

SS-1201 Lab 02 (Part 2/3): Selection Statements

Objectives:

1. Learn to use relational and equality operators
2. Learn to use **boolean** operators ! (not), && (logical AND), and || (logical OR)
3. Learn to use if and switch statements:

Completed?	Sub-topics
Yes / No	Two-way selection (if-else statement)
Yes / No	One-way selection (if-statement)
Yes / No	Multi-way selection with else option (if - else if – else)
Yes / No	Multi-way selection with no else option (if – else if – else if)
Yes / No	Nested if statements
Yes / No	switch statement with default option
Yes/ No	switch statement with no default option

4. Void Methods
5. Complete practice and increase your understanding using examples
6. Complete all Lab tasks/exercises

1. Relational and Equality Operators

In Java, conditions are **boolean** expressions that evaluate to either **true** or **false**. The following relational and equality operators are used to form simple conditions:

Operator	Meaning
==	equals
!=	Not equal
>	Greater than
>=	Greater or equal
<	less
<=	Less or equal

Examples:

x >= y

```
2*a*b != k * m
```

```
Math.pow(b, 2) - 4*a*c > 0
```

The **boolean** operators **&&** (and) and **||** (or) are used to form complex conditions:

```
x >= y && y <= z
```

```
grade < 0.0 || grade > 100.0
```

```
ch1 >= 'a' && ch1 <= 'z' || ch1 >= 'A' && ch1 <= 'Z'
```

1.1 Operator Precedence

In Java, mathematical and boolean expressions are evaluated according to the following precedence and associativity rules:

	Operators	Order of Evaluation of operands with same precedence (Associativity)
<div>Higher Priority</div> <div>↓</div> <div>Low Priority</div>	1 <i>(expression)</i> and method calls	Left to right
	2 var++ var--	Right to left
	3 !, unary +, unary −, ++var, --var, (type)	Right to left
	4 *, /, %	Left to right
	5 Binary +, binary −, String concatenation +	Left to right
	6 <, <=, >=, >	Left to right
	7 ==, !=	Left to right
	8 &&	Left to right
	9	Left to right
	10 =, *=, /=, %=, +=, -=	Right to left

Note: Parentheses are also used to group sub-expressions to force a different precedence; such parenthetical expressions can be nested and are evaluated from inner to outer.

2. Truth Tables

Let P and Q be **boolean** expressions

Truth table for the ! operator:

P	! P
true	false
false	true

Truth table for the **&&** operator:

P	Q	P && Q
true	true	true
true	false	false
false	true	false
false	false	false

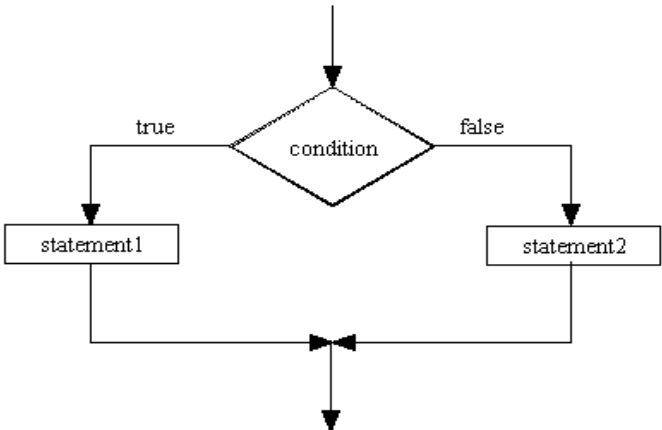
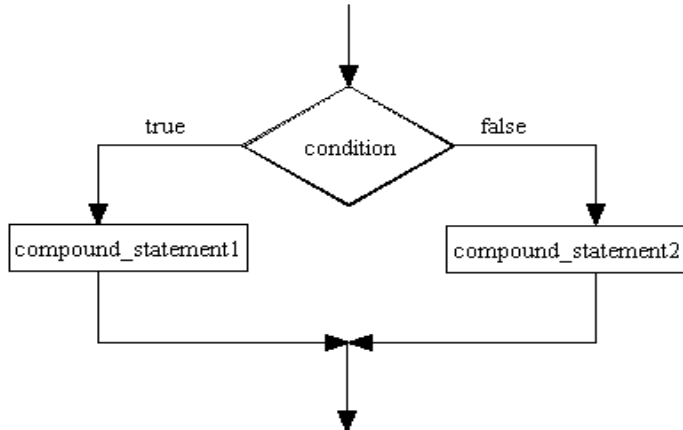
Truth table for the **| |** operator:

P	Q	P Q
true	true	true
true	false	true
false	true	true
false	false	false

3. Two-way selection (if-else statement)

An *if-else* statement is used to execute a statement or a compound-statement when a condition is **true**; and another statement or compound-statement when that condition is **false**.

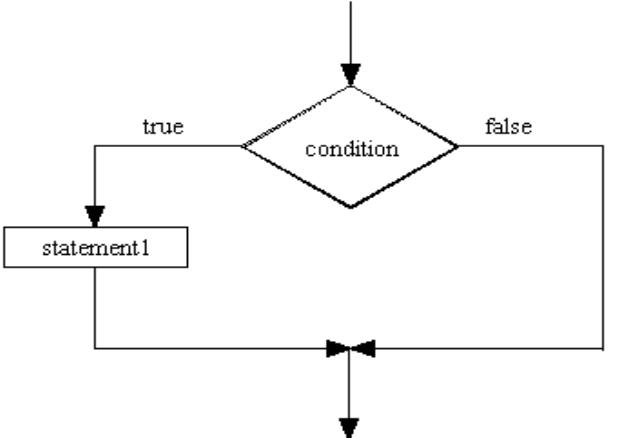
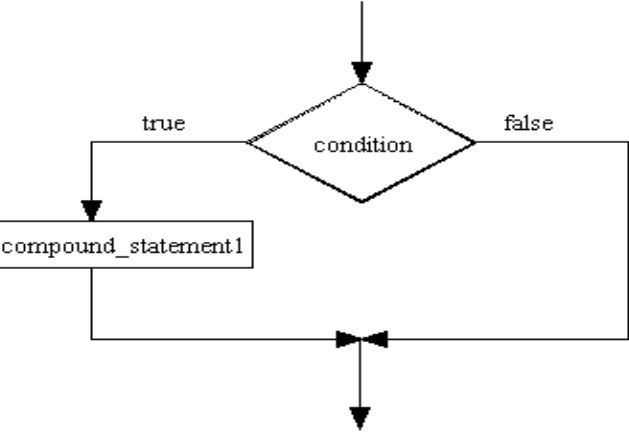
Single statements	Compound statements
<pre>if (condition) statement1; else statement2;</pre>	<pre>if (condition) compound_statement1 else compound_statement2</pre>
Execute <i>statement1</i> if <i>condition</i> is <i>true</i> else execute <i>statement2</i>	Execute <i>compound_statement1</i> if <i>condition</i> is <i>true</i> else execute <i>compound_statement2</i> Note: It is a syntax error to put a semicolon after the closing brace of <i>compound_statement1</i> A compound statement may contain zero or more statements

 <pre> graph TD Entry(()) --> Condition{condition} Condition -- true --> Statement1[statement1] Condition -- false --> Statement2[statement2] Statement1 --> Merge(()) Statement2 --> Merge Merge --> Exit(()) </pre>	 <pre> graph TD Entry(()) --> Condition{condition} Condition -- true --> CompoundStatement1[compound_statement1] Condition -- false --> CompoundStatement2[compound_statement2] CompoundStatement1 --> Merge(()) CompoundStatement2 --> Merge Merge --> Exit(()) </pre>
<p>Example: Finding max of two numbers</p> <pre> if(x >= y) max = x; else max = y; </pre>	<p>Example:</p> <pre> char currencyType = scanner.nextLine().charAt(0); if(currencyType == 's' currencyType == 'S'){ System.out.println("Enter positive amount "); amount = scanner.nextDouble(); riyalBalance = riyalBalance + amount; } else System.out.println("Wrong currency type"); </pre>

4. One-way selection (if-statement)

Used to execute a *statement* or a *compound_statement* when a condition is **true**.

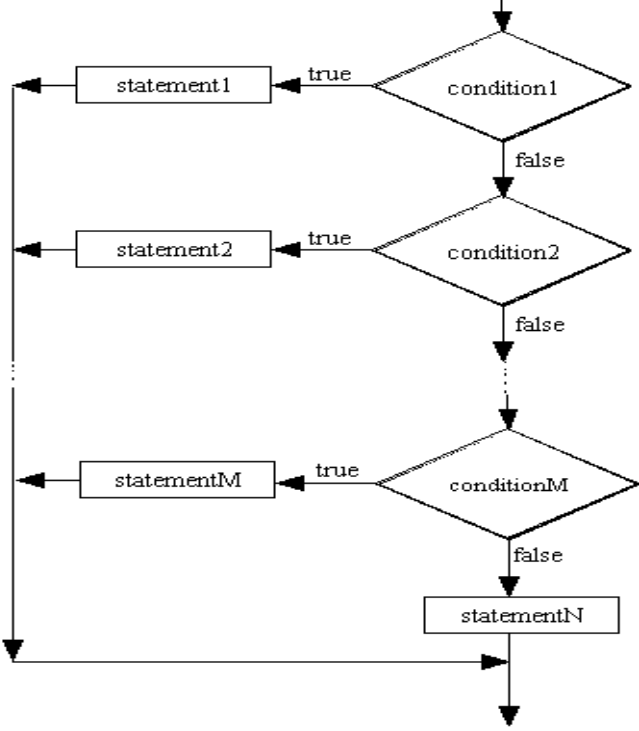
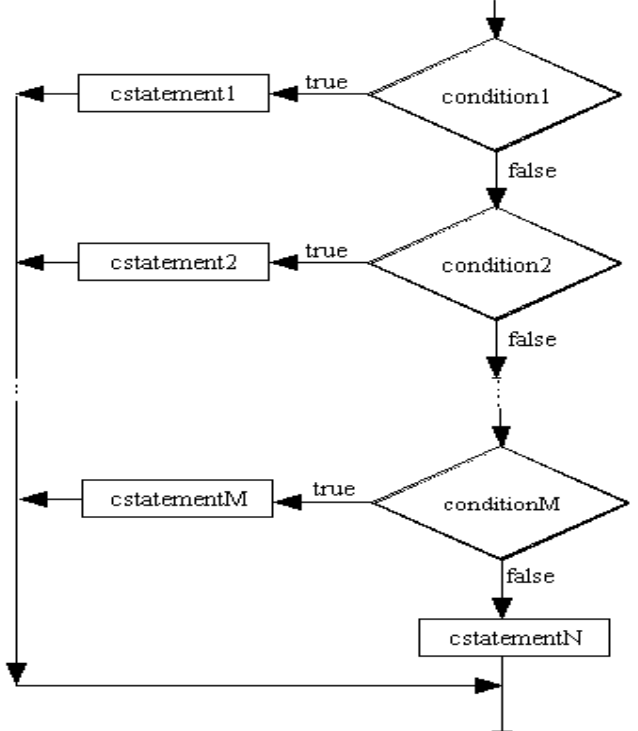
Single statements	Compound statements
<pre> if(condition) statement1; </pre>	<pre> if(condition) compound statement1 </pre>
Execute statement1 if condition is true	Execute compound_statement1 if condition is true

 <pre> graph TD Entry(()) --> Condition{condition} Condition -- true --> Statement1[statement1] Condition -- false --> Merge(()) Statement1 --> Merge Merge --> Exit(()) </pre>	 <pre> graph TD Entry(()) --> Condition{condition} Condition -- true --> CompoundStatement1[compound_statement1] Condition -- false --> Merge(()) CompoundStatement1 --> Merge Merge --> Exit(()) </pre>
<p>Example:</p> <pre> System.out.println("Enter a number: "); double num = scanner.nextDouble(); if (num > 0) System.out.printf("The square root is %f", Math.sqrt(num)); </pre>	<p>Example:</p> <pre> System.out.println("Enter a number: "); double num = scanner.nextDouble(); if (num > 0) { System.out.printf("The square root is %f", Math.sqrt(num)); System.out.printf("The natural logarithm is %f", Math.log(num)); } </pre>
<p>Example: Finding max of three numbers</p> <pre> max = num1; if(num2 > max) max = num2; if(num3 > max) max = num3; </pre>	<p>Example:</p> <pre> char currencyType = scanner.nextLine().charAt(0); if(currencyType == 's' currencyType == 'S'){ System.out.println("Enter positive amount "); amount = scanner.nextDouble(); riyalBalance = riyalBalance + amount; } </pre>

5. Multi-way selection with an else option (if- else if - else statement)

Used to execute the first *statement* or the first *compound_statement* whose corresponding condition is **true**. The statement in the else part is executed if each condition is **false**.

Single statements	Compound statements
-------------------	---------------------

<pre> if(condition1) statement1; else if(condition2) statement2; else if(condition3) statement3; . . else if(conditionM) statementM; else statementN; </pre>	<pre> if(condition1) compound_statement1 else if(condition2) compound_statement2 else if(condition3) compound_statement3 . . else if(conditionM) compound_statementM else compound_StatementN </pre>
<p>Note: There may be one or more <i>else if</i> branches</p>	<p>Note:</p> <ul style="list-style-type: none"> • There may be one or more <i>else if</i> branches. • It is a syntax error to put a semicolon after the closing brace of a compound statement in an <i>if</i> branch.
 <pre> graph TD Start(()) --> C1{condition1} C1 -- true --> S1[statement1] S1 --> Join1(()) C1 -- false --> C2{condition2} C2 -- true --> S2[statement2] S2 --> Join1 C2 -- false --> Dots1[...] Dots1 --> CM{conditionM} CM -- true --> SM[statementM] SM --> Join1 CM -- false --> SN[statementN] SN --> End(()) Join1 --> End </pre>	 <pre> graph TD Start(()) --> C1{condition1} C1 -- true --> CS1[cstatement1] CS1 --> Join1(()) C1 -- false --> C2{condition2} C2 -- true --> CS2[cstatement2] CS2 --> Join1 C2 -- false --> Dots1[...] Dots1 --> CM{conditionM} CM -- true --> CSM[cstatementM] CSM --> Join1 CM -- false --> CSN[cstatementN] CSN --> End(()) Join1 --> End </pre>

```

boolean validGrade = true;
double grade; char letterGrade = 'A';
System.out.println("Enter grade");
grade = scanner.nextDouble();
if(grade < 0.0 || grade > 100.0)
    validGrade = false;
else if(grade >= 85.0)
    letterGrade = 'A';
else if(grade >= 75.0)
    letterGrade = 'B';
else if(grade >= 65.0)
    letterGrade = 'C';
else if(grade >= 45.0)
    letterGrade = 'D';
else
    letterGrade = 'F';

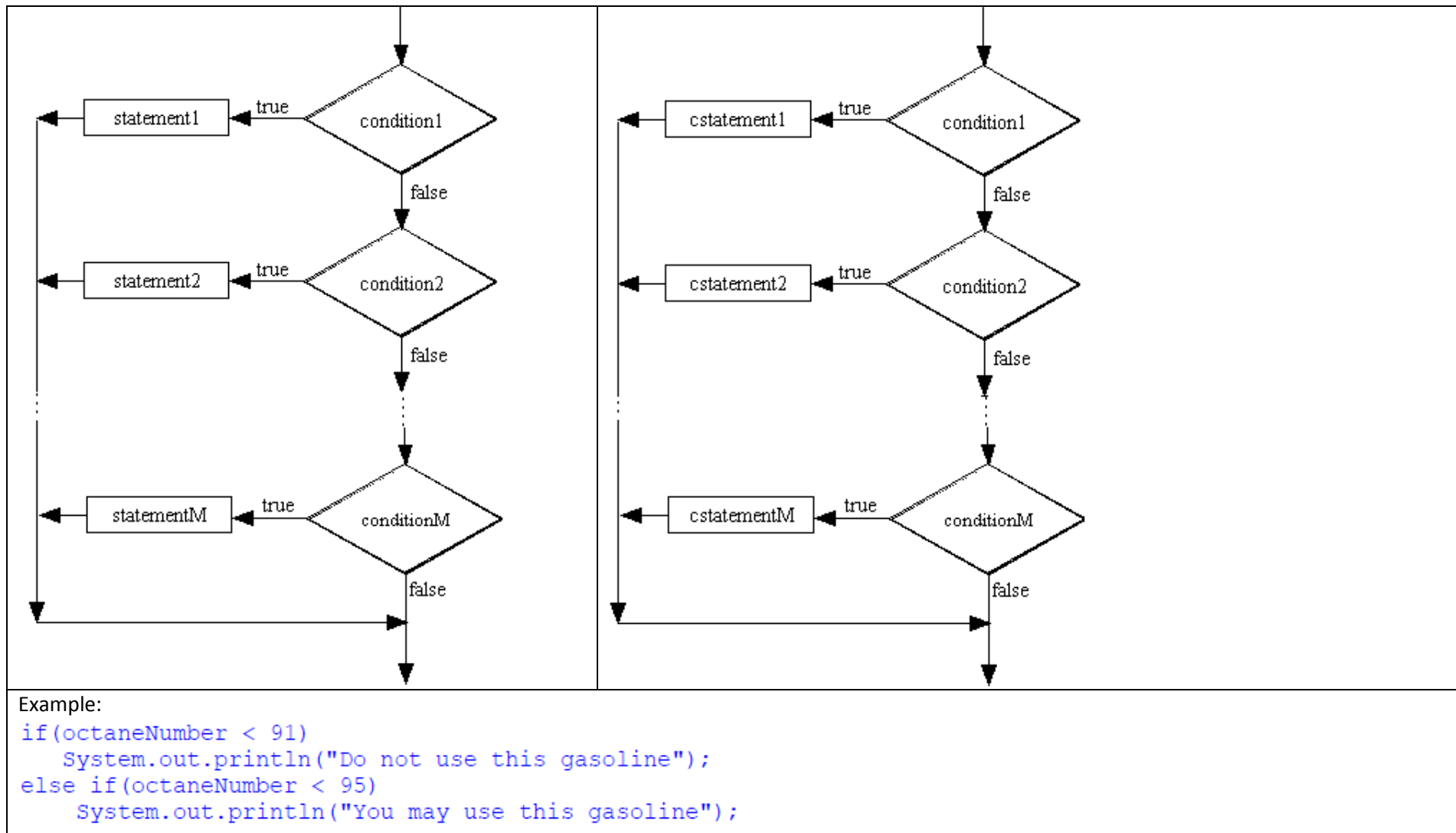
if(validGrade)
    System.out.printf("The letter grade is %c", letterGrade);
else
    System.out.println("Error: Invalid grade");

```

6. Multi-way selection without an else option (if - else if - else if statement)

Used to execute the first *statement* or *compound_statement* whose corresponding condition is **true**. No if-branch is executed if each condition is **false**.

Single statements	Compound statements
<pre> if(condition1) statement1; else if(condition2) statement2; . . else if(conditionM) statementM; else if(conditionN) statementN; </pre>	<pre> if(condition1) compound_statement1 else if(condition2) compound_statement2 . . else if(conditionM) compound_statementM else if(conditionN) compound_StatementN </pre>
Note: There may be one or more <i>else if</i> branches	Note: <ul style="list-style-type: none"> • There may be one or more <i>else if</i> branches. • It is a syntax error to put a semicolon after the closing brace of a compound statement in an <i>if</i> branch (except the last branch).



7. Nested if statements

The compound statement in an if-branch or an else-branch of an if-statement may contain one or more of any type of if-statement discussed in the previous pages.

Example:

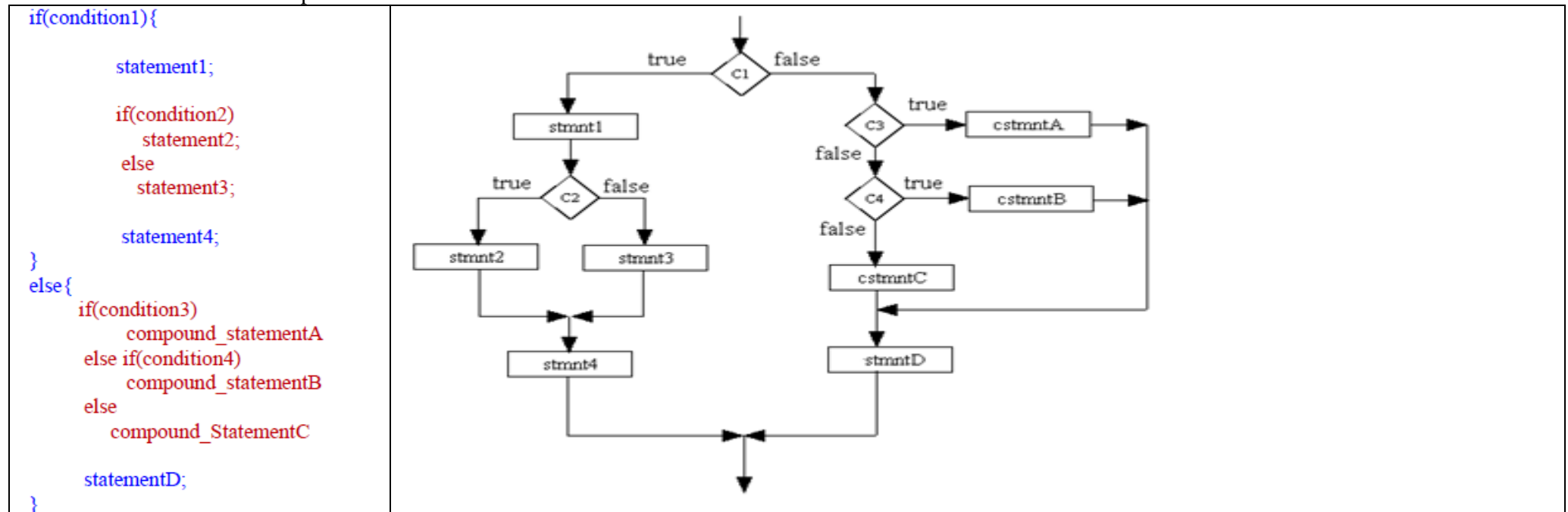

```

if(grade < 0.0 || grade > 100.0)
    System.out.println("Error: Invalid grade");
else{
    if(grade >= 85.0)
        letterGrade = 'A';
    else if(grade >= 75.0)
        letterGrade = 'B';
    else if(grade >= 65.0)
        letterGrade = 'C';
    else if(grade >= 45.0)
        letterGrade = 'D';
    else
        letterGrade = 'F';

    System.out.printf("The letter grade is %c", letterGrade);
}

```

Nested if structures can be complicated:



Note: In a nested if statement, the last else is associated with the closest unpaired if, unless braces are used to alter the default pairing:

Example:

```

if(hours < 9)
    if(distance > 500)
        System.out.println("Type 01");
else
    System.out.println("Type 02");

```

is equivalent to:

```

if(hours < 9){
    if(distance > 500)
        System.out.println("Type 01");
    else
        System.out.println("Type 02");
}

```

8. Switch statement

The *if- else if- else* statement is used in programming situations where one set of statements must be selected from many possible alternatives. The **switch** statement provides an alternative to this statement for cases that compare the value of a **char**, **byte**, **short**, **int**, or **String** expression to a specific **char**, **byte**, **short**, **int**, or **String** constant. [Note: Using **String** in switch expression is supported in Java 7 and above]

The general form of the switch statement is:

```

switch(expression){ // char, byte, short, int, or String expression
    case constant1:    statementList1;
                      break;
    case constant2:    statementList2;
                      break;
                      .
                      .
                      .
    case constantM:    statementListM;
                      break;
    default: statementListN;
}

```

Note:

The case constants must be distinct char, byte, short, int, or String constants; otherwise there is a syntax error.

A *statementList* may contain zero or more semi-colon separated statements. It is not necessary for *statementList* to be a compound-statement.

The **default** label together with its *statementList* may be missing.

The **default** label need not be the last label.

The **break** statement following a *statementList* may be missing.

The *switch expression* is evaluated and then the *statementList* of the case value that equals to the *expression* is executed. If there is a **break** statement, control passes to the statement after the switch; otherwise, the following *statementLists* are executed until a **break** statement is encountered, control then passes to the statement after the switch statement.

If switch *expression* is not equal to any case value, the *statementList* for the **default** label is executed, and if the default label is the last one or if its statement list is followed by a break statement, control passes to the statement after the switch statement.

If switch *expression* is not equal to any case value and there is no **default** label, control passes to the statement after the switch without executing any switch *statementList*.

switch example01: display the name of a digit:

```
int digit;
System.out.println("Enter an integer digit: ");
digit = scanner.Int();
switch(digit){
    case 0:    System.out.println("zero");
               break;
    case 1:    System.out.println("one");
               break;
    case 2:    System.out.println("two");
               break;
    case 3:    System.out.println("three");
               break;
    case 4:    System.out.println("four");
               break;
    case 5:    System.out.println("five");
               break;
    case 6:    System.out.println("six");
               break;
    case 7:    System.out.println("seven");
               break;
    case 8:    System.out.println("eight");
               break;
    case 9:    System.out.println("nine");
               break;
    default:   System.out.println("Error: Invalid digit");
}
```

switch example02: classify a character

```
char c1;
System.out.println("Enter an English alphabet: ");
c1 = scanner.nextLine().charAt(0);
if(c1 >= 'a' && c1 <= 'z' || c1 >= 'A' && c1 <= 'Z'){
    switch(c1){
        case 'a':
        case 'A':
        case 'e':
        case 'E':
        case 'i':
        case 'I':
        case 'o':
        case 'O':
        case 'u':
        case 'U': System.out.println("You entered a vowel");
                   break;
        default:  System.out.println("You entered a consonant");
    }
}else
    System.out.println("You entered a non-English alphabet");
```

switch example03: Determine month number from month string:

// Note: Strings in switch condition are supported in Java 7 and above

```
String month;
int monthNumber;
System.out.println("Enter a month name:");
month = scanner.next( );
switch (month.toLowerCase()) {
    case "january": monthNumber = 1;
                    break;
    case "february": monthNumber = 2;
                     break;
    case "march":    monthNumber = 3;
                     break;
    case "april":    monthNumber = 4;
                     break;
    case "may":      monthNumber = 5;
                     break;
    case "june":     monthNumber = 6;
                     break;
    case "july":     monthNumber = 7;
                     break;
    case "august":   monthNumber = 8;
                     break;
    case "september": monthNumber = 9;
                     break;
    case "october":  monthNumber = 10;
                     break;
    case "november": monthNumber = 11;
                     break;
    case "december": monthNumber = 12;
                     break;
    default:        monthNumber = 0;
                     break;      // optional
}

if(monthNumber != 0)
    System.out.printf("The month number for %s is %d\n",
                      month, monthNumber);
else
    System.out.println("Error: Invalid month name");
```

9. Void Methods

A **method** is a sequence of instructions with a name. Let's write a method to compute the volume of a cube with a given side length. When writing this method, you need to

- Pick a name for the method (cubeVolume).
- Declare a variable for each argument (double sideLength). These variables are called the **parameter variables**.
- Specify the type of the return value (**void**).
- Add the public static modifiers.

The body of a method is enclosed in braces. This has to be placed outside of the **main** method. Here is the complete method:

```
public static void cubeVolume(double sideLength) {  
    double volume = sideLength * sideLength * sideLength;  
    System.out.println("A cube with side length " + sideLength + " has volume " + volume);  
}
```

Let's put this method to **use**. We'll supply a main method that calls the cubeVolume method twice.

```
public static void main(String[] args) {  
    cubeVolume(2);  
    cubeVolume(10);  
}
```

Parameter Passing

When a method is called, variables are created for receiving the method's arguments. These variables are called **parameter variables**. (Another commonly used term is **formal parameters**.) The values that are supplied to the method when it is called are the **arguments** of the call. (These values are also commonly called the **actual parameters**.)

Each parameter variable is initialized with the corresponding argument.

<p>Example:</p> <pre>public static void mystery(int x, int y) { double z = x + y; z = z / 2.0; System.out.println(z); } public static void main(String[] args) { int a = 5; int b = 7; mystery(a, b); }</pre>	<p>Example:</p> <pre>public static void mystery(int x) { int y = x * x; System.out.println(y); } public static void main(String[] args) { int a = 4; mystery(a + 1); }</pre>
<p>Example:</p> <pre>public static void mystery(int n) { n++;</pre>	<p>Example:</p> <pre>public static void addTax(double price, double rate) { double tax = price * rate / 100;</pre>

<pre> n++; System.out.println(n) } public static void main(String[] args) { int a = 5; mystery(a); } </pre>	<pre> double result = price + tax; System.out.println (result); } public static void main(String[] args) { addTax(1000, 5); addTax(100, 20); } </pre>
<p>Often it is useful to pass more than one argument to a method. Here is a method that accepts two arguments:</p> <pre> public static void showSum(double num1, double num2){ double sum; // To hold the sum sum = num1 + num2; System.out.println("The sum is " + sum); } </pre> <p>Notice that two parameter variables, num1 and num2, are declared inside the parentheses in the method header. This is often referred to as a <i>parameter list</i>. Also notice that a comma separates the declarations. Here is an example of a statement that calls the method:</p> <pre> showSum(5, 10); </pre>	

10. Laboratory Tasks

Note: For each of task write a pseudo-code algorithm and then translate it into a complete Java program. A sample problem and its pseudo-code algorithm is:
Write a Java program that prompts for and reads a number greater than zero. If the number x entered is greater than zero, the program computes and displays x, $\log_{10}(x)$, $\ln(x)$, and e^x ; otherwise it displays an appropriate error message.

1. Prompt for a number x [$x > 0$]

2. Input: x

3. if($x > 0$) {

3.1 Output: x

3.2 Output: $\log_{10}(x)$

3.3 Output: $\ln(x)$

3.4 Output: e^x

}else

3.5 Output: “Error: $x \leq 0$ ”

4. Stop.

The translation of the pseudo-code algorithm is:

```

import java.util.Scanner;
public class Lab2_2Task01 {
    public static void main(String[] args) {
        double x;
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a value greater than zero: ");
        x = scanner.nextDouble();
        if(x > 0){
            System.out.printf("x = %.5f\n", x);
            System.out.printf("log10(x) = %.5f\n", Math.log10(x));
            System.out.printf("ln(x) = %.5f\n", Math.log(x));
            System.out.printf("e^x = %g\n", Math.exp(x));
        }else
            System.out.println("Error: x <= 0");
    }
}

```

Task 01: Write a Java program that prompts and reads two integer numbers. It then checks the numbers and prints one of the following messages accordingly:

You have entered two even numbers.
You have entered two odd numbers.
You have entered one even number and one odd number.

Hint: Use the modulus operator (%) for checking the numbers.

Task 02: Write a Java program that reads a character from the user. It then displays one of the following messages accordingly:

The character **x** is lowercase

The character **x** is uppercase

The character **x** is a character digit

where **x** is the character read from the console/keyboard

Task 03: Convert the program-fragment below to a complete Java program that uses **if-else if -else** statement instead of the **switch** statement:

```

char grade;
System.out.println("Enter the student letter grade (A, B, C, D, or F): ");
grade = scanner.nextLine().charAt(0);
switch (grade) {
    case 'A':
    case 'a':
    case 'B':
    case 'b':    System.out.println("Good standing");
                break;

    case 'C':
    case 'c':    System.out.println("O.K. ");
                break;

    case 'D':
    case 'd':
    case 'F':
    case 'f':    System.out.println("Poor, student is on probation");
                break;

    default:     System.out.println("Invalid letter grade");
}

```

Task 04: Write a Java program that displays the following menu:

1. Find area and perimeter of a rectangle
2. Find area and circumference of a circle

The program then reads the menu choice and behaves as in the following table:

Menu choice	Program behavior
Input other than 1, and 2	The program displays the following error message : Error: Wrong menu choice and then terminates.
1	The program prompts for and reads the length and the width of a rectangle. It then computes and displays the area and the perimeter of the rectangle. Sample input: length = 4.0, width = 3.0 Output: area = 12.00 square cm , perimeter = 14.00 cm
2	The program prompts for and reads the radius of a circle. It then computes and displays the area and the circumference of the circle. [$Area = \pi r^2$, $Circumference = 2\pi r$] Sample input: 3.5 Output: area = 38.48 square cm, circumference = 21.99 cm

Note:

Assume that the values read by the program for options 1, and 2 are in **centimeters**. For these options, your program must display appropriate units in the output. You can solve the problem by using if- or switch-statement.

Task05: Write a Java program that prompts for and reads two strings **str1** and **str2**. The program then displays one of the following messages accordingly:

str1 and **str2** are equal

str1 is greater than **str2**

str1 is less than **str2**

where **str1** and **str2** are the strings read. Your comparison must be case-sensitive.

Task06: The following algorithm yields the season (Spring, Summer, Fall, or Winter) for a given month and day.

If month is 1, 2, or 3, season = "Winter" Else if month is 4, 5, or 6, season = "Spring" Else if month is 7, 8, or 9, season = "Summer" Else if month is 10, 11, or 12, season = "Fall" If month is divisible by 3 and day >= 21 If season is "Winter", season = "Spring" Else if season is "Spring", season = "Summer" Else if season is "Summer", season = "Fall" Else season = "Winter"	Write a program that prompts the user for a month and day and then prints the season, as determined by this algorithm.
--	--

Task07: Write a method named timesTen. The method should accept a double argument, and prints a double value that is ten times the value of the argument.

Task08: Write a method named square that accepts an integer argument and prints the square of that argument.

Task09: Write a method named quartersToDollars. The method should accept an int argument that is a number of quarters, and return the equivalent number of dollars as a double. For example, if you pass 4 as an argument, the method should return 1.0; and if you pass 7 as an argument, the method should return 1.75.

Task10: Celsius Temperature Table

The formula for converting a temperature from Fahrenheit to Celsius is

$$C = \frac{5}{9}(F-32)$$

where F is the Fahrenheit temperature and C is the Celsius temperature. Write a method named celsius that accepts a Fahrenheit temperature as an argument. The method should print the temperature, converted to Celsius.

Task11: Distance Traveled Modification

The distance a vehicle travels can be calculated as follows: $Distance = Speed * Time$

Write a method named distance that accepts a vehicle's speed and time as arguments, and prints the distance the vehicle has traveled.

Task12: Kinetic Energy

In physics, an object that is in motion is said to have kinetic energy. The following formula can be used to determine a moving object's kinetic energy: $KE = \frac{1}{2} mv^2$

The variables in the formula are as follows: KE is the kinetic energy, m is the object's mass in kilograms, and v is the object's velocity, in meters per second.

Write a method named kineticEnergy that accepts an object's mass (in kilograms) and velocity (in meters per second) as arguments. The method should return the amount of kinetic energy that the object has. Demonstrate the method by calling it in a program that asks the user to enter values for mass and velocity.