جامعة الجلالة
GALALA UNIVERSITY

# CSE110 Principles of Programming

## Lecture 2: Control Statements Part 1

Professor Shaker El-Sappagh

Shaker.elsappagh@gu.edu.eg

Fall 2023

# Outline

1. Decision by if and if-else statements
2. Decision by conditional operator (?:)
3. Decision by switch statement

# Control statements in Java

**Java has only three kinds of control statements:**

1. **Sequence statement**
2. **Selection statements**
   - If statement
   - Conditional operator (?:)
   - Switch statement
3. **Iteration statements**
   - For statement
   - While statement
   - Do-while statement

# Sequence Structure in Java

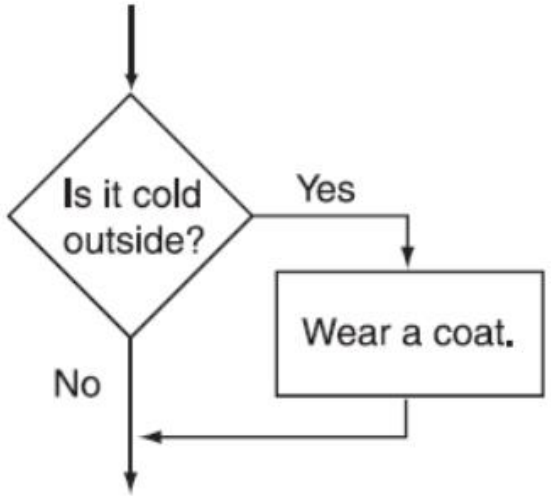
- The computer executes Java statements one after the other in the order in which they're written. (i.e., **sequence structure**)

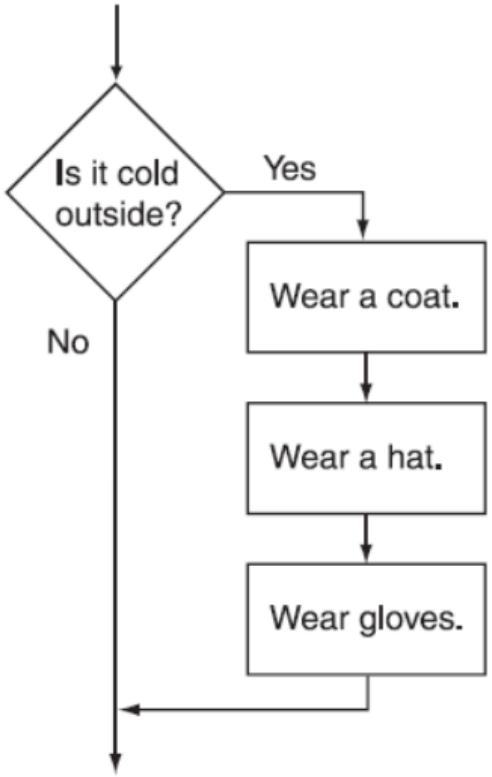
```
public class SquareArea
{

     public static void main(String[] args)
     {
          double length, width, area;

Step 1 ──► length = 10;
Step 2 ──► width = 5;
Step 3 ──► area = length * width;
Step 4 ──► System.out.print("The area is " + area);
     }

}
```

# Selection Statements in Java

**Java has three types of selection statements to make decisions:**

1. if statement either performs (selects) an action, if a condition is true, or skips it, if the condition is false. (single-selection statement )
2. The if…else statement performs an action if a condition is true and performs a different action if the condition is false. (double-selection statement)
3. The switch statement performs one of many different actions, depending on the value of an expression. (multiple-selection statement )

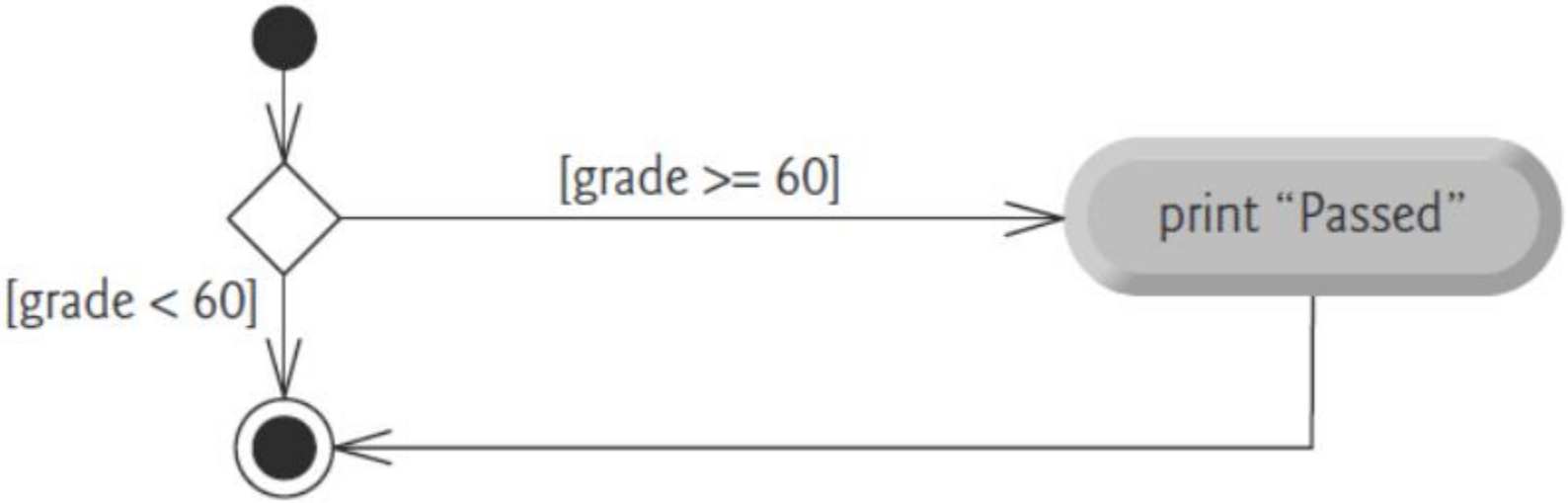


# if Single-Selection Statement

- The if statement is a single-entry/single-exit control statement

*If student's grade is greater than or equal to 60*
*Print "Passed"*




*Syntax*

```
if (condition)
{
    statements
}
```

```
if (studentGrade >= 60) {
    System.out.println("Passed");
}
```

# Using Relational operators to Form Conditions

- the boolean expression that is tested by an if statement is formed with a relational operator.
- A relational operator determines whether a specific relationship exists between two values.

| Relational Operators (in Order of Precedence) | Meaning |
|---|---|
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |
| == | Equal to |
| != | Not equal to |

| Expression | Meaning |
|---|---|
| x > y | Is x greater than y? |
| x < y | Is x less than y? |
| x >= y | Is x greater than or equal to y? |
| x <= y | Is x less than or equal to y? |
| x == y | Is x equal to y? |
| x != y | Is x not equal to y? |

# if Single-Selection Statement

- Be Careful with Semicolons

```
if (expression)
    statement;
```

No semicolon here

Semicolon goes here

?

```
if (floor > 13) ;
{
   floor--;
}
```

- Having Multiple Conditionally executed Statements

```
if (sales > 50000)
{
     bonus = 500.0;
     commissionRate = 0.12;
     daysOff += 1;
}
```

```
if (sales > 50000)
  bonus = 500.0;
  commissionRate = 0.12;
  daysOff += 1;
```

Only this statement is conditionally executed.

These statements are *always* executed.

# if Single-Selection Statement

- Comparing Characters: as numbers

```
if (ch == 'A')
    System.out.println("The letter is A.");
if (ch != 'A')
    System.out.println("Not the letter A.");
```

- In Unicode, letters are arranged in alphabetic order. Because ‘A’ comes before ‘B’, the numeric code for the character ‘A’ is less than the code for the character ‘B’.

```
if ('A' < 'B')
    System.out.println("A is less than B.");
```

# The if-else Statement

- The if-else statement will execute one group of statements if its boolean expression is true, or another group if its boolean expression is false.

```
if (BooleanExpression)
    statement or block
else
    statement or block
```

# The if-else Statement

A condition that is true or false.
Often uses relational operators:
== != < <= > >= (See Table 1.)

Braces are not required if the branch contains a single statement, but it's good to always use them.
See Programming Tip 3.2.

```
if (floor > 13)
{
   actualFloor = floor - 1;
}
else
{
   actualFloor = floor;
}
```

Don't put a semicolon here!
See Common Error 3.1.

If the condition is true, the statement(s) in this branch are executed in sequence; if the condition is false, they are skipped.

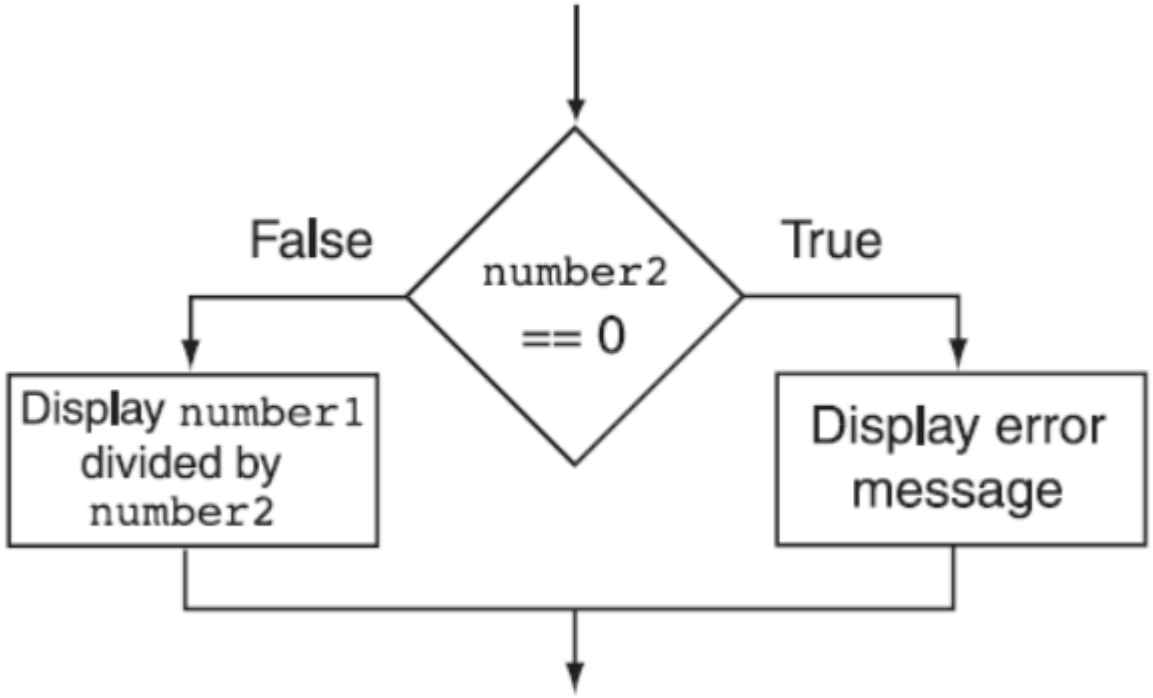
Omit the `else` branch if there is nothing to do.

If the condition is false, the statement(s) in this branch are executed in sequence; if the condition is true, they are skipped.

Lining up braces is a good idea.
See Programming Tip 3.1.

# The if-else Statement

uses the if-else statement to handle a classic programming problem: **division by zero**.



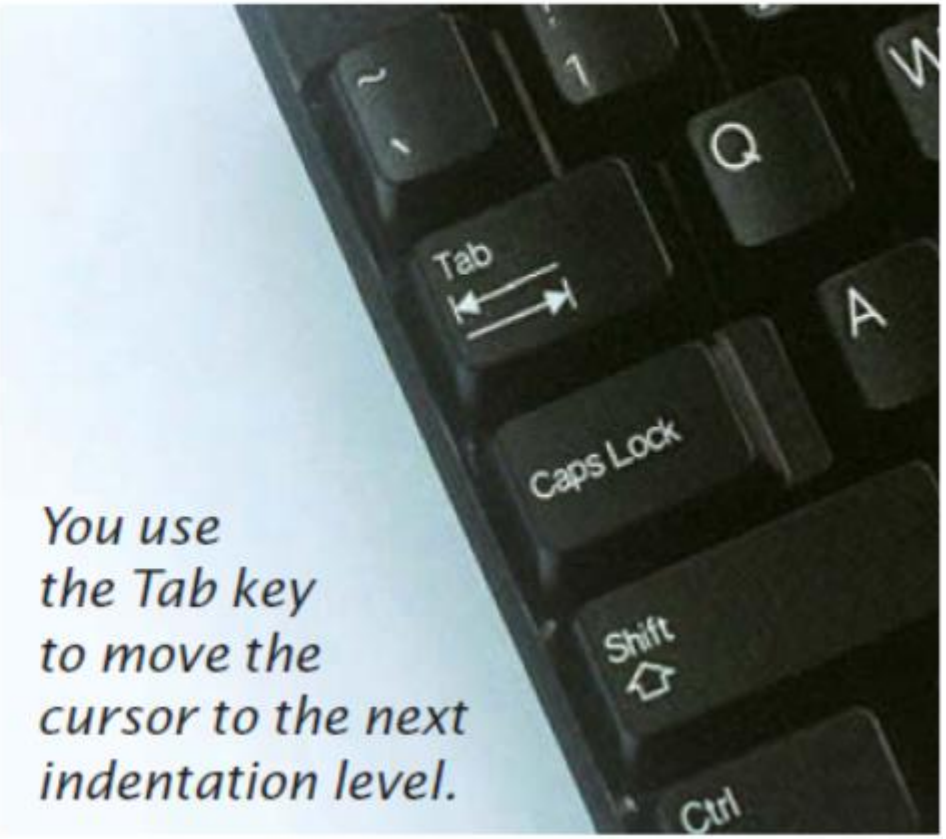

```
public class Division
{
   public static void main(String[] args)
   {
      double number1, number2; // Division operands
      double quotient;         // Result of division

      // Create a Scanner object for keyboard input.
      Scanner keyboard = new Scanner(System.in);

      // Get the first number.
      System.out.print("Enter a number: ");
      number1 = keyboard.nextDouble();

      // Get the second number.
      System.out.print("Enter another number: ");
      number2 = keyboard.nextDouble();

      if (number2 == 0)
      {
          System.out.println("Division by zero is not possible.");
          System.out.println("Please run the program again and ");
          System.out.println("enter a number other than zero.");
      }
      else
      {
          quotient = number1 / number2;
          System.out.print("The quotient of " + number1);
          System.out.print(" divided by " + number2);
          System.out.println(" is " + quotient);
      }
   }
}
```

# Writing style

## Tabs

Block-structured code has the property that nested statements are indented by one or more levels:

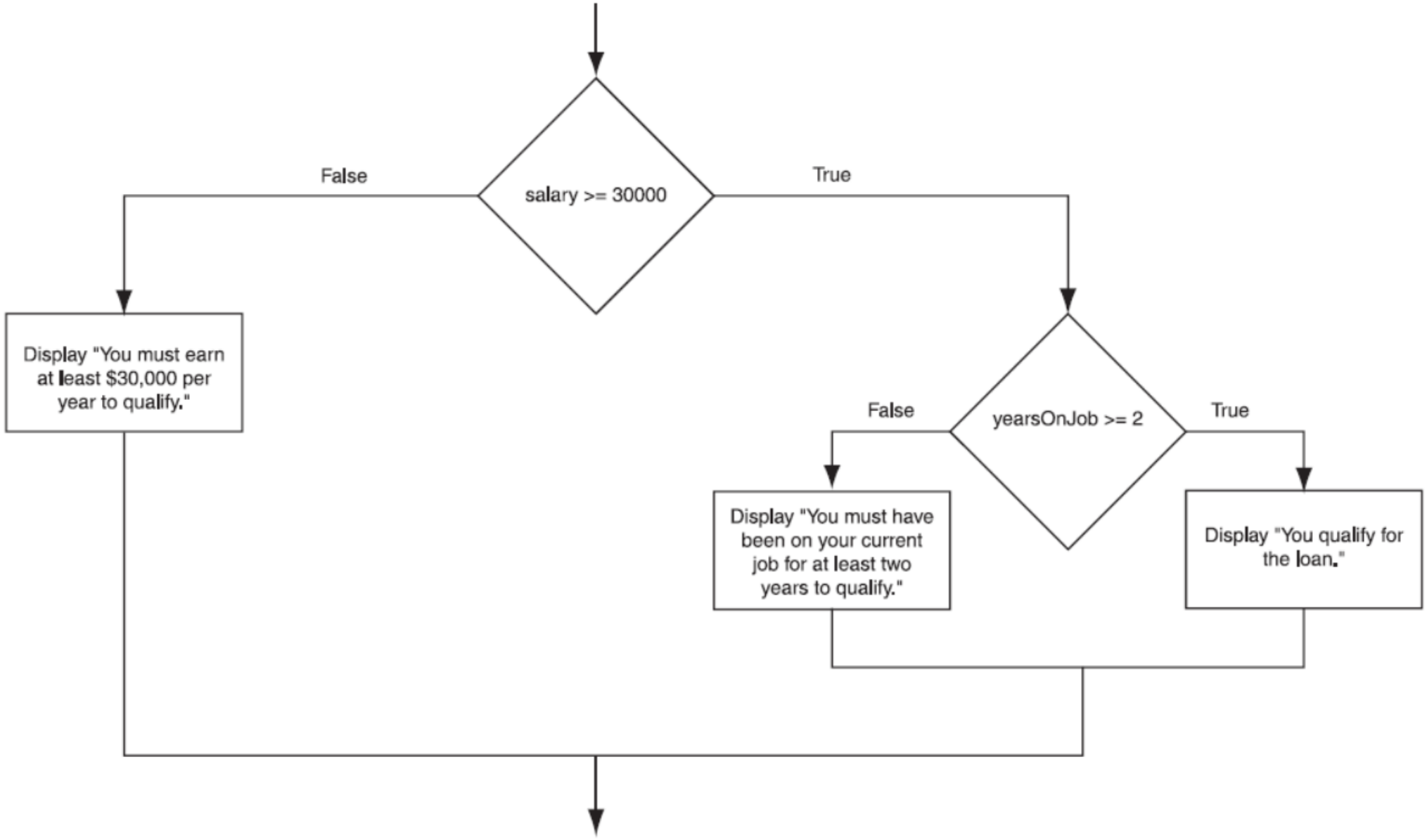
```
public class ElevatorSimulation
{
|  public static void main(String[] args)
|  {
|  |  int floor;
|  |  . . .
|  |  if (floor > 13)
|  |  {
|  |  |  floor--;
|  |  }
|  |  . . .
|  }
|  |  |  |
0  1  2  3    Indentation level
```




*You use the Tab key to move the cursor to the next indentation level.*

# nested if Statements

To test more than one condition, an if statement can be nested inside another if statement.



# nested if Statements

This `if` and `else` go together.

This `if` and `else` go together.

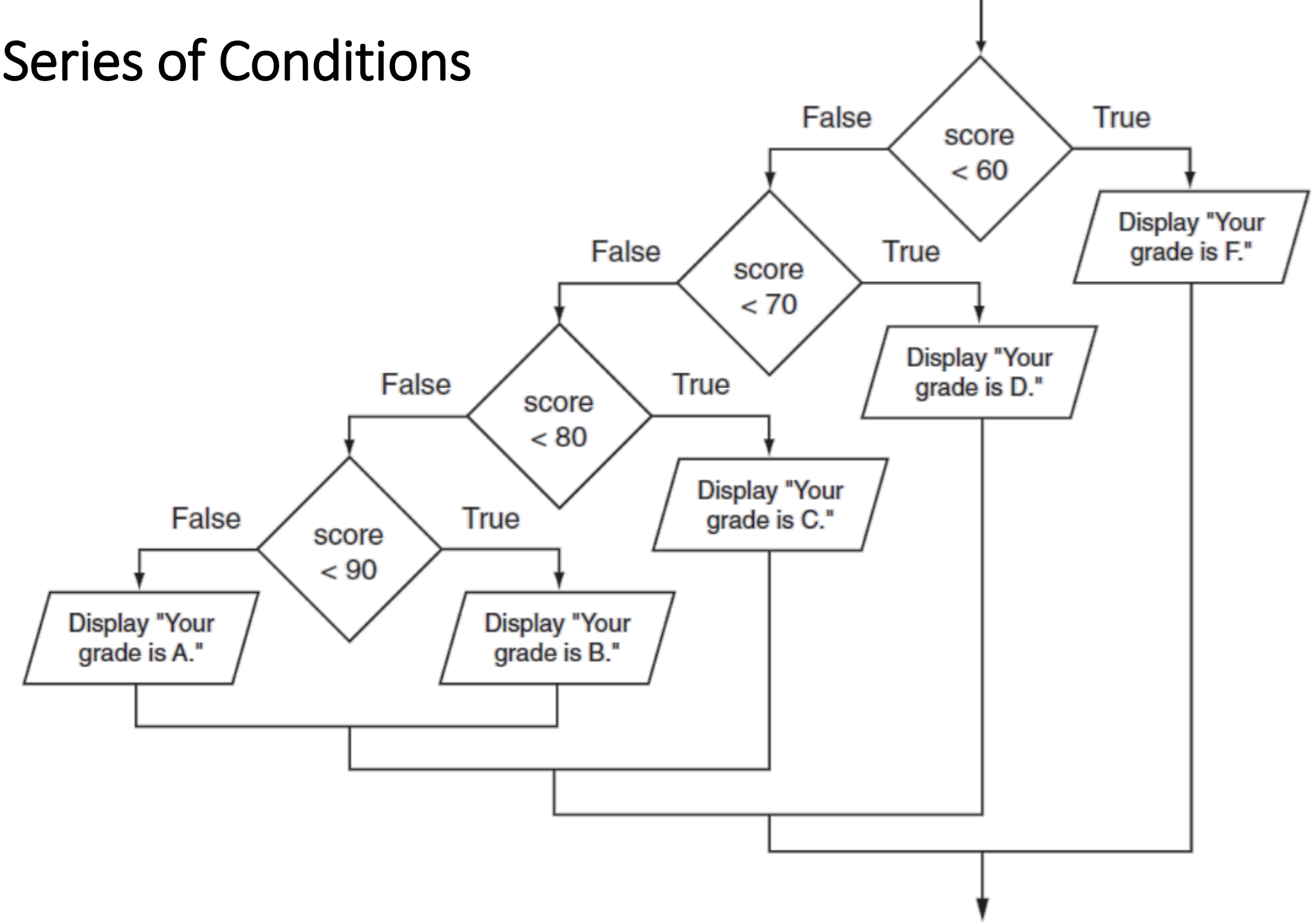
```
if (salary >= 30000)
{
    if (yearsOnJob >= 2)
    {
        JOptionPane.showMessageDialog(null, "You qualify " +
                                            "for the loan.");
    }
    else
    {
        JOptionPane.showMessageDialog(null, "You must have " +
                    "been on your current job for at least " +
                    "two years to qualify.");
    }
}
else
{
    JOptionPane.showMessageDialog(null, "You must earn " +
                "at least $30,000 per year to qualify.");
}
```

# Testing a Series of Conditions

| Test Score | Grade |
|---|---|
| 90 and above | A |
| 80–89 | B |
| 70–79 | C |
| 60–69 | D |
| Below 60 | F |

*Ask the user to enter a test score.*
*Determine the grade in the following manner:*
*If the score is less than 60, then the grade is F.*
*Otherwise, if the score is less than 70, then the grade is D.*
*Otherwise, if the score is less than 80, then the grade is C.*
*Otherwise, if the score is less than 90, then the grade is B.*
*Otherwise, the grade is A.*

# Testing a Series of Conditions



# Testing a Series of Conditions

```
public static void main(String[] args)
{
    int testScore;    // Numeric test score
    String input;     // To hold the user's input

    // Get the numeric test score.
    input = JOptionPane.showInputDialog("Enter your numeric " +
                  "test score and I will tell you the grade: ");
    testScore = Integer.parseInt(input);

    // Display the grade.
    if (testScore < 60)
    {
       JOptionPane.showMessageDialog(null, "Your grade is F.");
    }
    else
    {
       if (testScore < 70)
       {
         JOptionPane.showMessageDialog(null, "Your grade is D.");
       }
       else
          {
          if (testScore < 80)
          {
            JOptionPane.showMessageDialog(null, "Your grade is C.");
          }
          else
          {
            if (testScore < 90)
            {
              JOptionPane.showMessageDialog(null, "Your grade is B.");
            }
            else
            {
              JOptionPane.showMessageDialog(null, "Your grade is A.");
            }
       }
    }
 }

 System.exit(0);
}
```

# Testing a Series of Conditions (could also be written as:)

```
if (studentGrade >= 90) {
   System.out.println("A");
}
else {
   if (studentGrade >= 80) {
      System.out.println("B");
   }
   else {
      if (studentGrade >= 70) {
         System.out.println("C");
      }
      else {
         if (studentGrade >= 60) {
            System.out.println("D");
         }
         else {
            System.out.println("F");
         }
      }
   }
}
```

## The if-else-if Statement

- The if-else-if statement tests a series of conditions. It is often simpler to test a series of conditions with the if-else-if statement than with a set of nested if-else statements.

```
if (expression_1)
{
  statement
  statement
  etc.
}
else if (expression_2)
{
  statement
  statement
  etc.
}
Insert as many else if clauses as necessary
else
{
  statement
  statement
  etc.
}
```

*If* `expression_1` *is true these statements are executed, and the rest of the structure is ignored.*

*Otherwise, if* `expression_2` *is true these statements are executed, and the rest of the structure is ignored.*

*These statements are executed if none of the expressions above are true.*

# The if-else-if Statement

```
public static void main(String[] args)
{
  int testScore;     // Numeric test score
  String input;      // To hold the user's input

  // Get the numeric test score.
  input = JOptionPane.showInputDialog("Enter your numeric " +
                  "test score and I will tell you the grade: ");
  testScore = Integer.parseInt(input);

  // Display the grade.
  if (testScore < 60)
     JOptionPane.showMessageDialog(null, "Your grade is F.");
  else if (testScore < 70)
     JOptionPane.showMessageDialog(null, "Your grade is D.");
  else if (testScore < 80)
     JOptionPane.showMessageDialog(null, "Your grade is C.");
  else if (testScore < 90)
     JOptionPane.showMessageDialog(null, "Your grade is B.");
  else
     JOptionPane.showMessageDialog(null, "Your grade is A.");

  System.exit(0);
}
```

**Using the Trailing `else` to Catch errors**

# Dangling-else Problem

- We always enclose control statement bodies in braces ({ and }). This avoids a logic error called the “dangling-else” problem.

```
if (grade >= 60) {
   System.out.println("Passed");
}
else {
   System.out.println("Failed");
   System.out.println("You must take this course again.");
}
```

- if grade is less than 60, the program executes both statements in the body of the else and prints:

```
Failed
You must take this course again.
```

- Without the braces, the statement

```
System.out.println("You must take this course again.");
```

would be outside the body of the else part of the if…else statement and would execute regardless of whether the grade was less than 60.

# Exercise

```
public static void main(String[] args)
{
    int funny = 7, serious = 15;
    funny = serious % 2;
    if (funny != 1)
    {
        funny = 0;
        serious = 0;
    }
    else if (funny == 2)
    {
        funny = 10;
        serious = 10;
    }
    else
    {
        funny = 1;
        serious = 1;
    }
    System.out.println(funny + " " + serious);
}
```

# Logical operators to create compound conditions

| Operator | Meaning | Effect |
|---|---|---|
| `&&` | AND | Connects two `boolean` expressions into one. Both expressions must be `true` for the overall expression to be `true`. |
| `\|\|` | OR | Connects two `boolean` expressions into one. One or both expressions must be `true` for the overall expression to be `true`. It is only necessary for one to be `true`, and it does not matter which one. |
| `!` | NOT | The `!` operator reverses the truth of a `boolean` expression. If it is applied to an expression that is `true`, the operator returns `false`. If it is applied to an expression that is `false`, the operator returns `true`. |

# Logical operators to create compound conditions

## The && Operator

The && operator is known as the logical AND operator. It takes two boolean expressions as operands and creates a boolean expression that is true only when both subexpressions are true. Here is an example of an if statement that uses the && operator:

```
if (temperature < 20 && minutes > 12)
{
      System.out.println("The temperature is in the " +
                         "danger zone.");
}
```

**Truth table for the && operator**

| Expression | Value of the Expression |
|---|---|
| true && false | false |
| false && true | false |
| false && false | false |
| true && true | true |

# Logical operators to create compound conditions

## The || Operator

The || operator is known as the logical OR operator. It takes two `boolean` expressions as operands and creates a `boolean` expression that is `true` when either of the subexpressions is `true`. Here is an example of an `if` statement that uses the || operator:

```
if (temperature < 20 || temperature > 100)
{
      System.out.println("The temperature is in the " +
                         "danger zone.");
}
```

**Truth table for the || operator**

| Expression | Value |
|---|---|
| `true || false` | `true` |
| `false || true` | `true` |
| `false || false` | `false` |
| `true || true` | `true` |

# Logical operators to create compound conditions

## The ! Operator

The ! operator performs a logical NOT operation. It is a unary operator that takes a boolean expression as its operand and reverses its logical value. In other words, if the expression is true, the ! operator returns false, and if the expression is false, it returns true. Here is an if statement using the ! operator:

```
if (!(temperature > 100))
   System.out.println("This is below the maximum temperature.");
```

**Truth table for the ! operator**

| Expression | Value |
|---|---|
| !true | false |
| !false | true |

# The precedence of Logical operators

| Order of Precedence | Operators | Description |
|---|---|---|
| 1 | - (unary negation) ! | Unary negation, logical NOT |
| 2 | * / % | Multiplication, division, modulus |
| 3 | + - | Addition, subtraction |
| 4 | < > <= >= | Less than, greater than, less than or equal to, greater than or equal to |
| 5 | == != | Equal to, not equal to |
| 6 | && | Logical AND |
| 7 | \|\| | Logical OR |
| 8 | = += -= *= /= %= | Assignment and combined assignment |

# The precedence of Logical operators

The ! operator has a higher precedence than many of Java's other operators.

The && and || operators rank lower in precedence than the relational operators

```
(a > b) && (x < y) is the same as a > b && x < y
(x == y) || (b > a) is the same as x == y || b > a
```

The logical operators evaluate their expressions from left to right. In the following expression, a < b is evaluated before y == z.

```
a < b || y == z
```

In the following expression, y == z is evaluated first, however, because the && operator has higher precedence than ||.

```
a < b || y == z && m > j
```

For safety use this:

```
(a < b) || ((y == z) && (m > j))
```

# Checking numeric Ranges with Logical operators

When determining whether a number is inside a range, it's best to use the **&& operator**.

```
if (x >= 20 && x <= 40)
   System.out.println(x + " is in the acceptable range.");
```

When determining whether a number is outside a range, it's best to use the **|| operator**.

```
if (x < 20 || x > 40)
   System.out.println(x + " is outside the acceptable range.");
```

It's important not to get the logic of these logical operators confused. For example, the boolean expression in the following if statement **would never test true**:

```
if (x < 20 && x > 40)
   System.out.println(x + " is outside the acceptable range.");
```

# Comparing String objects

- You cannot use relational operators to compare String objects. Instead, you must use a String method.
- Remember that a String object is referenced by a variable that contains the object's memory address.
- When you use a relational operator with the **reference variable**, the operator works on the **memory address** that the variable contains, **not the contents of the String object**.

```
String name1 = "Mark";
String name2 = "Mary";
```

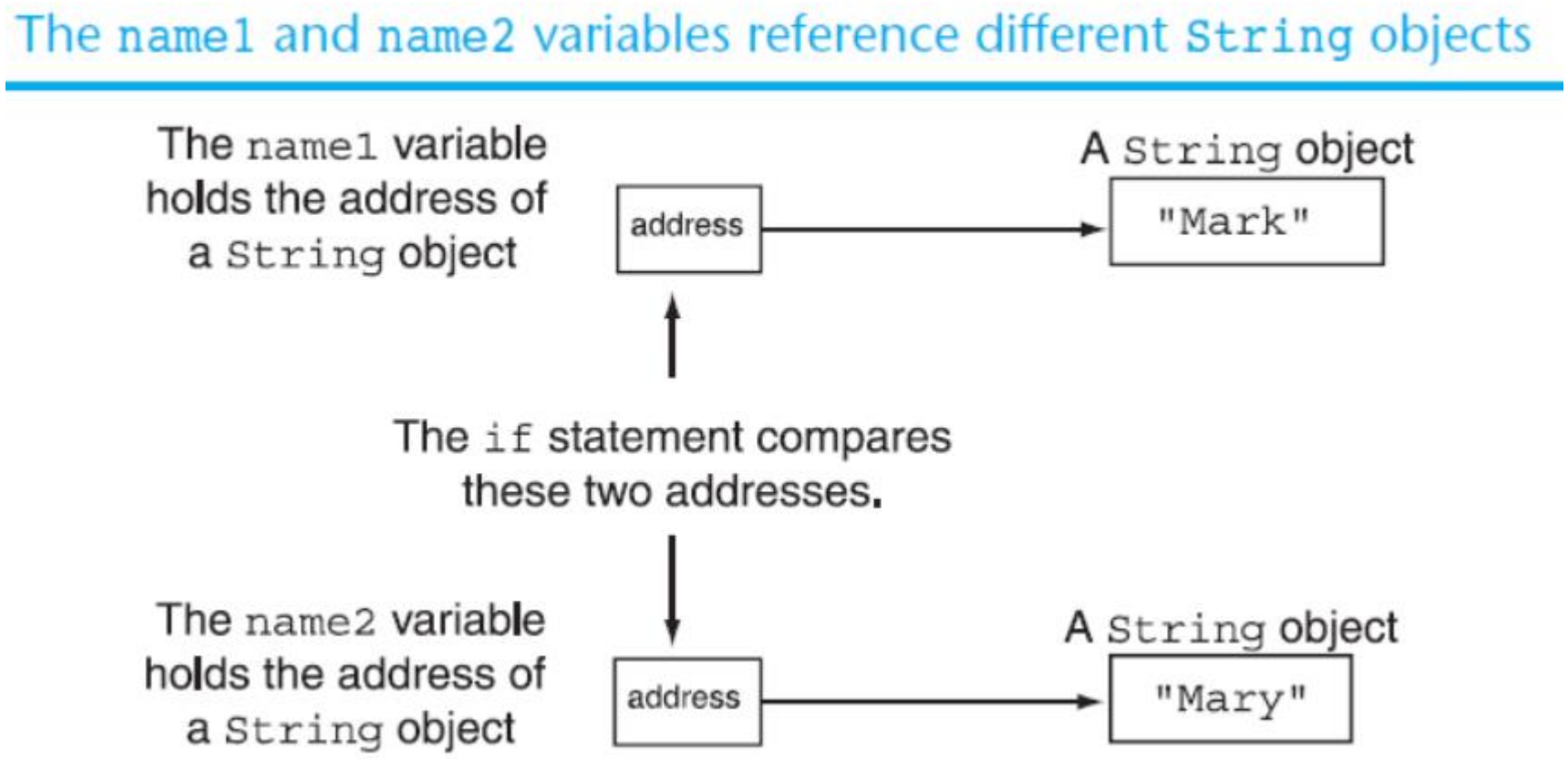
And later, the same program has the following `if` statement:

```
if (name1 == name2)
```

The expression `name1 == name2` will be `false`, but not because the strings "Mark" and "Mary" are different. The expression will be `false` because the variables `name1` and `name2` reference different objects.

# Comparing String objects

The correct way is to



# Comparing String objects

**The correct way:**

To compare the contents of two String objects correctly, you should use the String class's equals method. The general form of the method is as follows:

*StringReference1.equals(StringReference2)*

StringReference1 is a variable that **references** a String object, and StringReference2 is another variable that **references** a String object. The method returns true **if the two strings are equal**, or false **if they are not equal**. Here is an example:

*if (name1.equals(name2))*

# Comparing String objects

```
public static void main(String[] args)
{
   String name1 = "Mark",
          name2 = "Mark",
          name3 = "Mary";

   // Compare "Mark" and "Mark"

   if (name1.equals(name2))
   {
       System.out.println(name1 + " and " + name2 +
                          " are the same.");
   }
   else
   {
       System.out.println(name1 + " and " + name2 +
                          " are NOT the same.");
   }

   // Compare "Mark" and "Mary"

   if (name1.equals(name3))
   {
       System.out.println(name1 + " and " + name3 +
   " are the same.");
   }
   else
   {
       System.out.println(name1 + " and " + name3 +
                          " are NOT the same.");
   }
}
```

**Program Output**

```
Mark and Mark are the same.
Mark and Mary are NOT the same.
```

# Comparing String objects

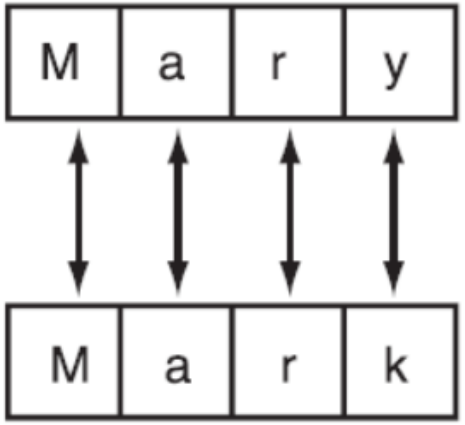
- The String class also provides the compareTo method, which can be used to determine whether one string is greater than, equal to, or less than another string. The general form of the method is as follows:

*StringReference.compareTo(OtherString)*

- The method returns an integer value that can be used in the following manner:

  - If the method's return value is negative, the string referenced by `StringReference` (the calling object) is less than the `OtherString` argument.
  - If the method's return value is 0, the two strings are equal.
  - If the method's return value is positive, the string referenced by `StringReference` (the calling object) is greater than the `OtherString` argument.

- The String class provides the equalsIgnoreCase and compareToIgnoreCase methods.



# The Conditional operator

- You can use the conditional operator to create short expressions that work like if-else statements.
- Syntax: *BooleanExpression ? Value1: Value2;*

```
if (x < 0)
    y = 10;
else
    y = 20;
```

```
y = x < 0 ? 10: 20;
```

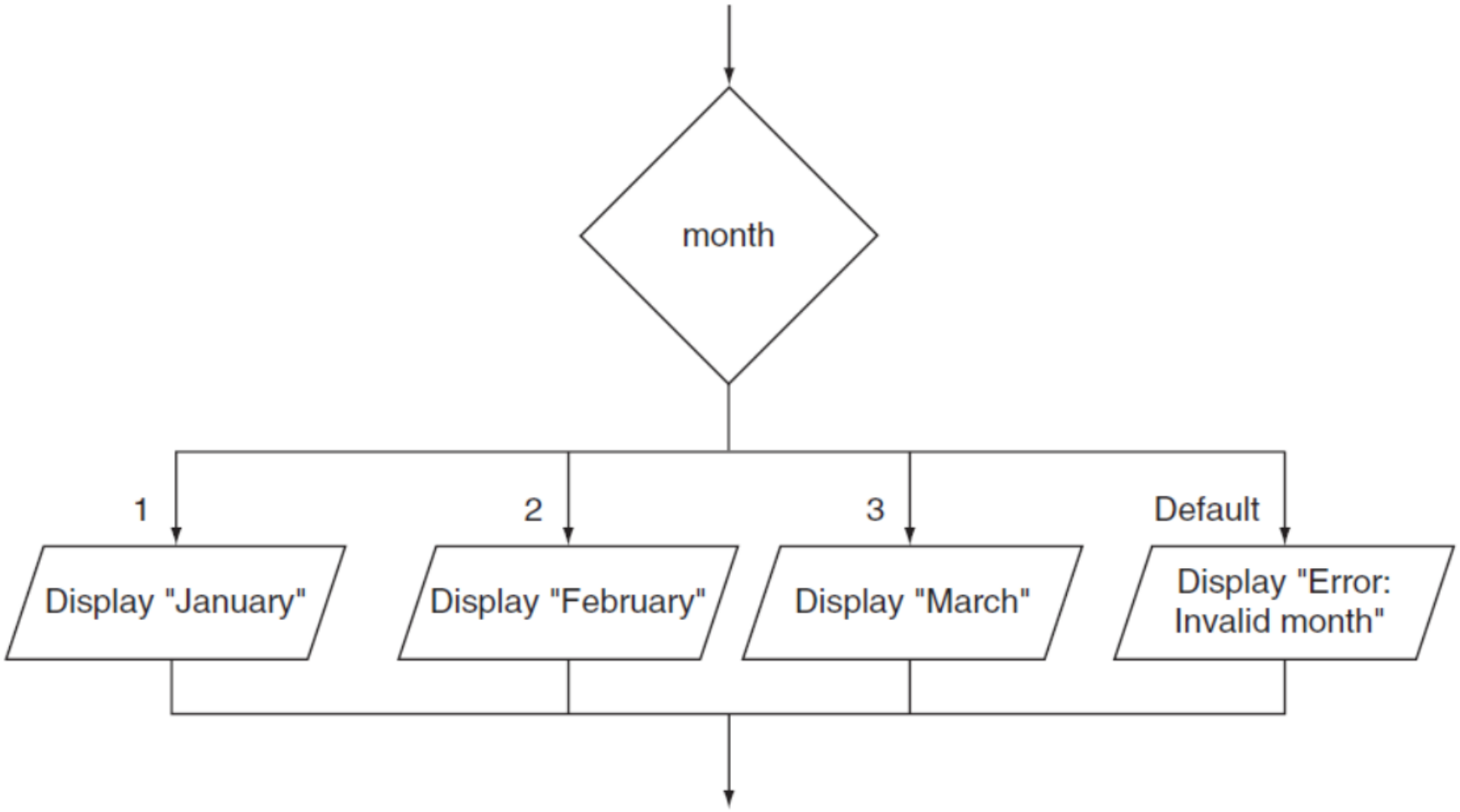
```
System.out.println("Your grade is: " +
            (score < 60 ? "Fail." : "Pass."));
```

Converted to an `if-else` statement, it would be written as follows:

```
if (score < 60)
    System.out.println("Your grade is: Fail.");
else
    System.out.println("Your grade is: Pass.");
```

# The switch Statement

- The switch statement lets the value of a variable or expression determine where the program will branch to.



# The switch Statement

The *testExpression* is a variable or an expression that gives a **char, byte, short, int, or string** value. (If you are using a version of Java prior to Java 7, the *testExpression* cannot be a string.)

```
switch (testExpression)
{
   case value_1:
      statement;
      statement;
      etc.
      break;

   case value_2:
      statement;
      statement;
      etc.
      break;

   case value_N:
      statement;
      statement;
      etc.
      break;

   default:
      statement;
      statement;
      etc.
      break;
}
```

*The `testExpression` is a variable or expression.*

*These statements are executed if the `testExpression` is equal to `value_1`.*

*These statements are executed if the `testExpression` is equal to `value_2`.*

*Insert as many `case` sections as necessary.*

*These statements are executed if the `testExpression` is equal to `value_N`.*

*These statements are executed if the `testExpression` is not equal to any of the `case` values.*

# The switch Statement

```
if (month == 1)
{
    System.out.println("January");
}

else if (month == 2)
{
    System.out.println("February");
}
else if (month == 3)
{
    System.out.println("March");
}
else
{
    System.out.println("Error: Invalid month");
}
```

```
switch (month)
{
    case 1:
        System.out.println("January");
        break;

    case 2:
        System.out.println("February");
        break;

    case 3:
        System.out.println("March");
        break;

    default:
        System.out.println("Error: Invalid month");
        break;
}
```

# The switch Statement (Example 1)

```
public class PetFood
{
   public static void main(String[] args)
   {
      String input;       // To hold the user's input
      char foodGrade;     // Grade of pet food

      // Create a Scanner object for keyboard input.
      Scanner keyboard = new Scanner(System.in);

      // Prompt the user for a grade of pet food.
      System.out.println("Our pet food is available in " +
                         "three grades:");
      System.out.print("A, B, and C. Which do you want " +
                       "pricing for? ");
      input = keyboard.nextLine();
      foodGrade = input.charAt(0);
```

# The switch Statement (Example 1)

```
switch(foodGrade)
{
    case 'a':
    case 'A':
       System.out.println("30 cents per lb.");
       break;
    case 'b':
    case 'B':
       System.out.println("20 cents per lb.");
       break;
     case 'c':
     case 'C':
        System.out.println("15 cents per lb.");
        break;
     default:
        System.out.println("Invalid choice.");
  }
}
}
```

# The switch Statement (Example 2)

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

    // Get a day from the user.
    System.out.print("Enter the name of a season: ");
    input = keyboard.nextLine();

    // Translate the season to Spanish.
    switch (input)
    {
       case "spring":
          System.out.println("la primavera");
          break;
       case "summer":
          System.out.println("el verano");
          break;
       case "autumn":
       case "fall":
          System.out.println("el otono");
          break;
       case "winter":
          System.out.println("el invierno");
          break;
       default:
          System.out.println("Please enter one of these words:\n"
                     + "spring, summer, autumn, fall, or winter.");
    }
}
```

# Relational Operator Examples

| Expression | Value | Comment |
|---|---|---|
| `3 <= 4` | `true` | 3 is less than 4; <= tests for "less than or equal". |
| `3 =< 4` | **Error** | The "less than or equal" operator is <=, not =<. The "less than" symbol comes first. |
| `3 > 4` | `false` | > is the opposite of <=. |
| `4 < 4` | `false` | The left-hand side must be strictly smaller than the right-hand side. |
| `4 <= 4` | `true` | Both sides are equal; <= tests for "less than or equal". |
| `3 == 5 - 2` | `true` | == tests for equality. |
| `3 != 5 - 1` | `true` | != tests for inequality. It is true that 3 is not 5 – 1. |
| `3 = 6 / 2` | **Error** | Use == to test for equality. |
| `1.0 / 3.0 == 0.333333333` | `false` | Although the values are very close to one another, they are not exactly equal. See Common Error 3.2 on page 93. |
| `"10" > 5` | **Error** | You cannot compare a string to a number. |
| `"Tomato".substring(0, 3).equals("Tom")` | `true` | Always use the `equals` method to check whether two strings have the same contents. |
| `"Tomato".substring(0, 3) == ("Tom")` | `false` | Never use == to compare strings; it only checks whether the strings are stored in the same location. See Common Error 3.3 on page 94. |

Thank you