BCS2143: OBJECT ORIENTED PROGRAMMING

Chapter 3

- CLASS STRUCTURES AND GUI

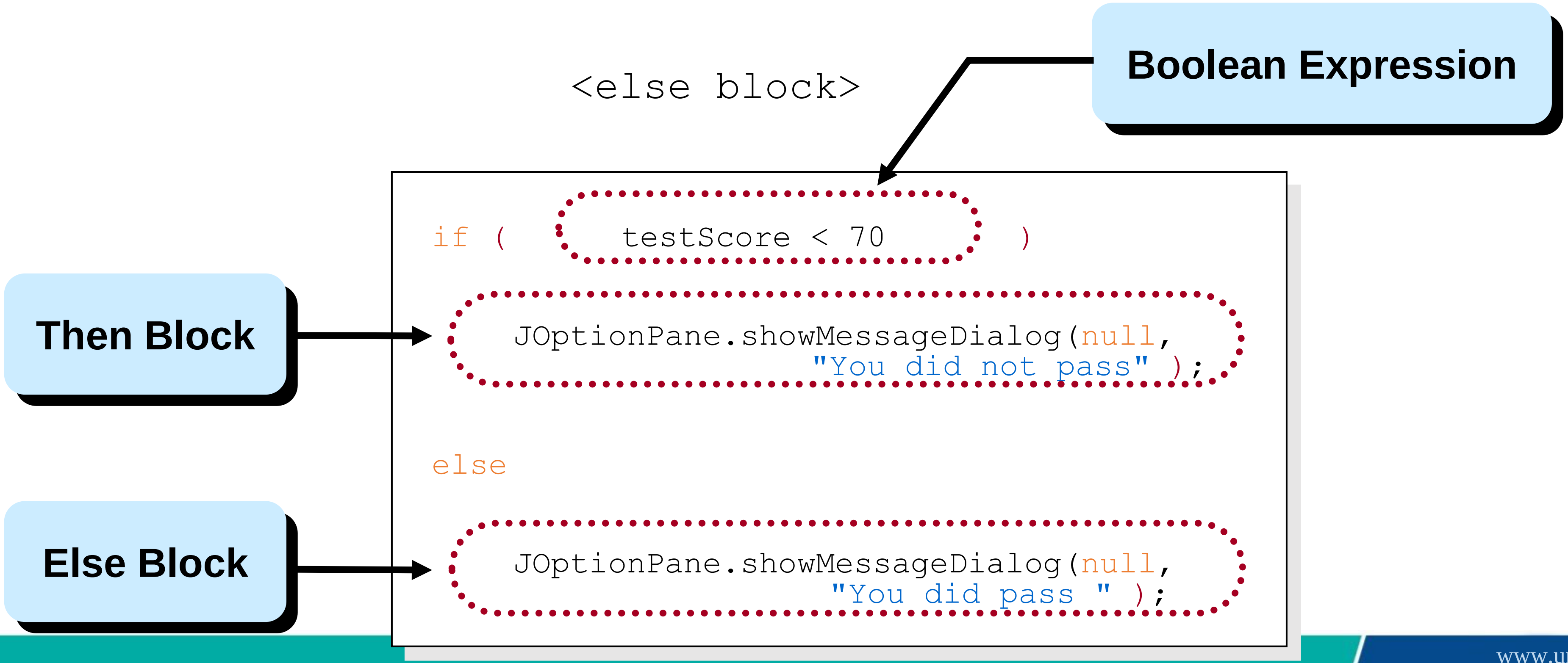
Control Structures

- Java has a sequence structure “built-in”
- Java provides three selection structures
  - `if`
  - `if…else`
  - `switch`
- Java provides three repetition structures
  - `while`
  - `do…while`
  - `for`
- Each of these words is a Java keyword

Selection

Syntax for the if Statement

```
if ( <boolean expression> )

        <then block>
else

        <else block>
```



```
if (   testScore < 70   )

   JOptionPane.showMessageDialog(null,
                      "You did not pass" );

else

   JOptionPane.showMessageDialog(null,
                       "You did pass " );
```

Relational Operators

```
<        //less than
<=       //less than or equal to
==       //equal to
!=       //not equal to
>        //greater than
>=       //greater than or equal to
```

```
testScore < 80
testScore * 2 >= 350
30 < w / (h * h)
x + y != 2 * (a + b)
2 * Math.PI * radius <= 359.99
```

Compound Statements

- Use braces if the <then> or <else> block has multiple statements.

```
if (testScore < 70)
{

     JOptionPane.showMessageDialog(null,
                          "You did not pass“ );

     JOptionPane.showMessageDialog(null,
                     “Try harder next time“ );

}
else
{

     JOptionPane.showMessageDialog(null,
                        “You did pass“ );

     JOptionPane.showMessageDialog(null,
                     “Keep up the good work“ );

}
```

Then Block

Else Block

Style Guide

```
if ( <boolean expression> ) {
     …
}
else {
     …
}
```

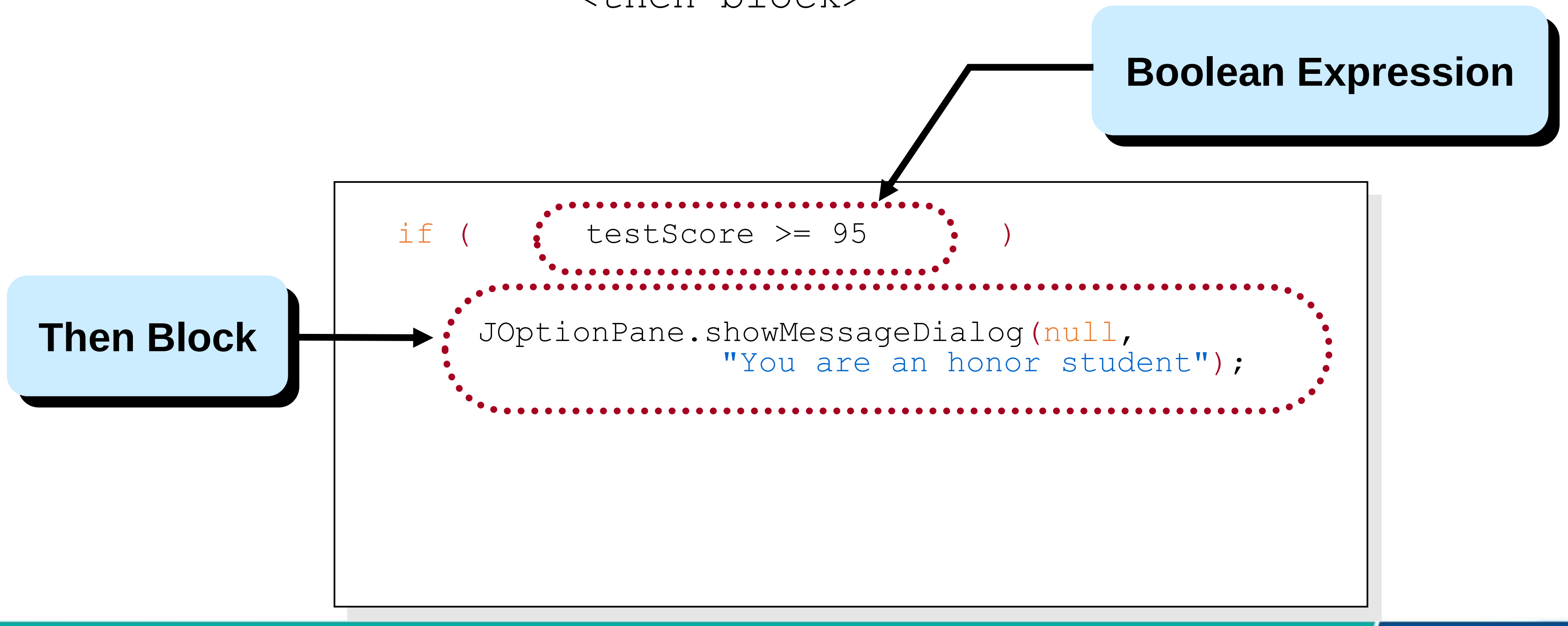
Style 1

```
if ( <boolean expression> )
{
     …
}
else
{
     …
}
```

Style 2

The if-then Statement

```
if ( <boolean expression> )

      <then block>
```



```
if (    testScore >= 95    )
    JOptionPane.showMessageDialog(null,
                     "You are an honor student");
```

if – else if Control

Test Score	Grade
$90 \le$ score	A
$80 \le$ score < 90	B
$70 \le$ score < 80	C
$60 \le$ score < 70	D
score < 60	F

```
if (score >= 90)
   System.out.print("Your grade is A");

else if (score >= 80)
    System.out.print("Your grade is B");

else if (score >= 70)
    System.out.print("Your grade is C");

else if (score >= 60)
    System.out.print("Your grade is D");

else
    System.out.print("Your grade is F");
```

The Nested-if Statement

- The then and else block of an if statement can contain any valid statements, including other if statements. An if statement containing another if statement is called a nested-if statement.

```
if (testScore >= 70) {
     if (studentAge < 10) {
          System.out.println("You did a great job");
     } else {
          System.out.println("You did pass"); //test score >= 70
     }                                      //and age >= 10
} else { //test score < 70
      System.out.println("You did not pass");
}
```

Boolean Operators

- A *boolean operator* takes boolean values as its operands and returns a boolean value.
- The three boolean operators are
  - and: &&
  - or: ||
  - not !

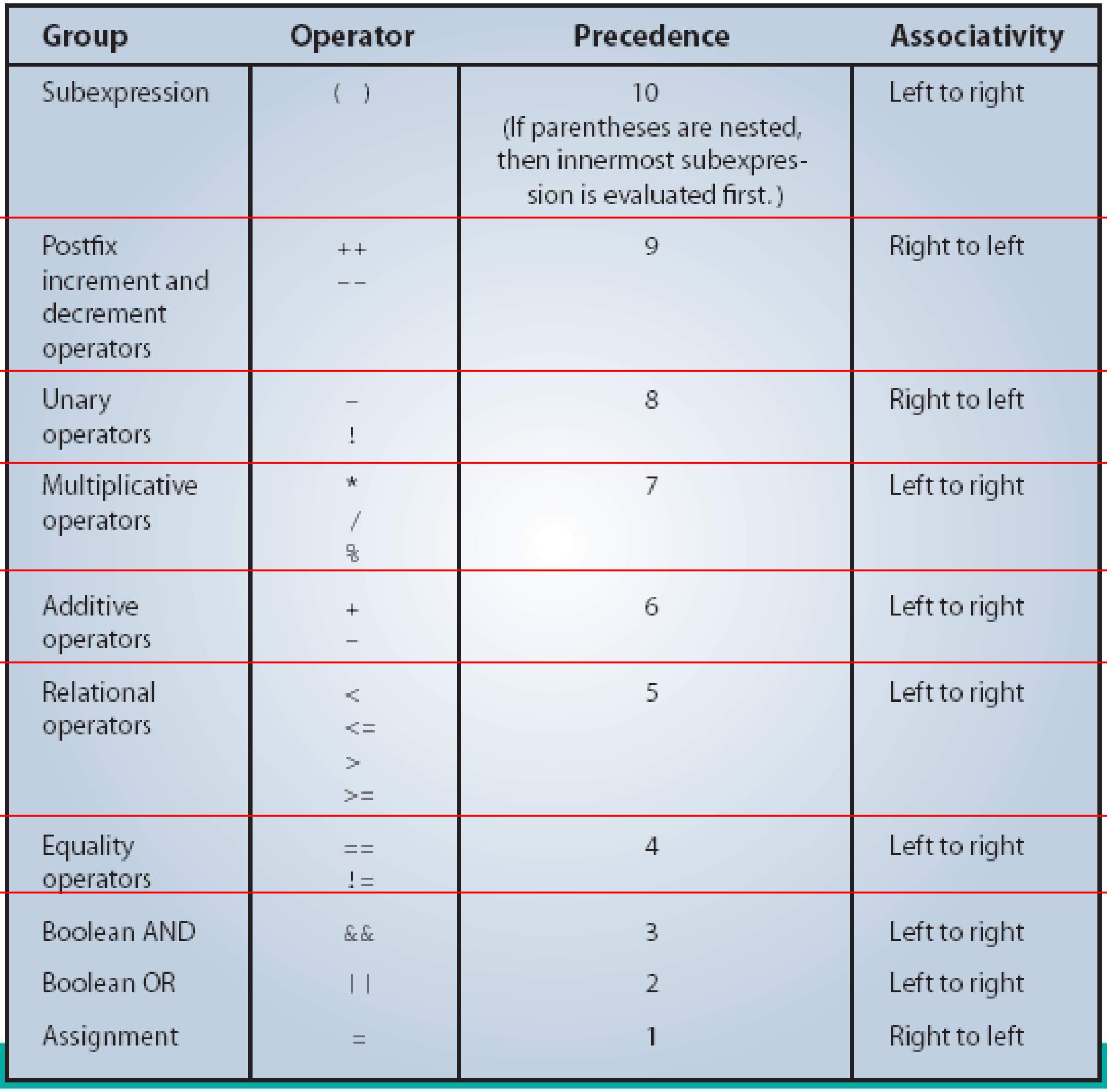
```
if (temperature >= 65 && distanceToDestination < 2) {
     System.out.println("Let's walk");
} else {
     System.out.println("Let's drive");
}
```

Semantics of Boolean Operators

- Boolean operators and their meanings:

P	Q	P && Q	P || Q	!P
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Operator Precedence Rules



Group	Operator	Precedence	Associativity
Subexpression	( )	10 (If parentheses are nested, then innermost subexpression is evaluated first.)	Left to right
Postfix increment and decrement operators	++ --	9	Right to left
Unary operators	– !	8	Right to left
Multiplicative operators	* / %	7	Left to right
Additive operators	+ –	6	Left to right
Relational operators	< <= > >=	5	Left to right
Equality operators	== !=	4	Left to right
Boolean AND	&&	3	Left to right
Boolean OR	||	2	Left to right
Assignment	=	1	Right to left

Comparing Objects

- With primitive data types, we have only one way to compare them, but with objects (reference data type), we have two ways to compare them.
  1. We can test whether two variables point to the same object (use `==`), or
  2. We can test whether two distinct objects have the same contents (use `.equals()` method）.

Using == With Objects (Sample 1)

```
String str1 = new String("Java");
String str2 = new String("Java");

if (str1 == str2) {
    System.out.println("They are equal");
} else {
    System.out.println("They are not equal");
}
```

```
They are not equal
```

Not equal because str1 and str2 point to different String objects.

Using == With Objects (Sample 2)

```
String str1 = new String("Java");
String str2 = str1;

if (str1 == str2) {
    System.out.println("They are equal");
} else {
    System.out.println("They are not equal");
}
```

```
They are equal
```

It's equal here because str1 and str2 point to the same object.

Using equals with String

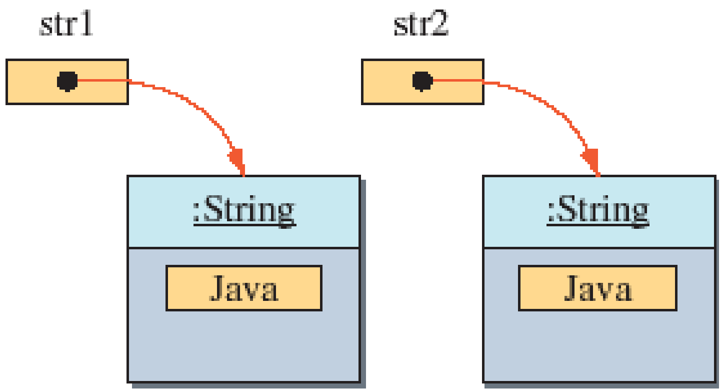
```
String str1 = new String("Java");
String str2 = new String("Java");

if (str1.equals(str2)) {
    System.out.println("They are equal");
} else {
    System.out.println("They are not equal");
}
```

```
They are equal
```

It's equal here because str1 and str2 have the same sequence of characters.

The Semantics of ==

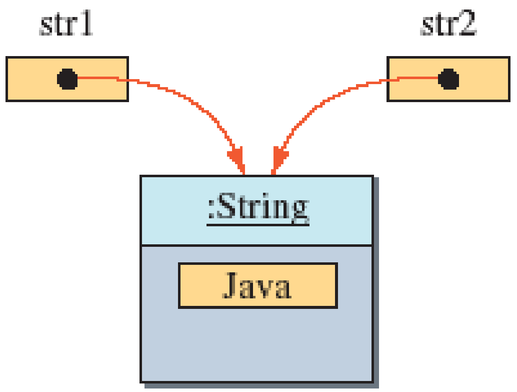
Case A: Two variables refer to two different objects.



```
String str1, str2;

str1 = new String("Java");
str2 = new String("Java");
```

str1 == str2 $\longrightarrow$ false

Case B: Two variables refer to the same object.



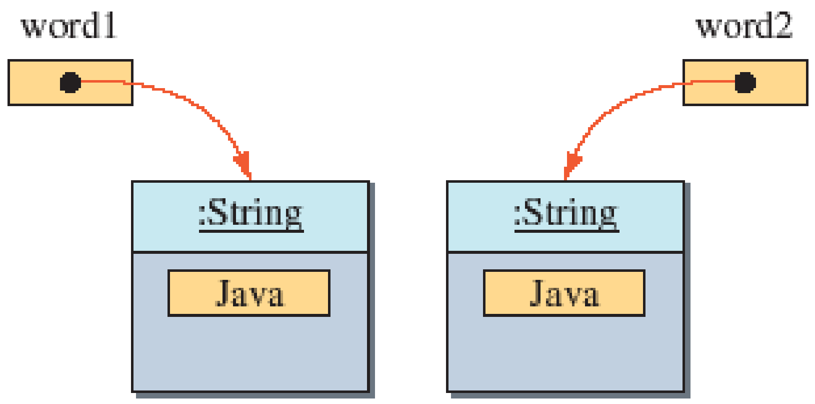
```
String str1, str2;

str1 = new String("Java");
str2 = str1;
```

str1 == str2 $\longrightarrow$ true

In Creating String Objects

```
String word1, word2;

word1 = new String("Java");

word2 = new String("Java");
```

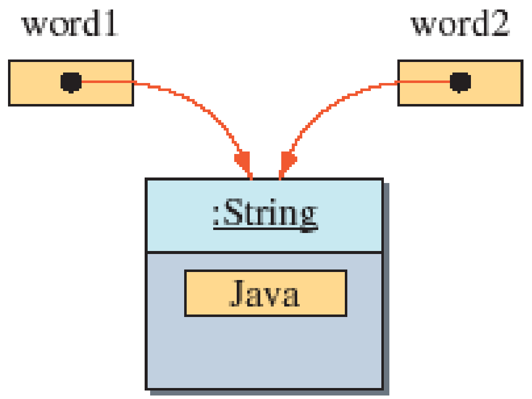
Whenever the new operator is used, there will be a new object.



word1 == word2 → false

```
String word1, word2;

word1 = "Java";

word2 = "Java";
```

Literal string constant such as "Java" will always refer to one object.



word1 == word2 → true

The switch Statement

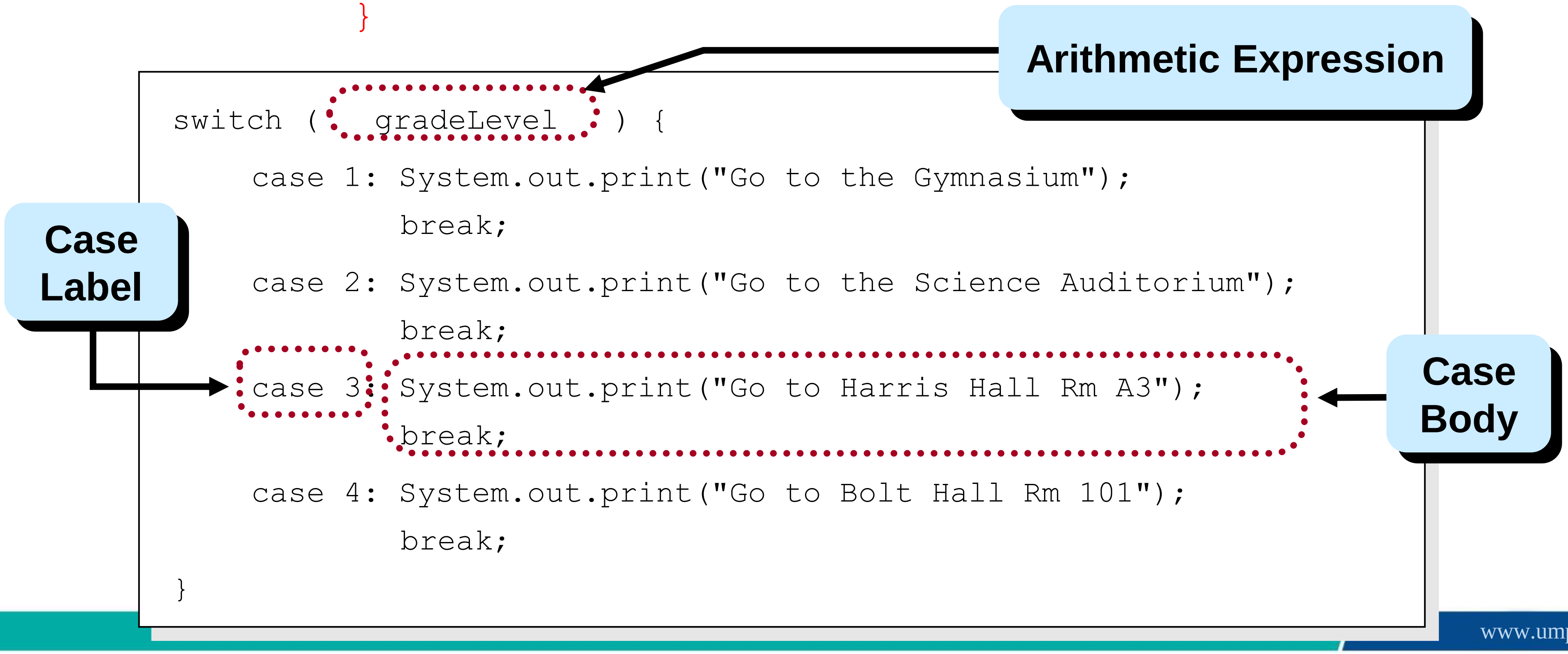
```
int gradeLevel;
gradeLevel = JOptionPane.showInputDialog("Grade (Frosh-1,Soph-2,...):" );

switch (gradeLevel) {

    case 1: System.out.print("Go to the Gymnasium");
            break;

    case 2: System.out.print("Go to the Science Auditorium");
            break;

    case 3: System.out.print("Go to Harris Hall Rm A3");
            break;

    case 4: System.out.print("Go to Bolt Hall Rm 101");
            break;
}
```

This statement is executed if the gradeLevel is equal to 1.

This statement is executed if the gradeLevel is equal to 4.

Syntax for the switch Statement

```
switch ( <arithmetic expression> ) {
        <case label 1> : <case body 1>
        …
        <case label n> : <case body n>
}
```

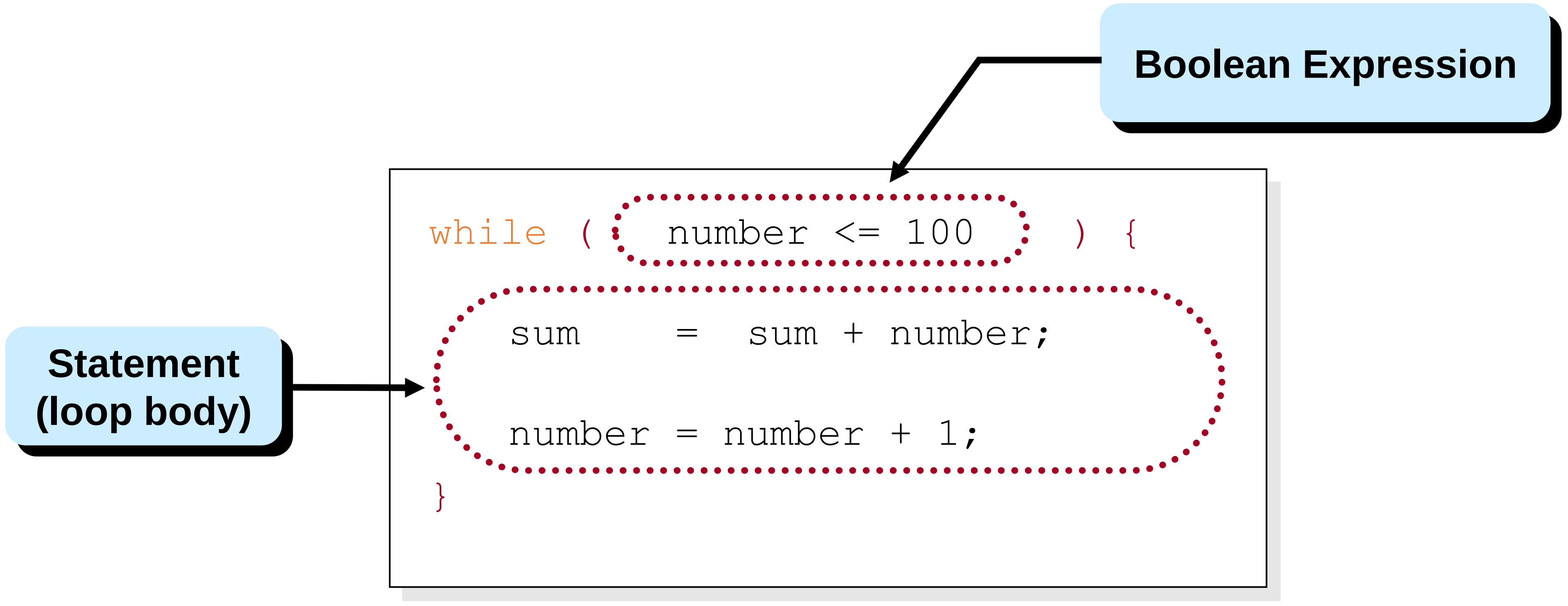


```
switch (  gradeLevel  ) {
    case 1: System.out.print("Go to the Gymnasium");
            break;
    case 2: System.out.print("Go to the Science Auditorium");
            break;
    case 3: System.out.print("Go to Harris Hall Rm A3");
            break;
    case 4: System.out.print("Go to Bolt Hall Rm 101");
            break;
}
```

Repetition

Syntax for the while Statement

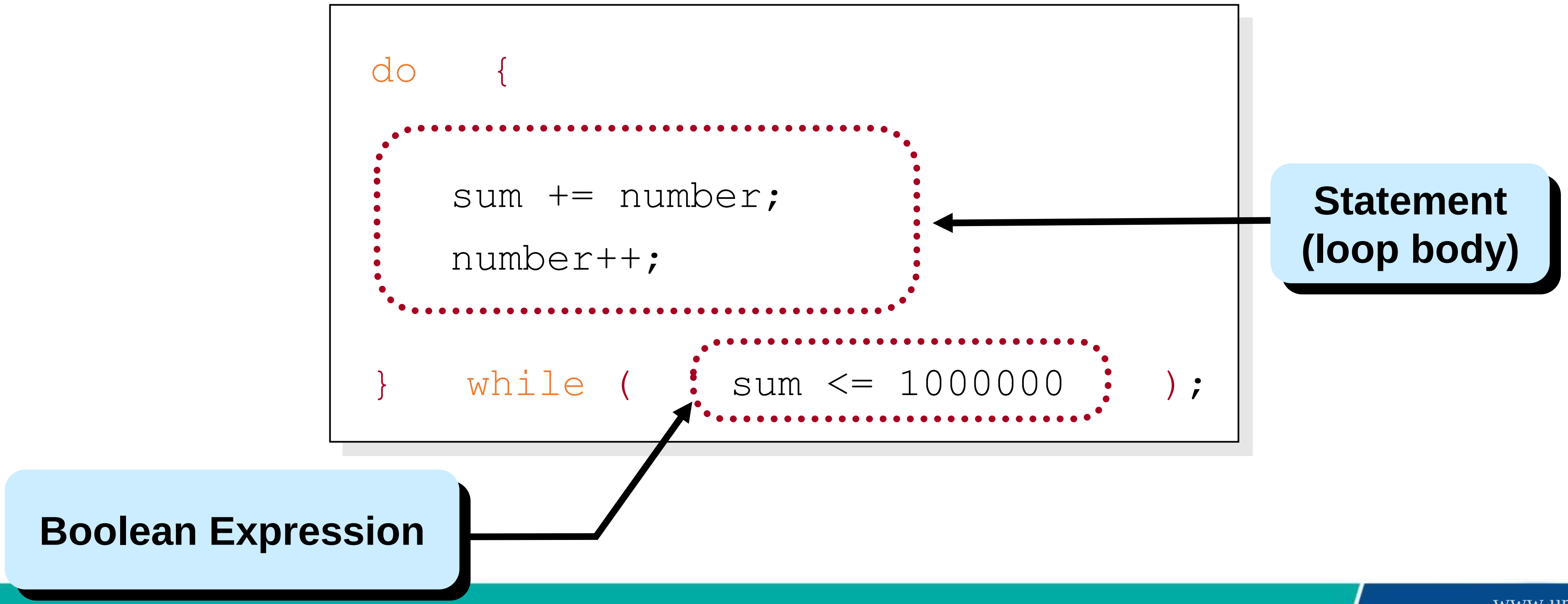
```
while ( <boolean expression> )

            <statement>
```



```
while ( number <= 100 ) {
    sum    =  sum + number;

    number = number + 1;
}
```

Syntax for the do-while Statement

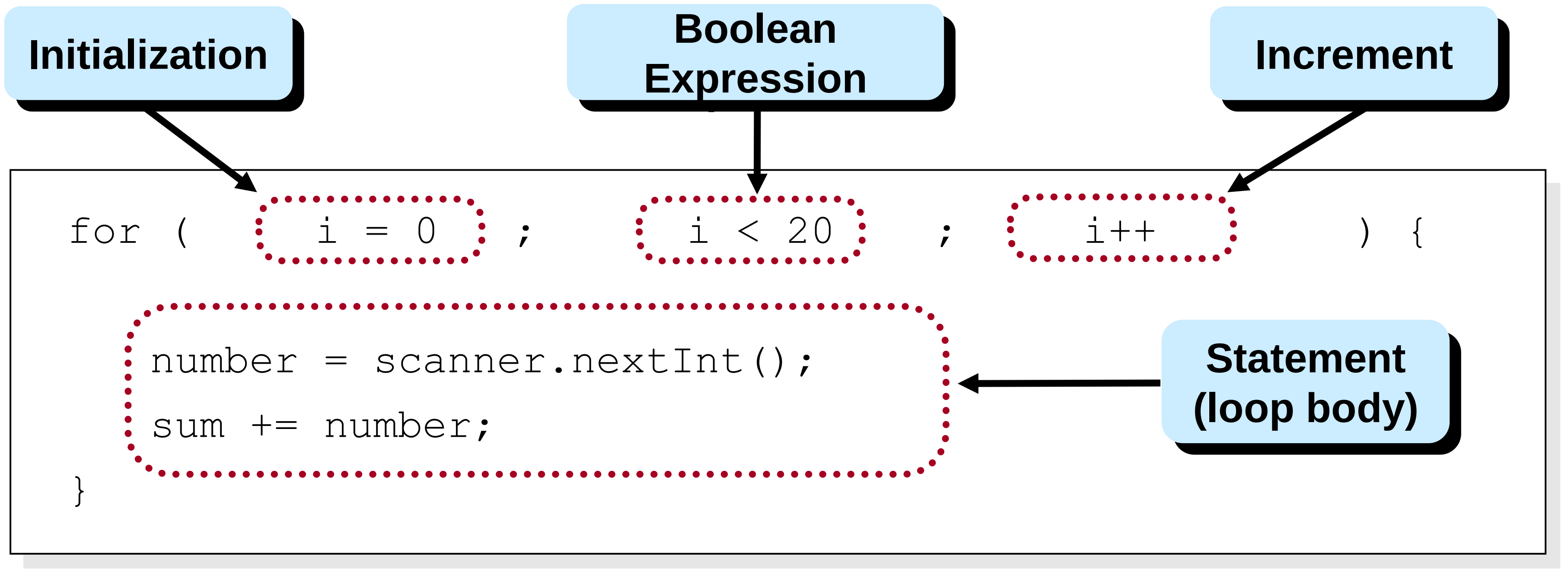
```
do
        <statement>

while ( <boolean expression> ) ;
```



```
do   {

    sum += number;
    number++;

}   while (  sum <= 1000000  );
```

Syntax for the for Statement

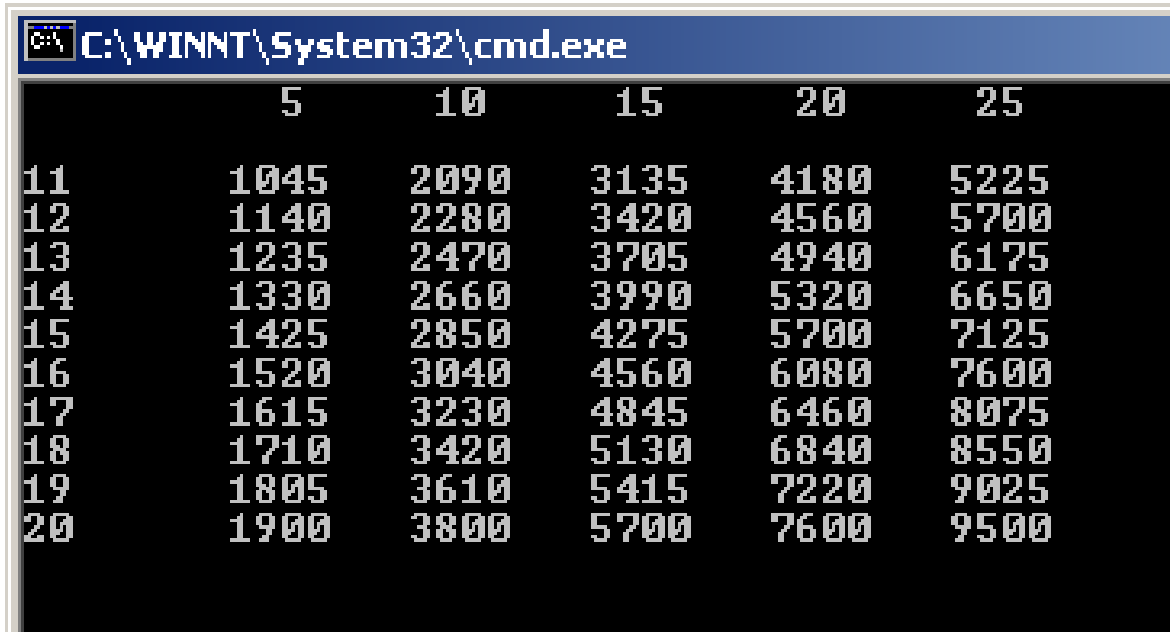
```
for ( <initialization>; <boolean expression>; <increment>  )

                    <statement>
```



```
for (  i = 0  ;   i < 20   ;   i++      ) {
    number = scanner.nextInt();
    sum += number;
}
```

The Nested-for Statement

- Nesting a for statement inside another for statement is commonly used technique in programming.
- Let’s generate the following table using nested-for statement.



C:\WINNT\System32\cmd.exe

```
                5       10       15       20       25

11           1045     2090     3135     4180     5225
12           1140     2280     3420     4560     5700
13           1235     2470     3705     4940     6175
14           1330     2660     3990     5320     6650
15           1425     2850     4275     5700     7125
16           1520     3040     4560     6080     7600
17           1615     3230     4845     6460     8075
18           1710     3420     5130     6840     8550
19           1805     3610     5415     7220     9025
20           1900     3800     5700     7600     9500
```

Generating the Table

```
int price;
for (int width = 11; width <=20, width++){

    for (int length = 5, length <=25, length+=5){


      price = width * length * 19; //$19 per sq. ft.
      System.out.print (“   “ + price);
    }
    //finished one row; move on to next row
    System.out.println(“”);
}
```

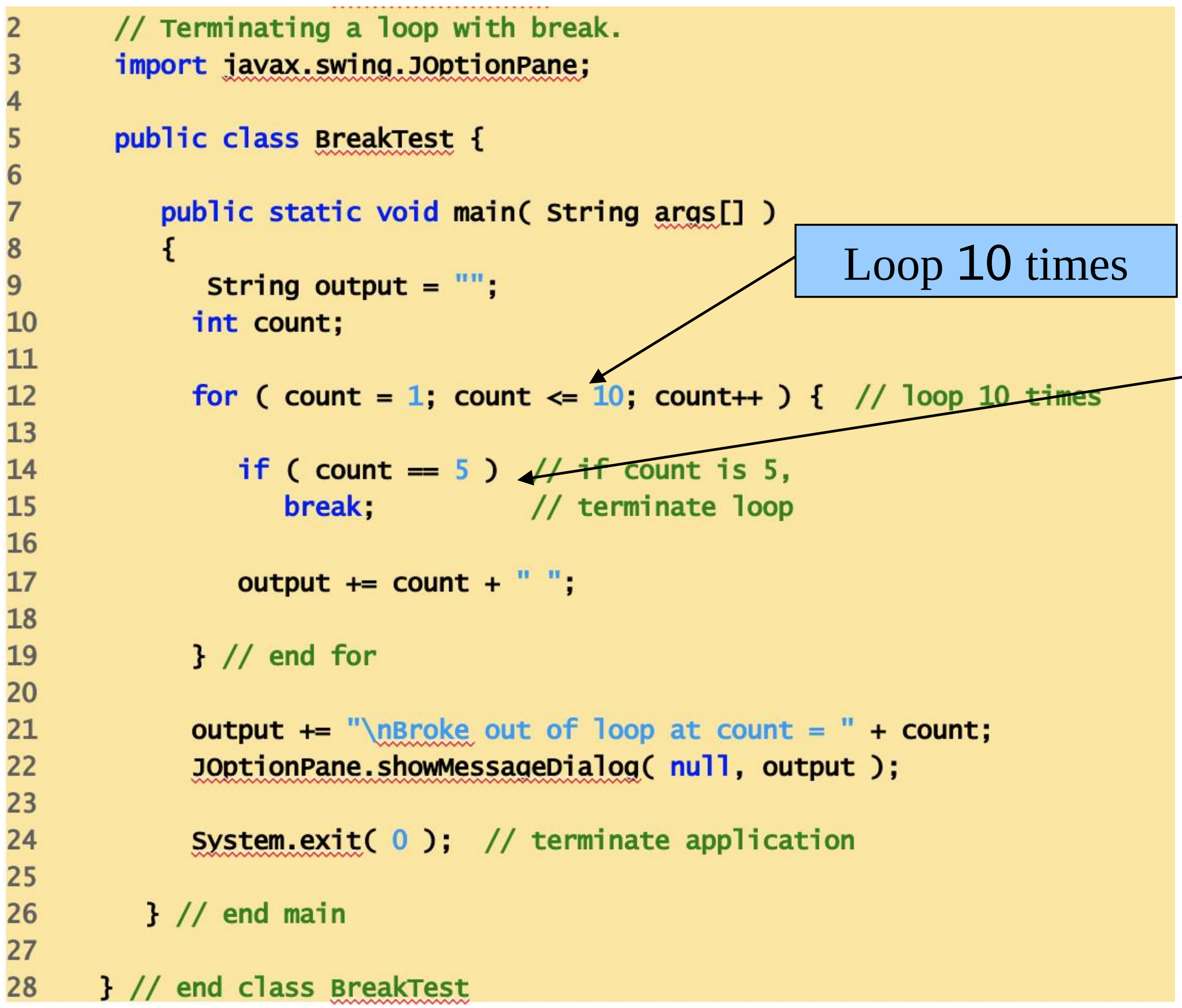
OUTER

INNER

Break and continue statement

- `break/continue`
  - Alter flow of control
- `break` statement
  - Causes immediate exit from control structure
    - Used in `while`, `for`, `do…while` or `switch` statements
    - Escape early from loop or skip remainder of switch
- `continue` statement
  - Skips remaining statements in loop body
  - Proceeds to next iteration
    - Used in `while`, `for` or `do…while` statements

Break statement example

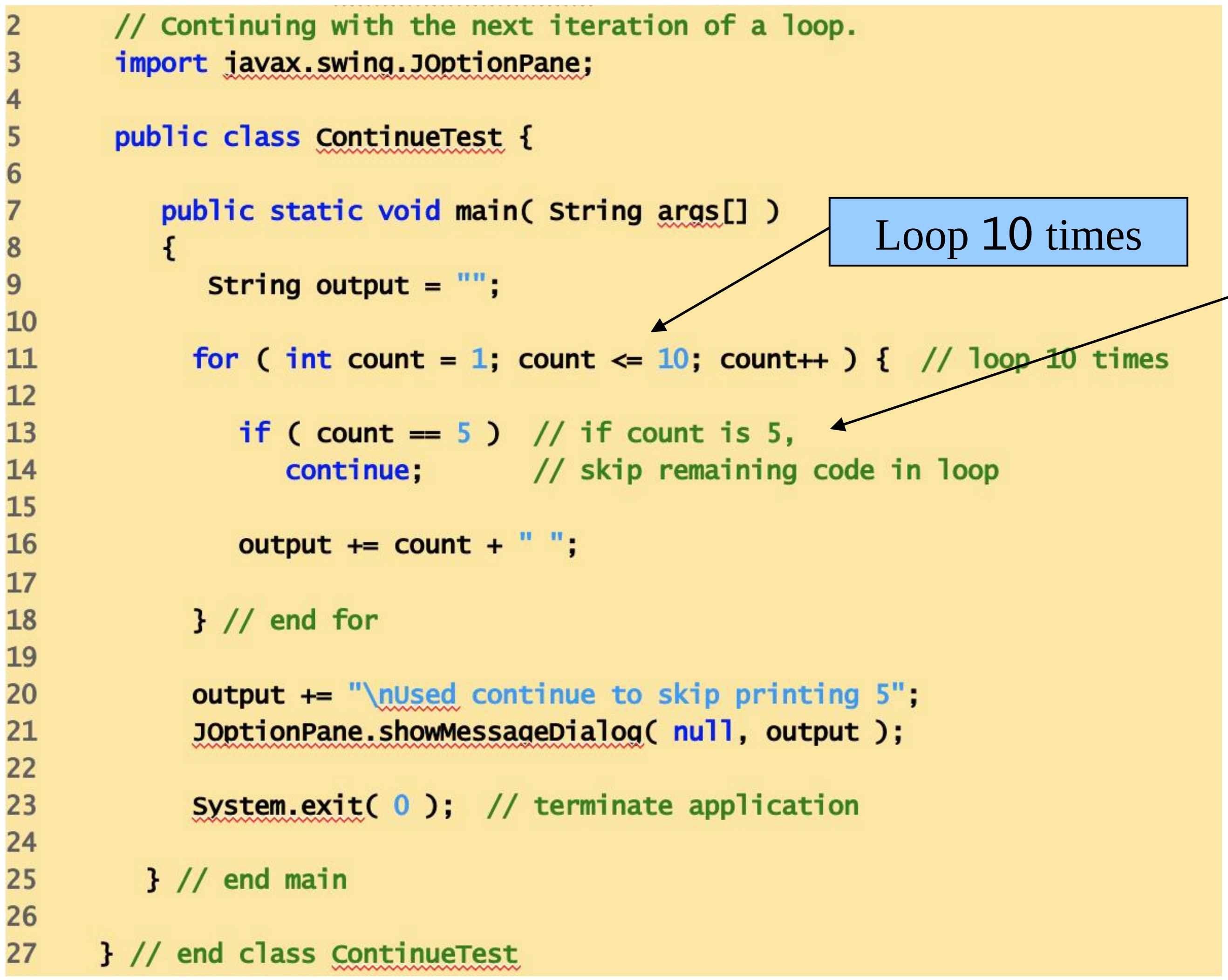


```java
// Terminating a loop with break.
import javax.swing.JOptionPane;

public class BreakTest {

  public static void main( String args[] )
  {
     String output = "";
    int count;

    for ( count = 1; count <= 10; count++ ) {  // loop 10 times

      if ( count == 5 )  // if count is 5,
        break;           // terminate loop

      output += count + " ";

    } // end for

    output += "\nBroke out of loop at count = " + count;
    JOptionPane.showMessageDialog( null, output );

    System.exit( 0 );  // terminate application

  } // end main

} // end class BreakTest
```

Loop 10 times

exit `for` structure (`break`) when `count` equals 5

Continue statement example



```
2     // Continuing with the next iteration of a loop.
3     import javax.swing.JOptionPane;
4
5     public class ContinueTest {
6
7        public static void main( String args[] )
8        {
9           String output = "";
10
11          for ( int count = 1; count <= 10; count++ ) {  // loop 10 times
12
13             if ( count == 5 )  // if count is 5,
14                continue;       // skip remaining code in loop
15
16             output += count + " ";
17
18          } // end for
19
20          output += "\nUsed continue to skip printing 5";
21          JOptionPane.showMessageDialog( null, output );
22
23          System.exit( 0 );  // terminate application
24
25       } // end main
26
27    } // end class ContinueTest
```

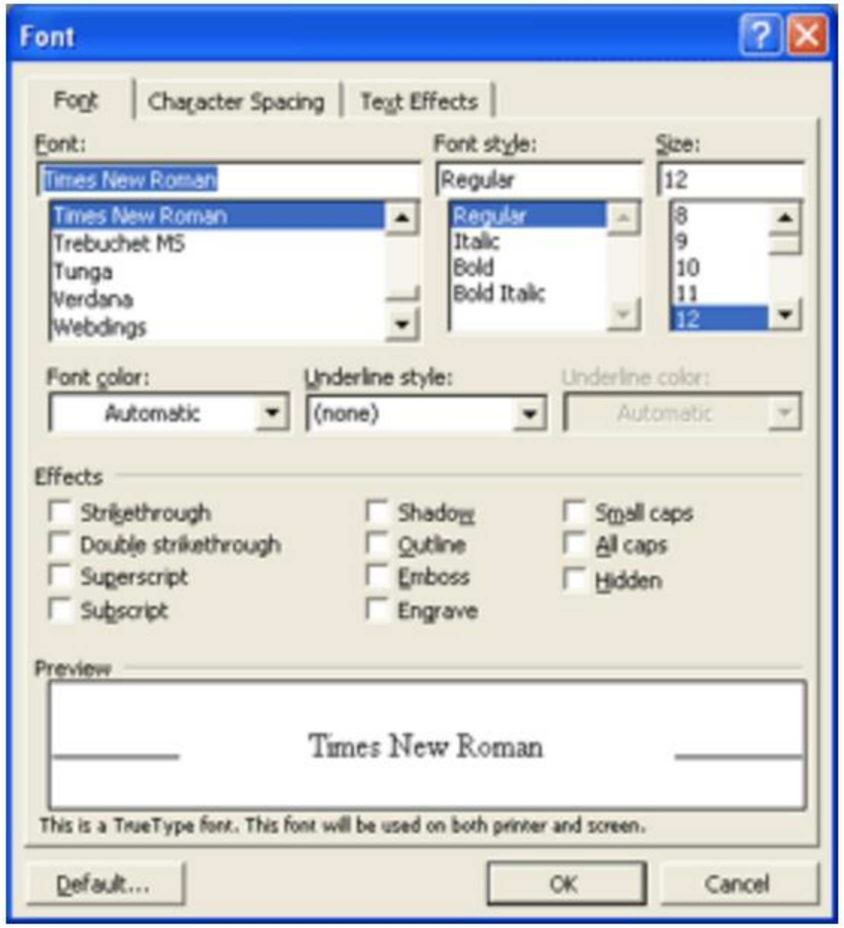
Loop 10 times

Skip line 16 and proceed to line 11 when `count` equals 5

JAVA GUI Applications

- Graphical User Interfaces (GUIs) are mechanisms for allowing users to enter data in the most economical and straightforward manner possible.
- example GUI that is designed to allow a user to choose a font type, style and size.

Figure 9.1. Example GUI: Font Dialog



Font: Times New Roman
Font style: Regular
Size: 12
Font color: Automatic
Underline style: (none)
Underline color: Automatic

This is a TrueType font. This font will be used on both printer and screen.

Basic GUI components

- A component is an object with a graphical representation that can be displayed on the screen and that can interact with the user.

Button (java.awt.Button)

```
Button()             // Constructs a Button with no label.
Button(String label) // Constructs a Button with the specified label.
```

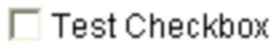
Figure 6.2. A Button Component

When a user presses on a Button object, an event is generated

Checkboxes (java.awt.Checkbox)

- Checkboxes are two states, on and off
- The state of the button is returned as the Object argument, when Checkbox event occurs

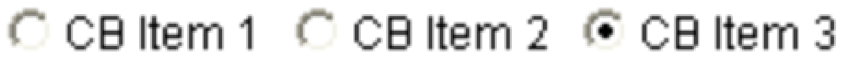
Figure 6.3. A Checkbox Component



Radio Buttons (java.awt.CheckboxGroup)

- Is a group of checkboxes, where only one of the items in the group can be selected at any one time

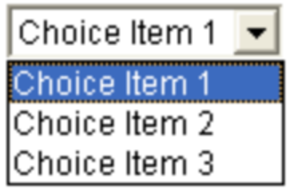
Figure 6.4. A Radio Button Component



Choice Buttons (java.awt.Choice)

- Like a radio button, where we make a selection, however it requires less space and allow us to add items to the menu dynamically using the *addItem()* method

Figure 6.5. A Choice Button Component



Labels (java.awt.Label)

- Allow us to add a text description to a point on the applet or application

Figure 6.6. A Label Component

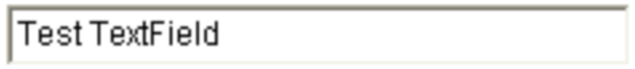
Test Label

TextFields (java.awt.TextField)

- Areas where user can enter text
- Useful for displaying and receiving text messages
- We can make it read-only or editable

```
TextField text1 = new TextField();                // no properties
TextField text2 = new TextField("Some text");     // a textfield with a
                                                  // predefined String
TextField text3 = new TextField(40);              // a textfield with a
                                                  // predefined size
TextField text4 = new TextField("Some text", 50); // combination of the two
```

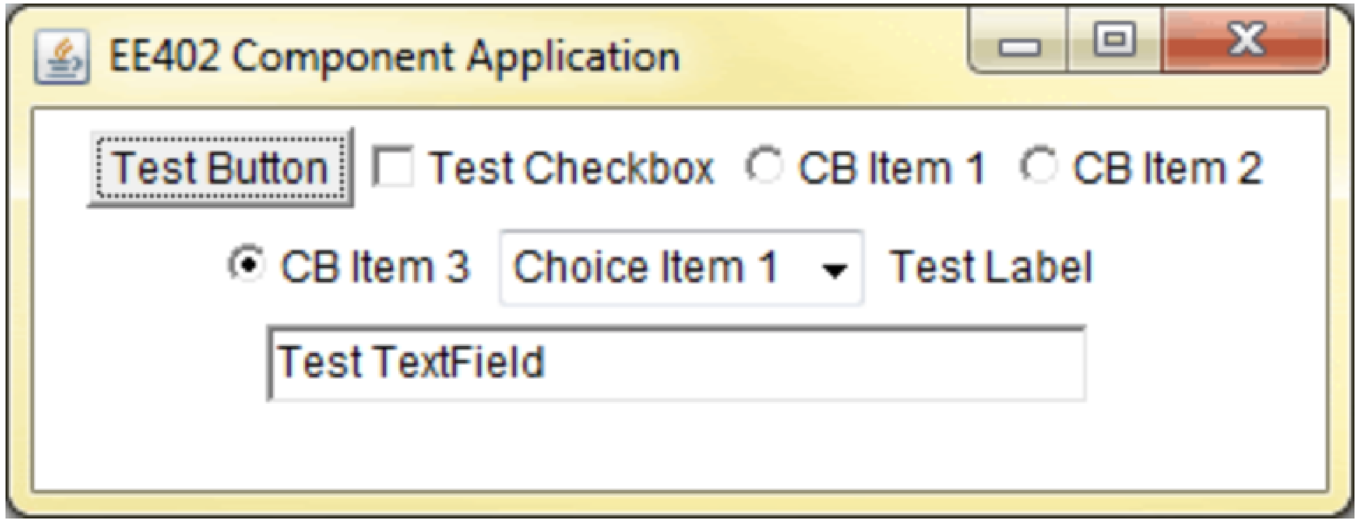
Figure 6.7. A TextField Component



An Example Component Application

- an example application that details all the previous components

Figure 6.8. A Component Application


EE402 Component Application
Test Button
Test Checkbox
CB Item 1
CB Item 2
CB Item 3
Choice Item 1
Test Label
Test TextField