Algoritma dan Struktur Data

Leon Andretti Abdillah

07

Control Flow Statements Selection by Using If and Else

Introduction

- Generally, there are three structure of an program, namely:
  - Sequence - execute one statement after another.
  - Choice/Decision/Selection - execute only one several statements depending on a condition (if and switch statements).
  - Repetition - execute some statements repetitively (while, for, do…while).

Introduction

- The statements inside your source files are generally executed from top to bottom (sequentially, from one statement to the next), unless you divert the flow using a flow-of-control construct.), in the order that they appear.
- *Control flow statements*, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to *conditionally* execute particular blocks of code.
- This section describes:
  - the decision-making statements (if-then, if-then-else, switch),
  - the looping statements (for, while, do-while), and
  - the branching statements (break, continue, return) supported by the Java programming language.

Decision, Conditional, Choice, Branch

- The if-then statement is the most basic of all the control flow statements. It tells your program to execute a certain section of code *only if* a particular test evaluates to true.
- The **if-then** and **if-then-else** conditional statements let a Java program make simple decisions about what to do next. They work in the same logical way as we do when making decisions in real life.
- For example, when making a plan with a friend, I could say "If Mike gets home before 5.00pm then we'll go out for an early dinner." When 5.00pm arrives, the condition (i.e., Mike is home), which determines whether we all go out for an early dinner, will either be true or false. It's works exactly the same in Java.

The syntax

- The simplest such construct in Java is the if statement, which reads like "if something is true, then do this before continuing." It is written:

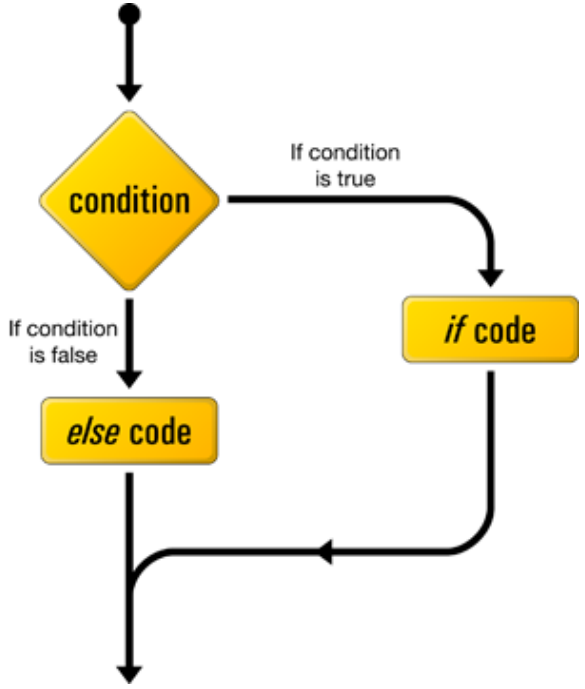
```
if (<Boolean expression>) {
        <Embedded statements>;
 }
```

Or

```
if (<Boolean expression>){
     <then-statements>;
 }
 else if {
    <else-statements>;
}
 else  {
    <else-statements>;
}
```

Or

```
if (<Boolean expression>){
     <then-statements>;
 }
 else {
    <else-statements>;
}
```



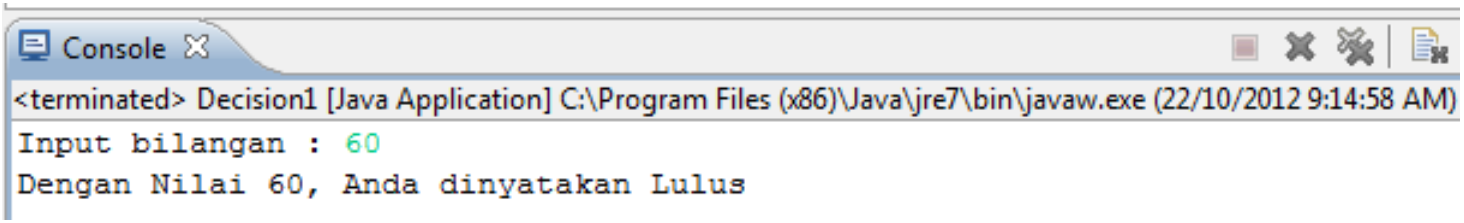
Notes

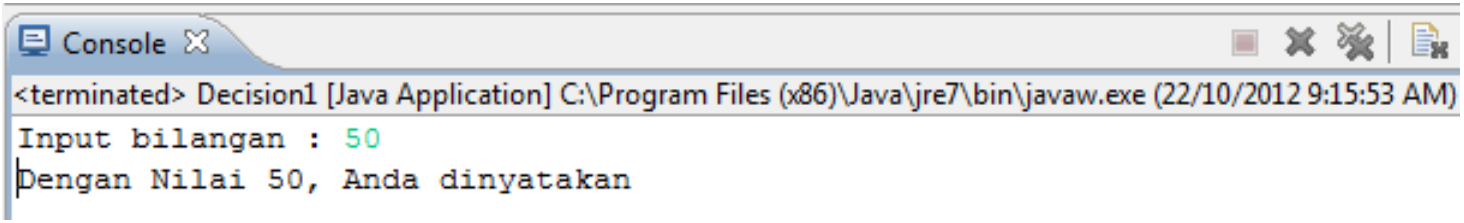
- The expression must be boolean.
- The braces are not necessary if there is only one statement, but it's a common style to always use them.

Example If with 1 condition 1/2

```
package Package04;

import java.util.Scanner;

public class Decision1 {

    public static void main(String[] args) {

        Scanner scInput = new Scanner(System.in);
        String ket = "";

        System.out.print("Input bilangan : ");
        int bil = scInput.nextInt();

        if (bil >= 55) {
            ket = "Lulus";
            }

        System.out.println("Dengan Nilai " + bil + ", Anda dinyatakan " + ket);
    }
}
```

Example If with 1 condition 2/2

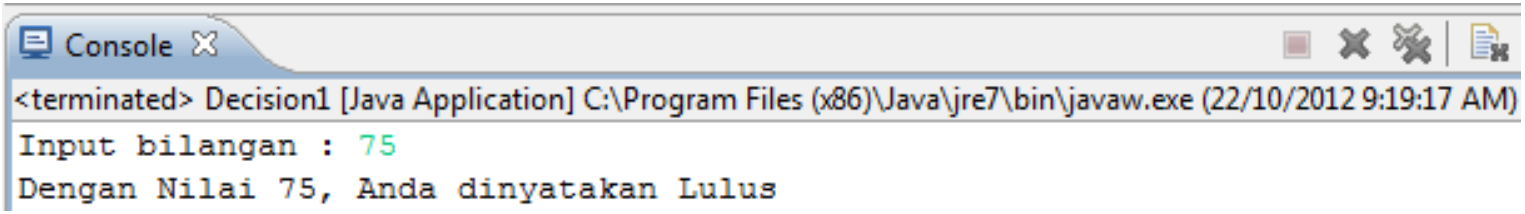


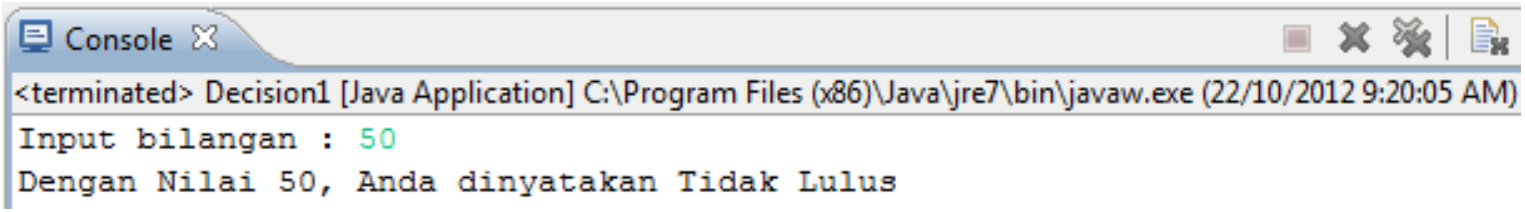


Example If with 2 conditions 1/2

```
package Package04;

import java.util.Scanner;

public class Decision1 {

    public static void main(String[] args) {

        Scanner scInput = new Scanner(System.in);
        String ket = "";

        System.out.print("Input bilangan : ");
        int bil = scInput.nextInt();

        if (bil >= 55) {
            ket = "Lulus";
            }
        else {
            ket = "Tidak Lulus";
        }

        System.out.println("Dengan Nilai " + bil + ", Anda dinyatakan " + ket);
    }
}
```

Example If with 2 conditions 2/2





The Equality and Relational Operators

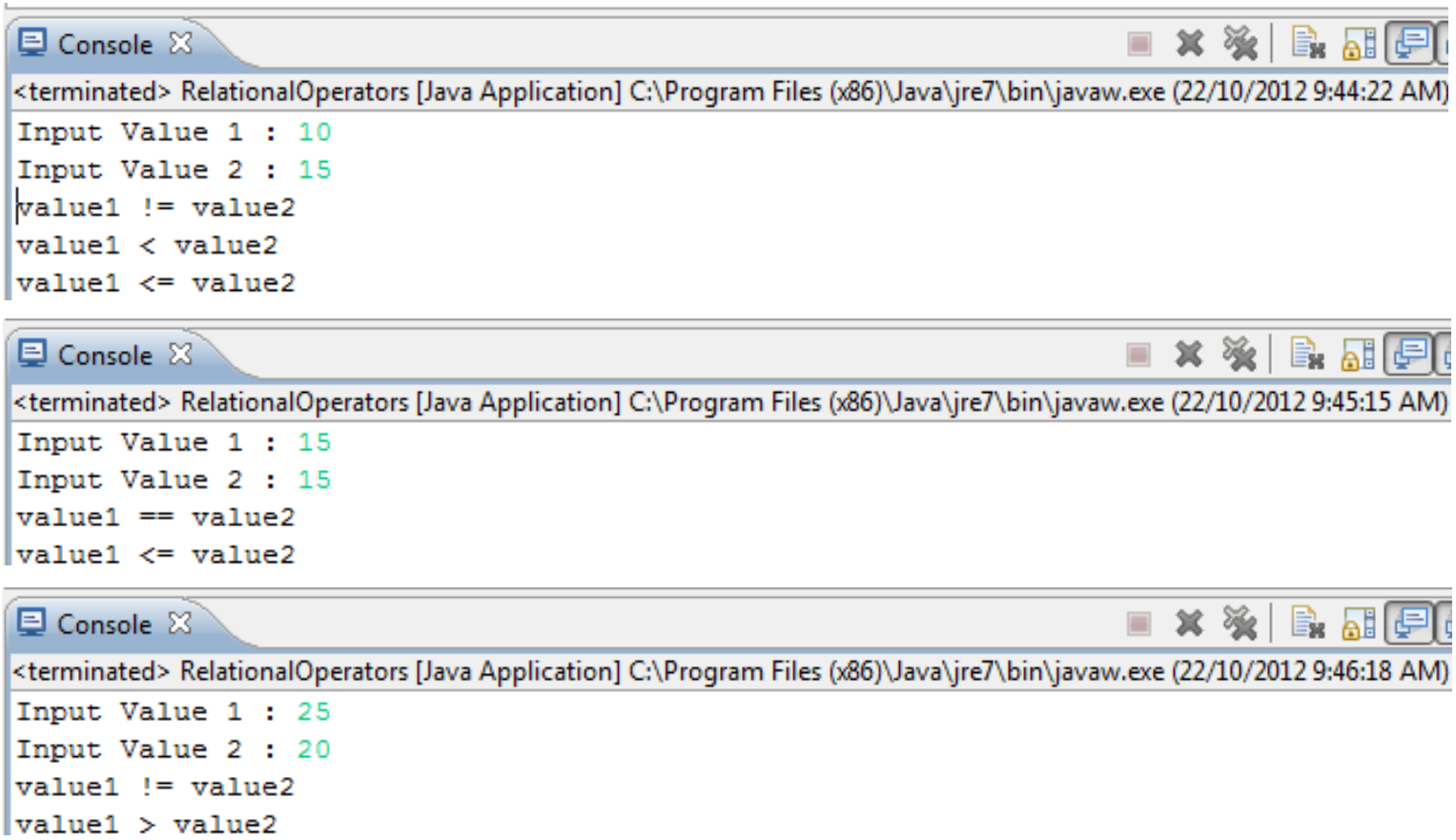
- The equality and relational operators determine if one operand is greater than, less than, equal to, or not equal to another operand. The majority of these operators will probably look familiar to you as well. Keep in mind that you must use "==", not "=", when testing if two primitive values are equal.

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

Example – Operational Operators 1/2

```
package Package04;

import java.util.Scanner;

public class RelationalOperators {

    public static void main(String[] args){

        Scanner scIn =  new Scanner(System.in);

        System.out.printf("%s : ", "Input Value 1");
        int value1 = scIn.nextInt();
        System.out.printf("%s : ", "Input Value 2");
        int value2 = scIn.nextInt();

        if(value1 == value2)
            System.out.println("value1 == value2");
        if(value1 != value2)
            System.out.println("value1 != value2");
        if(value1 > value2)
            System.out.println("value1 > value2");
        if(value1 < value2)
            System.out.println("value1 < value2");
        if(value1 <= value2)
            System.out.println("value1 <= value2");
    }
}
```

Example – Operational Operators 2/2



The Conditional (Logical) Operators

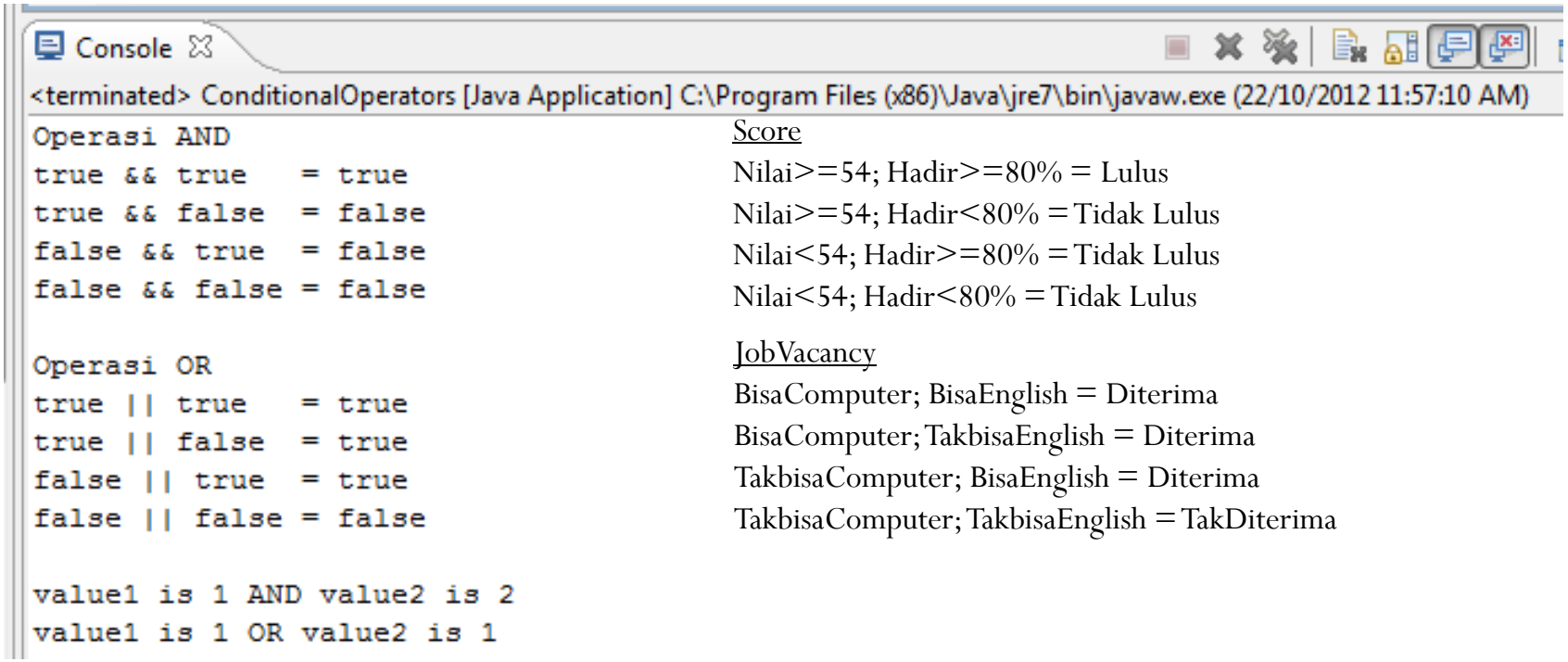
- The && and || operators perform *Conditional-AND* and *Conditional-OR* operations on two boolean expressions.
- These operators exhibit "short-circuiting" behavior, which means that the second operand is evaluated only if needed.

Operator	Meaning
&&	Conditional-AND
||	Conditional-OR

Example Conditional Operators 1/2

```
package Package04;

public class ConditionalOperators {

    public static void main(String[] args){

        System.out.println("Operasi AND");
        System.out.println("true && true   = " + (true && true));
        System.out.println("true && false  = " + (true && false));
        System.out.println("false && true  = " + (false && true));
        System.out.println("false && false = " + (false && false));

        System.out.println("\nOperasi OR");
        System.out.println("true || true   = " + (true || true));
        System.out.println("true || false  = " + (true || false));
        System.out.println("false || true  = " + (false || true));
        System.out.println("false || false = " + (false || false));

        int value1 = 1;
        int value2 = 2;

        if((value1 == 1) && (value2 == 2))
            System.out.println("\nvalue1 is 1 AND value2 is 2");
        if((value1 == 1) || (value2 == 1))
            System.out.println("value1 is 1 OR value2 is 1");
    }
}
```

Example Conditional Operators 2/2



Score

Nilai>=54; Hadir>=80% = Lulus

Nilai>=54; Hadir<80% = Tidak Lulus

Nilai<54; Hadir>=80% = Tidak Lulus

Nilai<54; Hadir<80% = Tidak Lulus

JobVacancy

BisaComputer; BisaEnglish = Diterima

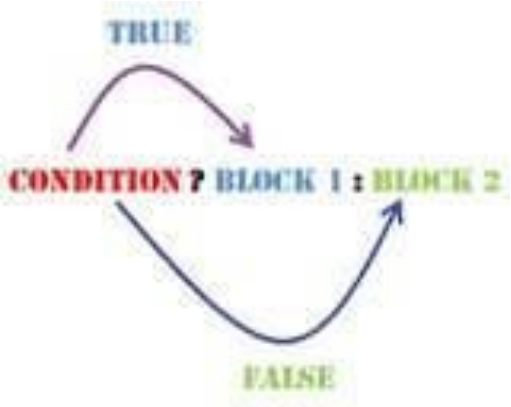
BisaComputer; TakbisaEnglish = Diterima

TakbisaComputer; BisaEnglish = Diterima

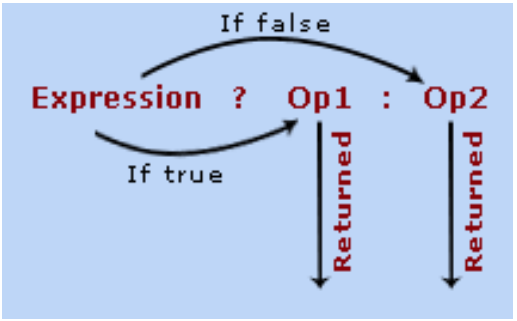
TakbisaComputer; TakbisaEnglish = TakDiterima

The Ternary Operator (? :)

- Another conditional operator is **? :**, which can be thought of as shorthand for an if-then-else statement.
  - `boolean expression ? operand1 : operand2;`
- This operator is also known as the *ternary operator* because it uses three operands.
- In the following example, this operator should be read as:
  - "If someCondition is true, assign the value of value1 to result.
  - Otherwise, assign the value of value2 to result."



The Ternary Operator (? :)



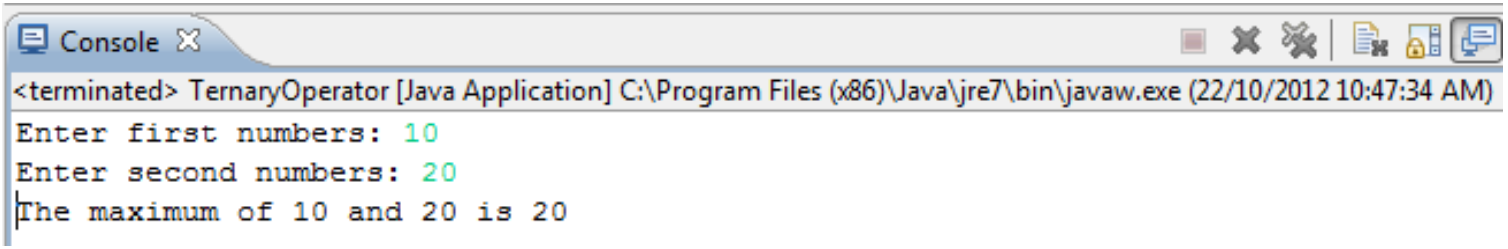

```
if (a > b) {
    max = a;
}
else {
   max = b;
}
```

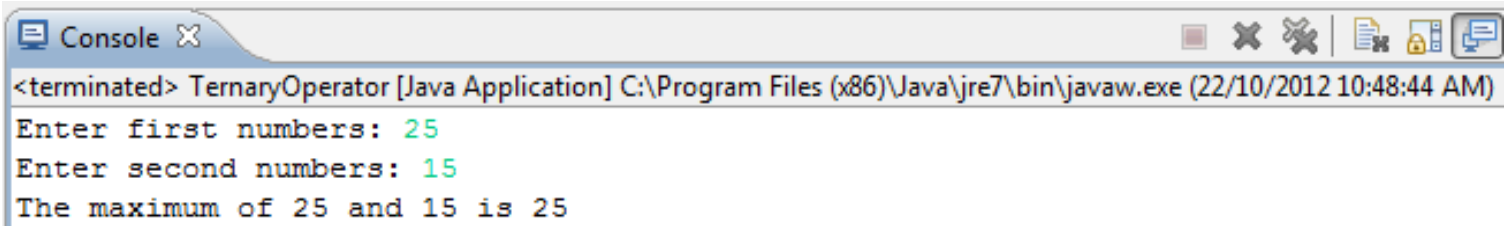
```
max = (a > b) ? a : b;
```

Example Ternary Operator 1/2

```
package Package04;

import java.util.Scanner;

public class TernaryOperator {

    public static void main(String[] args) {

        Scanner scnr = new Scanner(System.in);
        int number1 = 0;
        int number2 = 0;
        int maximum = 0;

        System.out.print("Enter first numbers: ");
        number1 = scnr.nextInt();
        System.out.print("Enter second numbers: ");
        number2 = scnr.nextInt();

        maximum = (number1 < number2) ? number2 : number1;

        System.out.print("The maximum of ");
        System.out.print(number1);
        System.out.print(" and ");
        System.out.print(number2);
        System.out.print(" is ");
        System.out.println(maximum);
    }
}
```

Example Ternary Operator 2/2





References

Abdillah, L. A. (2005). Validasi data dengan menggunakan objek lookup pada borland delphi 7.0. *Jurnal Ilmiah MATRIK, 7(1), 1-16.*

Abdillah, L. A. (2009). *Pemrograman II (Delphi Dasar) Edisi 4. Palembang: Pusat Penerbitan dan Percetakan Universitas Bina Darma (PPP-UBD) Press.*

Abdillah, L. A. (2009). *Pemrograman III (Delphi Database) Edisi 4.* Palembang: Pusat Penerbitan dan Percetakan Universitas Bina Darma.

Abdillah, L. A. (2010). Introduction to Java Programming, from http://eprints.binadarma.ac.id/3124/1/Introduction%20to%20Java%20Programming%20-%20%2001%20Introduction.pdf

Abdillah, L. A. (2013). Algorithms & Programming. Retrieved from http://blog.binadarma.ac.id/mleonaa/teaching/programming/algorithm-and-programming-2/

Abdillah, L. A. (2014). Data Structures & Algorithms. Retrieved from http://blog.binadarma.ac.id/mleonaa/teaching/programming/data-structures/

Hilfinger, P. N. (2002). Data Structures (Into Java) Retrieved from http://www.cs.berkeley.edu/~hilfingr/cs61b/f2002/public_html/data-structures.pdf

Lafore, R., & Waite, M. (2003). *Data structures & algorithms in Java (Second Edition ed.). Indianapolis, Indiana, USA: Sams Publishing.*

Mehlhorn, K., & Sanders, P. (2007). Algorithms and Data Structures *The Basic Toolbox (pp. 295).* Retrieved from http://users.dcc.uchile.cl/~nbaloian/cc3001-02/Libros/Mehlhorn-Sanders-Toolbox.pdf

ORACLE. (2015). The Java™ Tutorials from http://docs.oracle.com/javase/tutorial/java/index.html

Wirth, N. (1985). *Algorithms and Data Structures. Zurich, Switzerland.*