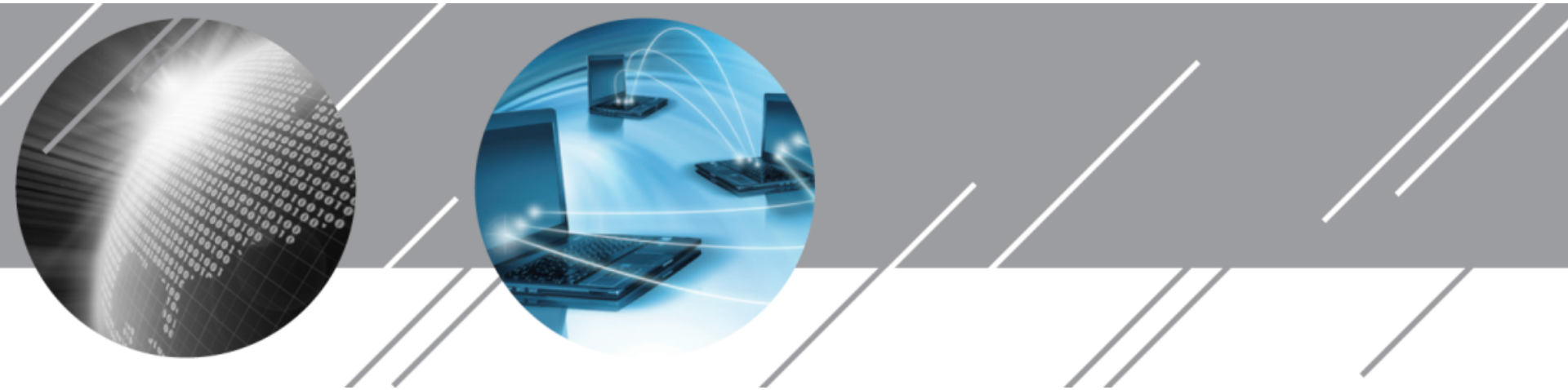
Object Oriented Programming

Introduction to Java

Ch. 3. Flow of Control : Branching

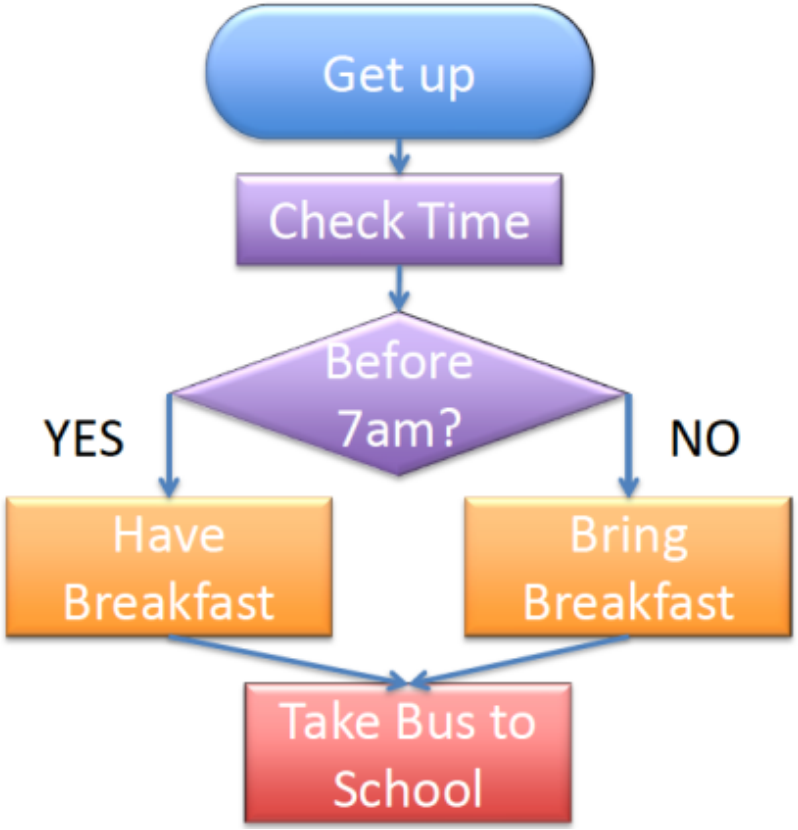


Dept. of Software, Gachon University
Ahyoung Choi, Spring

Flow of Control terminology

- *Flow of control*
  - The order in which a program performs actions.
  - The order has been sequential.
- *Branching*
  - Choose between two or more possible actions.
- *Loop*
  - Repeat an action until a stopping condition occurs.

Flow Chart




```
Student.getUp();
if (time < 7) {
    Student.haveBreakfast();
}
else { // time >= 7
    Student.bringBreakfast();
}
Student.takeBus();
```

if-else Statement

- A branching statement that chooses between two possible actions.
- Syntax

  if (*Boolean_Expression*)

  Statements_1

  else

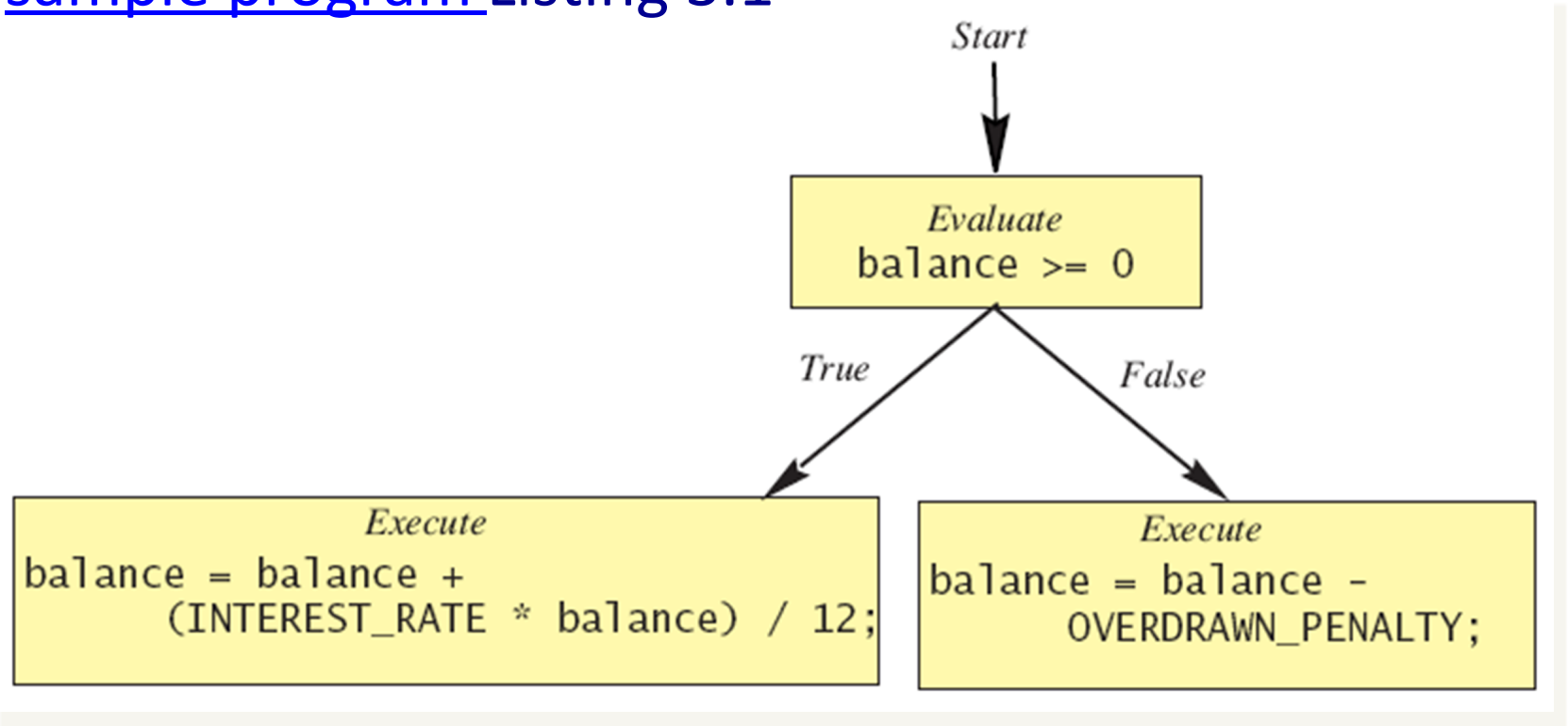
  Statements_2

- Example

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

The `if-else` Statement

- Figure 3.1 The Action of the `if-else` Statement
  sample program Listing 3.1



Lab: 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
```

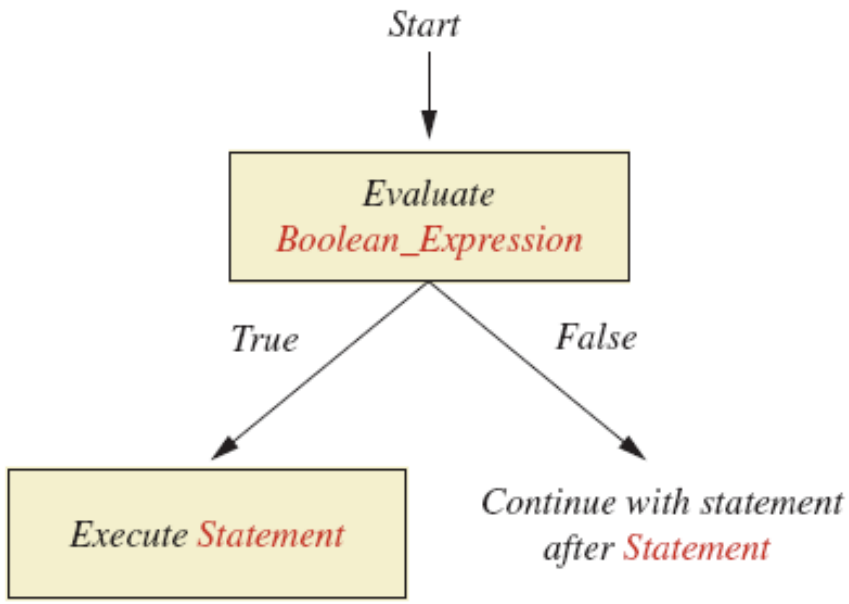
Lab: A Program Using `if-else`

```
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);
    }
}
```

Omitting the `else` Part

- FIGURE 3.3 The Semantics of an `if` Statement without an `else`

```
if (Boolean_Expression)
    Statement
```



Compound statements

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

```
if (balance >= 0)
{
   System.out.println("Good for you. You earned interest.");
   balance = balance + (INTEREST_RATE * balance) / 12;
}
else
{
   System.out.println("You will be charged a penalty.");
   balance = balance - OVERDRAWN_PENALTY;
}
```

Comparison operators

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

Boolean Expressions

- Expression?
  - An expression can be a variable, a value, or a combination made up by variables, values and operators
  - Arithmetic expression: a combination of numbers with a number value
    - 10, taxRate/100, (cost + tax) * discount
  - String expression: a combination of Strings with a String value
    - “Hello”, “The total cost is “ + totalCost

Boolean expression

- Boolean expression
  - An expression whose value is either *true* or *false*
  - Examples:
    - 5 == 3; // false
    - balance <= 0 // depending on the value of balance
- Compound Boolean expressions (logical operators)

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

Boolean expression && : and

- Boolean expressions can be combined using the "and" (&&) operator.
- Not allowed
  - if (0 < score <= 100)
- Example
  - if ((score > 0) && (score <= 100))

Boolean expression || : or

- What if you need ONE expression to be true out of many expressions
- Boolean expressions can be combined using the "or" (||) operator.
- Example
  - if ((quantity > 5) || (cost < 10))

Negating a Boolean Expression ! : not

- A boolean expression can be negated using the "not" (!) operator.
  - Syntax: !(Boolean_Expression)
  - Will be true if the expression is false
- NOTE: for some cases, use of ! is not recommended
  - You will get confused; try to write expresses straightforward
  - Example
    - (a || b) && !(a && b) -> ??
    - Use (cost != 3) instead of !(cost == 3)

! (*A Op B*) Is Equivalent to	(*A Op B*)
<	>=
<=	>
>	<=
>=	<
==	!=
!=	==

Effect of Boolean operators

Value of *A*	Value of *B*	Value of *A* && *B*	Value of *A* || *B*	Value of ! (*A*)
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Precedence rule

- Poor style: score < min / 2 – 10 || score > 90
- Rather write: (score < (min / 2 – 10)) || (score > 90)

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?

```
score < min/2 - 10 || score > 90
score < (min/2) - 10 || score > 90
score < ((min/2) - 10) || score > 90
(score < ((min/2) - 10)) || score > 90
(score < ((min/2) - 10)) || (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.
- A run-time error can result, for example, from an attempt to divide by zero.

```
if ((number != 0) && (sum/number > 5))
```

Using ==

- == is appropriate for determining if two integers or characters have the same value.

```
if (a == 3)
```

where **a** is an integer type

- Comparison operators connect values or variables
  - After connection, it's a boolean expression
  - *a > b*
  - *c == d*

Assignment (=) vs. Equal To (==)

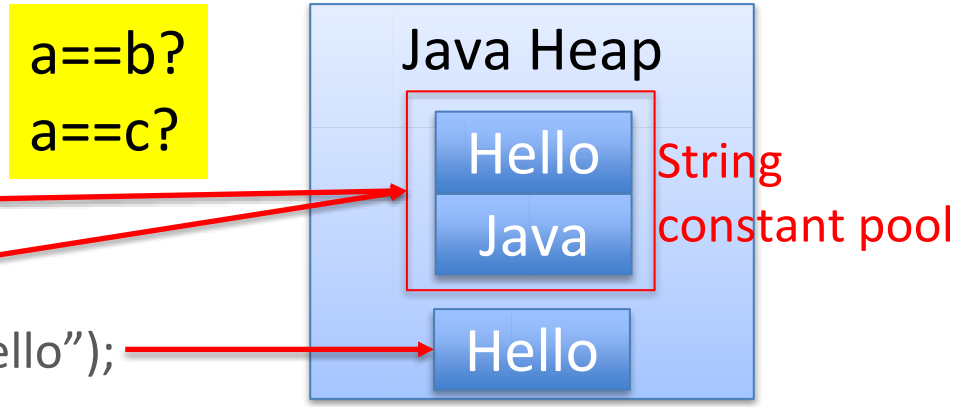
- *if (n1 = n2)* **vs.** *if (n1 == n2)*

- *if (n1 = n2)*
  - ***Error (if your purpose is to compare) !!***
    It's an assignment statement !
- *if (n1 == n2)*
  - *Correct!. It's a boolean expression now.*

Using "==" between two Strings

For two string objects `s1` and `s2`,
what does it mean `s1 == s2` ??

- `==` 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 the a common memory location.
  - E.g.
    - String a = "Hello";
    - String b= "Hello";
    - String c = new String("Hello");



Equality of Strings (objects)

- To test the equality of String values, use:
  s1.equals(s2) *or* s2.equals(s1)
- To test the equality ignoring the case, use:
  "Hello".equalsIgnoreCase("hello")

Lab: String memory test

```
package stringCompare;
public class stringCompare {
      public static void main(String[] args) {
      String a, b, c, d;
      a = "Hello";
      b = "hello";
      c = "hello";
      d = new String("Hello");

      if(a==b)System.out.println("true");
      else  System.out.println("false");

      if(a==c) System.out.println("true");
      elseSystem.out.println("false");

      if(b==c)System.out.println("true");
      else System.out.println("false");

      if(a==d) System.out.println("true");
      elseSystem.out.println("false");

      System.out.println();

      if(a.equals(d)) System.out.println("true");
      elseSystem.out.println("false");

      if(b.equals(d)) System.out.println("true");
      elseSystem.out.println("false");

      if(b.equalsIgnoreCase(d)) System.out.println("true");
      elseSystem.out.println("false");
      }
};
```

Result?

---

false
false
true
false

true
false
true

Comparing Strings

- Lexicographic order
  - Similar to *alphabetical* order; but it is based on the order of the characters in the ASCII (and Unicode) character set
    - All the digits come before all the letters
    - All the uppercase letters come before all the lower case letters
- Comparing Strings
  - string_1.compareTo(string_2) returns:
    - Negative value, if string_1 precedes string_2
    - Zero, if two strings are equal
    - Positive value, if string_1 is preceded by string_2

TIP: checking Alphabetic Order

- To see whether two Strings of letters are in alphabetic order, you must ensure that all the letters have the same case before using the method compareTo to compare the strings

```
String upperS1 = s1.toUpperCase();
String upperS2 = s2.toUpperCase();
if (upperS1.compareTo(upperS2) < 0)
   System.out.println(s1 + " precedes " + s2 "+"
                      "in ALPHABETIC ordering");
else if (upperS1.compareTo(upperS2) > 0)
   System.out.println(s1 + " follows " + s2 "+"
                      "in ALPHABETIC ordering");
else//s1.compareTo(s2) == 0
   System.out.println(s1 + " equals " + s2 + "
                      ignoring case");
```

Nested if-else statement

- An if-else statement can contain any statement(s) within it
- Syntax:

```
if (Boolean_Expression_1) {
    if (Boolean_Expression_2)
        Statements_1;
    else
        Statements_2;
}
else {
    if (Boolean_Expression_3)
        Statements_3;
    else
        Statements_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.

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;
```

Same?
Different?

Question

- Write a code doing the same task without “else”statement

```
if (time < 6){
   cook hams and scramble eggs;
}
else{
   if (time < 7){
      grab something from the fridge;
   }
   else{
      go to school;
   }
}
```

BE CAREFUL!!: Using If and Else

- Use if-else statement
  instead of two if-statements

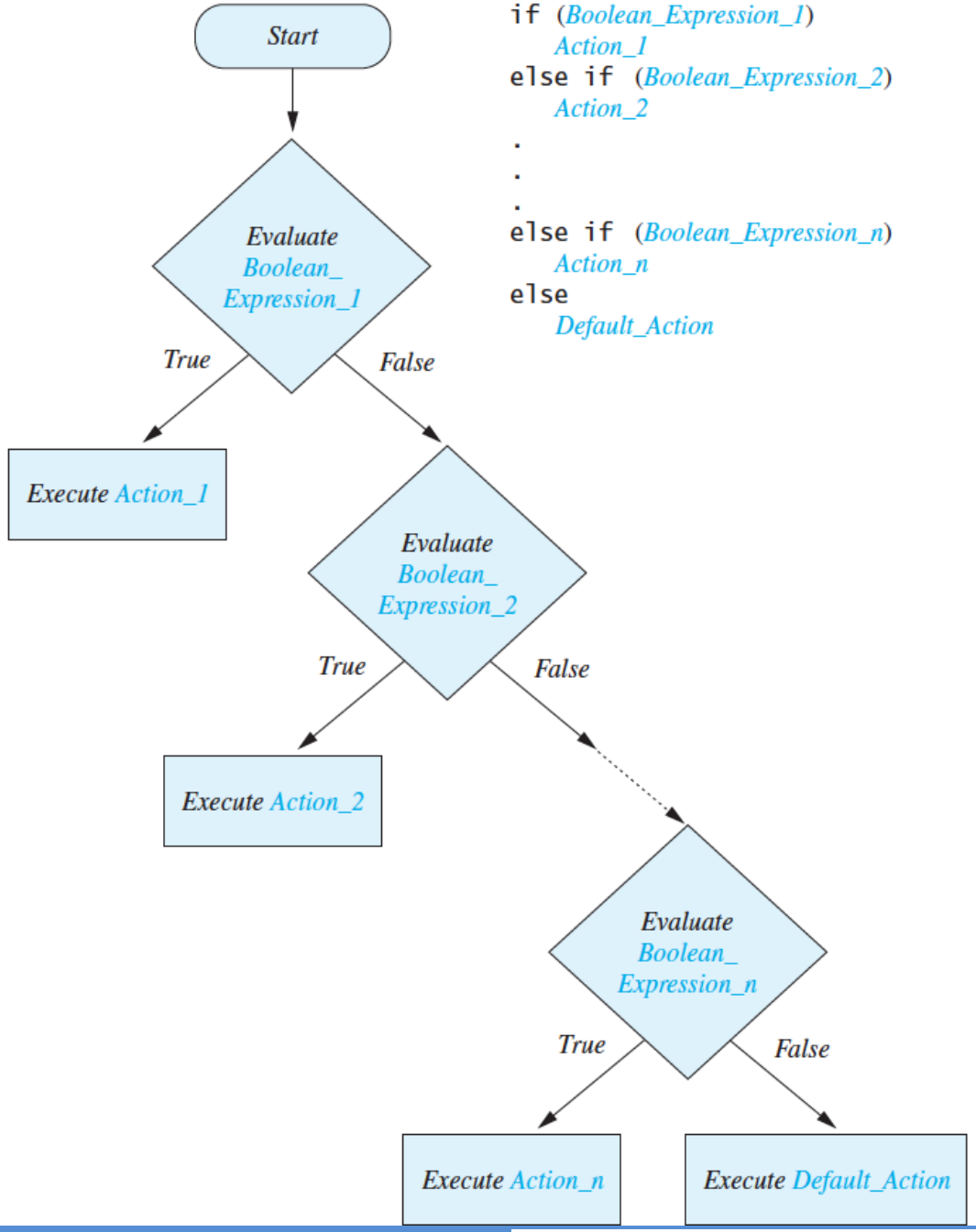
- Always pay attention to boundaries
  - Is it “>” or “>=” ?
  - Is it “<” or “<=” ?
  - Do you need a “==” ?

Conditional operator (?:)

- Conditional operator (?:)
  - if (n1 > n2) max = n1;
    else max = n2;
  - can be written as:
  - max = (n1 > n2) ? n1 : n2;
- Example:
  - System.out.println("You worked " + hours + ((hours > 1) ? " hours" ; " hour"));

Multibranch `if-else` Statements

- Figure 3.8 Semantics




```
if (Boolean_Expression_1)
    Action_1
else if (Boolean_Expression_2)
    Action_2
.
.
.
else if (Boolean_Expression_n)
    Action_n
else
    Default_Action
```

Multi-branch if-else statement

```
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);
    }
}
```

switch Statement

```java
if (year == 1)
     System.out.println("freshman");
else if (year == 2)
     System.out.println("sophomore");
else if (year == 3)
     System.out.println("junior");
else if (year == 4)
     System.out.println("senior");
else if (year == 5)
     System.out.println("super
senior");
else
     System.out.println("huh?");
```

```java
switch (year) {
  case 1:
    System.out.println("freshman");
    break;
  case 2:
    System.out.println("sophomore");
    break;
  case 3:
    System.out.println("junior");
    break;
  case 4:
    System.out.println("senior");
    break;
  case 5:
    System.out.println("super senior");
    break;
  default:
    System.out.println("unknown");
  break;
}
```

switch Statement

- A *multiway branch* based on an *integral* expression
  - Controlling expression return only int, short, byte, or char
- Each case consists of the keyword case followed by
  - A constant called the *case label*
  - A colon (:)
  - A list of ≥0 statements
  - break statement

```
switch (eggGrade) {
  case 'A':
  case 'a':
    System.out.println("Grade A");
    break;
  case 'C':
  case 'c':
    System.out.println("Grade C");
    break;
  default:
    System.out.println("We only buy
grade A and grade C.");
    break;
}
```

`switch` Statement

- A switch works with
  - Primitive data type: byte, short, char, and int
  - Enumerated types
  - Special classes that "wrap" certain primitive types: String, Character, Byte, Short, and Integer

exit() method

- 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);
}
```

Type `boolean`

- The type `boolean` is a primitive type with only two values: `true` and `false`.

- Naming Boolean variables
  - Choose names such as *isPositive* or *systemsAreOk*
  - Avoid names such as *numberSign* or *systemStatus*

Input and Output of Boolean Values

- Example

```
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);
```

Enumerations

- Consider a need to restrict contents of a variable to certain values
  - An enumeration lists the values a variable can have
  - Define in top level class, no need semicolon at the end

```java
package gradeTest;
import java.util.Scanner;
public class gradeTest {

    1 reference | 2 references
    enum Grade {Good, Gad, Soso}
    2 references
    static String name;
    2 references
    static Grade gr;

    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);

        name = keyboard.nextLine();
        gr = Grade.Good;

        System.out.println("Student name is "+ name);
        System.out.println("Grade is " + gr);

        keyboard.close();
    }
}
```

Problems | Declaration | Search | Console | Debug

```
<terminated> gradeTest [Java Application] C:₩Program Files₩Java₩jre1.8.0_19
Choi
Student name is Choi
Grade is Good
```

Enumerations

- To use in a **switch** statement

```
enum MovieRating {E, A, B}
MovieRating rating;
rating = MovieRating.A;
```

```
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.");
} // end switch
```

Practice 3.1

- Ex3_1a. Write a following program
  - Change the following if-else statement to switch
- Ex3_1b. Update 3_1a
  - Assign enum type case label instead of String

Define enum type here
`enum Rating {A, B, C, D, F}`

Define enum type here
`Rating rate;`

```
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);
    }
}
```

Change here

Practice 3.2

- Ex3_2. Write a program with a *switch* statement
  - Convert a letter grade to an equivalent numeric value
  - For A, B, C, and D, gradeValue is 4.0, 3.0, 2.0, and 1.0, respectively
  - For F and any other letter, gradeValue = 0.0
  - Use keyboard.next().charAt(0) to get a *char* value

Practice 3.3

- Ex3_3a. Write a following program
  - Print cashing charge depending on check(수표) amount
  - If < $10, charge = $1
  - Else if < $100, charge = 10 percent
  - Else if < $1000, charge = $5 + 5 percent
  - Else (≥ $1000) charge = $40 + 1 percent
- Ex3_3b. Extend Ex3_3a
  - Include input checking
  - Display an error message, if check amount< 1$, check amount not an integer multiple of 5

Assignment

- Read chapter 4.1~4.2