Java Programming: Guided Learning with Early Objects
Chapter 4
Control Structures I: Selection

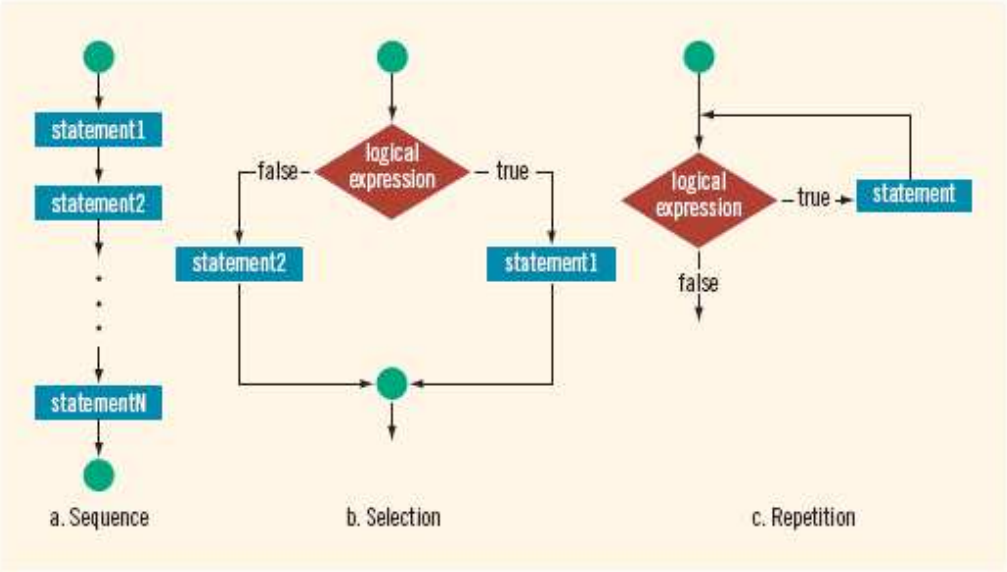
In this chapter, you will:

- Make decisions with the if and if…else structures
- Use compound statements in an if or if…else structure
- Nest if and if…else statements
- Use AND and OR operators
- Use the switch statement
- Understand precedence
- Use the conditional and NOT operators

Introduction :

Four ways a computer processes statements:
- Sequentially
- Selection
- Repetition
- Method calls

Branching: alter execution flow by selection
Looping: alter execution flow by repetition



Relational Operators

- Express conditions
- Make comparisons to make decisions
- **Logical** (**Boolean**) **expression**: has a value of either true or false
- **Relational operator**: allows comparisons in a program
- Relational operators are binary
- Binary operators require two operands
- Result of comparison is true or false

Operator	Description
==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

- Relational operators used with integral and floating-point data types

Expression	Meaning	Value
8 < 15	8 less than 15	True
6 != 6	6 not equal to 6	False
2.5 > 5.8	2.5 greater than 5.8	False
5.9 <= 7.5	5.9 less than or equal to 7.5	True

Comparing Floating-Point Numbers for Equality

Check absolute value of difference of two floating-point numbers

- Difference less than given tolerance
- Use method `abs` from `class Math`

Example: Math.abs(x – y) < 0.000001

Comparing Characters

- Expression using relational operators evaluates to true or false based on collating sequence
- Example: 'R' > 'T'

Expression	Value of the Expression	Explanation
`' ' < 'a'`	`true`	The Unicode value of `' '` is 32, and the Unicode value of `'a'` is 97. Because 32 < 97 is `true`, it follows that `' ' < 'a'` is `true`.
`'R' > 'T'`	`false`	The Unicode value of `'R'` is 82, and the Unicode value of `'T'` is 84. Because 82 > 84 is `false`, it follows that `'R' > 'T'` is `false`.
`'+' < '*'`	`false`	The Unicode value of `'+'` is 43, and the Unicode value of `'*'` is 42. Because 43 < 42 is `false`, it follows that `'+' < '*'` is `false`.
`'6' <= '>'`	`true`	The Unicode value of `'6'` is 54, and the Unicode value of `'>'` is 62. Because 54 <= 62 is `true`, it follows that `'6' <= '>'` is `true`.

Example :-

```
public class LogicalOperators
{
    public static void main(String[] args)
    {
        boolean found = true;
        boolean flag = false;
        double x = 5.2, num1=2567.58,num2=2567.58;
        double y = 3.4;
        int a = 5;
        int b = 8;
        int n = 20;
        char ch = 'B';

        System.out.println("!found evaluates to " + !found);
        System.out.println("x > 4.0 evaluates to " + (x > 4.0));
        System.out.println("!found && (x >= 0) evaluates to "
                           + (!found && (x >= 0)));
        System.out.println("!(found && (x >= 0)) evaluates to "
                           + !(found && (x >= 0)));
        System.out.println("x + y <= 20.5 evaluates to "
                           + (x + y <= 20.5));
        System.out.println("(n >= 0) && (n <= 100) evaluates to "
                           + ((n >= 0) && (n <= 100)));
        System.out.println("('A' <= ch && ch <= 'Z') evaluates to "
                           + ('A' <= ch && ch <= 'Z'));
        System.out.println("(a + 2 <= b) && !flag evaluates to "
                           + ((a + 2 <= b) && !flag));
        System.out.println("num1 = " + num1
                           + ", num2 = " + num2);

        System.out.println("The value of "
                           + "num1.equals(num2) is "
                           + num1.equals(num2));
        System.out.println("The value of "
                           + "num1 == num2 is "
                           + (num1 == num2));

    }    }
```


Comparing Strings

- Strings compared character by character
- Comparison continues until:
  - Mismatch found
  - Last characters compared and are equal
  - One string exhausted

- Shorter string less than larger string if comparison equal through shorter string

- Use method compareTo of class String

- Strings, the Assignment Operator, and the Operator new

Example 1:

```
String str1 = “Hello”;
String str2 = “Hello”;
(str1 == str2) evaluates to true
str1.equals(str2) evaluates to true
```

Example 2:

```
String str3 = new String(“Hello”);
String str4 = new String(“Hello”);
(str3 == str4) evaluates to false
str3.equals(str4) evaluates to true
```


Wrapper Classes (Revisited)

- Use method `compareTo` to compare values of two `Integer` objects
- Use method `equals` to compare values of two `Integer` objects for equality
- Relational operators compare values of Integer and Double objects
  - Using autoboxing and auto-unboxing
- Assignment operator always uses operator new to create Double object

Logical (Boolean) Operators and Logical Expressions

- **Logical** (**Boolean**) **operators** enable you to combine logical expressions
- Logical operators take logical values as operands
- Binary operators: && and ||
- Unary operator: !

Logical Operators :-

Logical Operator	Meaning	Evaluates
&&	AND	Both conditions must be true for the entire condition to be true
||	OR	Either condition or both conditions must be true for the entire condition to be true
!	NOT	Reverse the truth condition

Logical Operators Table (and &&)

Expression1	Expression2	Expression1 && Expression2
true	true	true
true	false	false
false	true	false
false	false	false

Logical Operators Table (or ||)

Expression1	Expression2	Expression1 || Expression2
true	true	true
true	false	true
false	true	true
false	false	false

Relational Operators :-

Relational Operator	Condition	Example
<	Less Than	(age < 25)
<=	Less Than or Equal	(age <= 25)
>	Greater Than	(age > 25)
>=	Greater Than or Equal	(age >= 25)
= =	Equal	(age = = 25)
!=	Not Equal	(! (age < 25))

Operator Precedence for Operators used

- Operations have higher and lower precedences
- Expression might contain arithmetic, relational, and logical operators
- Relational and logical operators evaluated left to right
  Left-to-right associativity
- The order in which you use operators makes a difference
- You can always use parentheses to change precedence or make your intentions clear

Precedence	Symbols
Highest	*** / %**
	+ -
	> >= < <=
	= = !=
	&&
	||
	? :
Lowest	**=**

Abbreviated version of if .. else

Short-Circuit Evaluation

- Logical expressions evaluated using efficient algorithm
- Short-circuit evaluation:
  - Logical expression evaluated left to right
  - Stops when value of entire expression known

boolean Data Type and Logical (Boolean) Expressions

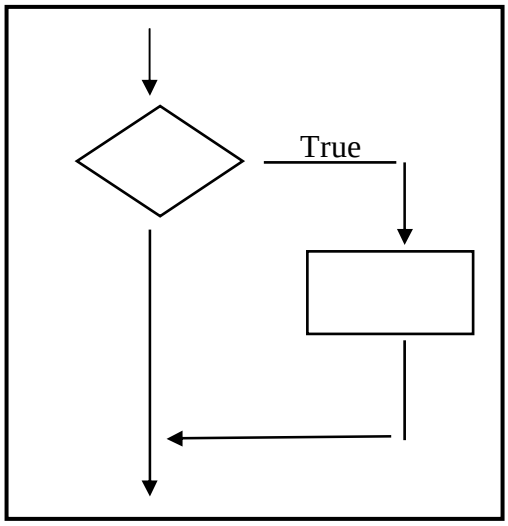
- boolean data type has values true and false
- Logical expressions manipulated using boolean data type
- boolean, true, false are reserved words

Example :-

```
public class StringComparison
{
    public static void main(String[] args)
    {
        String str1 = "Hello";
        String str2 = "Hi";
        String str3 = "Air";
        String str4 = "Bill";
        String str5 = "Bigger";

        System.out.println("Line 10: " +
                    "str1.compareTo(str2) evaluates to "
                   + str1.compareTo(str2));

        System.out.println("Line 11: " +
                    "str1.compareTo(\"Hen\") evaluates to "
                   + str1.compareTo("Hen"));

        System.out.println("Line 12: " +
                    "str4.compareTo(str3) evaluates to "
                   + str4.compareTo(str3));

        System.out.println("Line 13: " +
                    "str1.compareTo(\"hello\") evaluates to "
                   + str1.compareTo("hello"));

        System.out.println("Line 14: " +
                    "str2.compareTo(\"Hi\") evaluates to "
                   + str2.compareTo("Hi"));

        System.out.println("Line 15: " +
                    "str4.compareTo(\"Billy\") evaluates to "
                   + str4.compareTo("Billy"));

        System.out.println("Line 16: " +
                    "str5.compareTo(\"Big\") evaluates to "
                   + str5.compareTo("Big"));
    }    }
```


Making Decisions with the if and if…else Structures

- Making a decision involves choosing between alternate courses of action based on some value within a program
- The value the decision is based on is always Boolean-true or false
- You can use if or if…else statements to make a decision
    - ***Single alternative***- You only perform an action based on one alternative

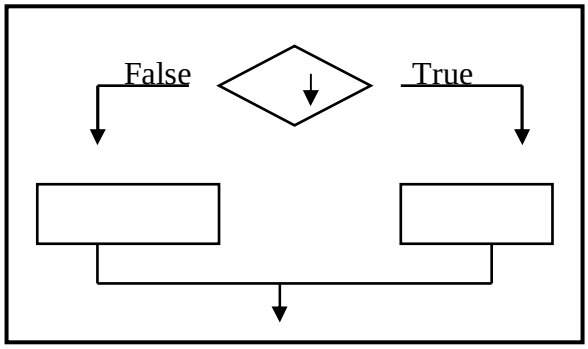


Example :-

```
                                                    import java.util.*;
public class onlyIf {
                                        static Scanner console = new Scanner(System.in);

   public static void main(String[] args)  throws Exception{
      char evType;
      String eventType ;
      System.out.println("This Example To illustrate IF  ...      statement");
      System.out.println("Enter Type of Event You Are Scheduling");
      System.out.println("Enter    C    For Coperate  Event ");
                                                          eventType = console.next();
;(0)charAt.eventType =evType
      if ( evType == 'C') {
         System.out.println("The  Manager Of This Event Will be  Carmen Daisy");
      }
       System.out.println("\n\nThis Part will always be printed");

  }  }
```

Dual alternative

- Requires two options for a course of action
- Provides the mechanism for performing one action when a Boolean expression evaluates as true and if it evaluates to false a different action occurs



Example

```
import java.util.*;
public class ifElse {
                                        static Scanner console = new Scanner(System.in);
   public static void main(String[] args)  throws Exception{
      char  evType;
      String eventType ;
      System.out.println("This Example To illustrate IF  ...   else  statement");
      System.out.println("Enter Type of Event You Are Scheduling");
      System.out.println("Enter    C    For Coperate  Event ");
      System.out.println("Enter    P    For Private Event");
      eventType = console.next();
     evType = eventType.charAt(0);

      System.out.print("The  Manager Of This Event Will be  ");

      if (evType == 'C') {
          System.out.println("Carmen Daisy ");
      } else {  System.out.println("Other than  Carmen Daisy");

      }
  }
```


Using Compound Statements in an if or if…else Structure

- To execute more than one statement that depends on the evaluation of a Boolean expression, use a pair of curly braces to place the dependent statements within a block

Example :-

```
import java.util.*;
public class ifElseCompound {
static Scanner console = new Scanner(System.in);
   public static void main(String[] args)  throws Exception{
      char evType;
      String eventType ;
      System.out.println("This Example TO illustrate COMPUND IF ..  else  statement");
      System.out.println("Enter Type of Event You Are Scheduling");
      System.out.println("Enter    C    For Coperate  Event ");
     eventType = console.next();
     evType = eventType.charAt(0);

      if (evType== 'C') {
         System.out.print("The  Manager Of This Event Will be  ");
         System.out.println("  Carmen Daisy ");
      } else
         if (evType == 'D')  {
            System.out.print("The  Manager Of This Event Will be  ");
            System.out.println("  Don Showla");
         }
      }
}
```


Nesting if and if…else statements

- Nesting if and if…else statements- Statements with an if inside another if
- Nested if statements are useful when two conditions must be met before some action can occur

Example :-

```
public class NestedIF {

   public static void main(String[] args) throws Exception {
     // int inTemp;
    int inTemp =   20;
     if ( inTemp > 32 )
     {
          if ( inTemp > 80 )
             {
                  System.out.println("It is very Hot");
              }   else
                    {
                         System.out.println("It is Moderate");
                    }
     }  else
     {
        System.out.println("It is Freezing");
     }
   }
}
```


Using the Switch Statement

- Switch statement- To test a single variable against a series of exact integer or character values
- The switch statement uses four keywords
- switch- starts the structure and is followed immediately by a test expression enclosed in parentheses
- case- is followed by one of the possible values for the test expression and a colon
- break- optionally terminates a switch structure at the end of each case
- default- optionally is used prior to any action that should occur if the test variable does not match any case

Example :-

```
public class caseStatement {

  public static void main (String[] args)
  {
    int num = 3;

    System.out.println("Integer Driven");
    switch(num)
    {
        case 1 : System.out.println("Number is 1"); break;
        case 2 : System.out.println("Number is 2"); break;
        case 3 : System.out.print("Number is 3 ,  It is the  ");
                  System.out.println("Third Sutition ");
                 break;
        default: System.out.println("Other number");
    }

    System.out.println("Character Driven");
    char ayear='f';
    switch (ayear){
       case 'F': System.out.println("Freshman"); break;
       case 'S': System.out.println("Sophomore");
                 System.out.println("Yes, the 2nd year.");
                 break;
       case 'T' : System.out.println("Junior"); break;
       case 'L': System.out.println("Senior"); break;
       default: System.out.println("Invalid year");
    }
    System.out.println("Program Ends");      }    }
```

What happens if one omits break ?????

Examples

```
If   ( tickets > 3   ||  age < 25    &&  gender = = 'M'   )
If   ( ( tickets > 3   ||  age < 25 )    &&  gender = = 'M'   )
```

Precedence	Symbols
Highest	*** / %**
	+ -
	> >= < <=
	= = !=
	&&
	||
	? :
Lowest	**=**

Abbreviated version of if .. else

Using the Conditional and NOT Operators

- Conditional operator- Requires three expressions separated with a question mark and a colon
  - Is used as an abbreviated version of the if…else structure
  - testExpression ? true Result : false Result

```
SmallNum  =  ( a < b )  ? a  :  b;
```


Using the Conditional and NOT Operators

- NOT operator- To negate the result of any Boolean expression
  - Written as the explanation point (!)

```
If ( ! ( age < 25 ) )  {  p = 125; }  else  { p = 200; }
```


Key Terms

- **action statement** – A statement that follows a condition in a selection statement.
- **boolean expression** – Another name for a logical expression.
- **boolean values** – The values `true` and `false`.
- **branch** – A selection or choice.
- **branching** – Altering the flow of program execution by making a selection or choice.
- **condition** – Another name for the logical expression in a selection statement.
- **conditional expression** – A statement written with the conditional operator.
- **conditional operator** – Another way to write an `if…else` statement; written as `? :`.
- **logical expression** – An expression that has a value of either `true` or `false`.
- **logical values** – The values `true` and `false`.
- **loop** – A structure that allows a program to execute a statement over and over.
- **looping** – Altering the flow of program execution by repetition of statements.
- **method calls** – Using a method inside of anther method.
- **pairing an `else` with an `if`** – In a nested `if` statement, Java associates an `else` with the most recent incomplete `if`.
- **postcondition** – A statement specifying what is true after the method call is completed.
- **precondition** – A statement specifying the conditions that must be true before the method is called.
- **relational operator** – An operator that allows you to make comparisons in a program.
- **repetition** – The program repeats particular statements a certain number of times, depending on one or more conditions.
- **sequential** – One after the other; one of the four ways a program can process statements.
- **short-circuit evaluation** – A process in which the computer evaluates a logical expression from left to right and stops as soon as the value of the expression is determined.
- **ternary operator** – An operator that takes three arguments.