UNSW Business School
Information Systems and Technology Management

INFS1609/2609 Fundamentals of Business Programming

Lecture 4
Selections

yenni.tim@unsw.edu.au

INFS1609/2609 Fundamentals of Business Programming

Topics for this week:

- `boolean` variables and relational operators
- The `if-else` statements
- The `switch` statements

Main references
- *Textbook: Chapter 3*
- *Other online references posted on Ed*

Remember the `ComputeArea` example?

- If you assigned a negative value for `radius` in the `ComputeAreaWithConsoleInput.java`, the program would print an invalid result
- If the radius is negative, you don't want the program to compute the area. How can you deal with this situation?

The `boolean` type and relational operators

- Often in a program you need to compare two values, such as whether `i` is greater than `j`
- Java provides six comparison operators (also known as **relational operators**) that can be used to compare two values
- The result of the comparison is a boolean value: `true` or `false`

```
boolean b = (1 > 2);
```

The `boolean` type and relational operators

Java Operator	Mathematics Symbol	Name	Example (radius is 5)	Result
`<`	<	less than	`radius < 0`	`false`
`<=`	≤	less than or equal to	`radius <= 0`	`false`
`>`	>	greater than	`radius > 0`	`true`
`>=`	≥	greater than or equal to	`radius >= 0`	`true`
`==`	=	equal to	`radius == 0`	`false`
`!=`	≠	not equal to	`radius != 0`	`true`

INFS1609/2609 Fundamentals of Business Programming

The `boolean` type and relational operators

```
public static void main (String[] args){
      System.out.println("10 > 9 is " + (10 > 9));

}
```

Logical Operators

Operator	Name	Description
!	not	logical negation
&&	and	logical conjunction
||	or	logical disjunction
^	exclusive or	logical exclusion

The Exclusive Or (^) Operator

true if and only if the two operands have **different** boolean values

p_1	p_2	p_1 ^ p_2	Example (assume age = 24, weight = 140)
false	false	false	
false	**true**	**true**	(age > 34) ^ (weight >= 140) is true, because (age > 34) is false but (weight >= 140) is true.
true	**false**	**true**	(age > 14) ^ (weight > 140) is true, because (age > 14) is true and (weight > 140) is false.
true	true	false	

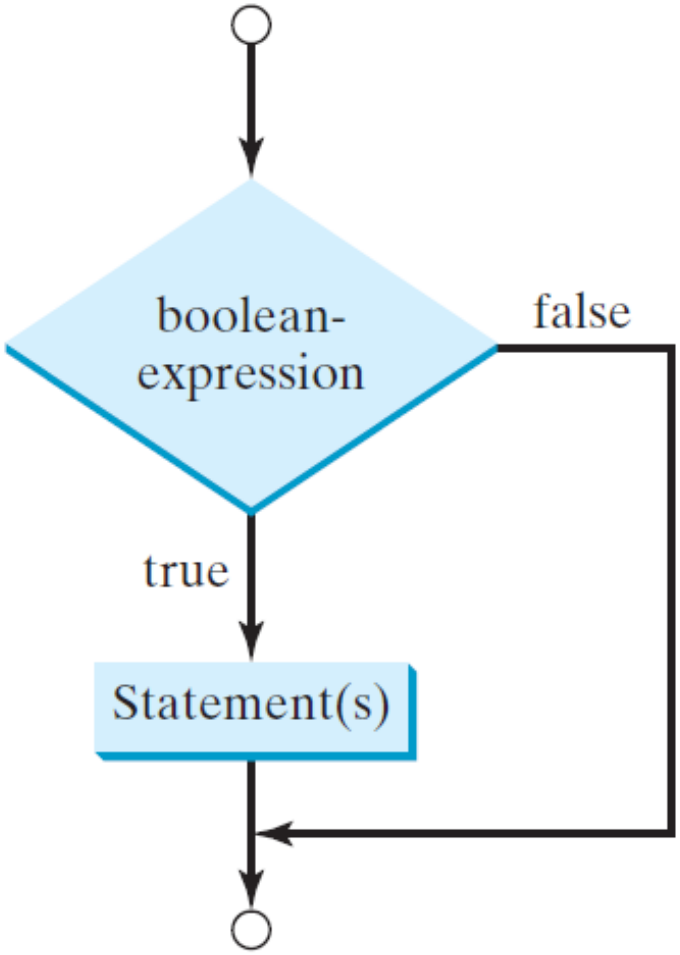
The `if` conditional statement:

- **Selectively** execute part of a program
- Syntax:

```
if(condition) statement;
```

- Condition is a `boolean` expression. If the condition is **true**, then the statement is **executed**

One way `if` statement:



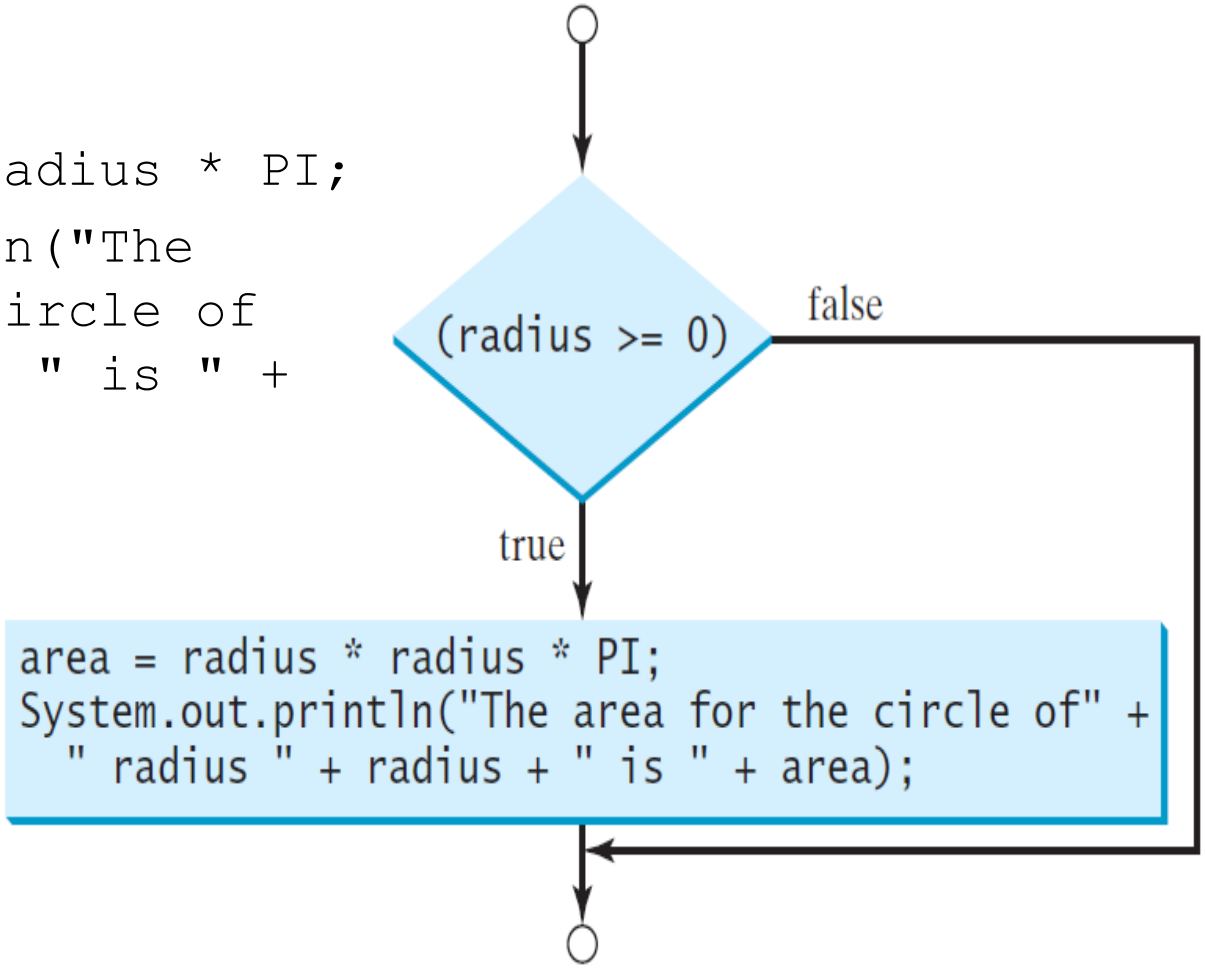

```
if i > 0 {
  System.out.println("i is positive");
}
```

(a) Wrong

```
if (i > 0) {
  System.out.println("i is positive");
}
```

(b) Correct

```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The
area" + " for the circle of
radius " + radius + " is " +
area);
}
```



The `if` conditional statement:

```
if(10<9) System.out.println("print me please?");
```

- Output?

```
if(10<9)
      System.out.println("print me please?");
System.out.println("Print me?");
```

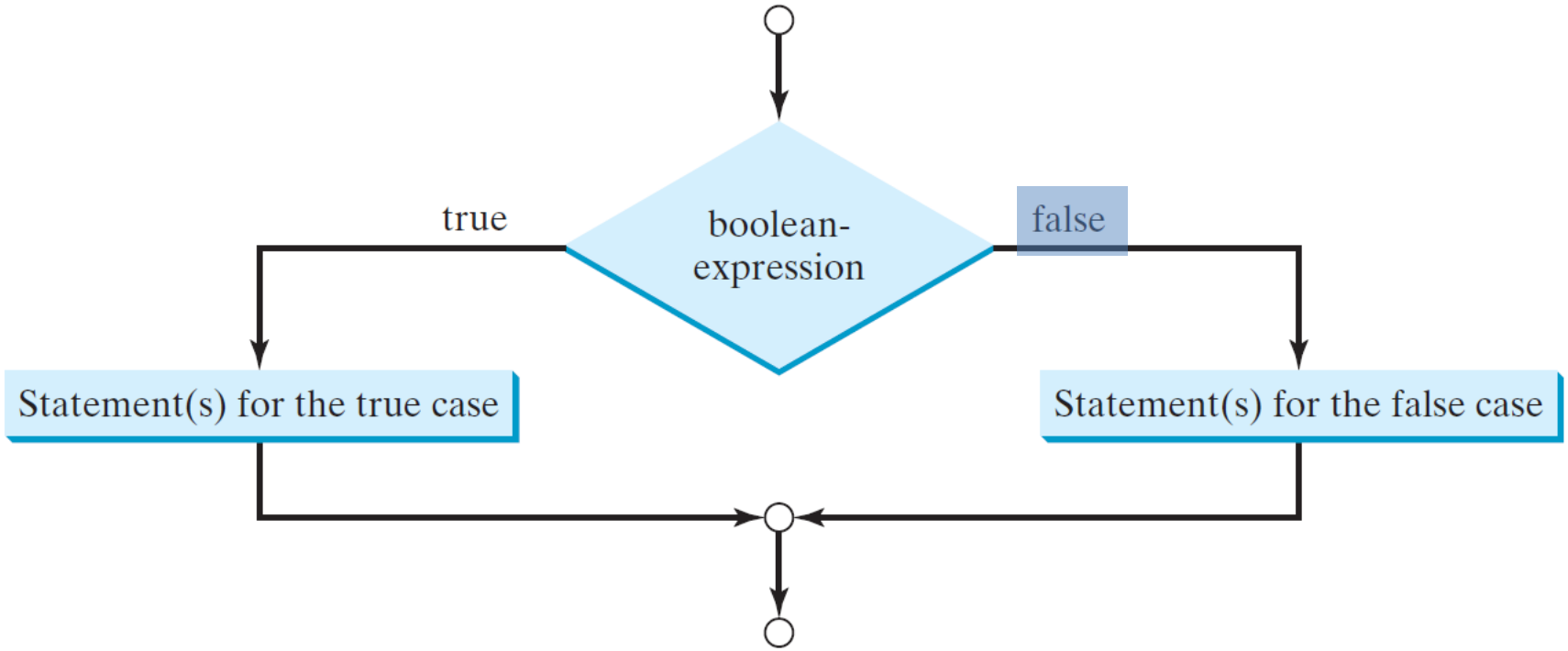
- How about this?

A simple `if` demo:

Write a program that prompts the user to enter an integer. If the number is a multiple of 5, print HiFive. If the number is divisible by 2, print HiEven.

https://liveexample.pearsoncmg.com/html/SimpleIfDemo.html

The two-way `if-else` conditional statement:



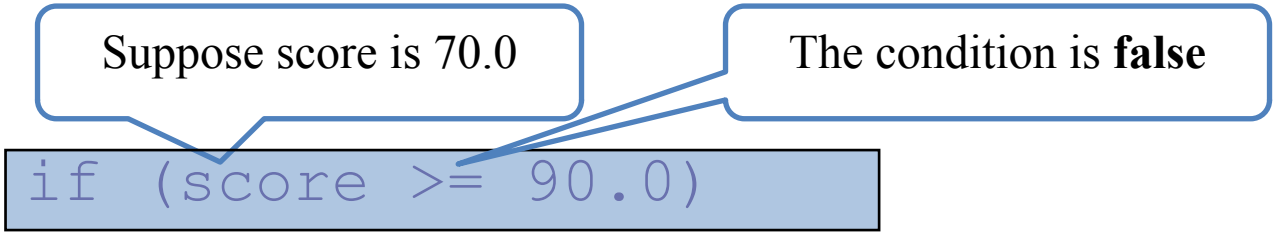
The two-way `if-else` conditional statement:

```
if (boolean-expression) {
  statement(s)-for-the-true-case;
}
else {
  statement(s)-for-the-false-case;
}
```

The two-way `if-else` conditional statement:

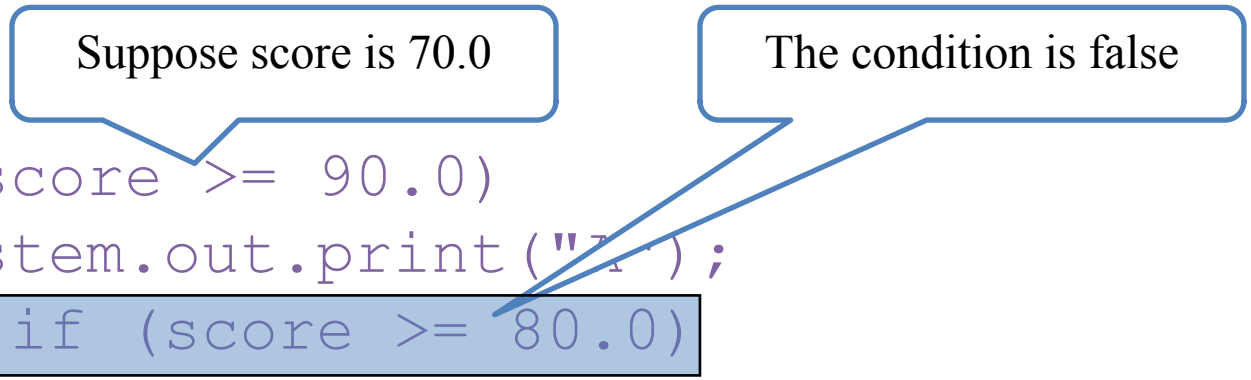
```
if (radius >= 0) {
  area = radius * radius * 3.14159;

  System.out.println("The area for the "
    + "circle of radius " + radius +
    " is " + area);
}
else {
  System.out.println("Negative input");
}
```

Trace two-way `if-else` conditional statement:



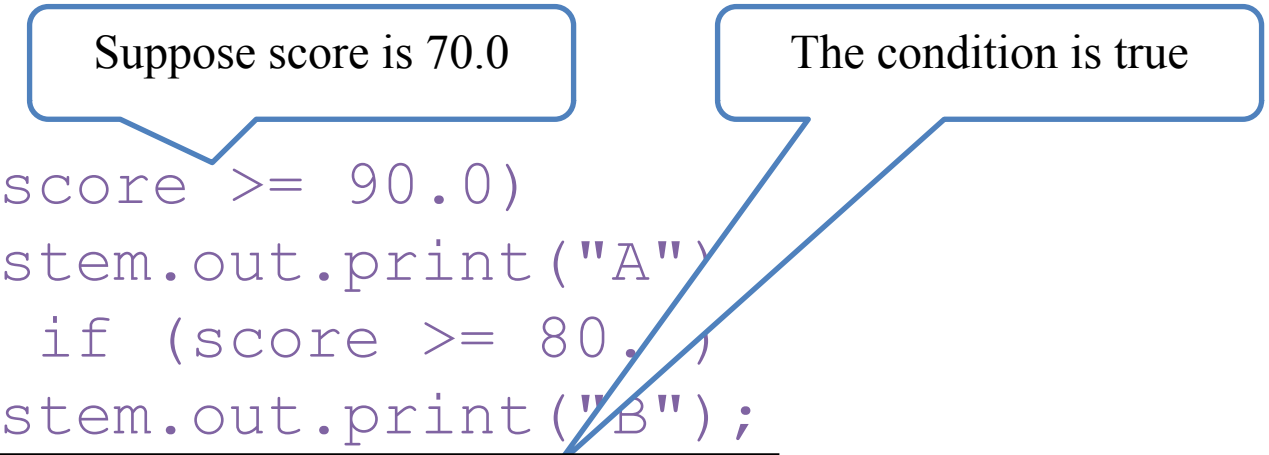

```
if (score >= 90.0)
   System.out.print("A");
else if (score >= 80.0)
   System.out.print("B");
else if (score >= 70.0)
   System.out.print("C");
else if (score >= 60.0)
   System.out.print("D");
else
   System.out.print("F");
```

Trace two-way `if-else` conditional statement:




```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

Trace two-way `if-else` conditional statement:

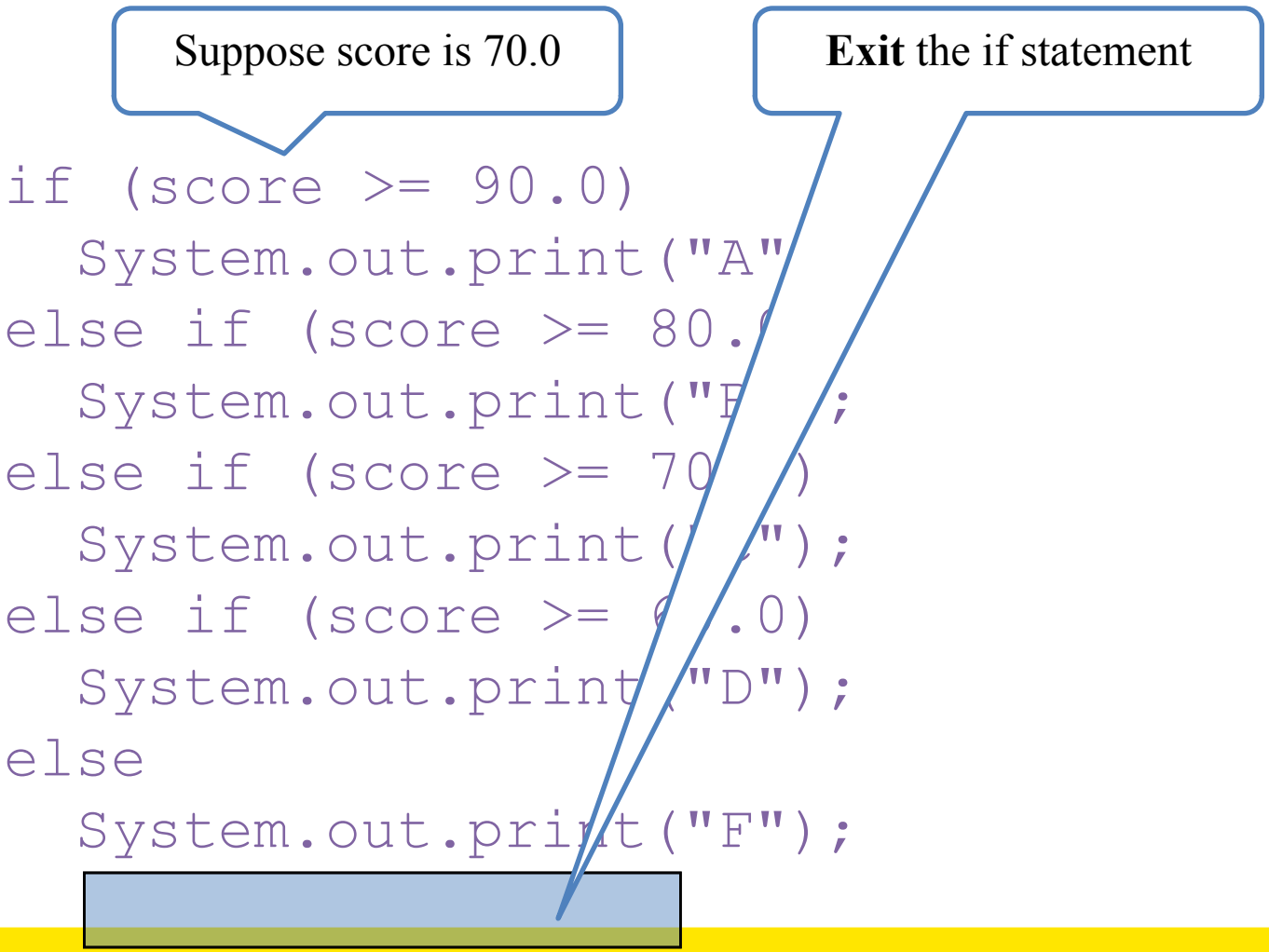



```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

Trace two-way `if-else` conditional statement:


```
if (score >= 90.0)
   System.out.print("A");
else if (score >= 80.0)
   System.out.print("B");
else if (score >= 70.0)
   System.out.print("C");
else if (score >= 60.0)
   System.out.print("D");
else
   System.out.print("F");
```

Trace two-way `if-else` conditional statement:



`if-else` statement with multiple alternatives

```
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

(a)

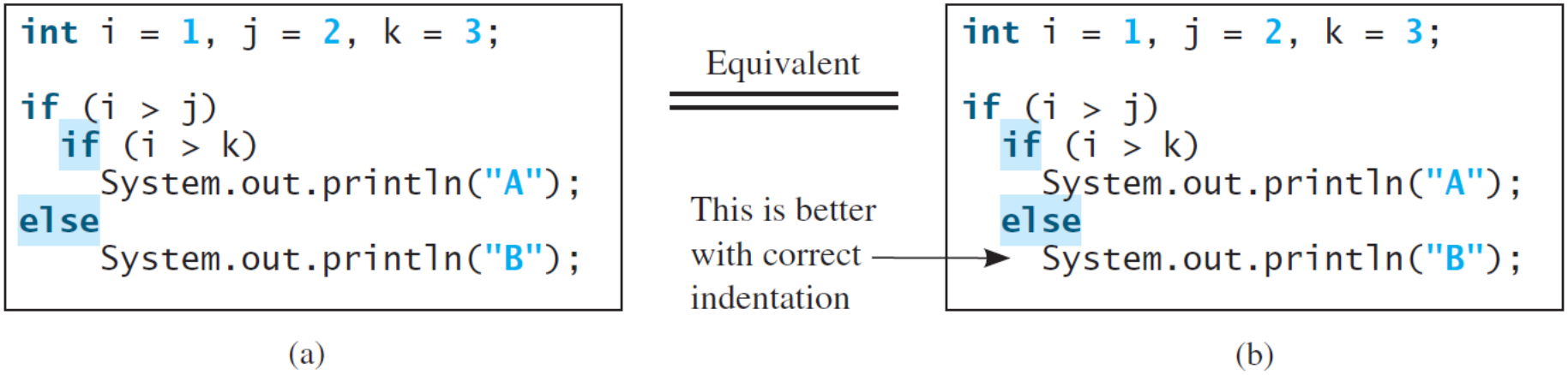
Equivalent

This is better

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

(b)

Note: The else clause matches the most recent if clause in the same block



So **where** you put the braces matters

To force the else clause to match the first if clause, you must add a pair of braces.

```
int i = 1;
int j = 2;
int k = 3;
if (i > j) {
   if (i > k)
      System.out.println("A");
}
else
   System.out.println("B");
```

This statement prints B

Adding a semicolon at the end of an if clause is a common mistake:

```
if (radius >= 0);
{
  area = radius*radius*PI;
  System.out.println(
    "The area for the circle of radius " +
    radius + " is " + area);
}
```

Wrong

This mistake is hard to find, because it is not a compilation error or a runtime error, it is a **logic error**. This error often occurs when you use the next-line block style.

Some tips!

```
if (number % 2 == 0)
  even = true;
else
  even = false;
```

(a)

Equivalent

```
boolean even
  = number % 2 == 0;
```

(b)

```
if (even == true)
  System.out.println(
    "It is even.");
```

(a)

Equivalent

```
if (even)
  System.out.println(
    "It is even.");
```

(b)

The **Conditional Operator** (a shortcut to if-else)

x ? y : z

- It takes three arguments that together form a conditional expression

```
grade > 70 ? "Passed" : "Failed"
```

- The first argument is a `Boolean`
- The second argument is the value of the operation if the condition is `true`
- The third argument is the value of the operation if the condition is `false`

The **Conditional Operator** (a shortcut to if-else)

x ? y : z

- It takes three arguments that together form a conditional expression

```
grade > 70 ? "Passed" : "Failed"
```

- This is equivalent to:

```
if (grade > 70)
      System.out.println("Passed");
else
      System.out.println("Failed");
```

Rewrite the following code using if-else statements

```
tax = (income > 1000) ? income * 0.2 : income
        * 0.17;
```

Example: Compute and Interpret Body Mass Index

Write a program that computes and interprets BMI. BMI can be calculated by taking your weight in kilograms and dividing by the square of your height in meters. The interpretation of BMI for people 16 years or older is as follows:

BMI	Interpretation
BMI < 18.5	Underweight
18.5 <= BMI < 25.0	Normal
25.0 <= BMI < 30.0	Overweight
30.0 <= BMI	Obese

https://liveexample.pearsoncmg.com/html/ComputeBMI.html

Example 2: Testing the boolean operators

https://liveexample.pearsoncmg.com/html/TestBooleanOperators.html

Questions before `switch` statement?

The `switch` statement

When to use the `switch` **statement?**

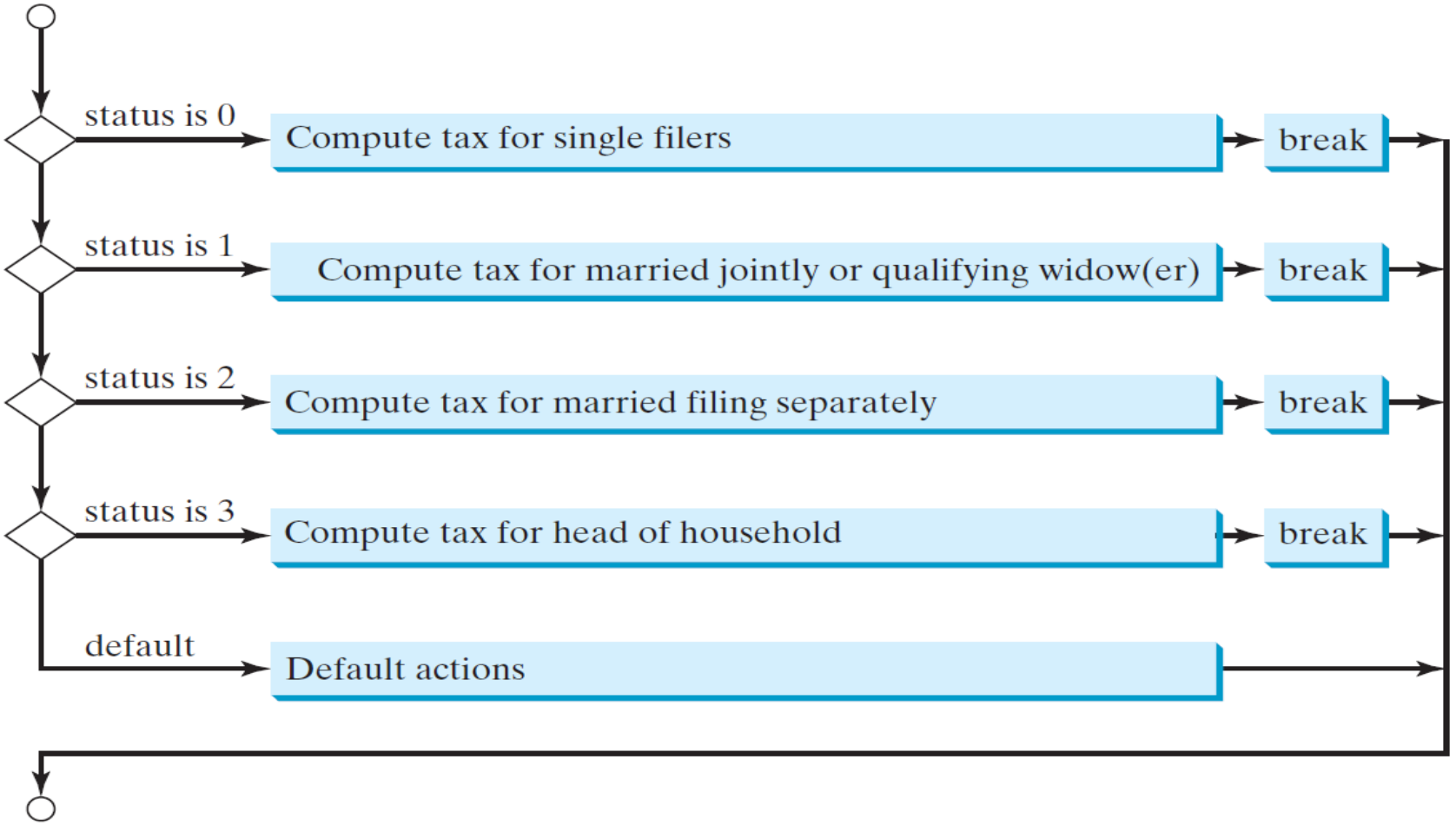
- A sequence of **comparisons** needed to be made against several **constant** alternatives

A switch statement evaluates an `expression` to determine a **value** and then matches that value with one of several possible `cases`

The `switch` statement

```
switch (switch-expression) {
  case value1:  statement(s)1;
          break;
  case value2: statement(s)2;
          break;
  …
  case valueN: statement(s)N;
          break;
  default: statement(s)-for-default;
}
```

The `switch` statement flow chart



The `switch` statement

The switch-expression must yield a value of **char**, **byte**, **short**, or **int** type and must always be enclosed in parentheses

```
switch (switch-expression) {
   case value1:  statement(s)1;
                 break;
   case value2: statement(s)2;
                 break;
   …
   case valueN: statement(s)N;
                 break;
   default: statement(s)-for-default;
}
```

The `switch` statement

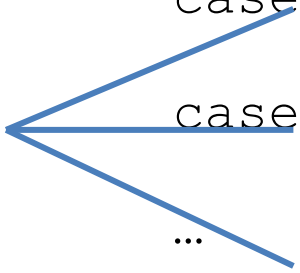
How about the **String**?

```
switch (switch-expression) {
  case value1:  statement(s)1;
                break;
  case value2: statement(s)2;
                break;
  …
  case valueN: statement(s)N;
                break;
  default: statement(s)-for-default;
}
```

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html

The `switch` statement

The value1, ..., and valueN must have the **same data type** as the value of the switch-expression.



```
switch (switch-expression) {
  case value1:  statement(s)1;
                break;
  case value2: statement(s)2;
                break;
  …
  case valueN: statement(s)N;
                break;
  default: statement(s)-for-default;
}
```

The `switch` statement

The resulting statements in the case statement are executed when the **value** in the case statement **matches** the value of the **switch-expression**

```
switch (switch-expression) {
  case value1:  statement(s)1;
                break;
  case value2:  statement(s)2;
                break;
  …
  case valueN:  statement(s)N;
                break;
  default: statement(s)-for-default;
}
```

The `switch` statement

Note that value1, ..., and valueN are constant expressions, meaning that they **cannot contain variables** in the expression, such as 1 + x

```
switch (switch-expression) {
  case value1:  statement(s)1;
                break;
  case value2: statement(s)2;
                break;
  …
  case valueN: statement(s)N;
                break;
  default: statement(s)-for-default;
}
```

The `switch` statement

The keyword break is optional, but it should be used at the end of each case in order to **terminate** the remainder of the switch statement. If the break statement is not present, the next case statement will be executed

```
switch (switch-expression) {
  case value1:  statement(s)1;
                break;
  case value2: statement(s)2;
                break;
  …
  case valueN: statement(s)N;
                break;
  default: statement(s)-for-default;
}
```

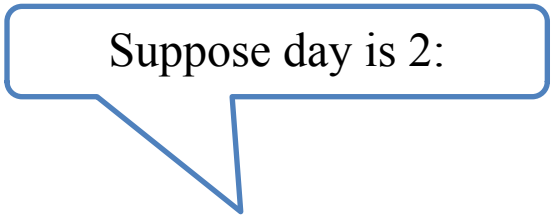
When the value in a **case** statement matches the value of the **switch-expression**, the statements *starting from this case* are executed until either a **break** statement or the end of the **switch** statement is reached.

The `switch` statement

The **default** case, which is optional, can be used to perform actions when **none** of the specified cases matches the switch-expression

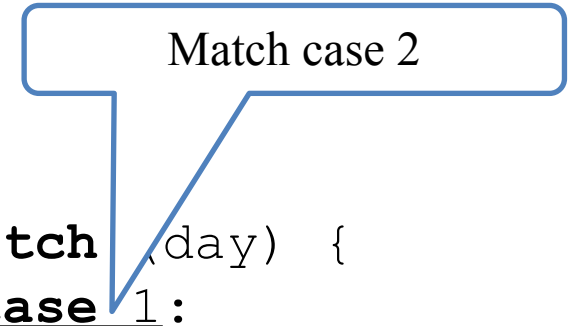
```
switch (switch-expression) {
  case value1:  statement(s)1;
                break;
  case value2: statement(s)2;
                break;
  …
  case valueN: statement(s)N;
                break;
  default: statement(s)-for-default;
}
```

Tracing the `switch` statement



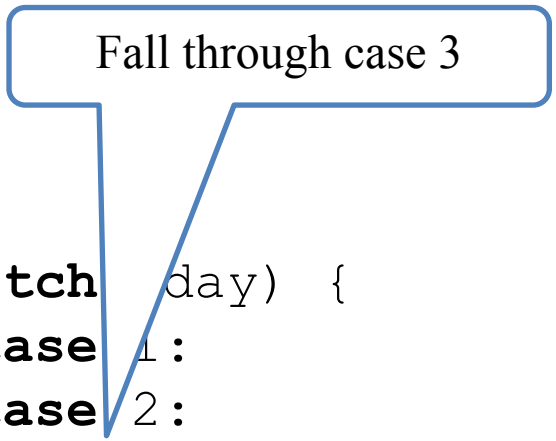

```
switch (day) {
   case 1:
   case 2:
   case 3:
   case 4:
   case 5: System.out.println("Weekday"); break;
   case 0:
   case 6: System.out.println("Weekend");
}
```

Tracing the `switch` statement



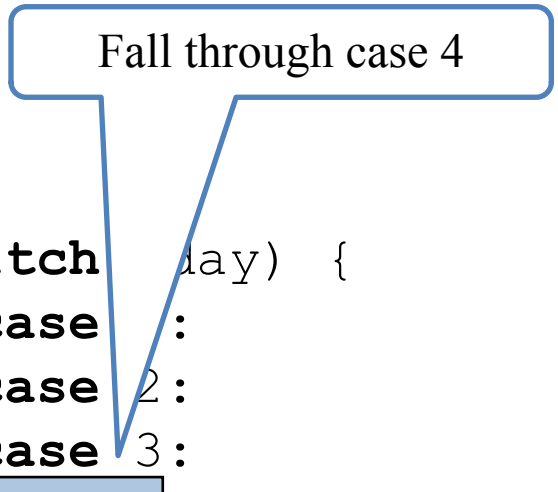

```
switch (day) {
   case 1:
   case 2:
   case 3:
   case 4:
   case 5: System.out.println("Weekday"); break;
   case 0:
   case 6: System.out.println("Weekend");
}
```

Tracing the `switch` statement




```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

Tracing the `switch` statement

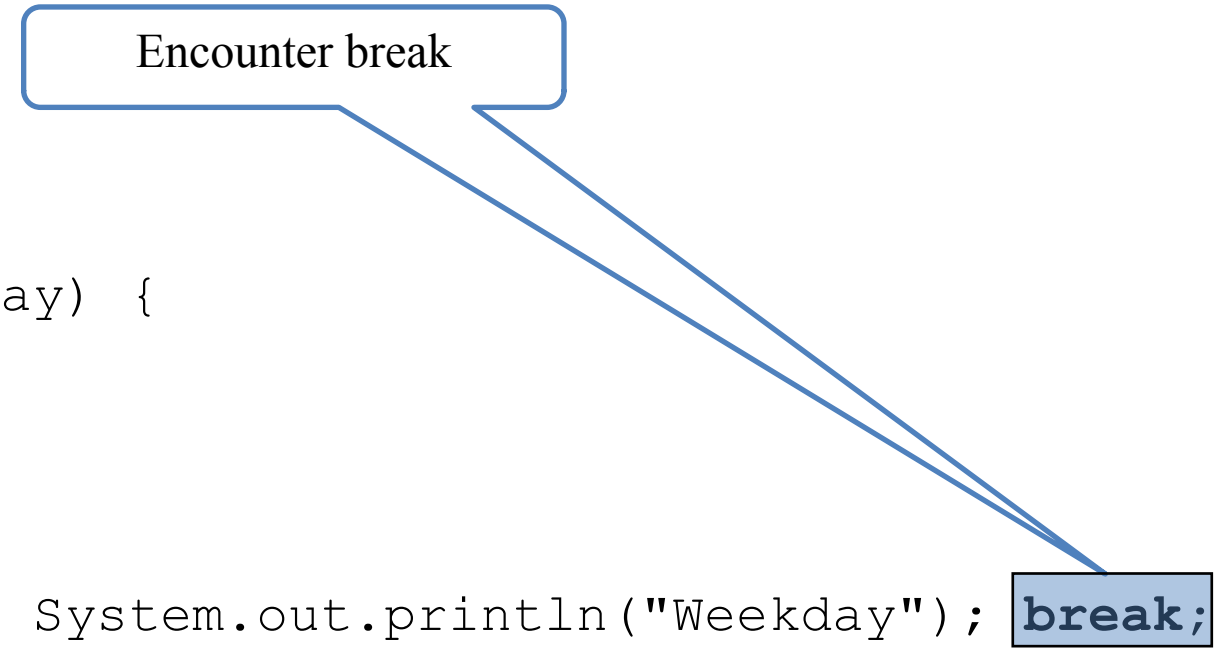



```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

Tracing the `switch` statement

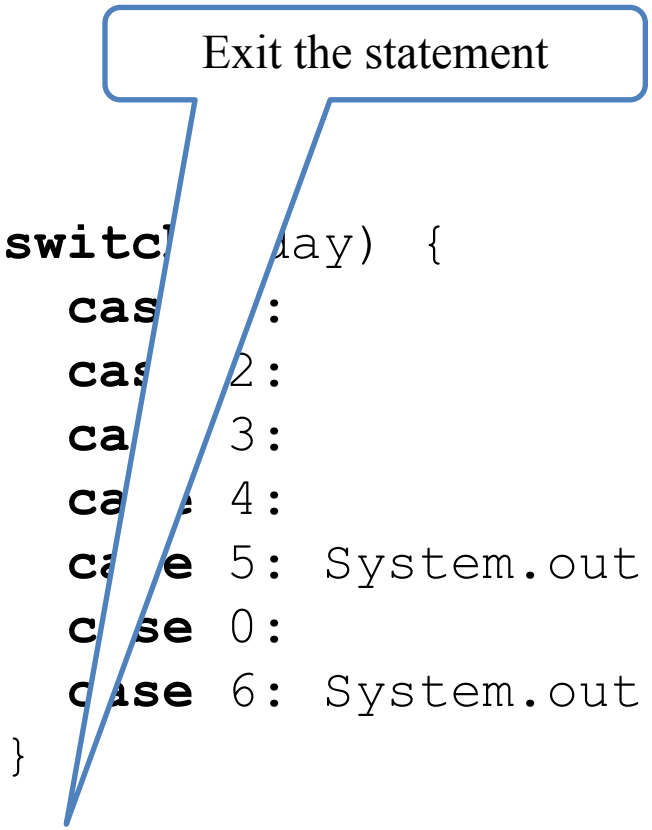

```
switch    ay) {
  case
  case  :
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

Tracing the `switch` statement




```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

Tracing the `switch` statement




```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

Do some coding!

Creating the same example using `String`

Example! The Chinese Zodiac

Write a program that prompts the user to enter a year and displays the animal for the year

year % 12 =
- 0: monkey
- 1: rooster
- 2: dog
- 3: pig
- 4: rat
- 5: ox
- 6: tiger
- 7: rabbit
- 8: dragon
- 9: snake
- 10: horse
- 11: sheep

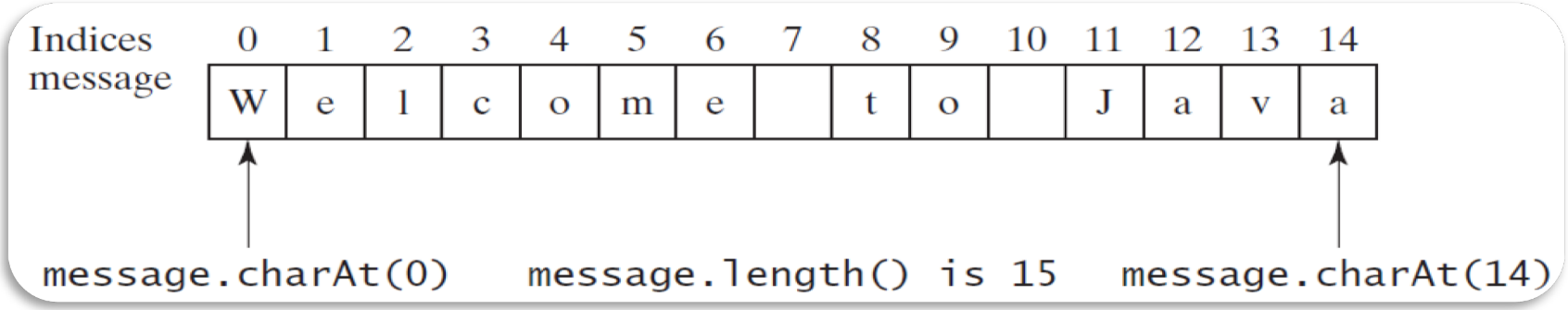
https://liveexample.pearsoncmg.com/html/ChineseZodiac.html

INFS1609/2609 Fundamentals of Business Programming

To cover some examples I missed last week:

Getting characters from a String

Working with the .charAt() method




```
String message = "Welcome to Java";
System.out.println("The first character
in message is " + message.charAt(0));
```

Converting Strings

`"Welcome".toLowerCase()` returns a new string `welcome`

`"Welcome".toUpperCase()` returns a new string `WELCOME`

`"  Welcome  ".trim()` returns a new string `Welcome`

String Concatenation

```
String s3 = s1.concat(s2); or String s3 = s1 + s2;

// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B'; // s1 becomes
SupplementB
```

Reading Strings from the console

```
Scanner input = new Scanner(System.in);
System.out.print("Enter three words separated
by spaces: ");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```

Reading a Character from the console

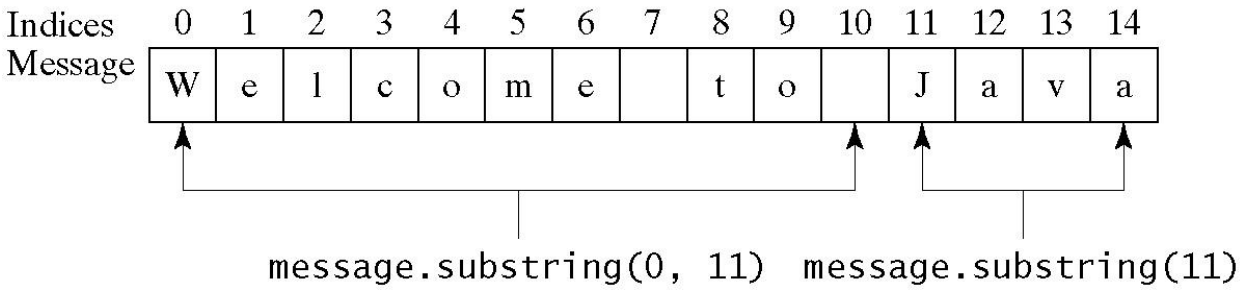
```
Scanner input = new Scanner(System.in);

System.out.print("Enter a character: ");

String s = input.nextLine();
char ch = s.charAt(0);

System.out.println("The character entered is " + ch);
```

Comparing Strings

Method	Description
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than `0`, equal to `0`, or less than `0` to indicate whether this string is greater than, equal to, or less than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.

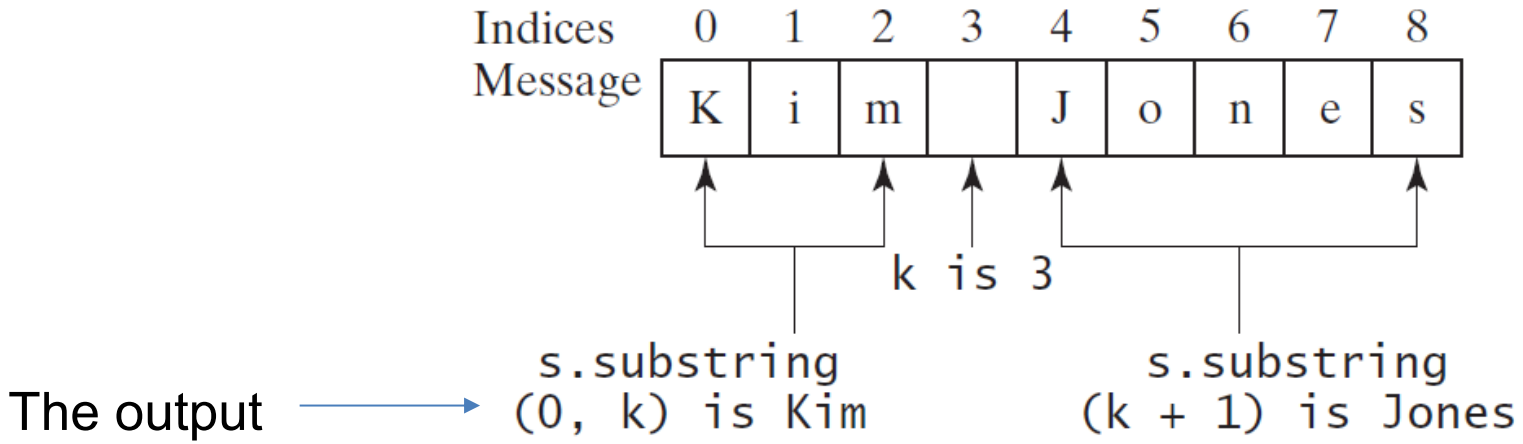
Obtaining substrings

Method	Description
`substring(beginIndex)`	Returns this string's substring that begins with the character at the specified `beginIndex` and extends to the end of the string, as shown in Figure 4.2.
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`, as shown in Figure 9.6. Note that the character at `endIndex` is not part of the substring.



Obtaining substrings

```
int k = s.indexOf(' ');
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```



Conversion between Strings and numbers

```
String intString = "123";
int intValue = Integer.parseInt(intString);
```


parseInt is a method in the **Integer** class

Try to perform `parseInt` with a **non-numerical** String! What is the output?

Extra! If we have time…

The `Math` class

Java provides many useful methods in the **Math** class for performing common mathematical functions

Class constants:

- `PI`
- `E`

Extra! If we have time…

The `random` method

- Generates a random double value greater than or equal to `0.0` and less than `1.0` (`0 <= Math.random() < 1.0`)

```
a + Math.random() * b
```
→ Returns a random number between a and a + b, excluding a + b.

Example

```
(int)(Math.random() * 10)
```
→ Returns a random integer between 0 and 9.

```
50 + (int)(Math.random() * 50)
```
→ Returns a random integer between 50 and 99.

INFS1609/2609 Fundamentals of Business Programming

Extra: combining `Math.random()` with if-else statement

Write a program that lets the user guess whether the flip of a coin results in heads or tails. The program randomly generates an integer 0 or 1, which represents head or tail. The program prompts the user to enter a guess, and reports whether the guess is correct or incorrect

(Answer on Ed)

Reflection

What have you learned today?

Thank you!