CHAPTER 8 JAVA OBJECT ORIENTED PROGRAMMING

You may have seen a picture of a mug with a vapor billowed with Java written on it such as in Figure 8.1. Yes, it is the logo of Java programming language that increases its popularity in recent years. Many games and applications that used in mobile equipment such as cellular phone and PDA is made using Java. Java is well known due to its portability and the support of object oriented programming.

This chapter covers two standard of competences, namely, create a program in object oriented programming language and create an application program in Java. This is due to the similarities between Java and object oriented programming. The standard of competence in creating a program in object oriented programming consists of four (4) basic competence, namely, data type and control of the program, class creation, inheritance usage, polymorphism, and overloading, and the usage of interface and package. Whereas the standard of competence in creating application program with Java consists of five (5) basic competences, namely, explaining I/O file, data type and variable, applying operator, explaining exception handling, applying multi- threading and explaining network programming.

Figure 8.1. Logo Java.

In this book, a section is indirectly referred to a basic competence. The summary is at the end of each chapter followed by exercise. Before studying this competence please review the operation system, the problem solving principles, the programming algorithm, the programming with VB and VB.Net and supporting materials from the other subjects.

OBJECTIVE

After studying this chapter, the reader is hoped to be able to:

- Understand the concept of object oriented programming.
- Explain I/O File, data type and variable in Java
- Use of operator.
- Implement program control.

- Explain Exception Handling.
- Implement Multi-Threading
- Explain Network Programming
- Create object oriented program using class
- Use inheritance principle, polymorphism and overloading
- Create object oriented program with interface and package.

8.1. OBJECT ORIENTED PROGRAMMING CONCEPT

Object Oriented Programming (OOP) is a programming paradigm that uses object and interaction to design an application and computer program. OOP is uncommon in the early 1990's. However, today it is commonly used. Programming languages such as Microsoft dotNet family (Visual Basic.Net, Visual C#, and Visual J), Borland Delphi, Java, Phyton, PHP version 5 and above, C++ and many others are OOP supported programming language.

What is an object? All objects in the world may be used as object. Even your Software Engineering teacher is an object. This software engineering handbook is also an object. Even the software engineering course is an object. Every object has certain characteristics and behavior. Characteristics is known as attribute, and behavior is known as behavior or method.

In object oriented, it is known to have class and object. Class defines the abstract characteristics of an object including attribute or characteristics or it tasks (method). For example, a car is a class as it has attributes, such as, color, maker, type etc. Car has also method, such as, forward, reverse, and stop etc. Please see Figure 8.2.

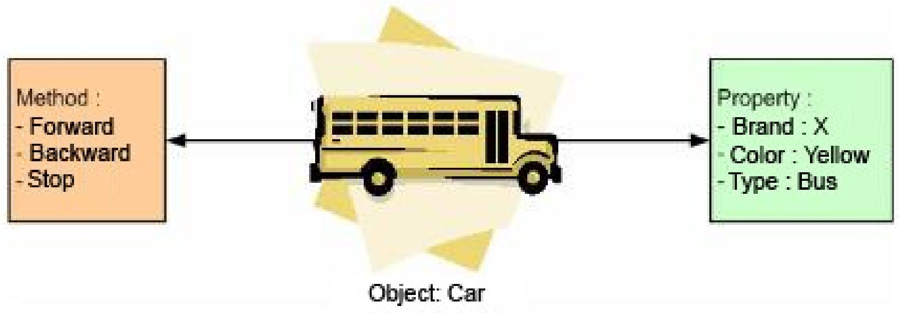


Figure 8.2. Class, Attribute and Method.

An object is an example of well define class. Attribute and method of a class is automatically inherent in an object with some certainties. As an illustration, please see Figure 8.2. In the Figure, we may easily identify that the class is a car with its attribute and method. The object may be a sedan, from Toyota, and red color. The sedan may have method to go forward, reverse, and stop. In this case, the sedan is known as instance or decedent from car class.

There are several important concepts that you must understand in object oriented programming, namely, abstraction, encapsulation, inheritance and polymorphism.

8.1.1. Abstraction

Abstraction is also known as composition is a simplification principle by modeling a complex using class in accordance to the problem. Please examine Figure 8.3, a car may be broken down into parts, such as, wheel, engine,frame, window glass, etc. as well as the other way around. In an object oriented programming, we may have several classes or object that may have many similar attributes and methods that may be merge into a super class.

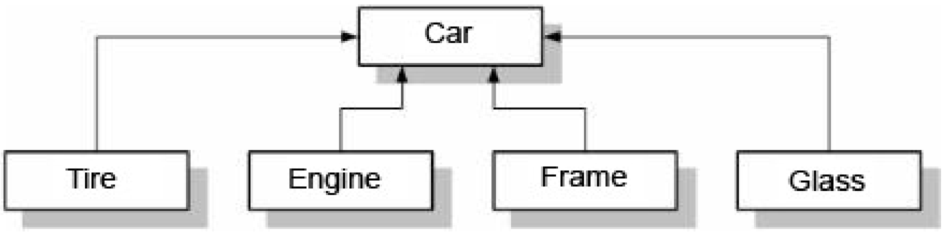


Figure 8.3. Abstraction Example.

8.1.2. Encapsulation

Encapsulation principle is to hide the detail of a class on object that interact with it. For example when we drive a car, we interact with the car and ask it to do its method, such as, forward, reverse, or stop. We interact only through some if car interface, such as, gears, throttle, break and several other parts. The detail process how to gp forward, backward, and stop, we don't need to know and understand.

8.1.3. Inheritance

Inheritance is an inheritance principles of the characteristics inheritance from its parents to the child or its descendants that is applied in the class. Parents has a more general attribute and method as compared with the child or its descendants. In Figure 8.4., it is shown that a car has its own attribute and method that are more general then that of sedan, truck, or bus. The car as the class that is inherited is known as super class, whereas sedan, truck and bus as the inherited class is known as sub class.

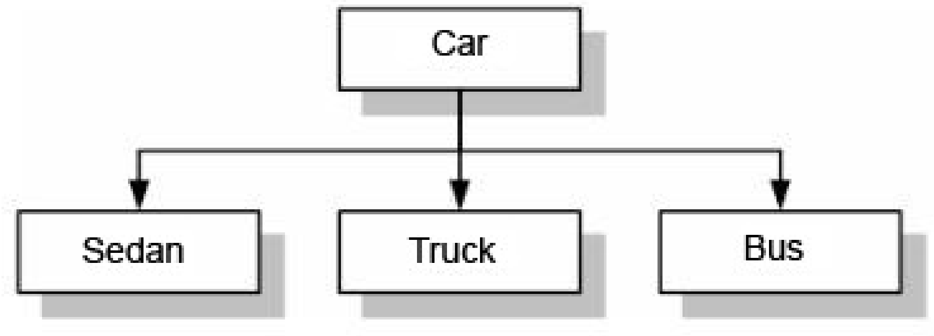


Figure 8.4. Inheritance.

8.1.4. Polymorphism

Polymorphism is possibly the most difficult concept to understand in object oriented programming. The meaning of polymorphism is the ability of an object to have more than one form. In other words, we may implement different things using the same way. For example, suppose we have four (4) different animals, such as, bird, snake, frog, and lion, if we ask them to move forward, then bird will fly, snake will crawl, frog will jump and lion will possibly run. So the same method may be implemented differently by different object.

8.2. INTRODUCTION TO JAVA

The Java programming language was born from The Green Project, that was 18 months, from early 1991 to summer 1992. This particular project used version named Oak. The name Oak is never used the name of Java release version as another software has been registered to use it as trade mark, this the replacement name is "Java".The name taken from coffee seeds. Today, Java is under the license from Sun Microsystems.

Based on Sun's definition, Java is a name for a collection of technology to create and to run software on a standalone computer or in a network environment. While people who work in programming world tend to see Java as a technology rather than a conventional programming language.

8.2.1. Software Requirement

To create a Java program, we need at least two (2) software, namely,

- Java 2 SDK Standard Edition (J2SE).

This software is used to compile the Java source code. More over, it has classes that may be used to build desktop application, graphics, security, database connectivity and networking. The software may be freely downloaded from Sun Microsystems site. After that, the software must be installed in our operating system.

- Text Editor.

  This software helps us in writing the source code. Notepad, vi, Gedit, are the example of text editor. However, there is help facilities to write Java source code in such text editor. Today, there are several free IDE available on the Internet. These IDE provide many facilities, such as, syntax colorig, auto completion, and template to make easier for one to write code in Java. NetBeans (www.netbeans.org) and Eclipse (www.eclipse.org) are the two (2) famous and powerful IDE. Figure 8.5. shows NetBeans IDE.

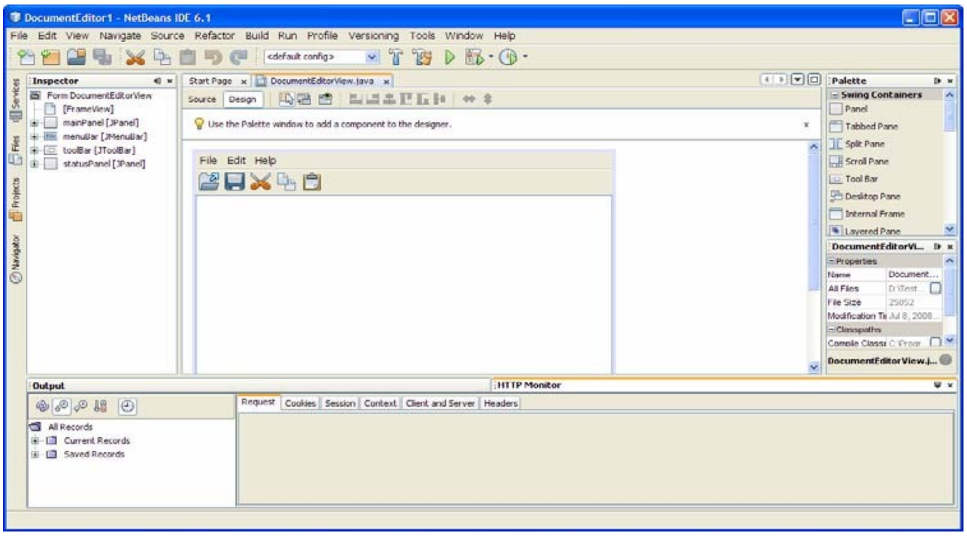


Figure 8.5. NetBeans IDE.

8.2.2. Source Code Compilation

Before create and compile a program, we need to set ClassPath. ClassPath is a system variable to let Java knows various location needed for running it. For example, we may put the source code in directory d:\TestCode\Java, then we must set the path so that we may use Java compiler from this directory. The following is the step by step process to set classpath:

- Open command-prompt in Windows then type:

  ```
  set PATH=C:\progra~1\java\jdk1.5.0\bin;%PATH%
  set CLASSPATH= . ;D:\TestCode\Java
  ```

  The above Path is typically used in JDK 1.5. We need to change slightly for other version.

- Check for correct setting by type Java command in any directory from the command prompt.

After classpath is set, try open notepad and type the following code.

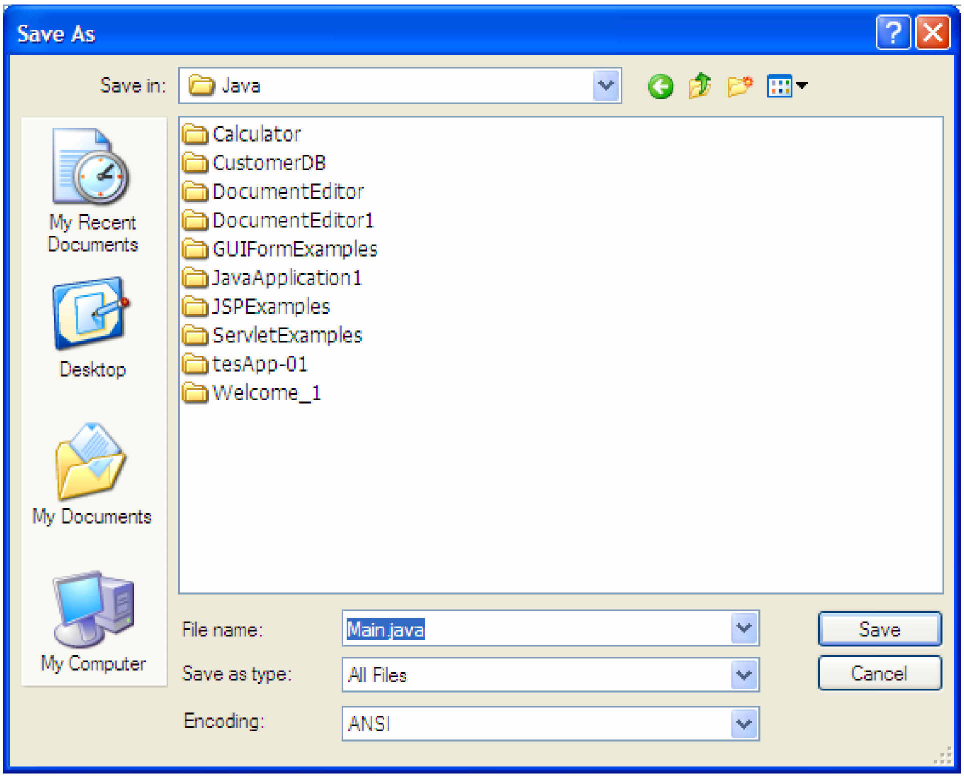
```
public class Main {
    //isi blok
    public static void main(String[] args) {
        System.out.println("Hallo ini Java lho");
    }
}
```

Then save the source code file using the same name as the class in the code, namely, main and add the extension .java. Save the code in the appointed directory, as shown in Figure 8.6. Java has a very tight writing rule of file name. If the filename of the member is not the same as the class name, the program will not be able to be executed.

In the above source code, the first line shows the class name, namely, Main. In Java all code must be placed in class declaration. We use keyword class. In addition, class with public access indicates that this class may have free access to other class in other package. A package is a collection of classes.

After class declaration follow by { sign as the beginning of the code block. It must be closed by } sign to end the block.

Line starts with // is a comment. The next line is the method. In our case the method is main (main method). We may create other methods other than main. After declaration followed by the block code. In the example block code, I t contains the System.out.println("Hallo ini Java lho"); statement. The statement System.out.println() will print the text between dual quotation marks (" ") into the screen.

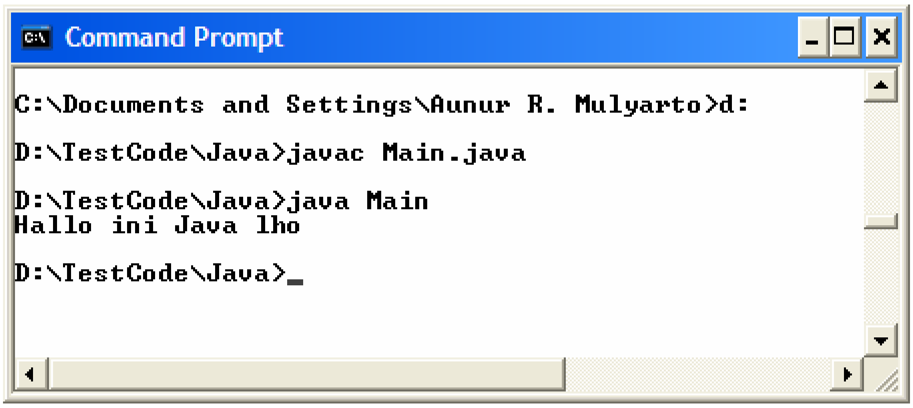


Calculator
CustomerDB
DocumentEditor
DocumentEditor1
GUIFormExamples
JavaApplication1
JSPExamples
ServletExamples
tesApp-01
Welcome_1

File name: Main.java
Save as type: All Files
Encoding: ANSI

Figure 8.6. File name and file location.

Now open the command prompt and select the directory where the file is and type the command as shown in Figure 8.7. Examine how to write it and the result.



```
C:\Documents and Settings\Aunur R. Mulyarto>d:

D:\TestCode\Java>javac Main.java

D:\TestCode\Java>java Main
Hallo ini Java lho

D:\TestCode\Java>_
```

Figure 8.7. Execute Java Program.

Prior to run, the source code (Main.java) must be compiled by using javac as shown in Figure 8.7. After a successful compilation, the program may be executed by using Java. The compilation process will produce Main.class file. Check your directory using dir command, look for the file with extension .class. The Java command calls the .class file, not the source code with extension .java.

8.3. DATA TYPE, VARIABEL, DAN INPUT/OUTPUT (I/O) STATEMENT

8.3.1. Data Type

There are 8 basic data types in Java, namely, boolean (for logic form), char (for text form), byte, short, int, long (integer), double and float (floating point). The table 8.1 shows the details of these data types.

Tabel 8.1. Data type in Java.

Data type	Comment
Logic (boolean)	Representing two (2) conditions: true and false.
Text (char)	Must be between quotes (' ')

Integer (byte, short, int & long)	Integer data type, default type is int.
	byte = 8 bits ranging from $-2^{7} – 2^{7}-1$
	short 16 bits ranging from $-2^{15} – 2^{15}-1$
	int = 32 bits ranging from $-2^{31} – 2^{31}-1$
	long = 64 bits ranging from $-2^{63} - 2^{63}-1$

Floating point (float and double)	Numbers data type, may be fractional. Default data type is double.
	Float = 32 bits ranging from $-2^{31} – 2^{31}-1$
	Double = 64 bits ranging from $-2^{63} - 2^{63}-1$

In Java, String is not a data primitive, but a Class. String represents data type consisted of several characters. String is written by using dual quotation marks (“”).

The following examples shows how to use the above data type. Type the following codes, then compile and run it.

Example 8.1. The use of integer data type.

```
public class ContohPerhitungan {
  public static void main(String[] args) {
    byte a = 1;
    short b = 12;
    int c = 300, d, e;
    d = a + b + c;
    e = a * b * c;
```

```
      System.out.println("Hasil penjumlahan = " + d);
      System.out.println("Hasil perkalian = " + e);
   }
}
```

Example 8.2. The use of float data type.

```
public class LuasLingkaran {
   public static void main(String[] args) {
     double pi = 3.1416;
     double r = 2.12;
     double luas;
     luas = pi * r * r;
     System.out.println("Luas Lingkaran = " + luas);
   }
}
```

Example 8.3. The use of char data type.

```
public class tipeChar {
   public static void main(String[] args) {
     char ch = 'A';
     System.out.println("ch = " + ch);
     ch++;
     System.out.println("ch = " + ch);
   }
}
```

8.3.2. Variable and Constant.

The naming (identifier) rule of variable and constant as written in Chapter 5 also apply for Java. Moreover the identifier in Java is case-sensitive. Similar variable with different upper case and lower case character is a different variable. Unlike Visual Basic, Java requires us to firstly declare variable and constant. If not then the source code would not be compiled.

Method to declare a variable is as follows.

```
<data type> <variable name> [= initial value];
```

The initial value is optional. Examine the example 8.1. The variable a, b, and c are initialized by the initial value. Whereas the variable d and e are initialized. Please examine example 8.2 and 8.3 on variable declaration.

8.3.3. Input / Output

In the above example, we have used one of the method to sent output to the screen, it uses the statement System out printing. We have not use command to get input. In the followings, we will study how use input and output statement in Java.

To catch input from keyboard, we must use the class Buffered Reader that is in java.io package. Thus, in the beginning of our program, we must include this class in the source code. Please examine the following example.

Example 8.4. Input statement in Java.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
public class InputKeyboard
{
    public static void main( String[] args ){
      BufferedReader dataIn = new BufferedReader(new
              InputStreamReader( System.in) );
      String name = "";
      System.out.print("Ketikkan nama anda:");
      try{
              name = dataIn.readLine();
               }catch( IOException e ){
              System.out.println("Error!");
      }
      System.out.println("Hai " + name +"!");
    }
}
```

The three (3) lines started with import shows that we will use the class BufferedReader, InputStreamReader and IOException in java.io package. Mode detailed explanation on package will be discussed in other section of this chapter.

In the following statements,

```
     BufferedReader dataIn = new BufferedReader(new
     InputStreamReader( System.in) );
```

We declare a variable named dataIn with type class BufferedReader. Then, we declare a String variable with identifier name. Those statements are used to keep user input. The variable name is initialized by using en empty String. The following line will print the string using System.out.print statement.

The following block code is a try-catch block. We will discuss on this in exception section.

```
try{
    name = dataIn.readLine();
}catch( IOException e ){
    System.out.println("Error!");
}
```

This will catch possible error in name = dataIn.readLine (); If mistake occurs that text “Error” will be printed. If no error, variable name will be the user input data via keyboard. It will be printed in the last statement.

To print the output, we may use the following statement.

```
System.out.println()
System.out.print()
```

System.out.println ()will create a new line, whereas System.out.print () will not make a new line.

8.4. OPERATOR

8.4.1. Arithmetic Operator

Arithmetics operator in Java is almost the same as VB. The only difference is in the notation of modulus operator. VB uses mod whereas Java uses % sign. Table 8.2 shows the list of arithmetics operator in Java.

Table 8.2. Arithmetics Operator in Java

Operator	Function	Example
+	Adding	3 + 5 = 8
-	Subtracting	7 – 2 = 5
*	Multiplying	5 * 2 = 10
/	Dividing	6 / 3 = 2
%	Remaining of a Dividing (modulus)	5 / 2 = 1
++	Increment by 1	C++ = C + 1
--	Decrement by 1	C-- = C - 1

The following example shows how to use the arithmetics operator. Type the following source code, compile and run. Examine the output of the program

Example 8.4. The use of Arithmetics Operator.

```
public class DemoAritmatika
{
    public static void main(String[] args)
   {
        int i = 21;
        int j = 38;
        double x = 9.123;
        double y = 12.78;
    //Cetak nilai variabel
        System.out.println("Nilai Variabel...");
        System.out.println("        i = " + i);
        System.out.println("        j = " + j);
        System.out.println("        x = " + x);
        System.out.println("          y   =    "    +    y);
    //penjumlahan
        System.out.println("Penjumlahan...");
        System.out.println("        i + j = " + (i + j));
        System.out.println("        x + y = " + (x + y));
    //pengurangan
        System.out.println("Pengurangan...");
        System.out.println("        i - j = " + (i - j));
        System.out.println("        x - y = " + (x - y));
    //perkalian
        System.out.println("Perkalian...");
        System.out.println("        i * j = " + (i * j));
        System.out.println("        x * y = " + (x * y));
    //pembagian
        System.out.println("Pembagian...");
        System.out.println("        i / j = " + (i / j));
        System.out.println("        x / y = " + (x / y));
    //modulus
        System.out.println("Sisa Hasil Bagi...");
        System.out.println("        i % j = " + (i % j));
        System.out.println("        x % y = " + (x % y));
    //increment
        System.out.println("Increment...");
        System.out.println("        i++ = " + (i++));
        System.out.println("        ++i = " + (++i));
        System.out.println("        j++ + i = " + (j++ + i));
        System.out.println("        ++j + i = " + (++j + i));
    }
}
```

8.4.2. Relational Operator

Relational or comparison operator in Java is also similar to VB. The only different in symbol same or not the same. In VB to compare two operand whether the same or not the same uses the operator = for the same, and < > for not the same. Whereas in java uses == for the same, and ! = for not the same. Type in the following example source code, compile and run it. Examine the output of the program.

Example 8.5. The use of Relational Operator.

```
public class DemoRelasional
{
    public static void main(String[] args) {
        int i = 20;
        int j = 16;
        int k = 16;
   //Cetak nilai variabel
        System.out.println("Nilai variabel...");
        System.out.println("        i = " + i);
        System.out.println("        j = " + j);
        System.out.println("        k = " + k);
   //lebih besar dari
        System.out.println("Lebih besar dari...");
        System.out.println("        i > j = " + (i > j));
        System.out.println("        j > i = " + (j > i));
        System.out.println("        k > j = " + (k > j));
   //lebih besar atau sama dengan
        System.out.println("Lebih        besar        atau     sama
dengan...");
        System.out.println("        i >= j = " + (i >= j));
        System.out.println("        j >= i = " + (j >= i));
        System.out.println("        k >= j = " + (k >= j));
   //lebih kecil dari
        System.out.println("Lebih kecil dari...");
        System.out.println("        i < j = " + (i < j));
        System.out.println("        j < i = " + (j < i));
        System.out.println("        k < j = " + (k < j));
   //lebih kecil atau sama dengan
        System.out.println("Lebih        kecil        atau     sama
dengan...");
        System.out.println("        i <= j = " + (i <= j));
        System.out.println("        j <= i = " + (j <= i));
        System.out.println("        k <= j = " + (k <= j));
   //sama dengan
        System.out.println("Sama dengan...");
        System.out.println("        i == j = " + (i == j));
        System.out.println("        k == j = " + (k == j));
    //tidak sama dengan
        System.out.println("Tidak sama dengan...");
        System.out.println("        i != j = " + (i != j));
        System.out.println("        k != j = " + (k != j));
    }
}
```

8.4.3. Logical Operator

There are three (3) logical operators in Java, namely, && (AND), || (OR), | and! (NOT). The use of these operators are the same as VB, only different in notation.

8.5. PROGRAM CONTROL STRUCTURE

As in VB, Java provides the control program structure for selection and looping. The statements is also fairly similar.

8.5.1. Selection Structure

The selection structure may use if, if... else, and if... else... if. It is fairly similar to Chapter 5 and Chapter 7. Please examine the following program snapshots.

Example 8.6. The use of if.

```
int nilai = 68;
if( nilai > 60 ) System.out.println("Selamat anda lulus!");
```

Example 8.6 uses if as selection structure. If the mark is more than 60 then the program will print “Selamat anda lulus!”

Example 8.7. The use of if ... else.

```
int nilai = 68;
if( nilai > 60   )        System.out.println("Selamat anda lulus!");
else System.out.println("Anda tidak lulus!");
```

In Example 8.7, we use the structure if... else. If the mark is more than 60 then the output will be “Selamat you passed! ” but if not (else) then the program will print “You are not pass! ”.

Example 8.8. The use of if ... else ... if.

```
int nilai = 68;
if( nilai > 90 ){
        System.out.println("Your mark is very good!");
}
else if( nilai > 60 ){
        System.out.println("Your mark is good!");
}
else{
        System.out.println("You are not passed");
}
```

Example 8.8 is the evolution from example 8.7. If the mark is more than 90 then the program will print “Your mark is very good! ”, but if less than 90 and more than 60 (else if) then the program will print “Your mark is good! ” and if not both of them (else) then the program will print “Your are not pass”

The selection structure provides many alternatives. However, the if structure may be quite complex. Java provides switch command. It is similar to Select .. case in VB. Examine the following example.

Example 8.9. The use of switch.

```
public class SwitchControl {
  public static void main(String[] args) {
    int a=2;
    int b;
    switch(a) {
      case 1:
         b = a + 1;
         break;
      case 2:
         b = a + 2;
         break;
      case 3:
         b = a + 3;
         break;
      case 4:
         b = a + 4;
         break;
      default: b = 0;
    }
    System.out.println("Nilai b: " + b);
  }
}
```

What is the results of the above source code in Example 8.9. If your answer is 4 means you have understand how the switch command works. Switch will check whether a value is the same as case, namely, 2. Inspection starts at the first case, I,e, 1. Break statement is used to stop matching to other cases. Try to eliminate break statement and run the source code again. How is the result?

8.5.2. Looping Structure

There are three (3) looping structure in Java, namely, for, while and do- while. In principle, the looping structure is similar to the one described in Chapter 5 and 7. The following is example of looping structure.

Example 8.10. The use of for in Java.

```
public class ForLoop {
  public static void main(String[] args) {
    int j=4;
    for (int x=0; x < 5; x++) {
      System.out.println("Nilai x: " + x);
      System.out.println("Nilai j: " + j);
      System.out.println();
      j--;
    }
  }
}
```

General syntax of for is: for (start value; condition; increment) follows by the block to be repeated. Examine how to use for structure as in Example 8.10. Initial value of x=0. Whereas x < 5 is the condition for looping.. Statement x++ is to increment by 1. Remembered that x++ is the same as x = x + 1. Please run the above program. What do you think the output would be?

Example 8.11. The use of While in Java.

```
public class WhileLoop {
  public static void main(String[] args) {
     int y = 4;
     while ( y > 0 ){
       System.out.print(y);
       y--;
     }
  }
}
```

In the example 8,11 we use while statement to do looping. In while structure, we need to initialize the variable before entering the while structure. In this case, the variable y is initialized to 4. The while condition to be met is y>0. In this example, decrement counter is used, examine the y --. Thus, the program will print 4321 on screen. What happen if the counter y – is removed? How many times looping will be performed?

Example 8.12. The of do-while in Java.

```
public class ContohDoWhile {
  public static void main(String[] args) {
    int z=3;
    do {
      System.out.println("Java");
      z++;
    } while (z < 6);
  }
}
```

Example 8.12 shows how to use do-while to repeatedly print “Java” on screen. Examine the syntax of do-while in this example. Do-while needs initialization and counter to do the looping. Try to run the program. How many times “Java” will appear? Now substitute the condition in while with z < 1. Is the program printing “Java”? Why? Read Chapter 7 in the section of control structure to understand this issue.

8.5.3. The use break and continue.

The break statement has three (3) functions, namely,

- Stopped selection process in switch statement.
- Stopped looping process or exit the loop body.
- Exit from certain labeled block.

The break statement may be used to break selection process in switch structure. We will study how to break a loop. Examine the following example.

Example 8.13. The use of break in a loop.

```
class BreakPengulangan {
   public static void main(String[] args) {
    for (int i=0; i<10; i++) {
      if (i == 5) {
         break;
      }
```

```
        System.out.println("baris ke-" + i);
      }
      System.out.println("Ini setelah break pengulangan”);
    }
}
```

In Example 8.13, the looping will be totaling 10 times, and is begin from 0 to 9. However, due to statement if (1 == 5) {break;} then the loop will stop when i =5. The program will exit the loop body and continue with statements after the ending sign of the loop.

The statement continue is used to force the program to continue the looping process. Examine the following example.

Example 8.14. The use of continue.

```
String nama[] = {"Joni", "Riko", "Denis", "Riko"};
int hitung = 0;
for( int i=0; i<names.length; i++ ){
        if( !nama[i].equals("Riko") ){
                continue;
        }
        hitung++;
}
System.out.println("Ada " + hitung + " Riko dalam daftar");
```

Example 8.14 is a program to count the number of Riko in the collection of name. The statement if (! Names [i] .equals (“Riko”) means that if the variable name is not “Riko” then follows continue statement. Continue statement forces the program to directly looping without executing the below statements. Thus, the line hitung++ will not be executed. So that the final result is 2.

8.6. EXCEPTION HANDLING

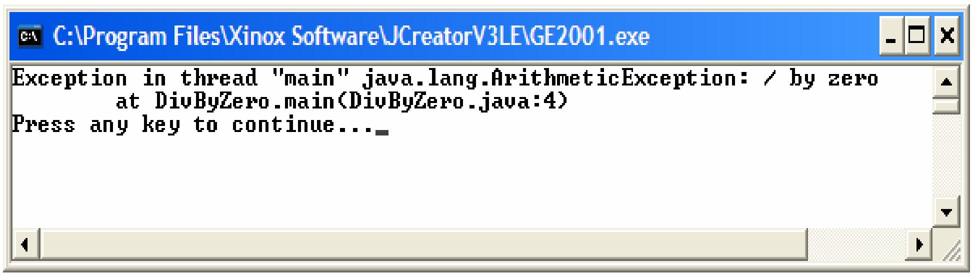
8.6.1. The understanding of Exception Handling

Mistakes in a program is fairly normal, even if it is written by a professional programmer. To reduce time in finding the mistakes, Java provides the mechanism for exception handling. Exception shorts for Exceptional Events. Its general definition, Exception is an abnormal condition that occurs during runtime. Runtime error during program execution will be trapped by exception. Exception may be automatically generated by Java runtime or deliberately generated by certain code for exception handling.

Examine the following code.

```
public class DivByZero {
        public static void main(String args[]) {
                int a = 5, b = 0, c;
                c = a/b;
                System.out.println("c = " + c);
        }
}
```

In the above source code, there is nothing wrong with the syntax. However, there is a fatal mistake, namely, divide by zero mistake. As a = 5 is divided b = 0. This type of mistake often happens as it is quite invisible or we are not conscientious. In this type of error / mistake, we may successfully compile the program. However, it will stop running and shows the message of the occurrence exception or the abnormal condition as shown in Figure 8.8 and the program will stop.



```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at DivByZero.main(DivByZero.java:4)
Press any key to continue...
```

Figure 8.8. Warning of Runtime Error.

As shown in Figure 8.8, it informs the exception type and the exception location at which line. This is the default action when an exception is nor handled. If there is no code to handle an exception, default action will be automatically used.

There are several general exception type, namely,

- ArithmeticException. Exception due to arithmetic related error, such as, divide by zero.
- ArrayIndexOutOfBoundException. Exception due to read Array index outside its limit.
- NullPointerException. Mistakes due to no value / null in the pointer.
- Etc.

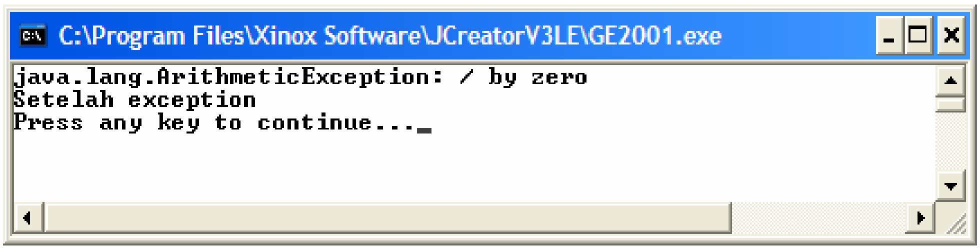
8.6.2. Try and Catch

Try is used to check on the block that may contain exception. If during runtime there is exception in this block it will be redirected to block that catch the exception that written using catch statement. Examine the following example.

Example 8.15. Exception with Try-Catch.

```
public class DivByZero {
        public static void main(String args[]) {
                int a = 5, b = 0, c;
                try {
                        c = a/b;
                } catch (ArithmeticException exc) {
                                //Reaksi jika terjadi exception
                                System.out.println(exc);
                }
                System.out.println("Setelah exception");
        }
}
```

The source code in Example 8.15 is an expansion from the previous source code. The statement c = a/b is the statement to be tested whether contains exception or not. When exception occurs it will be thrown to the catch code. Exception is checked by ArithmeticException. The reaction occurs when an exception happens while running the following System.out.println(exc); statement. Using the above mechanism, the program will not be forced to terminate, in fact, after the try-catch block, it will continue running. Figure 8.9. shows the program execution result. Compare it to the Figure 8.8.



```
java.lang.ArithmeticException: / by zero
Setelah exception
Press any key to continue..._
```

Figure 8.9. Output from Try-Catch.

In general, there may be more than one exception in one problem. In Example 8.16, we may use several exception type to check possible exception. Type the following source code, run and examine what happen.

Example 8.16. Exception with Try-Catch.

```
class BanyakEksepsi {
   public static void test(int a, int b) {
      try {
        int c = a / b;
        System.out.println("Hasil bagi: " + c);
        int[] Arr = {1,2,3,4,5}; // array dengan 5 elemen
        Arr[10] = 11; // mengakses indeks ke-10
      } catch (ArithmeticException ae) {
        System.out.println("Terdapat pembagian dengan 0");
        System.out.println(ae);
      } catch (ArrayIndexOutOfBoundsException oobe) {
        System.out.println("Indeks di luar rentang");
        System.out.println(oobe);
      }

  public static void main(String[] args) {
      test(4, 0); // menimbulkan ArithmeticException
      System.out.println();
      test(12,              4);             //            menimbulkan
ArrayIndexOutOfBoundsException
  }
}
```

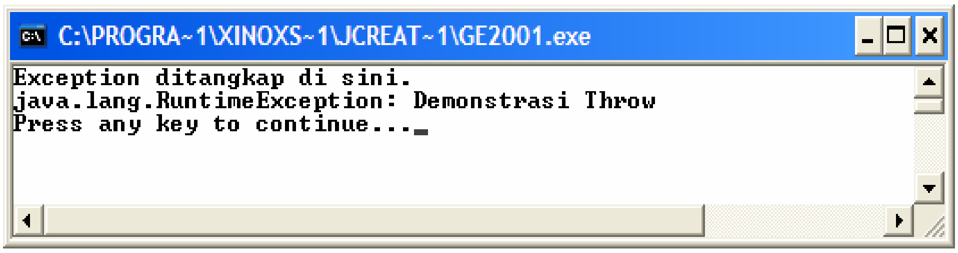
8.6.3. Throw

In addition to catch an exception, Java permits one to throw an exception. Examine the following Example 8.17.

Example 8.17. Exception with Try-Catch and Throw.

```
class ThrowDemo {
        public static void main(String args[]){
                String input = "Salah input";
                try {
                        if (input.equals("Salah input")) {
                              throw  new RuntimeException("Demonstrasi Throw");
                        } else {
                              System.out.println(input);
                        }
                        System.out.println("Setelah throw");
                        } catch (RuntimeException e) {
                              System.out.println("Exception ditangkap di sini.");
                              System.out.println(e);
                }
        }
}
```

Examine the statement starts with if statement. As shown clearly, if the input is the same as “Salah input” then threw exception and print “Demonstrasi Throw”. Output from this program is shown in Figure 8.10.



```
Exception ditangkap di sini.
java.lang.RuntimeException: Demonstrasi Throw
Press any key to continue...
```

Figure 8.10. Output program using throw.

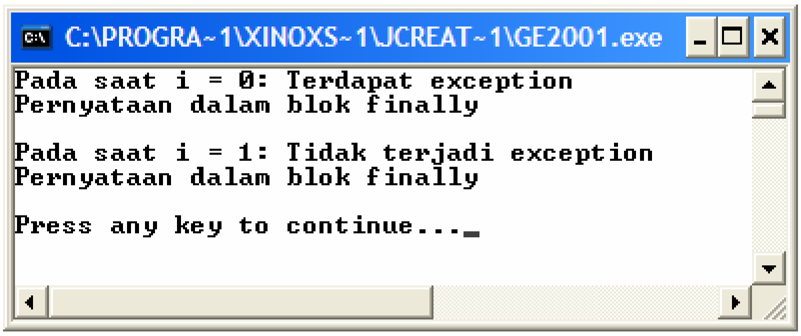
8.6.4. Finally

The finally block contains the handling code to be used after try and catch. This code block always executed after block Try. This code block will return value true although it may executed return, continue or break. Examine the following program.

Example 8.18. Exception with try-catch-finally.

```
class DemoFinally {
   private static int i = 0;
   public static void main(String[] args) {
      while (true) {
         try {
            System.out.print("Pada saat i = " + i + ": ");
            if (i++ == 0) {
              throw new Exception(); // melempar exception
            }
            System.out.println("Tidak terjadi exception");
           } catch (Exception e) {
            System.out.println("Terdapat exception");
           } finally {
            System.out.println("Pernyataan dalam blok finally\n");
            if (i == 2) {
              break;       // pada saat i==2, pengulangan akan berhenti
            }
         }
      }
   }
}
```

When the program runs, the output appears as Figure 8.11. Please note that statements in block finally will always be executed.



```
Pada saat i = 0: Terdapat exception
Pernyataan dalam blok finally

Pada saat i = 1: Tidak terjadi exception
Pernyataan dalam blok finally

Press any key to continue..._
```

Figure 8.11. Output of program with Try-Catch-Finally. 8.7.1. Concept of Thread

8.7. MULTI-THREADING

8.7.1. Concept of Thread

A thread is a control of program flow. Let us imagine, a threat as a process to be executed by a certain program. Thread is an independent program, not depends on other section of the program and may be simultaneously run. It means a threat may stop, or paused without stopping other thread. In Java, every threat is controlled by unique object of the Thread, is defined in java.lang package.

When a Java program is run, it, in fact, automatically runs a thread. This particular thread is normally called main thread. The main thread is the mother of all other thread. Despite the main thread is automatically run, it may be controlled through the object thread by using currentThread() method. Please examine the following example.

Example 8.19. Main Thread.

```
class ThreadUtama {
  public static void main(String[] args)
    throws InterruptedException {
    // mendapatkan thread yang sedang aktif
    Thread tUtama = Thread.currentThread();
    // menampilkan informasi tentang thread
    System.out.print("Informasi thread: ");
    System.out.println(tUtama.toString());
    for (int i=0; i<5; i++) {
      System.out.println("Detik ke-" + (i+1));
      Thread.sleep(1000); // membuat delay selama 1 detik
    }
  }
}
```

In the example above, we name the thread as tUtama. This variable is used to catch the running main thread using Thread.currentThread (). Then the information about this thread is printed on the screen. In the next line starts with for, we will use command to control running thread. We use sleep method to control thread to postpone a job for 1 second for each repetition time. Please type the code, then run it. Then try to remove the Thread.sleep line (1000);. Run the program again. What is the different?

8.7.2. Thread Creation and Usage

Thread may be created in two methods, namely, create a new class that apply interface Runnable or create a new class as a decedent of Thread class. These two (2) methods need java.lang package. By default this package has been automatically imported when we made a program in Java.

In this section, we will only discuss the first method. Whereas the second method will be studied in the multi-thread section. Please examine the following example.

Example 8.20. Create thread using interface Runnable.

```
class TestRunnable implements Runnable {
  // mengimplementasikan method run() yang dideklarasikan
  // di dalam interface Runnable
  public void run() {
    System.out.println("Thread anak dieksekusi");
  }
}
class PenerapanRunnable {
  public static void main(String[] args) {
    // (LANGKAH KE-1): membuat objek Runnable
    TestRunnable obj = new TestRunnable();
    //   (LANGKAH   KE-2):   membuat   objek  Thread   dengan
melewatkan objek Runnable
    Thread t = new Thread(obj);
    // (LANGKAH KE-3) : menjalankan thread
    t.start();
    System.out.println("Thread utama dieksekusi");
  }
}
```

In the above example, we create class TestRunnable that appliy Runnable. Please note the line class TestRunnable implements Runnable and its block code beneath it.

Then, we create object TestRunnable from this class, see the line TestRunnable obj = new TestRunnable (). This object is used to create new thread using constructor class thread, see the line Thread t = new Thread (obj). After created then could run the thread, examine the line t.start ().

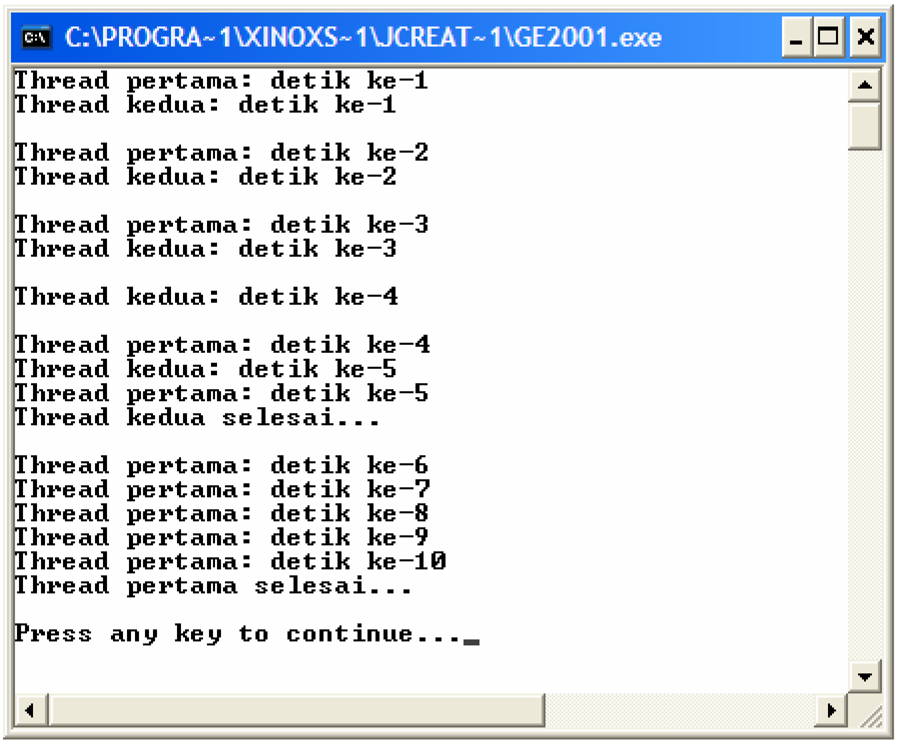
8.7.3. Multi-Thread

In Example 8.19 and 8.20, we are using one and two threads. However, Java allows us to create more than two (2) threads. This is known as Multi- thread. Examine the following example.

Example 8.21. Create a multi-thread.

```
class MyThread1 extends Thread {
  public void run() {
    try {
      for (int i=0; i<10; i++) {
        System.out.println("Thread pertama: detik ke-" + (i+1));
        if (i != 9) {
          sleep(1000);
        } else {
          System.out.println("Thread pertama selesai...\n");
        }
      }
    } catch (InterruptedException ie) {
      System.out.println(ie.getMessage());
    }
  }
}
class MyThread2 extends Thread {
  public void run() {
    try {
      for (int i=0; i<5; i++) {
        System.out.println("Thread kedua: detik ke-" + (i+1));
        if (i != 4) {
          System.out.println();
          sleep(1000);
        } else {
          System.out.println("Thread kedua selesai...\n");
        }
      }
    } catch (InterruptedException ie) {
      System.out.println(ie.getMessage());
    }
  }
}
class DemoMultipleThread {
  public static void main(String[] args) {
    MyThread1 t1 = new MyThread1();
    t1.start();
     MyThread2 t2 = new MyThread2();
     t2.start();
  }
}
```

The above source code shows how to create two (2) threads. The first thread is created in class MyThread1 by deriving from class Thread. This method is the second method as describe in the previous section. The second Thread using class MyThread2 is also created using the same method. Then in class DemoMultipleThread, we create object t1 from class MyThread1 and object t2 from class MyThread2. As we run the program, the following results will be presented as shown in Figure 8.12.



```
Thread pertama: detik ke-1
Thread kedua: detik ke-1

Thread pertama: detik ke-2
Thread kedua: detik ke-2

Thread pertama: detik ke-3
Thread kedua: detik ke-3

Thread kedua: detik ke-4

Thread pertama: detik ke-4
Thread kedua: detik ke-5
Thread pertama: detik ke-5
Thread kedua selesai...

Thread pertama: detik ke-6
Thread pertama: detik ke-7
Thread pertama: detik ke-8
Thread pertama: detik ke-9
Thread pertama: detik ke-10
Thread pertama selesai...

Press any key to continue..._
```

Figure 8.12. Execution result of multi-thread.

8.8. OBJECT ORIENTED PROGRAMMING APPLICATION USING JAVA

In the previous examples as well as the explanations, we have touched on class and object. In this section, we will study more detailed on class and object to implement object oriented programming principles.

Class may be defined as the framework that defines variables, general methods of a certain object. In object oriented programming, class is not that different from data type primitive. The difference is data type is used to declare normal variable, while class is used to declare object variable. Class is more abstract.

8.8.1. Class Creation

In Java, class is defined by keyword class. The general form in defining class is as follows.

```
class NamaKelas
      tipe data1;
      tipe data2;
      ...
      tipe dataN;
      tipe method1    (daftar parameter) {
            //blok   kode untuk method1
      }
      tipe method2    (daftar parameter) {
            //blok   kode untuk method2
      }
      ...
      tipe methodN    (daftar parameter) {
            //blok   kode untuk methodN
      }
}
```

Data or variable defined in a class is often known as instance variable. The values of these variable may be read through available methods. Therefore method is used as interface between the class user and the data in the class. Remember the encapsulation principle in the beginning of the chapter. Examine the following class example.

Example 8.22. Create a simple class.

```
class Siswa
{
        String name;
        String alamat;
        int usia;
}
```

In the above code, we create a class named Student. There are three (3) data in the class, namely, name, address and age. We have not add any method yet. In the above code, we have defined a new data called Student. The above source code is only a template. Thus, nothing will happen if we run the above program. We need to create the actual object using the above class. Please see the following.

Example 8.23. The use of class.

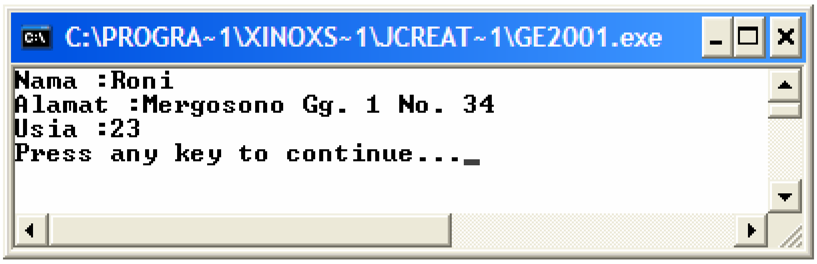
```
class Siswa {
        String nama;
        String alamat;
        int usia;
}

public class DataSiswa {
       public static void main(String[] args) {
              Siswa siswa1 = new Siswa();
              siswa1.nama = "Roni";
              siswa1.alamat = "Mergosono Gg. 1 No. 34";
              siswa1.usia = 23;
              System.out.println("Nama :" + siswa1.nama);
              System.out.println("Alamat :" + siswa1.alamat);
              System.out.println("Usia :" + siswa1.usia);
       }
}
```

The source code must be saved in DataSiswa.java file, and will open Siswa.java. This is because the main method is in class DataSiswa. In the above code, the class Siswa is in class DataSiswa. We made the actual object from class Student by typing

```
            Siswa siswa1 = new Siswa();
```

Siswa1 is the name of the actual object in class Siswa. Then, we may use variable or data in class Siswa. If we run it, the above source code will show the following output as shown in Figure 8.13.

After that just we could use the variable or the available data in the student's class. If being undertaken, the program code above will produce output like to the Picture 8,13.



```
Nama :Roni
Alamat :Mergosono Gg. 1 No. 34
Usia :23
Press any key to continue..._
```

Figure 8.13. Output result of class class DataSiswa.

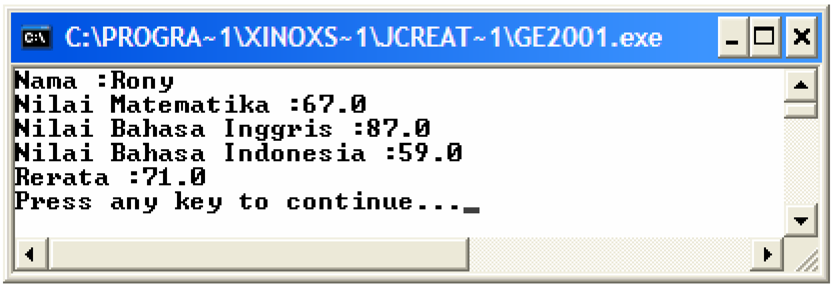
We will create a little bit complex class by including method into the class. Please examine the following example.

Example 8.23. Creating class with method.

```
class Siswa {
	String nama;
	String alamat;
	int usia;
	double nilaiMatematika;
	double nilaiBhsInggris;
	double nilaiBhsIndonesia;
	double rerata;
	// Menghasilkan nama dari Siswa
	public String getNama(){
		return nama;
	}
	// Mengubah nama siswa
	public void setNama( String temp ){
		nama = temp;
	}
	// Menghitung rata – rata nilai
	public double getRerata(){
		rerata = (
nilaiMatematika+nilaiBhsInggris+nilaiBhsIndonesia )/3;
		return rerata;
	}
}
public class DataSiswa {
	public static void main(String[] args) {
		Siswa siswa1 = new Siswa();
		siswa1.setNama("Rony");
		siswa1.nilaiMatematika = 67;
```

```
                siswa1.nilaiBhsInggris = 87;
                siswa1.nilaiBhsIndonesia = 59;
                System.out.println("Nama                  :" +  siswa1.getNama());
                System.out.println("Nilai Matematika   :" +  siswa1.nilaiMatematika);
                System.out.println("Nilai Bahasa Inggris :" +  siswa1.nilaiBhsInggris);
                System.out.println("Nilai Bahasa Indonesia :" +  siswa1.nilaiBhsIndonesia);
                System.out.println("Rerata :" + siswa1.getRerata());
        }
}
```

In the above code, we expand class Siswa by adding four (4) more variables, namely, nilaiMatematika, nilaiBhsInggris, nilaiBhsIndonesia and rerata. We also added three (3) more methods, namely, getNama, setNama and getRerata. getNama is a method to show value of variable nama. setNama is a method to set value to variable nams. GetRerata is a method to calculate the avarage value of the three (3) lessons and show the calculation result. Examine how these methods are used in class DataSiswa. The output result of the source code may be seen in Figure 8.14.



```
Nama :Rony
Nilai Matematika :67.0
Nilai Bahasa Inggris :87.0
Nilai Bahasa Indonesia :59.0
Rerata :71.0
Press any key to continue..._
```

Figure 8.14. Execution result of class with method.

There are several types of method in a class, namely, method that return no value, method that returns value and special method namely constructor. In general, method is similar to procedure or the function. Please review Chapter 6 and 7. Examine the following example.

Example 8.24. Method creation with no value in return.

```
Class Bangun {
  double panjang;
  double lebar;
  // Mendefinisikan method void (tidak mengembalikan nilai)
  void cetakLuas() {
    System.out.println(“Luas bangun = “ +
       (panjang * lebar));
  }
}
class pakaiBangun {
  public static void main(String[] args) {
    Bangun b1, b2;
    // instansiasi objek
    b1 = new Bangun();
    b2 = new Bangun();
    // mengisi data untuk objek b1
    b1.panjang = 4;
    b1.lebar = 3;
    // mengisi data untuk objek b2
    b2.panjang = 6;
    b2.lebar = 5;
    // memanggil method  cetakLuas()     untuk   masing-masing objek
    b1.cetakLuas();
    b2.cetakLuas();
  }
}
```

In the above source code, class Bangun has one method, namely, cetak Luas. This method is not returning any value. The final result of the method will be stored in the method. Examine the following exanple.

Example 8.25. Method creation that returns value.

```
Class Bangun {
  double panjang;
  double lebar;
  // Mendefinisikan method yang mengembalikan nilai
  double hitungLuas() {
      double luas = panjang * lebar;
      return luas;
  }
}
class pakaiBangun {
  public static void main(String[] args) {
     Bangun b1, b2;
     // instansiasi objek
     b1 = new Bangun();
     b2 = new Bangun();
     // mengisi data untuk objek b1
     b1.panjang = 4;
     b1.lebar = 3;
     // mengisi data untuk objek b2
     b2.panjang = 6;
     b2.lebar = 5;
```

```
    // memanggil method hitungLuas() untuk masing-masing objek
    System.out.println(“Luas b1 = “ + b1.hitungLuas());
    System.out.println(“Luas b2 = “ + b2.hitungLuas());
  }
}
```

In Example 8.25, we create a method hitungLuas that returns value. Please note that method declaration is not using void, but double as the returns value data type. To return a value, a keyword “return” is used. Compare calling method in Example 8.24 and 8.25, what is the difference?

Method may have argument similar to function or procedure. Examine the following example.

Example 8.26. Method creation with argument.

```
class Bangun {
  double panjang;
  double lebar;
  // method dengan argumen
  void isiData(double p, double l) {
      panjang = p;
      lebar = l;
  }
  // method yang mengembalikan nilai
  double hitungLuas() {
      double luas = panjang * lebar;
      return luas;
  }
}
class pakaiBangun {
  public static void main(String[] args) {
      Bangun b;
      // instansiasi obyek
      b = new Bangun();
      // memanggil method isiData dan mengisi argumennya
      b.isiData(6,8);
    // memanggil method hitungLuas() untuk objek b
    System.out.println("Luas b = " + b.hitungLuas());
  }
}
```

In Example 8.26, we add one (1) more method to class Bangun, namely, isiData. Since, it returns no value, we use void. This method has two (2) arguments, namely, p and l that is used to collect the input value. Examine how to use this methid, see the part of b.isiData (6, 8).

Constructor is a special method, it is defined in time and automatically called when defining an object. Constructor is usually used to initialize the data value in a particular class. The name of method constructor must be the same as its class. Constructor has no return value and not void. Examine the following example.

Example 8.27. Constructor creation with class..

```
class Bangun {
  double panjang;
  double lebar;
  // constructor dengan argumen
  Bangun(double p, double l) {
      panjang = p;
      lebar = l;
  }
  // method yang mengembalikan nilai
  double hitungLuas() {
      double luas = panjang * lebar;
      return luas;
  }
}
class pakaiBangun {
  public static void main(String[] args) {
      Bangun b;
      // instansiasi obyek
      b = new Bangun();
      // memanggil method isiData dan mengisi argumennya
      b.isiData(6,8);
    // memanggil method hitungLuas() untuk objek b
    System.out.println("Luas b = " + b.hitungLuas());
  }
}
```

The above code is nearly similar to Example 8.26, but in fact different. In this code, there is a constructor named Bangun same as its class. Constructor is similar to other method may or may not have argument. In the above example, the constructor Bangun has argument p and l.

8.8.2. Application of Inheritance

Inheritance principle in general has been studied in the beginning of the chapter. An inheritance is a big advantage in an object oriented programming as a characteristic or a method defined in superclass, will be automatically bequeathed to all subclasses. So we may have to define a method once in superclass and may be used in subclasses. Examine the following exampile.

Example 8.28. Application of Inheritance.

```
class SuperA {
  private int a;
  public void setSuperA(int nilai) {
    a = nilai;
  }
  public int getSuperA() {
    return a;
  }
}
// membuat kelas turunan (subclass) dari kelas A
class SubB extends SuperA {
  private int b;
  public void setSubB(int nilai) {
    b = nilai;
  }
  public int getSubB() {
    return b;
  }
}
class DemoKelasTurunan1 {
  public static void main(String[] args) {
    // melakukan instansiasi terhadap kelas B
    SubB ObyekB = new SubB();
    // mengeset nilai objek dari kelas B
    ObyekB.setSuperA(50);
    ObyekB.setSubB(200);
    // mendapatkan nilai yang terdapat dalam objek dari kelas B
    System.out.println("Nilai a : " + Obyek.getSuperA());
    System.out.println("Nilai b : " + Obyek.getSubB());
  }
}
```

In the above source code, class SuperA is a super class with one data, namely, a and two (2) methods, namely, setSuperA and getSuperA. Class SubB is the descendant from the class SuperA. Please note the declaration class SubB extends SuperA. Data and method available in class SuperA will be automatically brought to class SubB. Thus, class SubB will have two (2) data, namely, a and b. Data a is inheritance from class SuperA while data b is the property of class SubB. Method in class SubB consists of four (4) methods, namely, setSuperA and getSuperA that is the inheritance from class SuperA and class setSubB and getSubB that property of class SubB. Please type the program above source code and run, please examine the result.

Examine the following example.

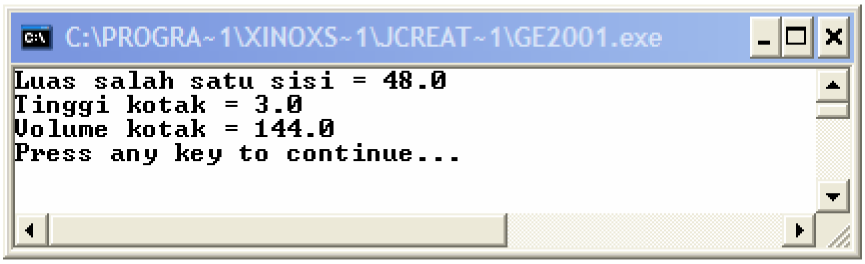
Example 8.29. Implementation of inheritance to calculate area and volume.

```
class Bangun {
  protected double panjang;
  protected double lebar;
  // constructor default
  Bangun() {
      panjang = lebar = 0;
  }
  Bangun(double p, double l) {
      panjang = p;
      lebar = l;
  }
  // method yang mengembalikan nilai
  public double hitungLuas() {
      double luas = panjang * lebar;
      return luas;
  }
}
class Box extends Bangun {
      private double tinggi;
      // constructor class Box
      Box (int p, int l, int t) {
            panjang = p;
            lebar = l;
            tinggi = t;
      }
      public double getTinggi() {
            return tinggi;
      }
      public double hitungVolume() {
            double volume = panjang * lebar * tinggi;
            return volume;
      }
}
class inheritBangun {
  public static void main(String[] args) {
      Box kotak;
      // instansiasi obyek
      kotak = new Box(6, 8, 3);
```

```
    // memanggil method hitungLuas(), getTinggi() dan hitung volume()
    System.out.println("Luas salah satu sisi = " +  kotak.hitungLuas());
    System.out.println("Tinggi kotak = " + kotak.getTinggi());
    System.out.println("Volume kotak = " + kotak.hitungVolume());
  }
}
```

The class Bangun is a superclass whereas Box is a subclass. In the default constructor Bangun, the length and wide value is initialized by 0. Examine in front of the declaration of the length and wide variable in the class Bangun includes key word protected means a decendants's class from Bangun may be able to access the value of the variable but not for those with no descendant relationship.

In Box, which is a subclass, we add one more variable, namely, tinggi, with added keyword private and its data type. Private means that variable tinggi may only be access internally in class Box. The opposite of private is public, that means to allow access by anyone from anywhere. In class Box, we also add one more method, namely, hitungVolume (). In Example 8.29, an object kotak is created based on class Box. Thus, Box is the decedent from class Bangun, and may access method hitungLuas () that is the inheritance from class Bangun. Of course, we could also access method getTinggi () and hitungVolume () that are the method in the class Box. When the program is executed, the followings will be shown.



```
Luas salah satu sisi = 48.0
Tinggi kotak = 3.0
Volume kotak = 144.0
Press any key to continue...
```

Figure 8.15. Execution result from class Bangun dan class Box.

8.8.3. Implementation of Overriding and Overloading

Sometimes, when we make a method inside a subclass, we want to make the method by using the same name as a method in the superclass but with the different application. For example, in class Bangun in Example 8.29, it is available method hitungLuas (). For example, we want to make a subclass Segitiga which is the descendant from the class Bangun. Then, we want to make method hitungLuas () that is no longer area = length x wide rather area = 0,5 x base length x high. In this condition, method hitungLuas () from superclass will be override by method in subclass. It is normally called overriding. Examine the following example.

Example 8.30. Implementation of overriding.

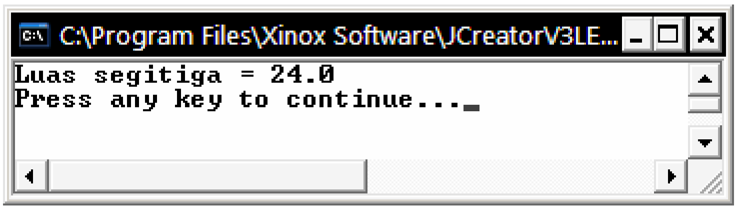
```
class Bangun {
  // method umum
  public double hitungLuas() {
        System.out.println("Method belum terdefinisi");
        Return 0;
  }
}
class Segitiga extends Bangun {
        private double alas;
        private double tinggi;
        Segitiga (int a, int t) {
              alas = a;
              tinggi = t;
        }
        // overriding method hitungLuas()
        public double hitungLuas() {
              double luas = 0.5 * alas * tinggi;
              return luas;
        }
}
class overridingBangun {
  public static void main(String[] args) {
        Segitiga s;
        // instansiasi obyek
        s = new Segitiga(6, 8);
    // memanggil method hitungLuas() dari subclass Segitiga
    System.out.println("Luas segitiga = " + s.hitungLuas());
  }
}
```

In the above example, class Bangun as superclass has a method hitungLuas(), but not yet defined. In the class Segitiga, method hitungLuas () is overriding to get the value of the area of the triangle. During program execution, the one that has been run is the method hitungLuas () is subclass Segitiga. Results of the program execution is shown in Figure 8.16. If we want to continue to run method hitungLuas () in class Bangun, it may be called by using keyword super. Change the method hitungLuas () in the class Segitiga by using the following code.

```
        public double hitungLuas() {
              super.hitungLuas();
              System.out.println();
              double luas = 0.5 * alas * tinggi;
```

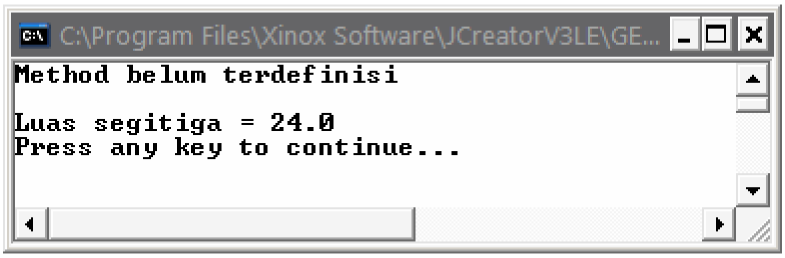
```
            return luas;
      }
```

Run the program, you will get the result as shown in Figure 8.17. Compare it with the previous result in Figure 8.16.


C:\Program Files\Xinox Software\JCreatorV3LE...

```
Luas segitiga = 24.0
Press any key to continue..._
```

Figure 8.16. Result of overriding in method hitungLuas().


C:\Program Files\Xinox Software\JCreatorV3LE\GE...

```
Method belum terdefinisi

Luas segitiga = 24.0
Press any key to continue...
```

Figure 8.17. Result of overriding dan uses super statement.

Overloading is similar to overriding in terms hiding a method from a superclass. However, it has some differences, namely, in overloading, a method has the same name as the method in the parent class, but has different argument list and implementation. Whereas in overriding, method has the same name and same argument list as the parent class, only different in its implementation. Examine the following example.

Example 8.31. Example of overloading.

```
class Bangun {
  // method umum
  public double hitungLuas() {
      System.out.println("Method belum terdefinisi");
      return 0;
  }
}
class BujurSangkar extends Bangun {
       private double sisi;
       // overload method hitungLuas()
```

```
        public double hitungLuas(double sisi) {
              double luas = sisi * sisi;
              return luas;
        }
}
class overloadBangun {
   public static void main(String[] args) {
        BujurSangkar b;
        // instansiasi obyek
        b = new BujurSangkar();
     //   memanggil method hitungLuas() dari subclass BujurSangkar
     System.out.println("Luas BujurSangkar = " +  b.hitungLuas(6));
   }
}
```

Examine the above code, method hitungLuas () in the class Bangun has no argument. Whereas in the class BujurSangkar, it has argument, namely, sisi.. This is known as overloading. Please compare overriding example in Example 8.29 and 8.30.

8.8.4. Application of Polymorphism

As explain in the beginning of the chapter, polymorphism is the capacity to translate a method into various actions. In fact, when you study overriding and overloading, it is indirectly the foundation of the polymorphism. Polymorphism permitts the parent class (superclass) to define general methods for its descendants. Its descendant classes may implement method in accordance to each class characteristics.

In the Example 8.30 and 8.31, we make class Bangun that is the parent class that has a method hitungLuas (). This is a general method. In Example 8.30, we implement the method to calculate triangle's area in the class Segitiga. Whereas in Example 8.31, the method is implemented to calculate the area of BujurSangkar in class BujurSangkar. Please examine the followings example.

Example 8.32. Application polymorphism.

```
class Bangun {
  public double hitungLuas() {
     System.out.println("Method umum");
     return 0;
  }
}
class BujurSangkar extends Bangun {
  private double sisi;
  BujurSangkar(int s) {
     sisi = s;
  }
  //overriding method hitungLuas()
  public double hitungLuas() {
       double luas = sisi * sisi;
       return luas;
  }
}
class Segitiga extends Bangun {
       private double alas;
       private double tinggi;
       Segitiga (int a, int t) {
               alas = a;
               tinggi = t;
       }
       // overriding method hitungLuas()
       public double hitungLuas() {
               double luas = 0.5 * alas * tinggi;
               return luas;
       }
}
class Lingkaran extends Bangun {
  private double jarijari;
  private final double PI = 3.1416;
  Lingkaran(int r) {
     jarijari = r;
  }
  //overriding method hitungLuas()
  public double hitungLuas() {
     double luas = PI * jarijari * jarijari;
     return luas;
  }
}
class DemoPolimorfisme2 {
   public static void main(String[] args) {
      Bentuk obyek;
      BujurSangkar b = new BujurSangkar(12);
      Segitiga s = new Segitiga(5, 6);
      Lingkaran l = new Lingkaran(4);
      // obyek mengacu pada objek BujurSangkar
      obyek = b;
      //     akan memanggil method yang terdapat pada BujurSangkar
      System.out.println("Luas bujursangkar : " +  obyek.hitungLuas());
      System.out.println();
      // obyek mengacu pada objek Segitiga
      obyek = s;
      // akan memanggil method yang terdapat pada Segitiga
      System.out.println(“Luas segitiga : “ + obyek.hitungLuas());
      System.out.println();
      // obyek mengacu pada objek Lingkaran
      obyek = l;
      // akan memanggil method yang terdapat pada Lingkaran
      System.out.println(“Luas lingkaran: “ +  obyek.hitungLuas());
      System.out.println();
   }
}
```

In Example 8.32, we combine the previous examples to show how polymorphism is formed. The parent class is Bangun and with subclass is BujurSangkar, Segitiga and Lingkaran. All of the subclasses have method hitungLuas () that is decendent from class Bangun. Please note that despite the name method hitungLuas () is in all subclasses, its implementation is different and depends on its subclass respectively.

8.8.5. How to use Package and Interface

In some section, we have touched on package. Packages in JAVA means grouping of several classes and interface into one unit. This feature provides way to manage a large number class and interface and to avoide disorder in class and file naming.

Examine closely Example 8.4 in previous section, we have indirectly used package concept. In the Example, we use import statement to use classes in Java.IO package. Java provides many packages that has been written by java developer team. These packages are to make the programming easier.

To create a package is fairly easy. Use keyword follow by package name.

```
package NamaPaket;
```

NamaPaket is the package name to store the program code and compiled file. Java uses system directory to store package. If we made the package named PaketBangun, then we must create a directory using the exact name as PaketBangun. File and class in one package must be stored in the same directory. Please examine the following example.

Example 8.33. Package Creation.

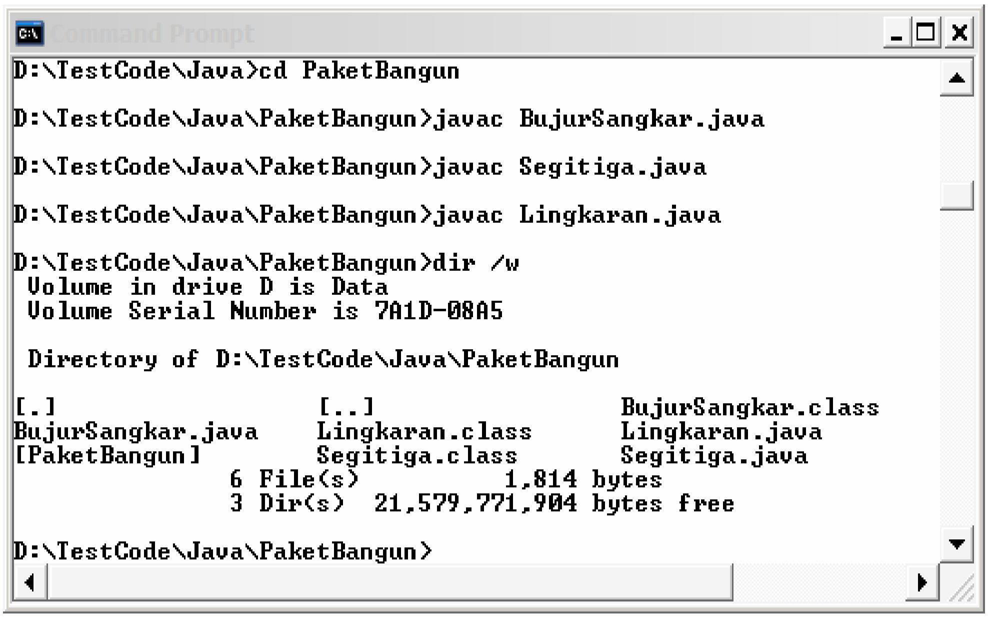
In this section, we will create a package that is modified from Example 8.32. The package name is PaketBangun with three (3) class members, namely, class BujurSangkar, class Segitiga and class Lingkaran. As an initial stage, create a directory named PaketBangun. In this example, the directory is located in D:\TestCode\Java\PaketBangun. Then, write the following source code, store and named according to the class in directory PaketBangun. Please note that in the beginning of a class must be started by statement package PaketBangun;. It shows that the class is a member of PaketBangun.

```
File : BujurSangkar.java
package PaketBangun;
class BujurSangkar extends Bangun {
   private double sisi;
   public BujurSangkar(int s) {
     sisi = s;
   }
   public double hitungLuas() {
       double luas = sisi * sisi;
       return luas;
   }
}
```

```
File : Segitiga.java
package PaketBangun;
class Segitiga extends Bangun {
        private double alas;
        private double tinggi;
        public Segitiga (int a, int t) {
                alas = a;
                tinggi = t;
         }
         public double hitungLuas() {
                double luas = 0.5 * alas * tinggi;
                return luas;
         }
}
```

```
File : Lingkaran.java
package PaketBangun;
class Lingkaran extends Bangun {
   private double jarijari;
   private final double PI = 3.1416;
   public Lingkaran(int r) {
     jarijari = r;
   }
   public double hitungLuas() {
     double luas = PI * jarijari * jarijari;
     return luas;
   }
}
```

Do classpath management as shown in the beginning of the chapter. Compile these three (3) files to get the result as shown in Figure 8.18.



```
D:\TestCode\Java>cd PaketBangun

D:\TestCode\Java\PaketBangun>javac BujurSangkar.java

D:\TestCode\Java\PaketBangun>javac Segitiga.java

D:\TestCode\Java\PaketBangun>javac Lingkaran.java

D:\TestCode\Java\PaketBangun>dir /w
 Volume in drive D is Data
 Volume Serial Number is 7A1D-08A5

 Directory of D:\TestCode\Java\PaketBangun

[.]                   [..]                  BujurSangkar.class
BujurSangkar.java     Lingkaran.class       Lingkaran.java
[PaketBangun]         Segitiga.class        Segitiga.java
               6 File(s)          1,814 bytes
               3 Dir(s)  21,579,771,904 bytes free

D:\TestCode\Java\PaketBangun>
```

Figure 8.18. Compile the three (3) files as part of the package.

After a successful compilation, create new file in directory D:\TestCode\Java, for example PakaiPaketBangun.java. Type the following code in the file. Compile and run the source code.

```
//import seluruh kelas pada PaketBangun
import PaketBangun.*;
class PakaiPaketBangun {
   public static void main(String[] args) {
     BujurSangkar b = new BujurSangkar(12);
     Segitiga s = new Segitiga(5, 6);
     Lingkaran l = new Lingkaran(4);
     System.out.println("Luas bujursangkar : " + b.hitungLuas());
     System.out.println();
     System.out.println("Luas segitiga : " + s.hitungLuas());
     System.out.println();
     System.out.println("Luas lingkaran: " + l.hitungLuas());
     System.out.println();
   }
}
```

Please examine how to write and call package. If we need only class Lingkaran, we must write import PaketBangun.Lingkaran;. but if we need all the classes then we use import PaketBangun. *;. If we run the result is the same as Example 8.33. The main difference is that in Example 8.32 collect all class in a single file. While in Example 8.33, classes are in respective file and grouped in package.

Interface in Java programming language is similar to class, but without class attribute and with declared method without content. Declaration method of an interface may be implemented in other class. A class may be implemented in more than one interface. Method in interface may be implemented in a class must be the same as the one in interface. Interface is used if we want unrelated class implement the same method. Please examine the following example.

Example 8.34. Create interface.

```
interface Bentuk {
        public double luas();
        public double volume();
}
```

In the above example, we make an interface named Bangun with two (2) methods, namely, area() and the volume(). Examine how to write interface. Both methods is declared without content. We may use the interface to create new class and implement the interface in the new class. Examine the following example.

Example 8.34. The use of Interface.

```
class Kubus implements Bentuk {
        int x = 10;
        public double luas( ) {
            return (6 * x * x);
        }
        public double volume() {
                return (x * x * x);
        }
}
class Lingkaran implements Bentuk {
        int radius = 10;
        public double luas() {
                return (Math.PI * radius * radius);
        }
        public double volume() {
                return 0;
        }
}
```

8.9. SUMMARY

- Important concepts in object oriented programming are class, object, abstraction, encapsulation, inheritance and polymorphism.
- There are eight (8) basic data types in Java, namely, boolean (for logic form), Char (for textual form), byte, short, int, long (integer), double and float (floating point).
- Operators in Java are arithmetic operator, relational, logic and bitwise.
- Control program structure for branching may be done by statements if, if... else, and if... else... if, whereas the looping may be done with for, while and do-while.
- Exception is an abnormal condition during runtime. Java provides mechanisms, such as, try, throw, catch and finally to handle exception.
- Thread is a program section that is not depend on other part of the main program and may be run simultaneously together.
- Java fully supports the concept of class, inheritance, overriding, overloading, and polymorphism.
- Package is a collection of classes. Whereas interface is a class without the attribute and has method declare without the contents.

8.10. EXERCISES

1. Create a program to print the value of the following variables with different data type and initial value, namely.
   a. Data type float, initial value = 3.45.
   b. Data type char, initial value = B.
   c. Data type int, initial value = 23.

2. Create a program to calculate the average value of the following three (3) variables, namely, number1 = 56,3, number2 = 12 and number3 = 2,78.

3. Using variables in problem no. 2, create a program to seek for the smallest value from these three (3) variables.

4. Get three (3) Exam values from user and calculate its average value. Print "Congratulation" if the average value is larger than 60, in addition of print the average value. Use BufferedReader to get input from user, and System.out to print results.

5. Create a program to print your name a hundred times. Create three (3) version of the program using while loop, do while and for-loop.

6. Using class, create a program on class of school staff. This is a superclass

which has subclass teacher and subclass administrative staff.

- The class school staff has variables, namely, name, address, children,start working date, and rank. Moreover, it has method work period, calculated from start working date and current date, and method to calculate basic salary. The basic salary is calculated from basic payment plus work period allowance, rank allowance and children allowance. Moreover, method to calculate total payment is not defined yet.

- Class teacher, in addition to variables and methods in superclass, it has additional variables, namely, field of expertise and method to calculate teaching allowance based on the hours time the allowance per hour. Define method to calculate total payment as main salary plus teaching allowance.

- Class administrative staff, in addition to variables and methods in superclass, it has additional variables, namely, weekly working hours and method to calculate overtime. Overtime allowance is calculated as the additional hours from 40 hours per week working hour. Overtime allowance is overtime hours times overtime charges per hour. Define method to calculate total payment as main salary plus overtime allowance.