

3. Flow of Control: Branching

[ITP20003] Java Programming

Agenda



- The if-else statement
- The type Boolean
- The switch statement

Flow of Control



- Flow of control is the order in which a program performs actions.
 - By default, the order has been sequential.
- A branching statement chooses between two or more possible actions.
- A loop statement repeats an action until a stopping condition occurs.
 - ➔ Next chapter

The if-else Statement

- A branching statement that chooses between two possible actions.

- Syntax

if (Boolean_Expression)

Statement_1

[else

// else part can be omitted

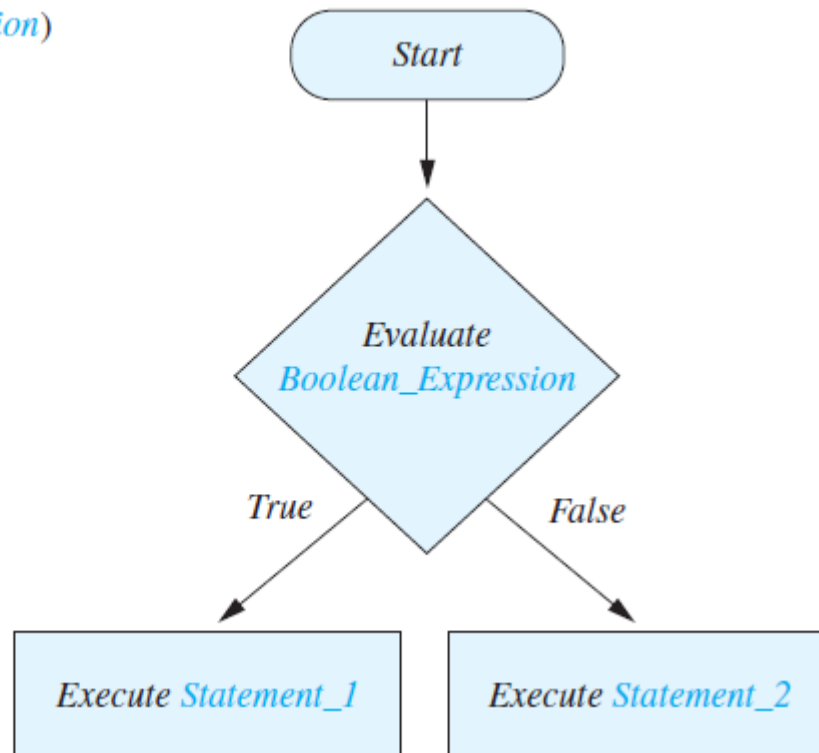
Statement_2]

Ex)

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

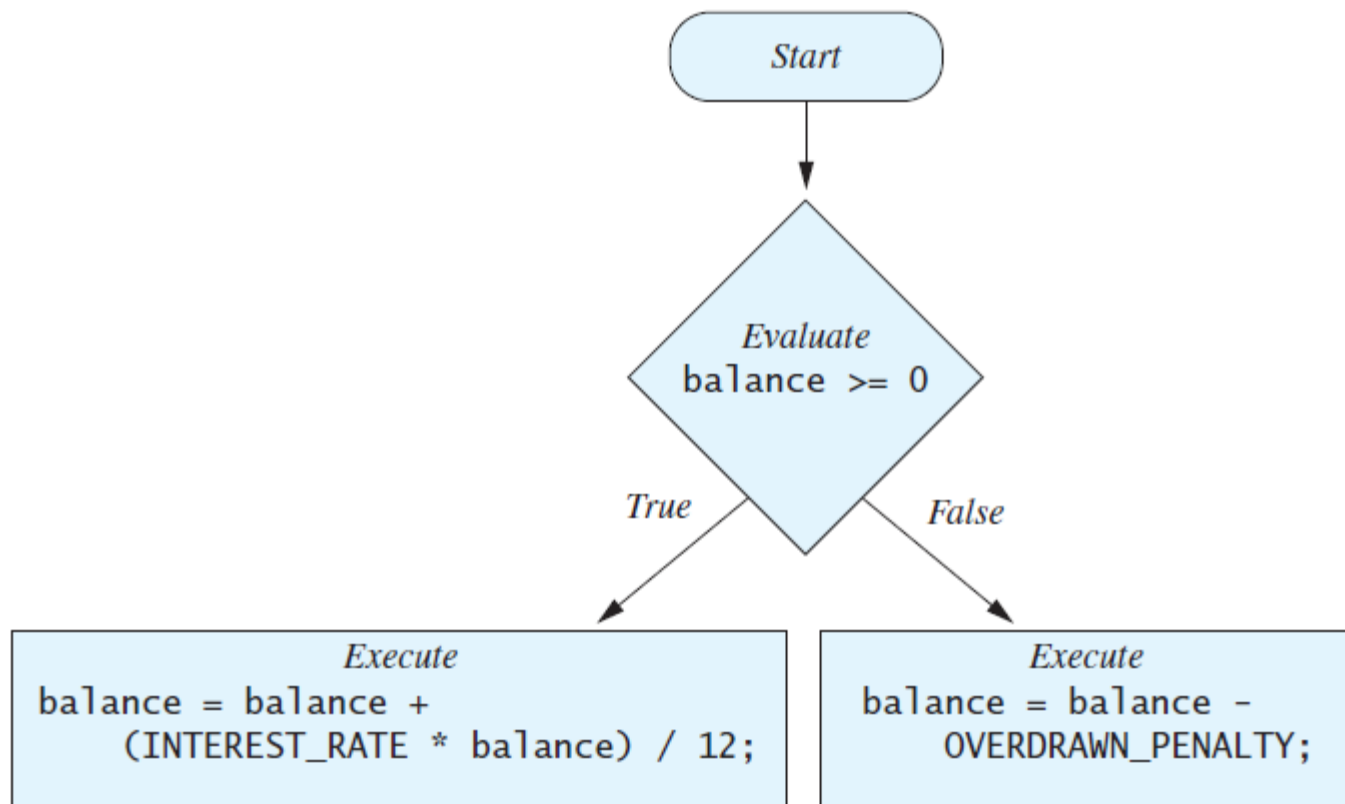
The if-else Statement

```
if (Boolean_Expression)  
    Statement_1  
else  
    Statement_2
```



The if-else Statement

Ex) Calculating balance of a bank account



BankBalance

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

The if-else Statement



Enter your checking account balance: \$505.67

Original balance \$505.67

After adjusting for one month of interest and penalties,
your new balance is \$506.51278

Enter your checking account balance: \$-15.53

Original balance \$-15.53

After adjusting for one month of interest and penalties,
your new balance is \$-23.53

Compound Statements



- To include multiple statements in a branch, enclose the statements in braces.

```
if (count < 3)
{
    total = 0;
    count = 0;
}
```

or

```
if (count < 3){
    total = 0;
    count = 0;
}
```

Boolean Expressions

- The value of a **boolean** expression is either **true** or **false**.

Examples

- `time < limit`
- `balance <= 0`
- Comparison operators

Math Notation	Name	Java Notation	Java Examples
=	Equal to	==	<code>balance == 0</code> <code>answer == 'y'</code>
≠	Not equal to	!=	<code>income != tax</code> <code>answer != 'y'</code>
>	Greater than	>	<code>expenses > income</code>
≥	Greater than or equal to	>=	<code>points >= 60</code>
<	Less than	<	<code>pressure < max</code>
≤	Less than or equal to	<=	<code>expenses <= income</code>

Boolean Operators

Name	Java Notation	Java Examples
Logical <i>and</i>	<code>&&</code>	<code>(sum > min) && (sum < max)</code>
Logical <i>or</i>	<code> </code>	<code>(answer == 'y') (answer == 'Y')</code>
Logical <i>not</i>	<code>!</code>	<code>!(number < 0)</code>

Value of <i>A</i>	Value of <i>B</i>	Value of <i>A && B</i>	Value of <i>A B</i>	Value of <i>! (A)</i>
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Compound Boolean Expressions



Examples)

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

...

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

...

- Wrong example

```
if (0 < score <= 100)
```


- Note! & and | are used for other purpose (bitwise and/or)

Exclusive OR Operator



- XOR (^)
 - Allows one or the other, but not both to be true.
 - $\text{true} \wedge \text{true} = \text{false}$
 - $\text{true} \wedge \text{false} = \text{true}$
 - $\text{false} \wedge \text{true} = \text{true}$
 - $\text{false} \wedge \text{false} = \text{false}$

Using == Operator



- == is appropriate for determining if two **integers** or **characters** have the same value.
 - if (a == 3)
 - where a is an integer type
- == is **not appropriate** for determining if two **floating points values** are equal. Use < and some appropriate tolerance instead.
 - if (abs(b - c) < epsilon)
 - where b, c, and epsilon are floating point types

Using == Operator

- == 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 to a **common memory location**.

Ex)

```
public class Test {  
    public static void main(String args[]){  
        Scanner keyboard = new Scanner(System.in);  
  
        String s1 = keyboard.nextLine(); // user types "Hello"  
        String s2 = keyboard.nextLine(); // user types "Hello"  
  
        System.out.println("s1 = " + s1);  
        System.out.println("s2 = " + s2);  
        System.out.println("Expression s1 == s2 is " + (s1==s2));  
    }  
}
```

s1 = Hello

s2 = Hello

Expression s1 == s2 is false

String Comparison



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

`String.equals(other_String)`

Ex)

`s1.equals(s2)`

or

`s2.equals(s1)`

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

`String.equalsIgnoreCase(Other_String)`

Ex) `"Hello".equalsIgnoreCase("hello")`

Testing Strings for Equality



```
import java.util.Scanner;
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 ();
        s2 = keyboard.nextLine ();
        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.");
    }
}
```

Testing Strings for Equality

■ Result

```
Enter two lines of text:  
Java is not coffee.  
Java is NOT COFFEE.  
The two lines are not equal.  
The two lines are not equal.  
But the lines are equal, ignoring case.
```

Comparing Strings



- Strings can be compared using method `compareTo`.
- `String.compareTo` method
 - Syntax
 - `String_1.compareTo(String_2)`
 - Returns
 - a negative number if `String_1` precedes `String_2`
 - zero if the two strings are equal
 - a positive number if `String_2` precedes `String_1`.

Comparing Strings



- `compareTo` can be combined with methods `toUpperCase` or `toLowerCase`.

Ex) `String1.toUpperCase().compareTo(String2.toUpperCase())`

Ex)

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

Nested Statements



- An if-else statement can contain any sort of statement within it.

```
if (Boolean_Expression_1)
    if (Boolean_Expression_2)
        Statement_1;
    else
        Statement_2;
else
    if (Boolean_Expression_3)
        Statement_3;
    else
        Statement_4;;
```

Nested Statements



- Each *else* is paired with **the nearest unmatched if**.
- If used properly, indentation communicates which *if* goes with which *else*.
- Braces can be used like parentheses to group statements.

Nested Statements

- Subtly different forms

First Form

```
if (a > b)
{
    if (c > d)
        e = f;
}
else
    g = h;
```

Second Form

```
if (a > b)
    if (c > d)
        e = f;
    else
        g = h;

// oops
```

Compound Statements



- When a list of statements is enclosed in braces ({}), they form a single **compound statement**.
- A compound statement can be used wherever a statement can be used.

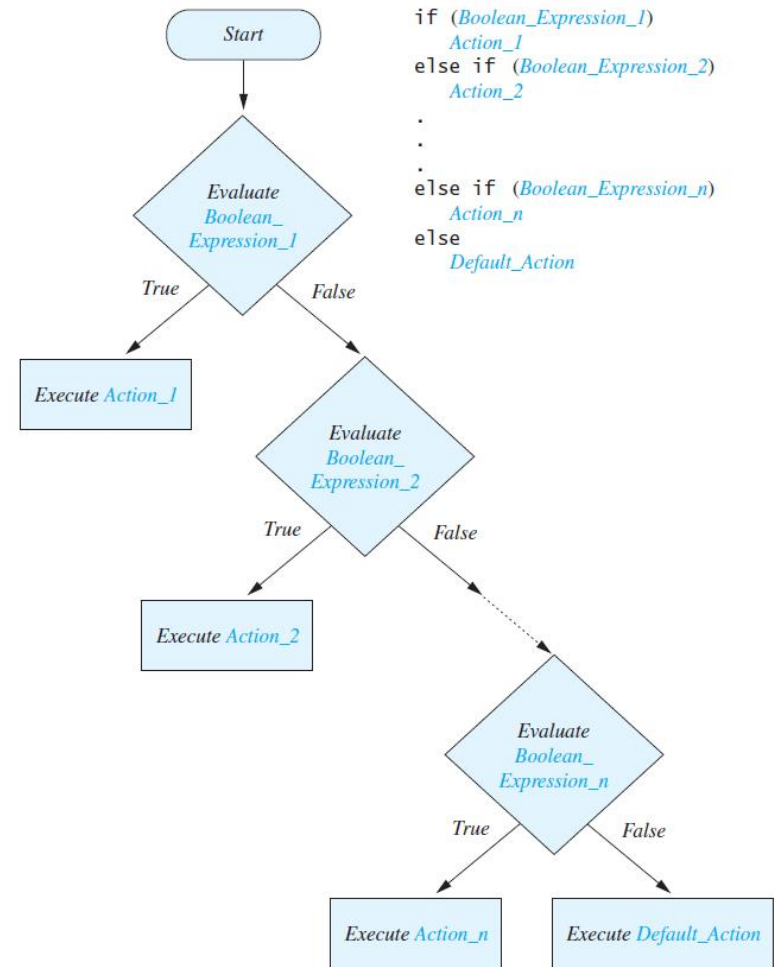
Ex)

```
if (total > 10)
{
    sum = sum + total;
    total = 0;
}
```


Multibranch if-else Statements

■ Syntax

```
if (Boolean_Expression_1)
    Statement_1
else if (Boolean_Expression_2)
    Statement_2
else if (Boolean_Expression_3)
    Statement_3
else if ...
    ...
else
    Default_Statement
```



Multibranch if-else Statements

```
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);
    }
}
```

Multibranch if-else Statements



■ Result

Enter your score:

85

Score = 85

Grade = B

Multibranch if-else Statements



■ Equivalent code

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

The Conditional Operator



```
if (n1 > n2)
    max = n1;
else
    max = n2;
```

can be written as

```
max = (n1 > n2) ? n1 : n2;
```

- The ? and : together are called the conditional operator (ternary operator)
- The conditional operator is useful with print and println statements.
System.out.print("You worked " + ((hours > 1) ? "hours" ; "hour"));

The *exit* Method



- Sometimes a situation arises that makes continuing the program pointless.
- A program can be terminated normally by `System.exit(0)`.

```
if (numberOfWinners == 0) {  
    System.out.println("Error: Dividing by zero.");  
    System.exit(0);  
} else {  
    oneShare = payoff / numberOfWinners;  
    System.out.println("Each winner will receive $" + oneShare);  
}
```

The Type boolean



- The type `boolean` is a primitive type with only two values: `true` and `false`.
- Boolean variables can make programs `more readable`.
if (systemsAreOK)

instead of

```
if((temperature <= 100) && (thrust >= 12000)
    && (cabinPressure > 30) && ...)
```

Boolean Expressions and Variables



- Variables, constants, and expressions of type **boolean** all evaluate to either **true** or **false**.
- A **boolean** variable can be given the value of a **boolean** expression by using an assignment operator.
 boolean isPositive = (number > 0);
 ...
 if (isPositive) ...
- Naming **boolean** variable
 - Good examples - **isPositive** or **systemsAreOk**
 - Bad examples - **numberSign** or **systemStatus**

Precedence Rules



- Parentheses should be used to indicate the order of operations.
- When parentheses are omitted, the order of operation is determined by **precedence rules**.
 - Operations with higher precedence are performed before operations with lower precedence.
- Operations with equal precedence are done left-to-right (except for unary operations which are done right-to-left).

Precedence Rules

Highest Precedence

First: the unary operators `+`, `-`, `++`, `--`, and `!`

Second: the binary arithmetic operators `*`, `/`, `%`

Third: the binary arithmetic operators `+`, `-`

Fourth: the boolean operators `<`, `>`, `<=`, `>=`

Fifth: the boolean operators `==`, `!=`

Sixth: the boolean operator `&`

Seventh: the boolean operator `|`

Eighth: the boolean operator `&&`

Ninth: the boolean operator `||`

Lowest Precedence

Precedence Rules



- In what order are the operations performed?

$\text{score} < \text{min}/2 - 10 \parallel \text{score} > 90$

Short-circuit Evaluation



- Sometimes only part of a boolean expression needs to be evaluated to determine the value of the entire expression.
 - If the first operand associated with an `||` is true, the expression is true.
 - If the first operand associated with an `&&` is false, the expression is false.
- This is called **short-circuit** or **lazy evaluation**.

Short-circuit Evaluation



- Short-circuit evaluation is not only efficient, sometimes it is essential!
 - A run-time error can result, for example, from an attempt to divide by zero.
 - `if((number != 0) && (sum/number > 5))`
- Complete evaluation can be achieved by substituting `&` for `&&` or `|` for `||`.

Input and Output of Boolean Values



Ex)

```
boolean booleanVar = false;  
System.out.println(booleanVar);  
System.out.println("Enter a boolean value:");  
Scanner keyboard = new Scanner(System.in);  
booleanVar = keyboard.nextBoolean();  
System.out.println("You entered " + booleanVar);
```

■ Result

false

Enter a boolean value: *true* // the user typed *true*

You entered true

The *switch* Statement

- The switch statement is a **multiway branch** that makes a decision based on an integral (integer or character) expression.
 - Java 7 allows String expressions

```
switch(Controlling_Expression)
{
    case Case_Label:
        Statements;
        break;
    case Case_Label:
        ...
    default:
        ...
}
```

The *switch* Statement



- The list is searched for a case label matching the controlling expression.
 - The action associated with a matching case label is executed.
 - The action for each case typically ends with the word **break**.
- If no match is found, the case labeled **default** is executed.
 - The default case is optional, but recommended, even if it simply prints a message.
- Repeated case labels are not allowed.

The switch Statement

- View [sample program](#) Listing 3.5
class MultipleBirths

```
Enter number of babies: 1  
Congratulations.
```

```
Enter number of babies: 3  
Wow. Triplets.
```

```
Enter number of babies: 4  
Unbelievable; 4 babies.
```

```
Enter number of babies: 6  
I don't believe you.
```

Enumerations



- Consider a need to restrict contents of a variable to certain values

Ex) days in a week, HGU building names, ...

- An enumeration lists the values a variable can have

Ex)

```
enum MovieRating { E, A, B }
```

```
MovieRating rating;
```

```
rating = MovieRating.A;
```

- Internally, enumeration values are treated as integers.
 - ➔ Can be used for controlling expression of switch statement

Enumerations

- Now possible to use in a switch statement

```
switch (rating)
{
    case E: //Excellent
        System.out.println("You must see this movie!");
        break;
    case A: //Average
        System.out.println("This movie is OK, but not great.");
        break;
    case B: // Bad
        System.out.println("Skip it!");
        break;
    default:
        System.out.println("Something is wrong.");
}
```

Reading enum with Scanner Object

```
import java.util.Scanner;

public class Main {
    enum Days { SUN, MON, TUE, WED, THU, FRI, SAT };

    public static void main(String args[]){
        Days a;
        System.out.println("What's your favorite day? ");
        Scanner keyboard = new Scanner(System.in);
        a = Days.valueOf(keyboard.next());

        System.out.println("Your favorite day is " + a);
    }
}
```