
    if (RATE >= 0)
        balance = balance + (RATE * balance)/12;
    else
          balance = balance - OVERDRAWN_PENALTY;
```

Multibranch selection: **`if-else  if-else if-…-else`**

- One way to handle situations with more than two possibilities
- Syntax:

```
if(Boolean_Expression_1)
   Action_1
else if(Boolean_Expression_2)
   Action_2
     .
     .
     .
else if(Boolean_Expression_n)
   Action_n
else
   Default_Action
```

`if-else if-else if-…-else` Example

```
if(score >= 90)
   grade = 'A');
else if (score >= 80)
   grade = 'B';
else if (score >= 70)
   grade = 'C';
else if (score >= 60)
   grade = 'D';
else
   grade = 'E';
```

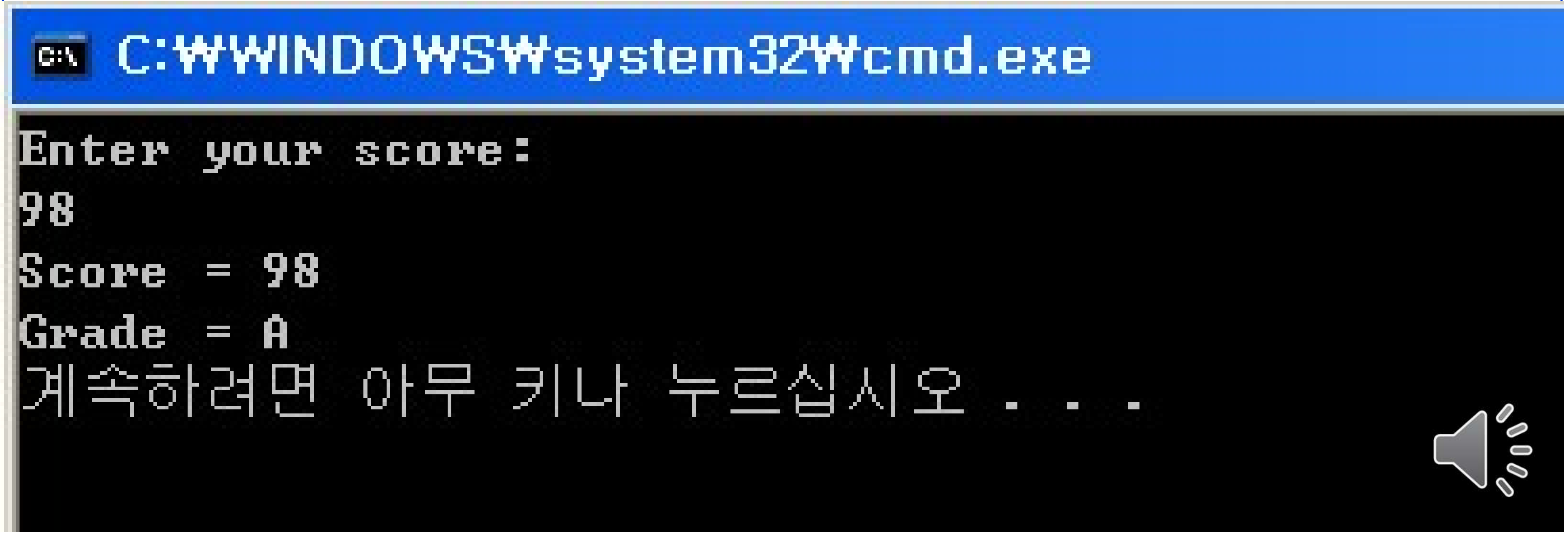
```
// GRADER.JAVA
```

Note indentation.

Even though these are nested if statements, they are all indented the same amount to indicate a multibranch selection.

// Listing 3.3

```
import java.util.Scanner;
public class Grader
{
    public static void main (String [] args)
    {
        int score;
        char grade;
        System.out.println ("Enter your score: ");
        Scanner keyboard = new Scanner (System.in);
        score = keyboard.nextInt ();
        if (score >= 90)
            grade = 'A';
        else if (score >= 80)
            grade = 'B';
        else if (score >= 70)
            grade = 'C';
        else if (score >= 60)
            grade = 'D';
        else
            grade = 'F';
        System.out.println ("Score = " + score);
        System.out.println ("Grade = " + grade);
    }
}
```



```
Enter your score:
98
Score = 98
Grade = A
계속하려면 아무 키나 누르십시오 . . .
```

The [redacted] operator

```
If (n1 > n2) max = n1 ;
Else max = n2;
➔ Max = (n1 > n2) ?  n1:n2;
```

```
If (houseWorked <= 40)
      pay = hourseWOrked*payRate;
Else
      pay = hoursWorked*payRate + 1.5*(hoursWorked – 40) * payRate;
```

```
Pay = (hoursworked <= 40) ?
```

- The conditional operator is useful with print and println statements.

```
System.out.print("You worked " +
    ((hours > 1) ? "hours" ; "hour"));
```

끝