Software Development

5. Selection Statements

Revision

- Data types
  - Declaring variables
  - Declaring instance variables
  - Declaring constants
- User input/ Display output
- Instantiable Classes

Revision: boolean – Primitive Data Type

- **boolean** - has only two possible values: **true** or **false**
- Declaring a variable of type boolean
  - boolean <variableName>;
  - e.g. Does the customer have a loyalty card?

    boolean hasLoyaltyCard;
- Assigning a value to the boolean variable
  - hasLoyaltyCard = **true;** // yes
  - hasLoyaltyCard = **false;** // no

Problem

- Create an application to compute basic arithmetic operations. The application prompts the user to enter the arithmetic operation the user wants to perform (e.g. addition, subtraction, multiplication or division). The application prompts the user to provide the two numbers, computes the result of the required operation, and prints the result of the operation.
  - Use the instantiable class SimpleCalculator.java

Problem

- Create an application to compute basic arithmetic operations. The application prompts the user to enter the arithmetic operation the user wants to perform (e.g. addition, subtraction, multiplication, or division). The application prompts the user to provide the two numbers, computes the result of the **required** operation, and prints the result of the operation.
  - Use the instantiable class SimpleCalculator.java
  - We need to know **how to select** the action to be performed

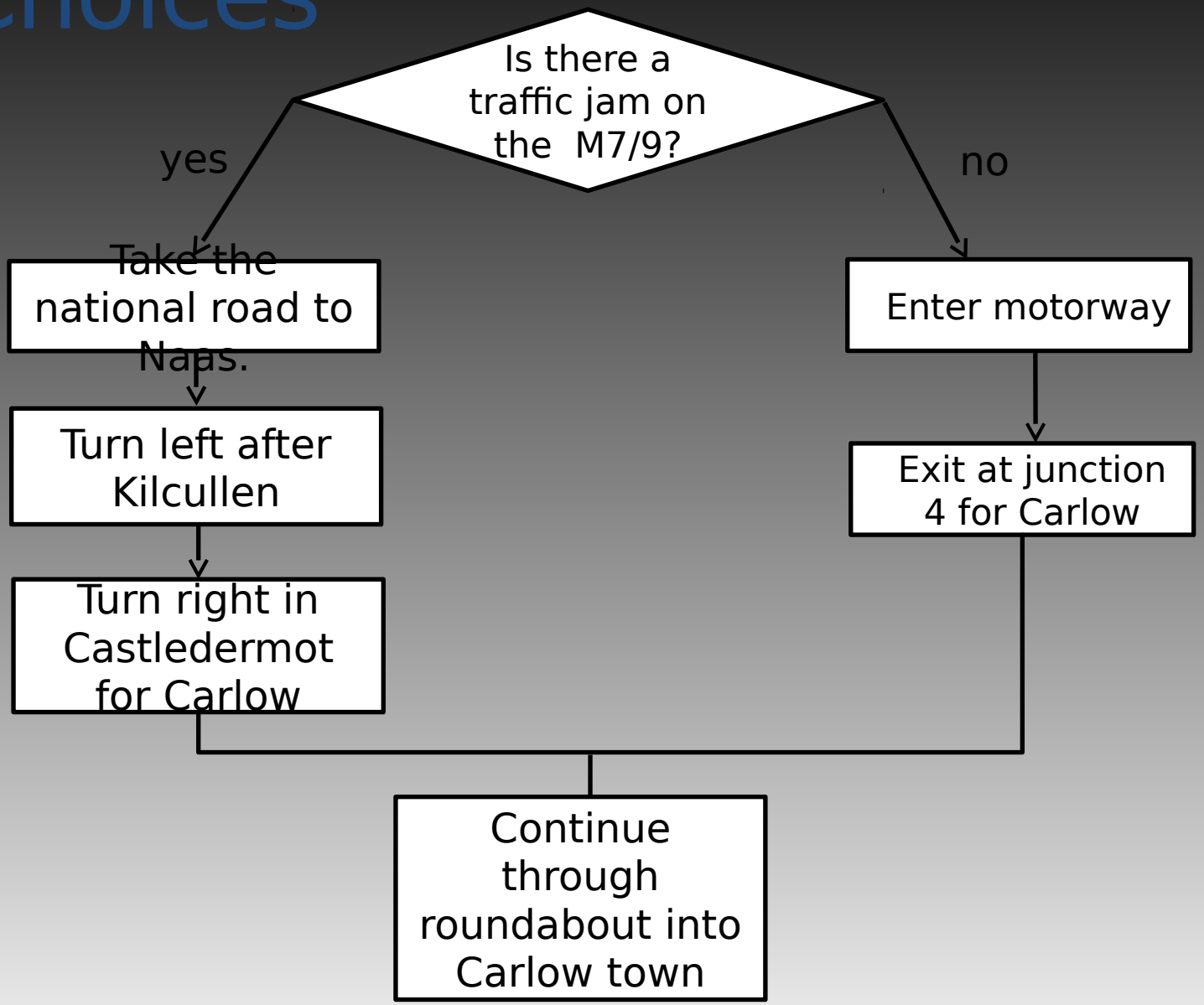
Approach for Writing a Program

- Planning the logic of your program
  - Identify input
  - Identify process
  - Identify output
  - Use class diagrams
  - Pseudo-code
    - Write (on a piece of paper) in english statements the tasks you need to perform to implement your program
  - Flow-charts
    - Similar with pseudo-code, but we write the steps in diagrams formed from different boxes connected by arrows (the arrows show the flow of the solution)

Approach for Writing a Program

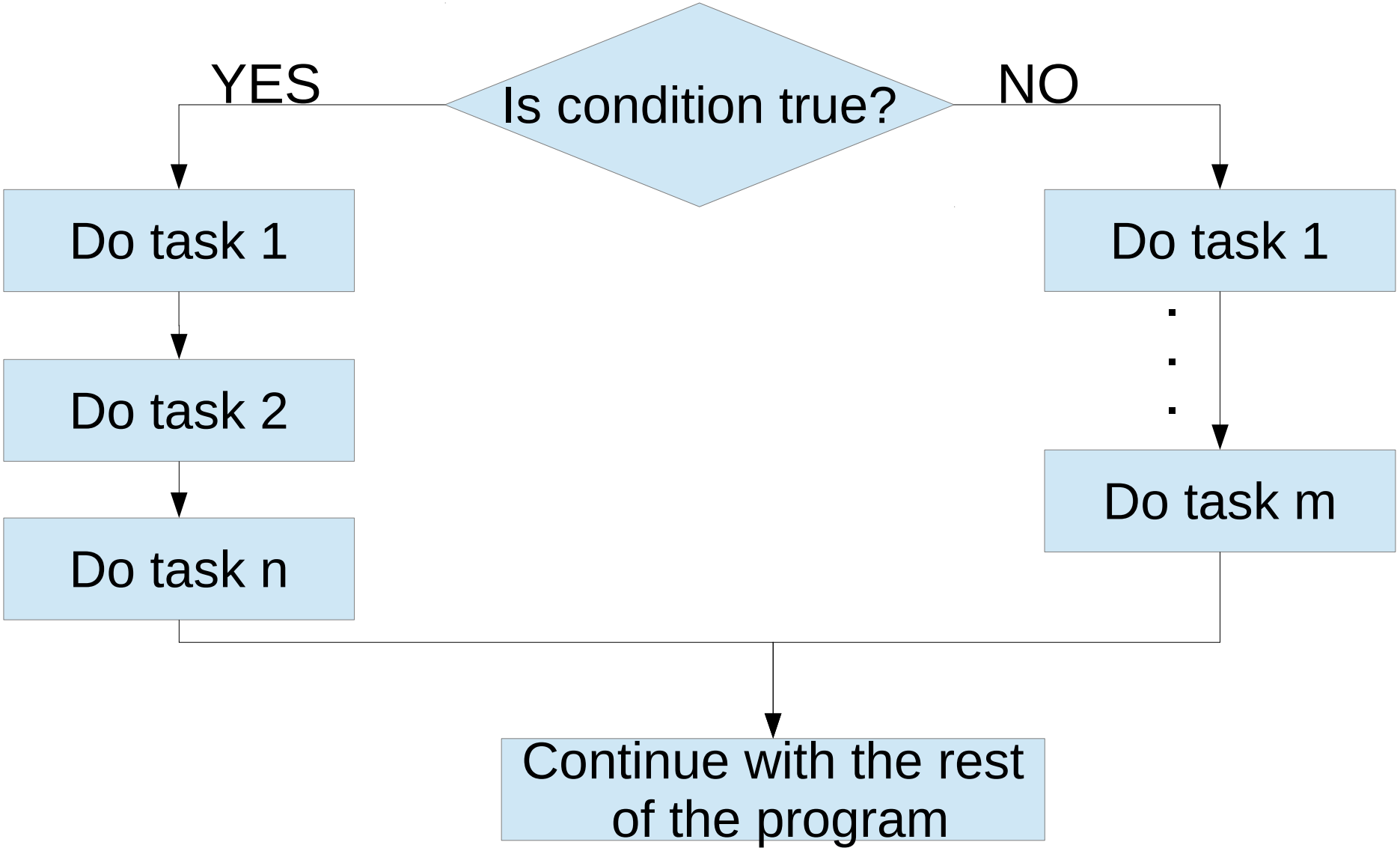
- Planning the logic of your program
  - Identify input
  - Identify process
  - Identify output
  - Use class diagrams
  - Pseudo-code
    - Write (on a piece of paper) in english statements the tasks you need to perform to implement your program
  - **Flow-charts**
    - Similar with pseudo-code, but we write the steps in diagrams formed from different boxes connected by arrows (the arrows show the flow of the solution)

Flow Chart – Making Choices

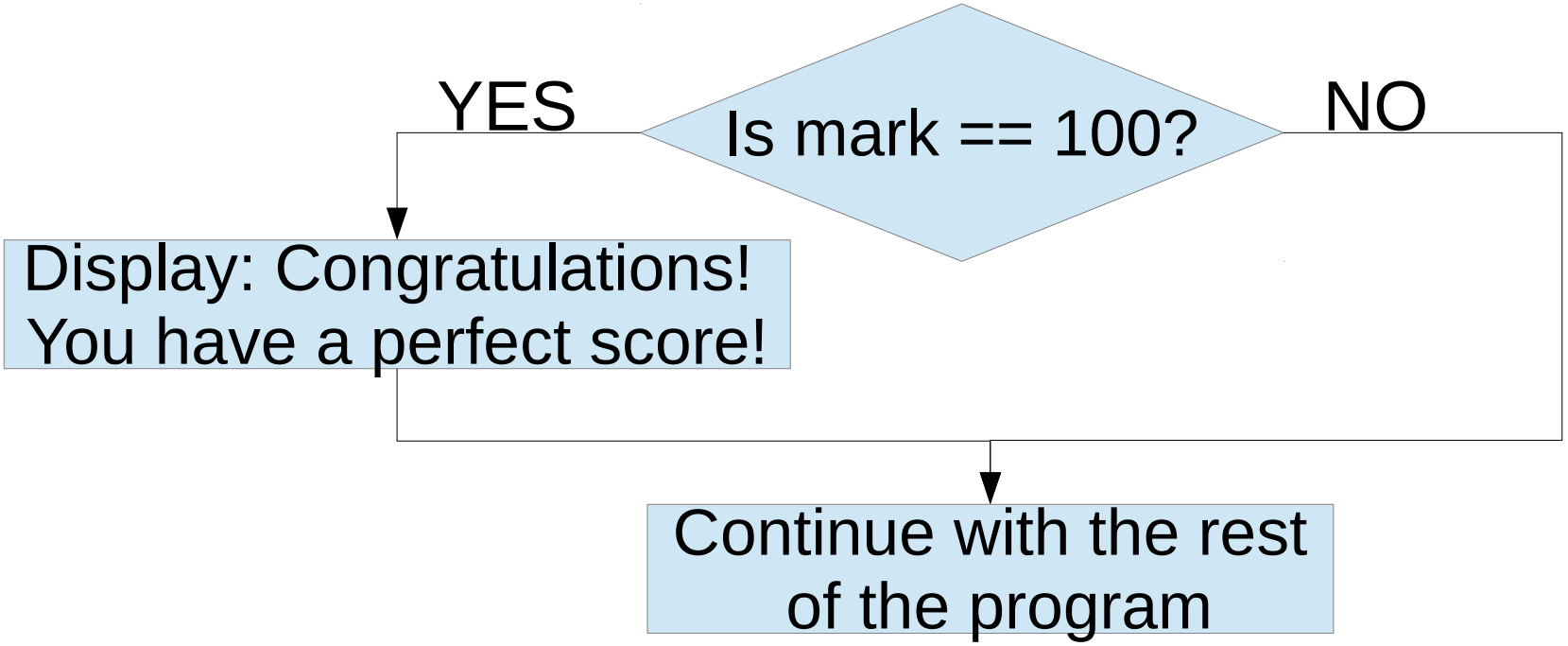


source: Carole McGloughlin's lecture notes

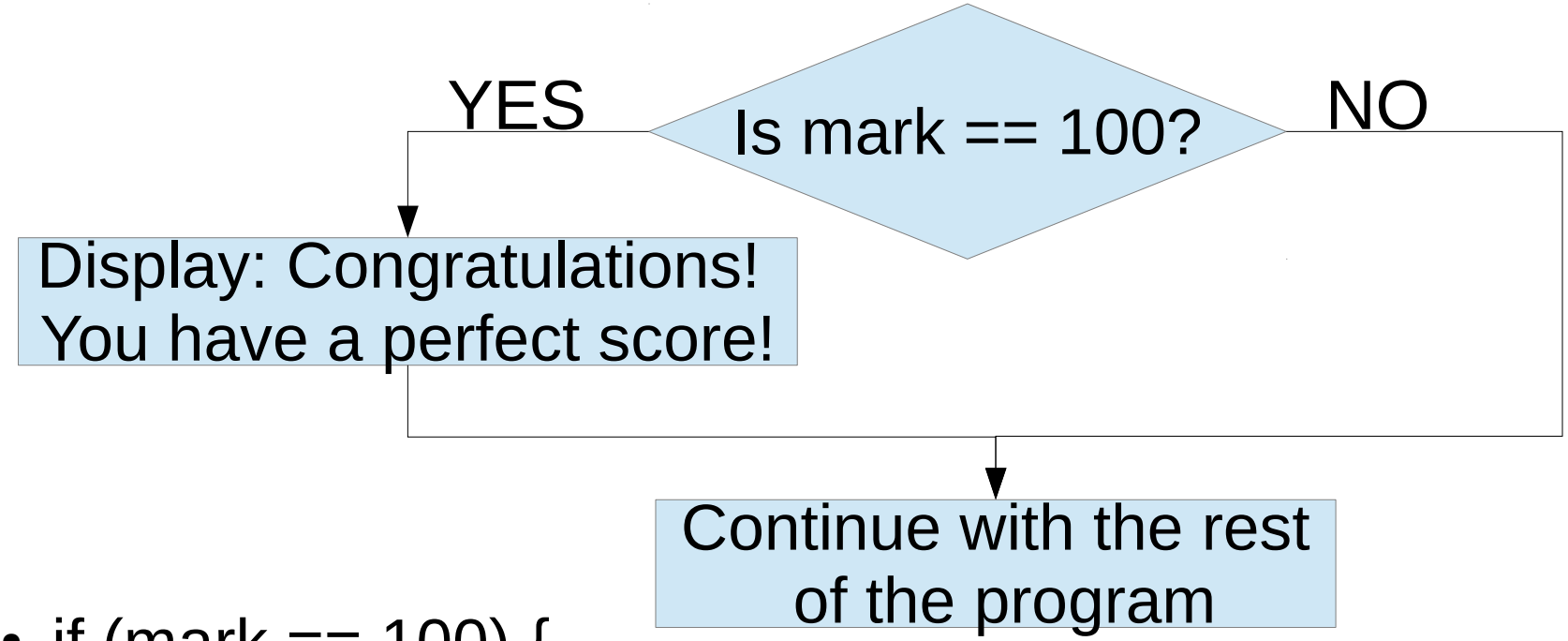
Flow Chart


Is condition true?
YES
NO
Do task 1
Do task 2
Do task n
Do task 1
Do task m
Continue with the rest of the program

The if Selection Statement



The if Selection Statement



- if (mark == 100) {

  System.out.println("Congratulations! You have a perfect score!");

  }

Equality Operator

- if (mark == 100) {

  System.out.println("Congratulations! You have a perfect score!");

  }
- == is the equality operator – checks if the two values are equal to each other or not
- Recall: one single = is the assignment operator!

The if Selection Statement

- General syntax

```
if (<boolean expression>) {
   <block of statements>
}
```

- If the **boolean expression** is **true** only then Java executes the block of statements/actions enclosed in the if's { }
- We use boolean exppressions to write **conditions**
- For example, mark == 100 is a boolean expression
  - It evaluates to either **true** or **false**
  - If mark == 100 is true then the message is displayed
  - If mark == 100 is false then the message is not displayed

Boolean expressions

- A boolean expression evaluates to either true or false
- We already know a data type to represent false or true values
  - boolean
- We can store the result of a boolean expression into a variable of type boolean
  - e.g. boolean condition = mark == 100; // or
  - e.g. boolean condition = (mark == 100);

The if Selection Statement

- if (mark == 100) {

  System.out.println("Congratulations! You have a perfect score!");

  }
- Another alternative for writing the above if statement is
- boolean isPerfectScore; // declare a boolean variable

  isPerfectScore = (mark == 100);

  if (isPerfectScore) {

  System.out.println("Congratulations! You have a perfect score!");

  }

The if Selection Statement

There is no “;” !

- if (mark == 100) {

  System.out.println(“Congratulations! You have a perfect score!”);

  }

- Another alternative for writing the above if statement is
- boolean isPerfectScore; // declare a boolean variable

  isPerfectScore = (mark == 100);

  if (isPerfectScore) {

  System.out.println(“Congratulations! You have a perfect score!”);

  }

Equality and Relational Operators

- **==** **equal to** e.g. if (result == 50)
- **!=** **not equal to** e.g. if (result != 50)
- **>** **greater than** e.g. if (result > 50)
- **>=** **greater than or equal to** if (result >=50)
- **<** **less than** e.g. if (result < 50)
- **<=** **less than or equal to** e.g. if (result <= 50)

Relational Operators Example

```
public class RelationalOperatorsDemo {
    public static void main(String args[]){
        int a = 10, b = 15;
        System.out.println(a<b);
        System.out.println(a>b);
    }
}
```

Relational Operators Example

```
public class RelationalOperatorsDemo {
    public static void main(String args[]){
        int a = 10, b = 15;
        System.out.println(a<b);
        System.out.println(a>b);
    }
}
```

Q: What is the output of the program?

Relational Operators Example

- Java evaluates the value of the boolean expression, namely whether the expression is true or false
- System.out.println(a<b);
  - Java evaluates a<b
  - a is 10 and b is 15
  - 10 < 15, indeed, 10 is less than 15, therefore the boolean expression is true, and the printing statement prints true

Relational Operators Example

- Java evaluates the value of the boolean expression, namely whether the expression is true or false
- System.out.println(a>b);
  - Java evaluates a>b
  - a is 10 and b is 15
  - 10 > 15 , 10 is not greater than 15, therefore the boolean expression is false, and the printing statement prints false

Relational Operators Example

```
public class RelationalOperatorsDemo {
    public static void main(String args[]){
        int a = 10, b = 15;
        System.out.println(a<b);  // true
        System.out.println(a>b);  // false
    }
}
```

Q: What is the output of the program?

A:
true
false

Equality and Relational Operators

- Arithmetic Operators and Equality and Relational Operators can be used to form boolean expressions/conditions
- int a =10, b = 15, c = 20; boolean condition;

  condition = **b < a**;

  condition = **b >= a + 5**;

  condition = **c - 5 != b**;

  condition = **a + b > c**;

What is the value of each boolean expression?

Equality and Relational Operators

- Arithmetic Operators and Equality and Relational Operators can be used to form boolean expressions/conditions
- int a =10, b = 15, c = 20; boolean condition;

  condition = **b < a**; // false

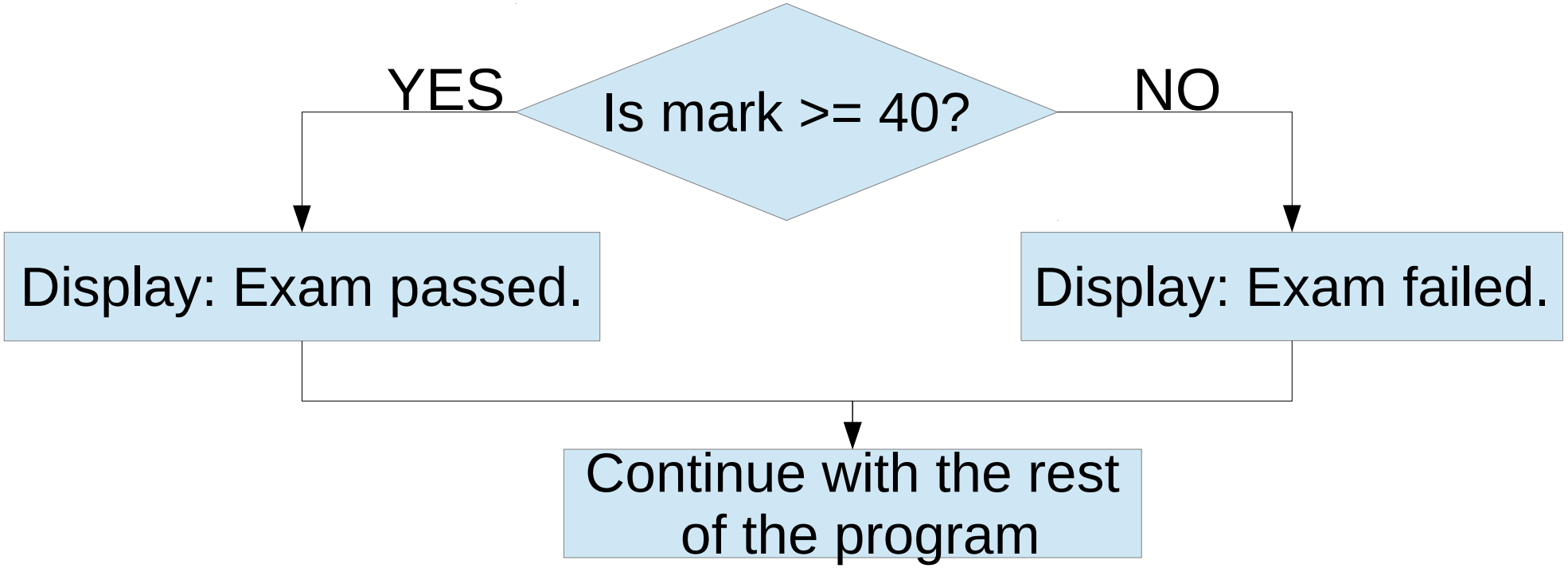
  What is the value of each boolean expression?

  condition = **b >= a + 5**; // true

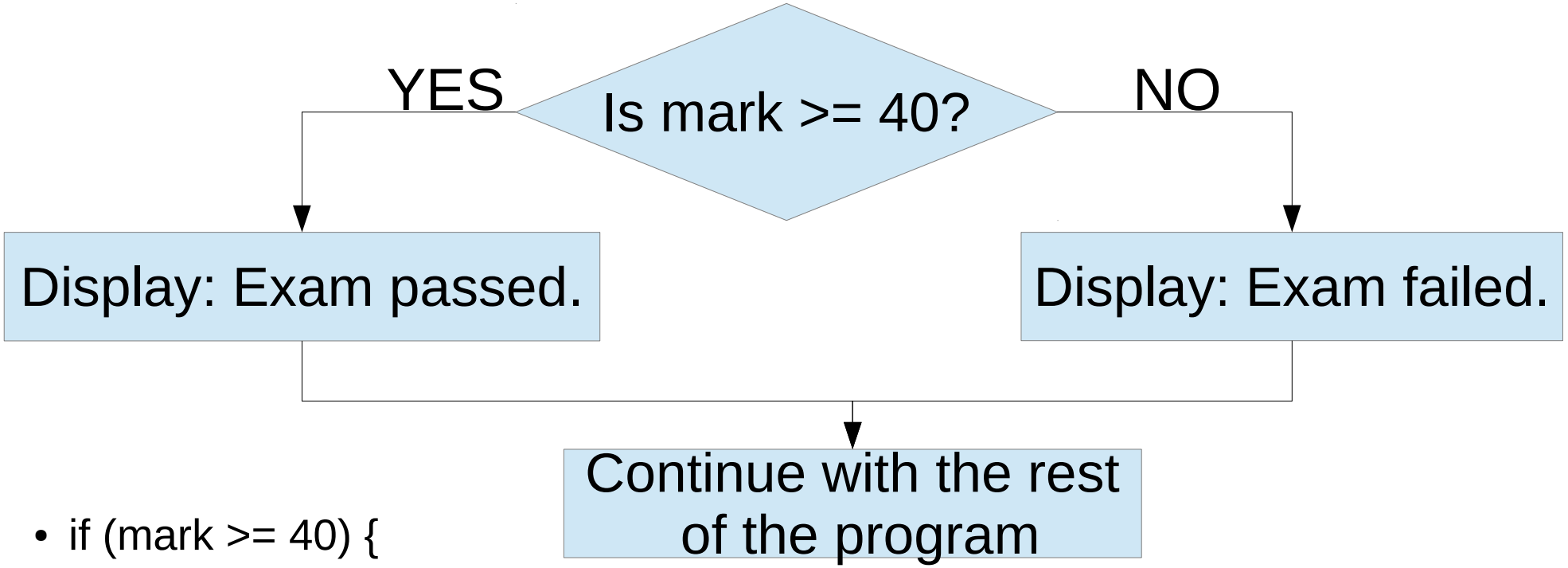
  condition = **c - 5 != b**; // false

  condition = **a + b > c**; // true

The if ... else selection statement



The if ... else selection statement



- if (mark >= 40) {

  System.out.println(“Exam passed.”);

  } else {

  System.out.println(“Exam failed.”);

  }

The if ... else Selection Statement

- General Syntax

```
if (<boolean expression>) {
    <block 1 of statements>
} else {
    <block 2 of statements>
}
```

- If the **boolean expresion** is **true**, Java executes the block of statements written after the if (i.e. enclosed in the if's { })
- If the **boolean expression** is **false**, Java executes the block of statements written after the else (i.e. enclosed in the else's { })

The if ... else Selection Statement

- The if ... else selection statement allows us to specify the actions/statements that should be performed when there are 2 mutually exclusive conditions/ decisions/ choices

The if ... else Selection Statement

- int mark = 55;

```
if (mark >= 40) {
    System.out.println("Exam passed.");
} else {
    System.out.println("Exam failed.");
}
```

Q: What is the output of the program?

The if ... else Selection Statement

There is no “;” !

- int mark = 55;

```
if (mark >= 40) {  // true: 55 is greater than or equal to 40
    System.out.println(“Exam passed.”);
} else {
    System.out.println(“Exam failed.”);
}
```

- The code above will display the message

  Exam passed.

The if ... else Selection Statement

- int mark = 37;

```
if (mark >= 40) {
    System.out.println("Exam passed.");
} else {
    System.out.println("Exam failed.");
}
```

Q: What is the output of the program?

The if ... else Selection Statement

There is no “;” !

- int mark = 37;

```
if (mark >= 40) {
    System.out.println(“Exam passed.”);
} else {// false: because 37 is less than 40
    System.out.println(“Exam failed.”);
}
```

- The code above will display the message

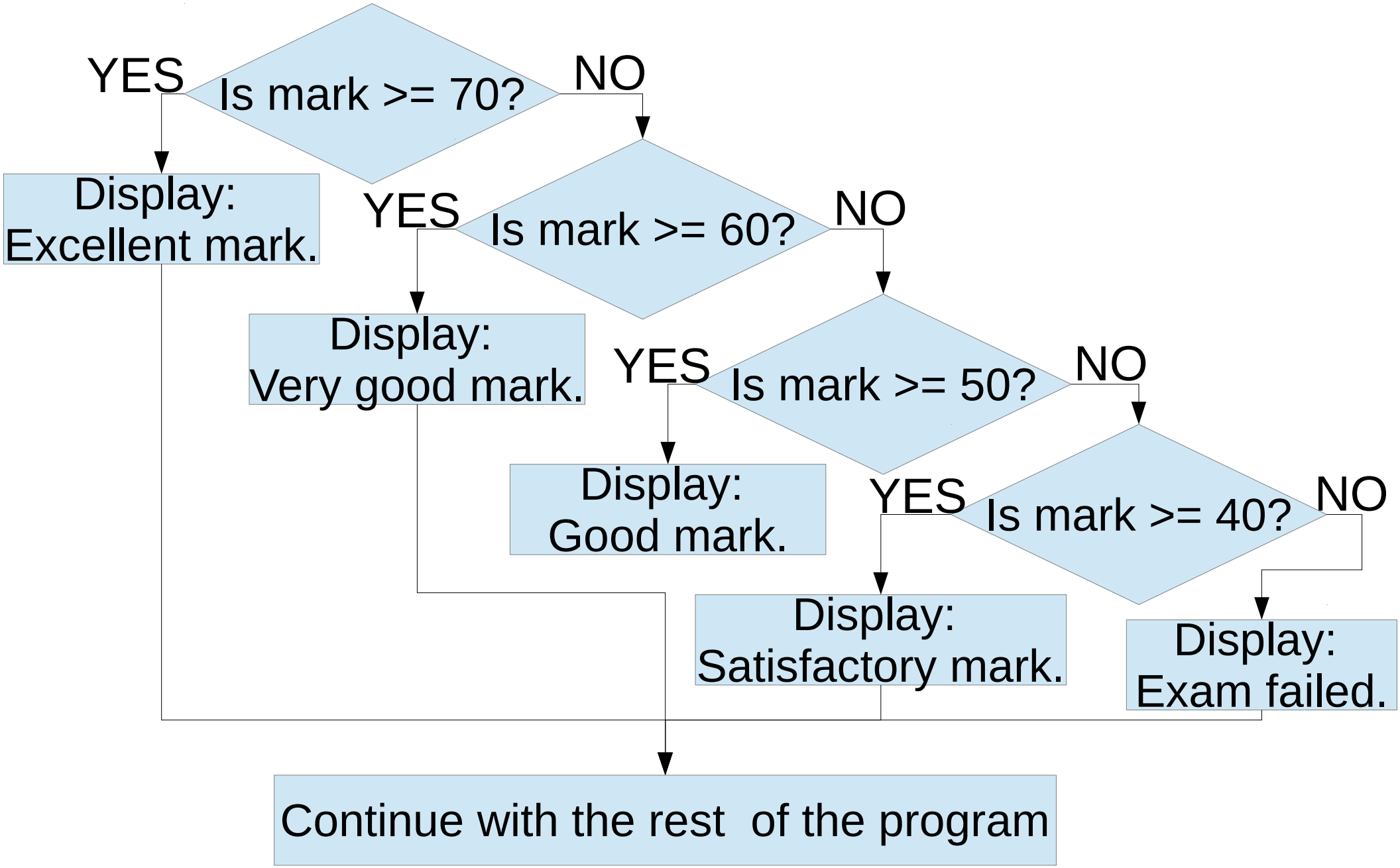
  Exam failed.

Nested if ... else selection statements

- Write an application that prompts the user to input a mark between 0 and 100, and then prints a message according to the conditions:

Mark Interval	**Message**
[70, 100]	Excellent mark.
[60, 70)	Very good mark.
[50, 60)	Good mark.
[40, 50)	Satisfactory mark.
[0, 40)	Exam failed.

Nested if ... else selection statements



Nested if ... else Selection Statement

- The nested if ... else selection statement allows us to specify the actions/tasks that should be performed when there exist multiple conditions/ decisions/ choices

```
if (<boolean expression 1>) {
    <block 1 of statements>
} else if (<boolean expression 2>) {
    <block 2 of statements>
} else {
    <block 3 of statements>
}
```

Nested if ... else Selection Statement

- if (<boolean expression 1>) {

  <block 1 of statements>

  } else if (<boolean expression 2>) {

  <block 2 of statements>

  } else {

  <block 3 of statements>

  }

- If the **boolean expresion 1** is **true**, Java executes the block 1 of statements written after the if (i.e. enclosed in the if's { })
- **Only if** the boolean expression 1 is false, Java evaluates the **boolean expression 2**. If the **expression** is **true**, Java executes the block 2 of statements written after the else if (i.e. enclosed in the else if's { })
- If the **boolean expression 2** is **false**, Java executes the block 3 of statements written after the else (i.e. enclosed in the else's { })

Nested if ... else Selection Statement

```
if (mark >= 70) {
        System.out.println("Excelent mark.");
} else if (mark >= 60) {
        System.out.println("Very good mark.");
} else if (mark >= 50) {
        System.out.println("Good mark.");
} else if (mark >= 40) {
        System.out.println("Satisfactory mark.");
} else {
        System.out.println("Exam failed.");
}
```

Nested if ... else Selection Statement

```
int mark = 62;
if (mark >= 70) {
        System.out.println("Excelent mark.");
} else if (mark >= 60) {
        System.out.println("Very good mark.");
} else if (mark >= 50) {
        System.out.println("Good mark.");
} else if (mark >= 40) {
        System.out.println("Satisfactory mark.");
} else {
        System.out.println("Exam failed.");
}
```

Q: What is the output when the mark is 62?

Nested if ... else Selection Statement

```
int mark = 62;
if (mark >= 70) {
        System.out.println("Excelent mark.");
} else if (mark >= 60) {
        System.out.println("Very good mark.");
} else if (mark >= 50) {
        System.out.println("Good mark.");
} else if (mark >= 40) {
        System.out.println("Satisfactory mark.");
} else {
        System.out.println("Exam failed.");
}
```

Q: What is the output when the mark is 62?

A: The message displayed is:
Very good mark.

Conditional Operators

- || Conditional **OR**

```
if (mark < 0 || mark >100){
    System.out.println(mark + " is not a valid mark.");
}
```

- At least one of the boolean expressions must be true to execute the code within the body of the if statement

Conditional Operators

- || Conditional **OR**

```
if (mark < 0 || mark >100){
    System.out.println(mark + " is not a valid mark.");
}
```

- Common mistake:

```
if (mark < 0 || >100){
    System.out.println(mark + " is not a valid mark.");
}
```

This is wrong! Because the second boolean expression is incomplete as one operand is missing!

Conditional Operators

- **&&** Conditional **AND**

```
if (mark >= 70 && mark < 100){
    System.out.println("Congrats! The result is not
      perfect, but it is an excellent result!")
 }
```

- All boolean expressions must be true to execute the code within the body of the if statement

Conditional Operators

- **&&** Conditional **AND**

```
if (mark >= 70 && mark < 100){
    System.out.println("Congrats! The result is not
      perfect, but it is an excellent result!")
}
```

- Common mistake

```
if (mark >= 70 &&  < 100){
    System.out.println("Congrats! The result is not
      perfect, but it is an excellent result!")
}
```

This is wrong! Because the second boolean expression is incomplete as one operand is missing!

Logical NOT/ complement operator

- **!** is the logical **NOT** operator / logical complement operator and it is used to negate the result of a boolean expression
- A boolean expression that evaluates to true becomes false when the NOT operator is applied to it

```
boolean a, b; a = true; b = !a;  // b is false
```

- A boolean expression that evaluates to false becomes true when the NOT operator is applied to it

```
boolean a, b; a = false; b = !a; // b is true
```

Logical NOT/ complement operator

- Example 1

```
if (mark <= 50)
```

- can be rewritten with the NOT operator as

```
if (!(mark > 50)) // the opposite of bigger than 50
```

- Example 2

```
boolean hasLoyaltyCard = true;
if (!hasLoyaltyCard){  // perform some operations }
```

Operators precedence

Precedence	Operator	Symbol
Highest	Logical Not	!
	Cast	()
	Multiplication, Division, Modulus	*, /, %
	Addition, Subtraction String concatenation	+, -, +
	Relational	>, >=, <, <=
	Equality	==, !=
	Logical AND	&&
	Logical OR	||
Lowest	Assignment	=

Operators precedence

- When you are not sure about the order in which a statement is evaluated you can enclose in brackets the portions you want to be evaluated first
- For example

```
if ( a + 3 > 5 && b > c)
```

- can be rewritten as

```
if (((a+3) > 5 ) && (b > c))
```

switch Multiple-Selection Statement

- Write an application that prompts a student to input a day of the week, and then it will display whether there are classes scheduled for that day

Day	**Message**
Monday	Evening classes
Tuesday	No classes
Wednesday	Evening classes
Thursday	No classes
Friday	No classes
Saturday	Whole day classes
Sunday	It's finally Sunday! :-)

switch Multiple-Selection Statement

- switch (<expression>) {

```
case <value1>:
    <block 1 of statements>
    break;
case <value2>:
    <block 2 of statements>
    break;
default:
    <block of statements>
    break;
}
```

switch Multiple-Selection Statement

- The switch selection statement allows to perform different actions/tasks based on the possible values of a constant expression.
- The type of the expression can be
  - byte, short, int, char
  - String (starting with Java 7)

switch Multiple-Selection Statement

- **switch** starts the selection statement and is followed immediately by an expression/variable enclosed in round brackets ()
- **case** is followed by one of the possible values for the expression/variable and a colon i.e. **:**
  - Each case corresponds to one possible value
- **break** terminates a switch statement, and it is placed at the end of the block statements for each case and default labels
- **default** is optional, and is used when there is no case specified for the current value

switch Multiple-Selection Statement Example

- String day = “TueSday”; day = day.toLowerCase();

```
switch (day) {
    case “monday”: System.out.println(“Evening classes”); break;
    case “tuesday”: System.out.println(“No classes”); break;
    case “wednesday”: System.out.println(“Evening classes”); break;
    case “thursday”: System.out.println(“No classes”); break;
    case “friday”: System.out.println(“No classes”); break;
    case “saturday”: System.out.println(“Whole day classes”); break;
    case “sunday”: System.out.println(“It's finally Sunday! :-)”); break;
    default: System.out.println(“Unknown input”); break;
}
```

switch Multiple-Selection Statement Example – Compact Alternative

- String day = “TueSday”; day = day.toLowerCase();

```
switch (day) {
    case “monday”:
    case “wednesday”:
            System.out.println(“Evening classes”); break;
    case “tuesday”:
    case “thursday”:
    case “friday”:
            System.out.println(“No classes”); break;
    case “saturday”:
            System.out.println(“Whole day classes”); break;
    case “sunday”:
            System.out.println(“It's finally Sunday! :-)”); break;
    default:
            System.out.println(“Unknown input”); break;
}
```

switch Multiple-Selection Statement Example – Compact Alternative

- String day = “TueSday”; day = day.toLowerCase();

```
switch (day) {
    case “monday”:
    case “wednesday”:
            System.out.println(“Evening classes”); break;
    case “tuesday”:
    case “thursday”:
    case “friday”:
            System.out.println(“No classes”); break;
    case “saturday”:
            System.out.println(“Whole day classes”); break;
    case “sunday”:
            System.out.println(“It's finally Sunday! :-)”); break;
    default:
            System.out.println(“Unknown input”); break;
}
```

There is no “;” !

Summary

- The if selection statement
- The if... else selection statement
- Boolean expresions
- Equality and relational operators
- Conditional operators
- Nested if... else selection statement
- The switch selection statement

Resources

- Java Language Keywords
  - http://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html