PP/018/III/R3

LABORATORIUM
INFORMATICS ENGINEERING
FACULTY OF INDUSTRIAL TECHNOLOGY
UNIVERSITAS AHMAD DAHLAN

PRACTICUM INSTRUCTION

OBJECT ORIENTED PROGRAMMING

Team:
Drs. Tedy Setiadi, M.T.
Ali Tarmuji, S.T., M.Cs.
Supriyanto, S.T., M.T.
Adhi Prahara, S.Si., M.Cs.
Faisal Fajri Rahani S.Si., M.Cs.

2020

PREFACE

Our gratitude goes to the presence of Allah SWT who has bestowed His grace and guidance so that we can complete this Practicum Instructions for the International Class. This module is made to guide students in their Object Oriented Programming labs work. This module has been arranged into ten meetings and adjusted to the RPS of OOP Course.

We realize that there are still many imperfections in this writing. We always accept criticism and suggestions to improve the quality of guidelines and practicum implementation. We would like to thank the management team of the International Class of Informatics Department and anyone who was involved in making this practicum guide. Hopefully the results obtained from the implementation of the OOP practicum through this guide can provide benefits and contributions in the advancement of science.

Yogyakarta, August 2020

OOP Team

AUTHORS

Drs. Tedy Setiadi, M.T.

Ali Tarmuji, S.T., M.Cs.

Supriyanto, S.T., M.T.

Adhi Prahara, S.Si., M.Cs.

Faisal Fajri Rahani S.Si., M.Cs.

REVISION HISTORY

The undersign below:

Name : Drs. Tedy Setiadi, M.T.

NIY : 60030475

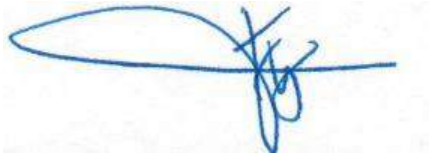
Position : Object Oriented Programming Lecturer

Hereby declare that the Revision of the OOP Practicum Guidelines for the Informatics Department of Universitas Ahmad Dahlan has been carried out with the following explanation:

No	Revision History	Date	Module Number
1	a. Change the module arrangement from 12 meetings to 10 meetings. b. The first meeting will be combination from the previous first and second meeting which is introduction to JAVA and IDE and basic programming. c. The eleventh meeting is removed.	22 August 2019	PP/018/III/R2
2	a. Adding practicum scenario	22 August 2020	PP/018/III/R3

Yogyakarta, 23 August 2020

OOP Team Coordinator



Drs., Tedy Setiadi, M.T.

NIY. 60030475

LETTER OF STATEMENT

The undersigned below:

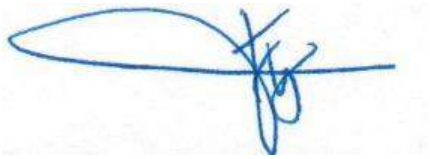
Name : Lisna Zahrotun, S.T., M.Cs.

NIK/NIY : 60150773

Position : The Head of Informatics Engineering Laboratory

Explain clearly that this Practicum Module has been reviewed and will be used for practicum activity in Academic Year 2020-2021 in the Laboratory of Informatics Engineering, Informatics Department, Faculty of Industrial Technology, Universitas Ahmad Dahlan.

Yogyakarta, 23 August 2020

Acknowledged, Chief of RELATA Scientific Group	Head of Informatics Engineering Laboratory
	
Drs., Tedy Setiadi, M.T.	**Lisna Zahrotun, S.T., M.Cs.**
NIY. 60030475	NIY. 60150773

VISI DAN MISI PRODI TEKNIK INFORMATIKA

VISI

Menjadi Program Studi Informatika yang diakui secara internasional dan unggul dalam bidang Informatika serta berbasis nilai-nilai Islam.

MISI

1. Menjalankan pendidikan sesuai dengan kompetensi bidang Informatika yang diakui nasional dan internasional
2. Meningkatkan penelitian dosen dan mahasiswa dalam bidang Informatika yang kreatif, inovatif dan tepat guna.
3. Meningkatkan kuantitas dan kualitas publikasi ilmiah tingkat nasional dan internasional
4. Melaksanakan dan meningkatkan kegiatan pengabdian masyarakat oleh dosen dan mahasiswa dalam bidang Informatika.
5. Menyelenggarakan aktivitas yang mendukung pengembangan program studi dengan melibatkan dosen dan mahasiswa.
6. Menyelenggarakan kerja sama dengan lembaga tingkat nasional dan internasional.
7. Menciptakan kehidupan Islami di lingkungan program studi.

TATA TERTIB LABORATORIUM TEKNIK INFORMATIKA

DOSEN/KOORDINATOR PRAKTIKUM

1. Dosen harus hadir saat praktikum minimal 15 menit di awal kegiatan praktikum dan menandatangani presensi kehadiran praktikum.
2. Dosen membuat modul praktikum, soal seleksi asisten, pre-test, post-test, dan responsi dengan berkoordinasi dengan asisten dan pengampu mata praktikum.
3. Dosen berkoordinasi dengan koordinator asisten praktikum untuk evaluasi praktikum setiap minggu.
4. Dosen menandatangani surat kontrak asisten praktikum dan koordinator asisten praktikum.
5. Dosen yang tidak hadir pada slot praktikum tertentu tanpa pemberitahuan selama 2 minggu berturut-turut mendapat teguran dari Kepala Laboratorium, apabila masih berlanjut 2 minggu berikutnya maka Kepala Laboratorium berhak mengganti koordinator praktikum pada slot tersebut.

PRAKTIKAN

1. Praktikan harus hadir 15 menit sebelum kegiatan praktikum dimulai, dan dispensasi terlambat 15 menit dengan alasan yang jelas (kecuali asisten menentukan lain dan patokan jam adalah jam yang ada di Laboratorium, terlambat lebih dari 15 menit tidak boleh masuk praktikum & dianggap Inhal).
2. Praktikan yang tidak mengikuti praktikum dengan alasan apapun, wajib mengikuti INHAL, maksimal 4 kali praktikum dan jika lebih dari 4 kali maka praktikum dianggap GAGAL.
3. Praktikan harus berpakaian rapi sesuai dengan ketentuan Universitas, sebagai berikut:
   a. Tidak boleh memakai Kaos Oblong, termasuk bila ditutupi Jaket/Jas Almamater (Laki-laki / Perempuan) dan Topi harus Dilepas.
   b. Tidak Boleh memakai Baju ketat, Jilbab Minim dan rambut harus tertutup jilbab secara sempurna, tidak boleh kelihatan di jidat maupun di punggung (khusus Perempuan).
   c. Tidak boleh memakai baju minim, saat duduk pun pinggang harus tertutup rapat (Laki-laki / Perempuan).
   d. Laki-laki tidak boleh memakai gelang, anting-anting ataupun aksesoris Perempuan.
4. Praktikan tidak boleh makan dan minum selama kegiatan praktikum berlangsung, harus menjaga kebersihan, keamanan dan ketertiban selama mengikuti kegiatan praktikum atau selama berada di dalam laboratorium (tidak boleh membuang sampah sembarangan baik kertas, potongan kertas, bungkus permen baik di lantai karpet maupun di dalam ruang CPU).
5. Praktikan dilarang meninggalkan kegiatan praktikum tanpa seizin Asisten atau Laboran.
6. Praktikan harus meletakkan sepatu dan tas pada rak/loker yang telah disediakan.
7. Selama praktikum dilarang NGENET/NGE-GAME, kecuali mata praktikum yang membutuhkan atau menggunakan fasilitas Internet.
8. Praktikan dilarang melepas kabel jaringan atau kabel power praktikum tanpa sepengetahuan laboran
9. Praktikan harus memiliki FILE Petunjuk praktikum dan digunakan pada saat praktikum dan harus siap sebelum praktikum berlangsung.
10. Praktikan dilarang melakukan kecurangan seperti mencontek atau menyalin pekerjaan praktikan yang lain saat praktikum berlangsung atau post-test yang menjadi tugas praktikum.
11. Praktikan dilarang mengubah setting software/hardware komputer baik menambah atau mengurangi tanpa permintaan asisten atau laboran dan melakukan sesuatu yang dapat merugikan laboratorium atau praktikum lain.
12. Asisten, Koordinator Praktikum, Kepala laboratorium dan Laboran mempunyai hak untuk menegur, memperingatkan bahkan meminta praktikan keluar ruang praktikum apabila dirasa anda

mengganggu praktikan lain atau tidak melaksanakan kegiatan praktikum sebagaimana mestinya dan atau tidak mematuhi aturan lab yang berlaku.

13. Pelanggaran terhadap salah satu atau lebih dari aturan diatas maka Nilai praktikum pada pertemuan tersebut dianggap 0 (NOL) dengan status INHAL.

ASISTEN PRAKTIKUM

1. Asisten harus hadir 15 Menit sebelum praktikum dimulai (konfirmasi ke koordinator bila mengalami keterlambatan atau berhalangan hadir).
2. Asisten yang tidak bisa hadir WAJIB mencari pengganti, dan melaporkan kepada Koordinator Asisten.
3. Asisten harus berpakaian rapi sesuai dengan ketentuan Universitas, sebagai berikut:
   a. Tidak boleh memakai Kaos Oblong, termasuk bila ditutupi Jaket/Jas Almamater (Laki-laki / Perempuan) dan Topi harus Dilepas.
   b. Tidak Boleh memakai Baju ketat, Jilbab Minim dan rambut harus tertutup jilbab secara sempurna, tidak boleh kelihatan di jidat maupun di punggung (khusus Perempuan).
   c. Tidak boleh memakai baju minim, saat duduk pun pinggang harus tertutup rapat (Laki-laki / Perempuan).
   d. Laki-laki tidak boleh memakai gelang, anting-anting ataupun aksesoris Perempuan.
4. Asisten harus menjaga kebersihan, keamanan dan ketertiban selama mengikuti kegiatan praktikum atau selama berada di laboratorium, menegur atau mengingatkan jika ada praktikan yang tidak dapat menjaga kebersihan, ketertiban atau kesopanan.
5. Asisten harus dapat merapikan dan mengamankan presensi praktikum, Kartu Nilai serta tertib dalam memasukan/Input nilai secara Online/Offline.
6. Asisten harus dapat bertindak secara profesional sebagai seorang asisten praktikum dan dapat menjadi teladan bagi praktikan.
7. Asisten harus dapat memberikan penjelasan/pemahaman yang dibutuhkan oleh praktikan berkenaan dengan materi praktikum yang diasisteni sehingga praktikan dapat melaksanakan dan mengerjakan tugas praktikum dengan baik dan jelas.
8. Asisten tidak diperkenankan mengobrol sendiri apalagi sampai membuat gaduh.
9. Asisten dimohon mengkoordinasikan untuk meminta praktikan agar mematikan komputer untuk jadwal terakhir dan sudah dilakukan penilaian terhadap hasil kerja praktikan.
10. Asisten wajib untuk mematikan LCD Projector dan komputer asisten/praktikan apabila tidak digunakan.
11. Asisten tidak diperkenankan menggunakan akses internet selain untuk kegiatan praktikum, seperti Youtube/Game/Medsos/Streaming Film di komputer praktikan.

LAIN-LAIN

1. Pada Saat Responsi Harus menggunakan Baju Kemeja untuk Laki-laki dan Perempuan untuk Praktikan dan Asisten.
2. Ketidakhadiran praktikum dengan alasan apapun dianggap INHAL.
3. Izin praktikum mengikuti aturan izin SIMERU/KULIAH.
4. Yang tidak berkepentingan dengan praktikum dilarang mengganggu praktikan atau membuat keributan/kegaduhan.
5. Penggunaan lab diluar jam praktikum maksimal sampai pukul 21.00 dengan menunjukkan surat ijin dari Kepala Laboratorium Prodi Teknik Informatika.

Yogyakarta, 23 Agustus 2020

Kepala Laboratorium Praktikum Teknik Informatika

Lisna Zahrotun, S.T., M.Cs.
NIY. 60150773

TABLE OF CONTENTS

TABLE OF FIGURES

PRACTICUM SCENARIO

Labs Course : Object Oriented Programming

Number of Meeting : 10 practicums + 1 final labs work

Online Practicum Scenario

Meeting-	Meeting Title	Duration	Practicum Scenario
1	INTRODUCTION TO JAVA AND IDE	1 week	Pre-test and post-test are given and uploaded at e-learning. Practicum will be held online/using tutorial video and whatsapp group to facilitate Q&A. Pre-test is given with duration 15 minutes after practicum started. Additional time will be given until practicum ended. Post-test and weekly report can be done in a week. The report consists of: Cover, Introduction (A brief theory about the materials), Pre-test (Answer of pre-test questions), Practicum (Answer of post-test questions usually code and screenshot of the result), Discussion (A brief discussion about the post-test result).
2	CLASS AND OBJECT IN JAVA	1 week	
3	CONSTRUCTOR, ENCAPSULATION & INFORMATION HIDING	1 week	
4	INHERITANCE	1 week	
5	POLYMORPHISM	1 week	
6	CLASS RELATION	1 week	
7	ABSTRACT CLASS	1 week	
8	INTERFACE	1 week	
9	EXCEPTION HANDLING AND I/O	1 week	
10	GAME	1 week	
11	Final labs work	1 week	Final labs work is given and uploaded at e-learning. The final labs work report consists of: Cover, Introduction (A brief theory about the materials), Pre-test (Answer of pre-test questions), Practicum (Answer of post-test questions usually code and screenshot of the result), Discussion (A brief discussion about the post-test result).

MEETING 1: INTRODUCTION TO JAVA AND IDE

Meeting : 1

Time Allocation : 90 minutes
- Pre-Test : 15 minutes
- Practicum : 55 minutes
- Post-Test : 20 minutes

Scoring : 100%
- Pre-Test : 20 %
- Practicum : 40 %
- Post-Test : 40 %

1.1. GOALS AND LEARNING OUTCOMES

By following this session, students are expected to be able to:
1. Understand the object oriented programming paradigm
2. Explain the technology that implements object oriented programming.
3. Explain the work of compiler
4. Explain basic programming lesson

Learning outcomes, students are expected to have the ability to:
1. Understand the object oriented programming paradigm
2. Use IDE to implement OOP
3. Understand the work of compiler
4. Implement basic programming

1.2. LITERATURE STUDY

As a programming language, Java can be used to make any kind of application, desktop, web, etc. Java is an object oriented programming language that can be run in any kind of operating system platform. The development of JAVA is applied to all operating system and it is also open source.

Java programming language is created by James Gosling in 1996 as a part of Sun Microsystem Java Platform. Java syntaxs is inherited from C and C++ but it is more simple and have limited access to OS. Java is proposed as programming language that easy to learn.

Java application is written as file with .java extension which compiled to .class file. Class file is a bytecode that can be run in every Java Virtual Machine, that not depend on OS and processor architecture. Java is a language that proposed to all concurrent needs, class-based, object oriented and designed to not depend on the environment where the application is excecuted.

Java platform consists of three profiles: Java ME (Java Mircro Edition) is a Java that can run on embedded system such as Java Card and Handphone. Java SE (Java Standard Edition) is a Java that can run on PC or server as standalone application or desktop application. Java EE (Java Enterprise Edition) is a Java profile that aimed to create Enterprise application such as web application (servlet) and Enterprise Java Bean (EJB).

Before we learn further about class, first, we need to learn about data type, variable, operator, control statement in the Java programming language.

Data Type

There are primitive data types in Java such as the following:

Data Type	Description
boolean	true or false
char	character
byte	-128 - 127
short	-32768 - 32767
int	-2147483648 - 2147483647
long	-9223372036854775808 - 9223372036854775807
double	4.9E-324 - 1.7976931348623157E308
float	1.4E-45 - 3.4028235E38

String is not a data type in Java. String is an object. String has unique property which is it can be created without creating an object.

Variable

Variable is used to store data. A variable must be inside a class or method. Variable can be created as follow:

```
Type namaVariabel;              // variable declaration
Type namaVariabel= nilai;       // variable initialization
```

Example 1.1 Variable declaration:

```
Int nilai;
Int nilai=0;
```

Operator

Operator is a special character that can be used to produce a result/value.

Logic Operator

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder

Assignment Operator

Operator	Description
=	To give operation result
+=	Number addition
-=	Number subtraction
*=	Number multiplication
/=	Number division
%=	Produce a remainder

Comparison Operator

Operator	Keterangan
==	equal
!=	Not equal
>=	More than equal
<=	Less than equal

>	More than
<	Less than

Logic Operator

Operator	Description
&&	and
||	or

Branching in Java

Branching using IF...ELSE

Syntax if

```
if (boolean expression) {
      statement or block
} else if (boolean
expression) {
      statement or block
} else {
      statement or block
}
```

Example:

```
public Class Percabangan {
    public static void main(String[] args) {
        int a = 2;
        if (a < 5) {
            System.out.println("Nilai a lebih kecil
dari 5");
        } else {
            System.out.println("Nilai a lebih besar
dari 5");
        }
    }
 }
```

Branching using SWITCH

Syntax switch

```
Syntax switch sebagai berikut :
switch (expression) {
      case constant1 :
      statements;
                  break;
      case constant2 :
      statements;
                  break;
      default :     statements;
            break;
      }
```

Example:

```
switch(x){
case 1: System.out.println("Senin");
      break;
case 2: System.out.println("Selasa");
      break;
case 3: System.out.println("Rabu");
      break;
case 4: System.out.println("Kamis");
      break;
case 5: System.out.println("Jum'at");
      break;
case 6: System.out.println("Sabtu");
      break;
case 7: System.out.println("Minggu");
      break;
default: System.out.println("Salah input
hari");
      break;
}
```

Branching using Conditional Expression

It is used to replace if … else ….. The syntax is:

Conditional expression syntax	Example:

`exp1 ? exp2 : exp3`
 	
If exp1 is true, exp2 is executed but if exp1 is false then exp3 is executed
	`int a = 50;`
	`int b = 100;`
	`int nilai = 3 > 2 ? a : b;`
	`Output:  nilai = 50`

Looping in Java

Looping using for

Syntax for	Example

`for (init_expr;boolean testexpr;alter_expr) {`
`  statement or block`
	`public Class PerulanganFOR {`
	`  public static void main(String[] args) {`
	`  for (int i = 1; i <=5; i++) {`

```
}
```

```
        System.out.println(i+"=Belajar Javaitu
        mudah");
      }
    }
}
```

Looping using while

Syntax while	Example:

`while (boolean testexpr) {`
`  statement or block`
`}`
	`public Class PerulanganWHILE {`
	`  public static void main(String[] args) {`
	`  int i = 0;`
	`  while (i<10) {`
	`  System.out.println(i);`
	`  i++;`
	`  }`
	`  }`
	`}`

Looping using do....while

Syntax do/while	Example

`do {`
`  statement or block`
`} while (boolean testexpr)`
	`public Class PerulanganDOWHILE {`
	`  public static void main(String[] args) {`
	`  int i = 0;`
	`  do {`
	`  System.out.println(i);`
	`  i++;`
	`  } while (i<10);`
	`  }`
	`}`

ARRAY

Array declaration

Array usually used to group some data or objects that have same data type. Array allows to point a group of object using the same name. Array can be declared with any type, both primitive or object. For example:

```
char s[];
point p[]; // point is a class
```

In the Java language, Array is an object though it consists of primitive element. Like any other class, when declaring an Array, an object array is not created yet. Array declaration only create a reference that point to an object array.

The memory capacity that used by elements in the array will be allocated dynamically using new statement or array initializer. The previous example shows how to write an array using square bracket after variable name. This is the standard array declaration in C, C++ and Java. The format is little bit harder to read. Therefore, Java use alternative array declaration using square bracket in the left of variable name as follow.

```
char [] s;
point [] p; // point is a class
```

An array declaration can be written as both of the example above but should be consistent to one form of declaration. In the array declaration, the size of array is not defined. The square bracket array in the left of variable name is applied to any variable on the right.

Creating an Array

Array is created using keyword new. The example below shows how to create an array of char.

```
s = new char[26];
```

Index of array is started from 0. The valid range of array index is 0 to size of array – 1. If program accesses outside the valid index, runtime exception will occur. Example below shows how to create an array with object as its element:

```
p = new point[10];
```

The statement does not create 10 objects of Point. Use this statement to create 10 objects of Point.

```
p[0] = new point(0, 1);
p[1] = new point(1, 2);
```

Array Initialization
Java has a simple way to create an array object with initialization value for every elements:

```
String names[] = {"Joko","Joni","Jujuk"};
Above code is similar to:
String names[];
names[0] = "Joko";
names[1] = "Joni";
names[2] = "Jujuk";
```

This shorter code can be used to create any kind of array:

```
MyDate dates[] = {
new MyDate(22, 7, 1964),
new MyDate(1, 1, 2000),
new MyDate(22, 12, 1964),
};
```

Multidimensional Array
Array can be created from array. It is called multidimensional array. For example, to create two dimensional array, one can use:

```
int twoDim [][] = new int[4][];
twoDim[0] = new int[5];
twoDim[1] = new int[5];
```

Non rectangular Array of array can be created as follow:

```
twoDim[0] = new int[2];
twoDim[1] = new int[4];
twoDim[2] = new int[6];
twoDim[3] = new int[8];
```

To create rectangular array of array:

```
int twoDim[][] = new int[4][5];
```

Array Limitation
Array index started from 0. The number of element in an array is saved as part of array object in the length attribute. This attribute can be used in the iteration process as follow:

```
int list[] = new int[10];
for(int i = 0; i < list.length; i++) {
System.out.println(list[i]);
```

By using length attribute, the program does not need to know the size of element in an array in the compilation step.

Array Manipulation
Changing the Size of Array
After creating an object array, the size of array can not be changed. But, can be referred using variable to point to a new object array as shown in this example:

```
int myArray[] = new int[6];
myArray = new int[10];
```

Above code will erase the first object array except another variable is made to refer to that object array.

Copying *Array*
Java provide a specific method to copy an array using arrayCopy() method from System class:

```
int myArray[] = {1, 2, 3, 4, 5, 6};
// array with more number of element
int hold[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
//copy elements to hold
// start from index 0
System.arraycopy(myArray, 0, hold, 0,myArray.length);
```

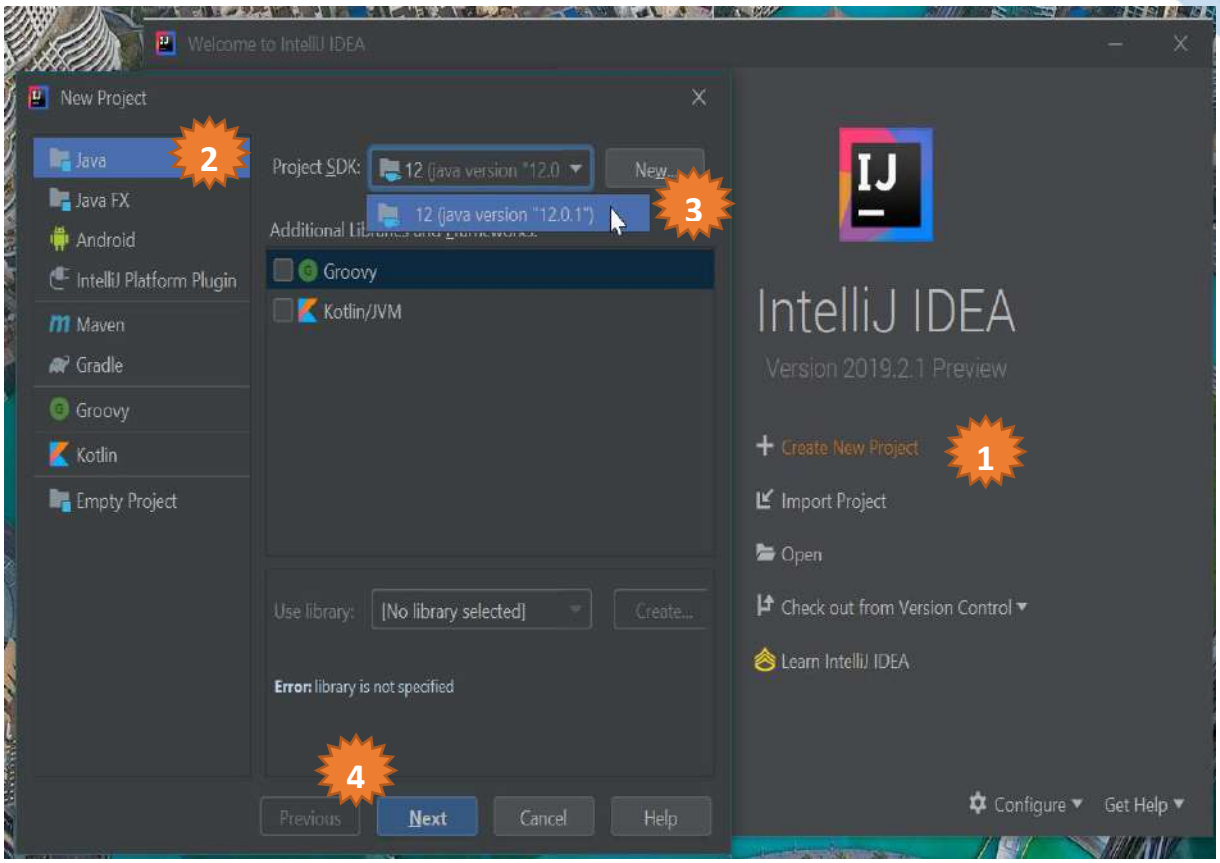
After copying process is done, the elements in the array are 1, 2, 3, 4, 5, 6, 4, 3, 2, 1.

1.3. TOOLS AND MATERIALS

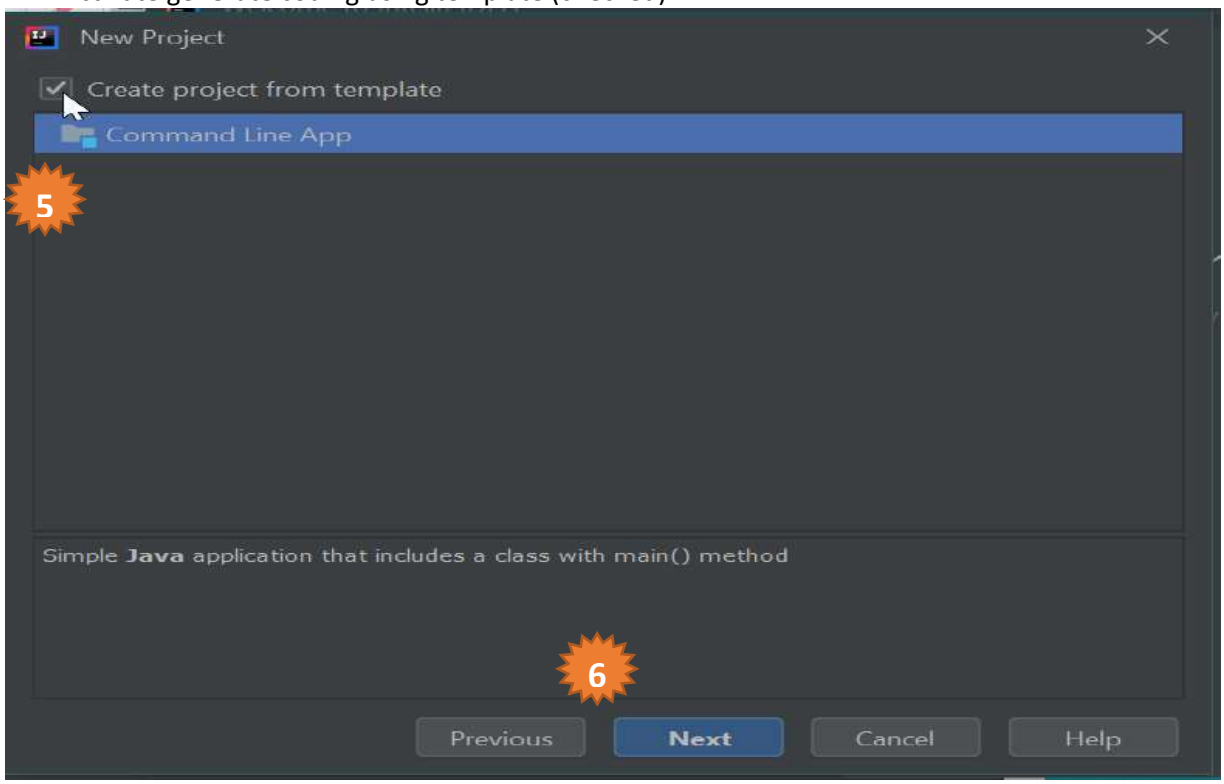
Tools and materials used in this meeting:
1. Computer.
2. Intellij IDEA Community
3. Java SDK

1.4. INSTRUCTIONS

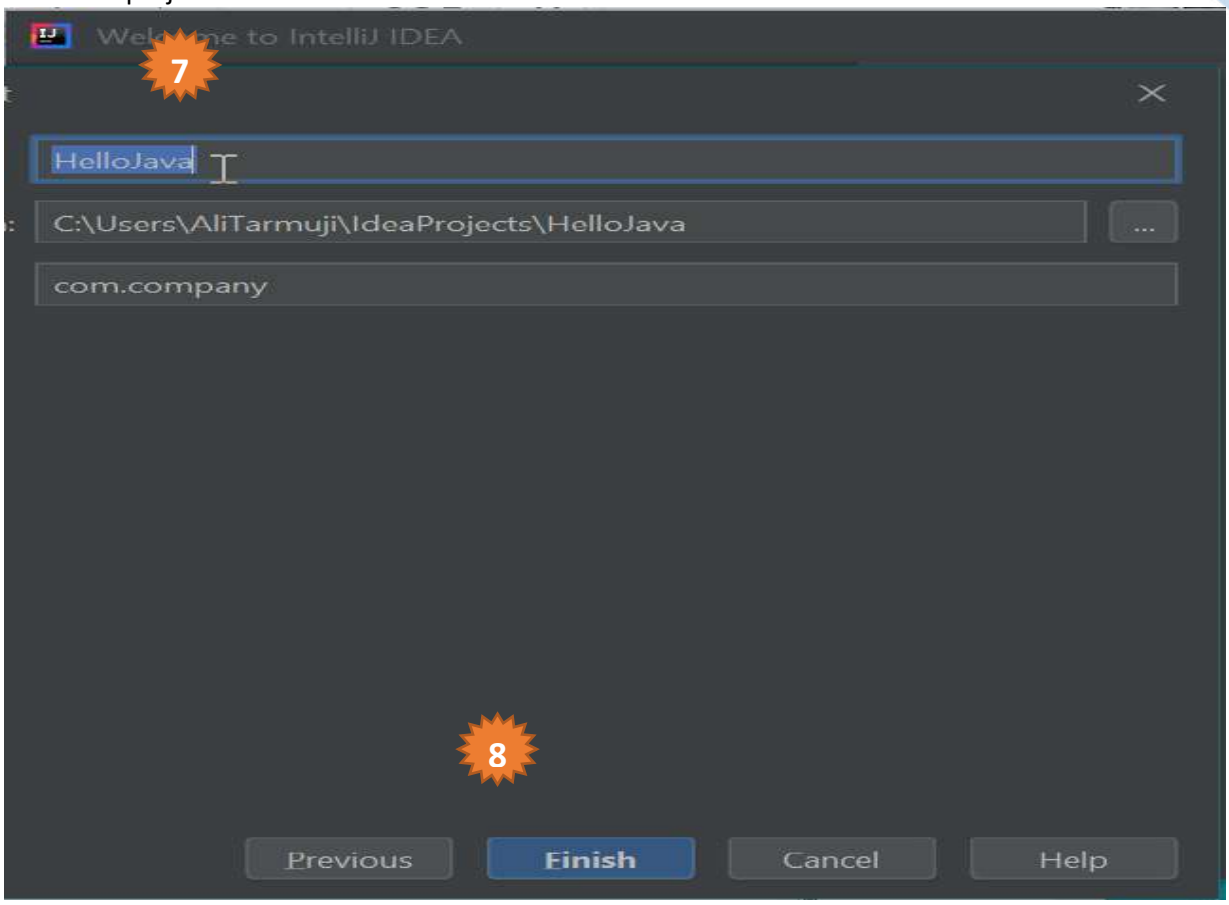
1. Open Java IDE program, Intellij IDEA Community (if not installed, it can be downloaded at: http://bit.ly/intellijideacommunity). Click **Create New Project**, Click **Java**, choose **Project SDK**: SDK Java (use the newest version if available), Click **Next**



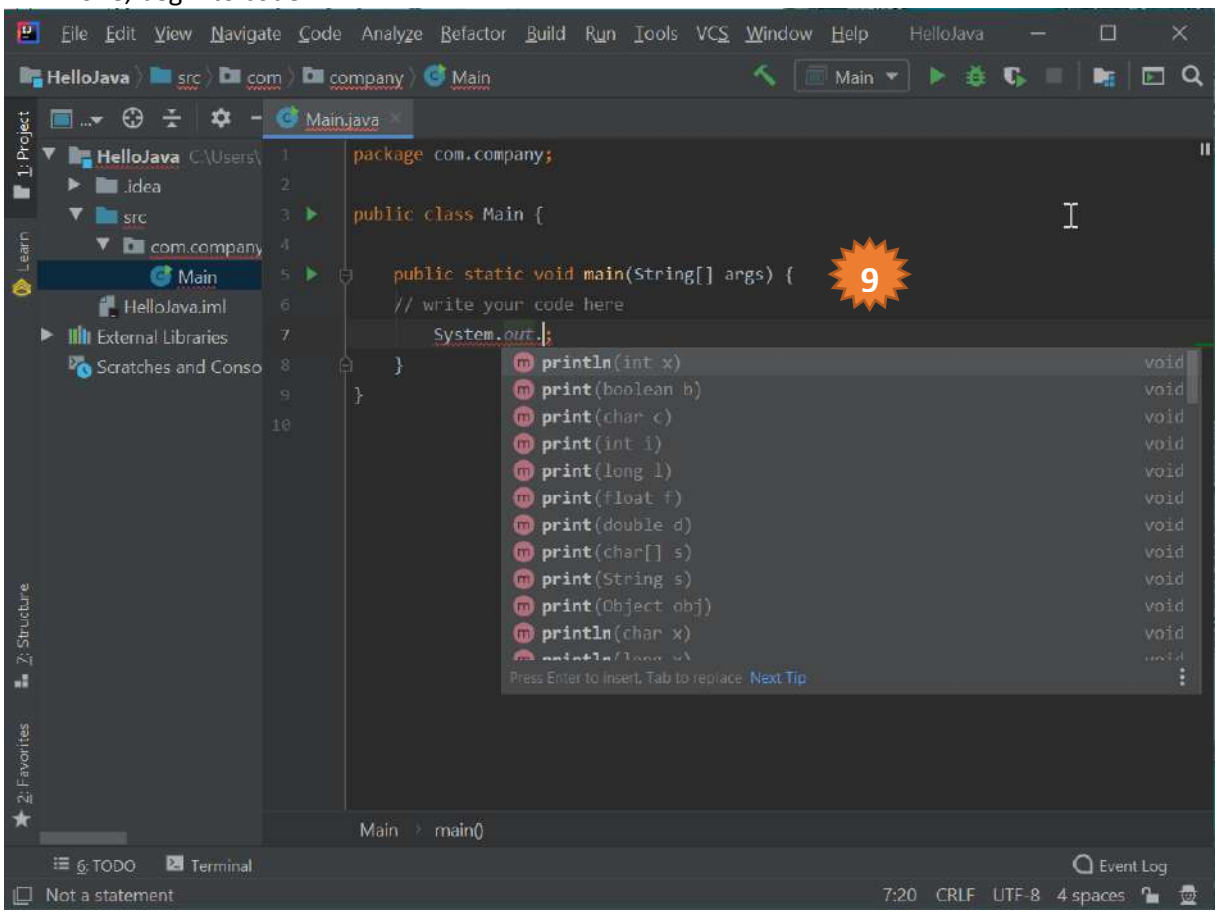
2. Activate generate coding using template (checked)



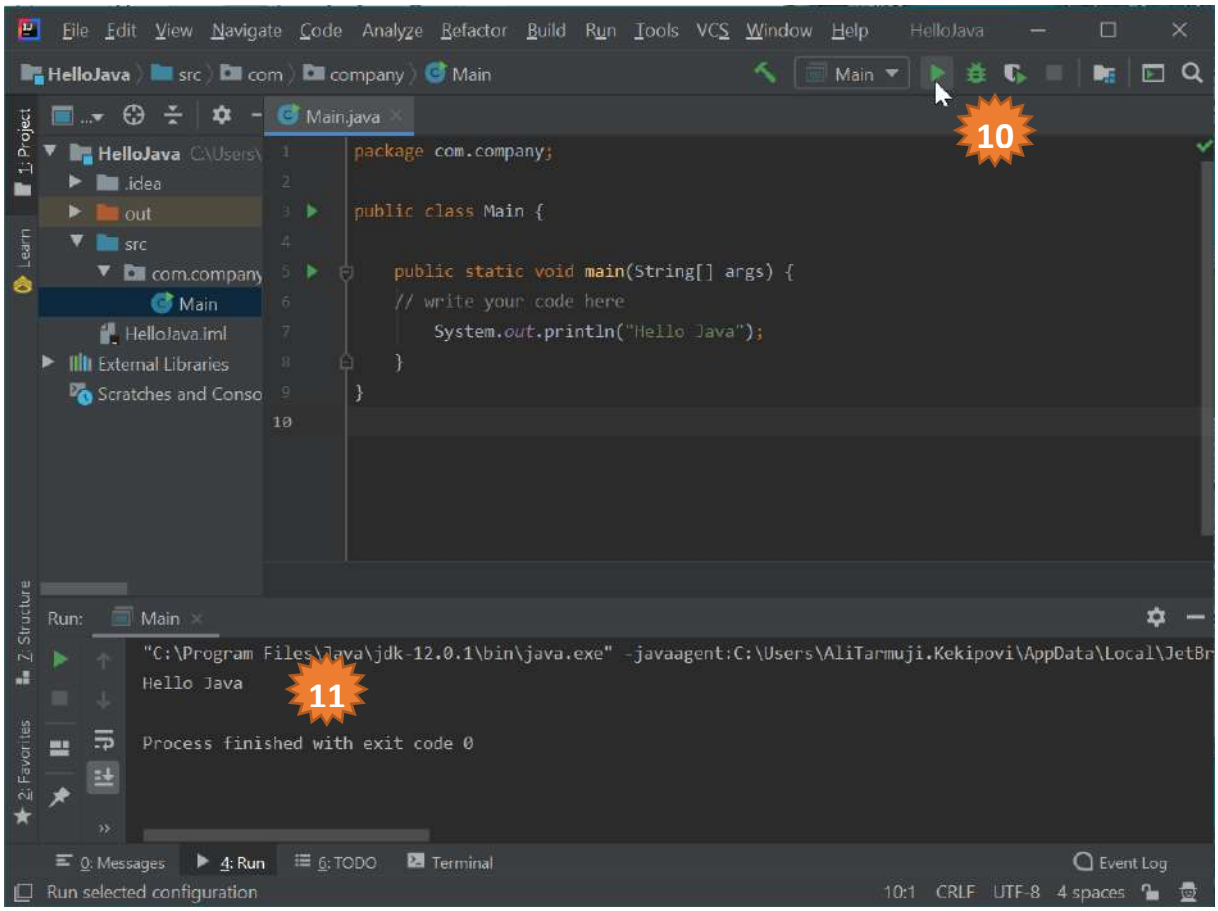
3. Write the project name.



4. Done, begin to code



5. To run the program (compile, linking, Excute). Click play button in the top right corner or select Run in the menu.



1.5. ASSIGNMENT

Build a program to:

1. Print 100 rows of hello world sentences.
2. Read the radius of a circle then calculate and print the perimeter and area of circle.
3. Read three integers then print the maximum and minimum value from that integers.
4. Read a number then print the letter using the following rule (A:80-100, B:65-79, C:55-64, D:40-54, E :0-53)
5. Create a program using array to show prime numbers between 1 to 100!
6. Create a program using array to read N data of positive integers then print:
   a. Sum of that numbers
   b. Average value
   c. Maximum value
   d. Minimum value

MEETING 2: CLASS AND OBJECT IN JAVA

Meeting : 2

Time Allocation : 90 minutes
- Pre-Test : 15 minutes
- Practicum : 55 minutes
- Post-Test : 20 minutes

Scoring : 100%
- Pre-Test : 20 %
- Practicum : 40 %
- Post-Test : 40 %

2.1. GOALS AND LEARNING OUTCOMES

By following this session, students are expected to be able to:
1. Explain about object oriented programming concept
2. Explain about **object**
3. Explain about **Class**
4. Explain about **instance**
5. Explain simple case study using OOP concept

Learning outcomes, students are expected to have the ability to:
1. Understand the concept of object oriented programming
2. Mention the example of object in daily life or application
3. Code **Class**
4. Code **instance** and its application.

2.2. LITERATURE STUDY

A Class defines a structure and behavior of an object or a group of object. Class is a prototype which define variables and methods generally. In Java, there is a rule when giving a name to a class. A class name must be started with capital letter. This rule is used to differentiate between Class and Object. Class is defined by keyword Class. Code below shows how to declare a class:

```
Class Mahasiswa {
    String nim;                 // variable or attribute declaration
    String nama;                // variable or attribute declaration
}
```

An object is instantiated from a class. Class can have many objects and every object has similar characteristics as defined in the parent class. To give a name to an object, the name must be started with lower letter. See the example below:

```
Mahasiswa mahasiswa;                    //deklarasi objek
mahasiswa = new Mahasiswa();            //instansiasi dari kelas Mahasiswa
Mahasiswa mahasiswa= new Mahasiswa();   //dijadikan satu
```

Every object has unique identity. For example: Student has NIM which the student's NIM is different with other student. Class consists of:
1. Constructor : use to instantiate an object

2. Variable : an attribute that state the condition from class and object.
3. Method : a function or procedure.

Variable and method can have one of the characteristic as follow:

1. **Private**, can not be called outside the parent class.
2. **Protected**, can be called only by the parent and the children class that inherite the parent class.
3. **Public**, can be called by any class.

In Java, there are two methods:

1. Function, a method that have return value. Function is created by defining the return value then write the function name.
2. Procedure, a method that does not have return value. Procedure can be created the same way as function but the return value is void.

POJO class in Java is a class which has attributes and **getter** and **setter** methods. The attribute is private and the getter and setter method are public. Getter method is used to get the value of an attribute while the setter method is used to change the value of an attribute. For example:

```
public Class Mahasiswa {
    private String nim;
    private String nama;

    public String getNama() {                // method is a Function
        return nama;
    }

    public void setNama(String nama) {       // method is a Procedure
        this.nama = nama;
    }

    public String getNim() {                 // method is a Function
        return nim;
    }

    public void setNim(String nim) {         // method is a Procedure
        this.nim = nim;
    }
}
```

Constructor is a method that the first one to be executed when creating an object. Constructor has characteristics as follow:

- It has the same name as the class.
- Does not have return value.
- Use to instantiate an object.
- It has access modifier; no other keyword can be placed before the method's name in a constructor declaration. For example:

```
Class Mahasiswa {
    String nim;                    // variable declaration
    String nama;
    public Manusia(){}             // default constructor
    public Manusia(String nim, string nama){   // parameter of constructor
        this.nim = nim;
        this.nama= nama;
    }
}
```

After creating a new project, the main function is automatically created. The example below will show how to call a method inside POJO class in the main.java.
Example: **main.java**

Source code	Result

`public Class Prak2 {`
`  public static void main(String[] args) {`
`  Mahasiswa m= new Mahasiswa();`
`  m.setNim("08018222");`
`  m.setNama("Rudi");`
`  System.out.println("Nim :"+ m.getNim());`
`  System.out.println("Nama :"+ m.getNama());`
`  }`
`}`
	Nim :08018222
	Nama :Rudi
	BUILD SUCCESSFUL (total time: 1 second)

2.3. TOOLS AND MATERIALS

Tools and materials used in this meeting:

1. Computer.
2. Intellij IDEA Community
3. Java SDK

2.4. INSTRUCTIONS

1. Create a new project and give a name: prak2.
2. Write the source code in the above example and run the application.
3. See and try to understand the result.
4. Create a new POJO class and give a name: **Rectangle**.
5. Define the attributes: width and height and methods getter, setter, calculate the area and perimeter of rectangle.
6. Show the result.
7. Try to do the same thing for class **Circle**.

2.5. ASSIGNMENT

Create a program to:

1. Make a **class Point** with attribute axis dan ordinate, method or operation to define a point, calculate the distance of two points, and find the center between two points.
2. Make a **class Timer** with attributes hour, minute, and second, method or operation that define a timer, and calculate the elapsed time.

MEETING 3: CONSTRUCTOR, ENCAPSULATION & INFORMATION HIDING

Meeting : 3

Time Allocation : 90 minutes
- Pre-Test : 15 minutes
- Practicum : 55 minutes
- Post-Test : 20 minutes

Scoring : 100%
- Pre-Test : 20 %
- Practicum : 40 %
- Post-Test : 40 %

3.1 GOALS AND LEARNING OUTCOMES

By following this session, students are expected to be able to:
1. Explain about **constructor**
2. Explain about **encapsulation** and **Information hiding**
3. Implement the topics with Java

Learning outcomes, students are expected to have the ability to:
1. Use **constructor** in application
2. Use **constructor** in **instance object**
3. Implement **encapsulation** and **information hiding** in Java application
4. Explain the difference when using **information hiding**

3.2 LITERATURE STUDY

Constructor is a pseudo-method that made in an object. In Java, constructor is an instance method that the name is the same as its class. Constructor can be called using **new**. For example:

```
public Class Bicycle{
    public Bicycle() {
           gear = 1
           speed = 0;
    }
}
```

Constructor will be called when an object is instantiated like the following:

```
Bicycle myBike = new Bicycle();
```

`new Bicycle()` will create a space in memory and initialize the object. A constructor is allowed to use overloading with the other constructor. Overloading is an identifier that refer to multiple items in the same scope. In Java, a method can be overloaded but not with variable and operator.
This example below shows how to implement overloading constructor:

```
public Class Bicycle{
    public Bicycle() {
```

```
            gear = 1
            speed = 0;
        }
        public Bicycle(int startSpeed, int startGear) {
            gear = startGear;
            speed = startSpeed;
        }
    }
```

To instantiate object from Bicycle Class, we can write:

```
Bicycle myBike = new Bicycle(30, 8);
```

There are two rules to create constructor:

1. Constructor name must be the same as the class name.
2. Constructor must not have a return type.

There are two types of constructor:

1. Default constructor is a constructor that does not have argument.
2. Parametric constructor is a constructor that has arguments.

There are several differences between constructor and method:

Java Constructor	Java Method
Constructor is used to initialize state in an object.	Method is used to show the behavior from object.
Constructor must not have a return type.	Method can have a return type.
Constructor can be called implicitly.	Method can be called explicitly
Java Compiler provides default constructor if the constructor is not defined.	Method is not provided by the compiler
The constructor's name must be the same as its class	The method's name can be different from its class.

Access modifier in Java

There are two types of modifier in Java namely access modifier and non-access modifiers.
Access modifiers in Java is specialized for accessibility of data member, method, constructor or class.
There are four types of access modifier in Java:

1. Private
2. Default
3. Protected
4. Public

There are several types of non-access modifiers such as static, abstract, synchronized, native, volatile, transient and so on. The example below shows how to use access modifier:

```
Class Students{
    private int data=40;
    private void msg(){
           System.out.println("Hello java");
    }
}

public Class Tester{
    public static void main(String args[]){
       Students obj = new Students();
       System.out.println(obj.data);//Compile Time Error
```

```
        obj.msg();//Compile Time Error
        }
}
```

The use of access modifiers:

Access Modifier	Class	Package	Subclass	Public
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Encapsulation di Java

In Java, encapsulation is a process to encapsulate code and data in a single unit. The advantage of encapsulation such as provide a setter and getter method and can make a Class become read-only or write-only. The encapsulation provides a control for accessing the data. The example below shows how to implement encapsulation in Java:

Example 3.1 **Class** declaration with filename Student.java

```
package com.uad;
public Class Student{
    private int age=20;
    private String nama;
    public String getNama(){
       return name;
    }

    public void setName(String name){
       this.name=name
    }
}
```

Example 3.2 **Class** declaration with filename Test.java

```
package com.uad;
Class Test{
   public static void main(String[] args){
     Student s=new Student();
     s.setName("Romeo");
     System.out.println(s.age);
     System.out.println(s.getNama());
   }
}
```

3.3 TOOLS AND MATERIALS

Tools and materials used in this meeting:

1. Computer.
2. Intellij IDEA Community
3. Java SDK

3.4 INSTRUCTIONS

1. Create a new project, give a name prak3.
2. Try the code in example 3.1 dan 3.2.
3. Modify the object **Class** (example 3.1) by adding a default constructor and parametric constructor. Declare a variable with defined value in the default constructor and save the value to a variable in the parametric constructor.

4. Instantiate an object using default constructor then print the value of variable from that object
5. Instantiate an object using parametric constructor then print the value of variable from that object.

3.5 ASSIGNMENT

1. Create a driver Class named Tester.java and an object Class named Karyawan.java.
2. Create a default constructor that has variable named gaji tetap and assign a value to that variable.
3. Create a parametric constructor that has a parameter then assign the parameter to variable named gaji tetap.
4. Create three variables named nomor pegawai, nama pegawai and gaji bonus.
5. Create a mutator and accessor method for those three variables.
6. Create a method to calculate the salary of new employees which equal to the value of gaji tetap.
7. Create a method to calculate salary of permanent employees which equal to the result of gaji tetap added by gaji bonus.
8. Print the information on the screen that includes the employee ID, employee name, and the result of salary computation for new employee and permanent employee.

MEETING 4: INHERITANCE

Meeting : 4

Time Allocation : 90 minutes
- Pre-Test : 15 minutes
- Practicum : 55 minutes
- Post-Test : 20 minutes

Scoring : 100%
- Pre-Test : 20 %
- Practicum : 40 %
- Post-Test : 40 %

4.1 GOALS AND LEARNING OUTCOMES

By following this session, students are expected to be able to understand about inheritance and its implementation.

Learning outcomes, students are expected to have the ability to implement inheritance in programming.

4.2 LITERATURE STUDY

Inheritance is the main characteristic of object oriented programming where the methods in the parent class will be inherited to the children class. The parent class called superclass and the new class inherited from the parent class called subclass. The subclass will inherit the parent class' methods and can have its own specific methods that not owned by the parent class. For example, Human class has name and address. Students class has NIM, name and address. The parent class of Human inherites the methods to its subclass Students.

Example 4.1 POJO parent class (Manusia.java)

```
public class Manusia {
    private String nama;
    private String alamat;

    public Manusia(){}

    public Manusia(String nama, String alamat) {
        this.nama = nama;
        this.alamat = alamat;
    }
    public String getAlamat() {
        return alamat;
    }
    public void setAlamat(String alamat) {
        this.alamat = alamat;
    }
    public String getNama() {
        return nama;
    }
    public void setNama(String nama) {
        this.nama = nama;
    }
}
```

Example 4.2 POJO children class (Mahasiswa.java)

```
public class Mahasiswa {
    private String nim;
    private String nama;
    private String alamat;

    public Mahasiswa() {
    }
    public Mahasiswa(String nim, String nama, String alamat) {
        this.nim = nim;
        this.nama = nama;
        this.alamat = alamat;
    }
    public String getAlamat() {
        return alamat;
    }
    public void setAlamat(String alamat) {
        this.alamat = alamat;
    }
    public String getNama() {
        return nama;
    }
    public void setNama(String nama) {
        this.nama = nama;
    }
    public String getNim() {
        return nim;
    }
    public void setNim(String nim) {
        this.nim = nim;
    }

}
```

See the attributes and methods in the above example. Some attributes and methods with the same/duplicate name are redundant. They should be merge using inheritance. For example: Human inherites its methods to Students because Students have all the methods in Human but Human does not have all methods in Students. To declare an inheritance of class X to class Y, we can use keyword “extend”. Therefore, we can change the Students class as follows:

Example 4.3 Class Inheritance (Mahasiswa.java)

```
public class Mahasiswa extends Manusia{
    private String nim;

    public Mahasiswa() {
    }
    public Mahasiswa(String nim) {
        this.nim = nim;
    }
    public String getNim() {
        return nim;
    }
    public void setNim(String nim) {
        this.nim = nim;
    }
}
```

Although Students class does not have attributes and methods of name and address, Students class actually has them via inheritance from Human class.

Example 4.4 Main.java

Source Code	Result

public class Main {
public static void main(String[] args) {
Mahasiswa m=new Mahasiswa();
m.setNim("06018251");
m.setNama("Tejo");
m.setAlamat("Sleman");
System.out.println("Nim :"+m.getNim());
System.out.println("Nama :"+m.getNama());
System.out.println("Alamat :"+m.getAlamat());
}
	Nim :06018251
	Nama :Tejo
	Alamat :Sleman
	BUILD SUCCESSFUL (total time: 1 second)

4.3 TOOLS AND MATERIALS

Tools and materials used in this meeting:

1. Computer.
2. Intellij IDEA Community
3. Java SDK

4.4 INSTRUCTIONS

1. Create a new project called Praktikum4.
2. Try all of the above examples.

4.5 ASSIGNMENT

Create a new project and follow the instructions above but using a case study to create a class named Circle and class named Square. Create all of the necessary methods that inherited from a parent class named Point.

MEETING 5: POLYMORPHISM

Meeting : 5

Time Allocation : 90 minutes
- Pre-Test : 15 minutes
- Practicum : 55 minutes
- Post-Test : 20 minutes

Scoring : 100%
- Pre-Test : 20 %
- Practicum : 40 %
- Post-Test : 40 %

5.1 GOALS AND LEARNING OUTCOMES

By following this session, students are expected to be able to implement polymorphism in OOP.
Learning outcomes, students are expected to have the ability to implement polymorphism in OOP.

5.2 LITERATURE STUDY

Polymorphism means having many form. Two objects called polymorphic if the objects have identical interface but different behaviors. Polymorphism uses single name to define objects in the different classes that related to a common superclass. Using polymorphism, it can be recognized and exploited a similiraty between different classes. There are two form of polymorphism:

- **Overloading**:
  Using single name to define some different methods in a class. The methods are differentiated based on the number of parameter and the data type of the parameters.
- **Overriding**:
  A modified (overrided) method that declared using the same name and parameters from the superclass.

In polymorphism also known a term called dynamic binding that used in class type variable. If we have a superclass type variable, this variable can be initiated with the superclass object or subclass object without changing the type.

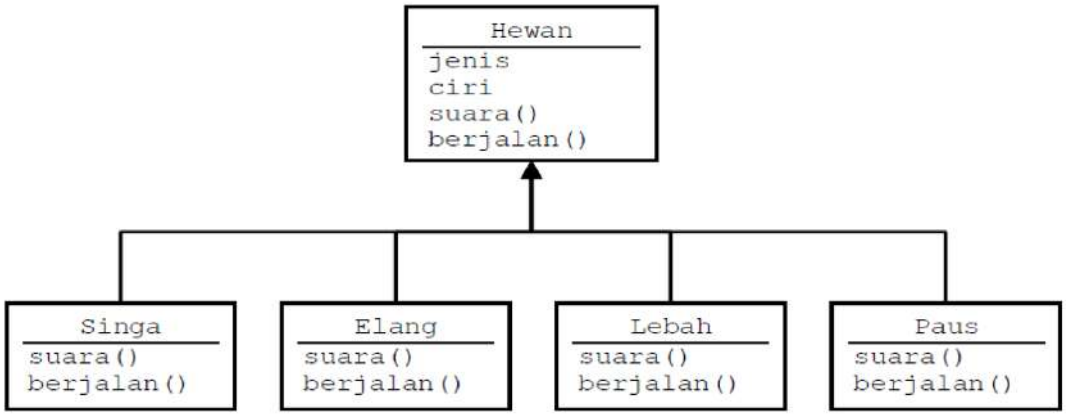


Figure 5.1. The example of Polymorphism

In the figure above, the method suara() from superclass Hewan can be any type depend on the subclass.

Example 1:

```
class EkspresiWajah{
    public String respons() {
        return("Perhatikan ekspresi wajah saya");
    }
}
class Gembira extends EkspresiWajah{
    public String respons() {
        return("ha ha ha…");
    }
}
class Sedih extends EkspresiWajah{
    public String respons() {
        return("hik hik ngeee ngeee ngeee");
    }
}
class Marah extends EkspresiWajah{
    public String respons() {
        return("Hai kurang ajar…!");
    }
}


class MainEkspresiWajah{
    public static void main(String args[]) {
       EkspresiWajah objEkspresi = new EkspresiWajah();
        Gembira objGembira = new Gembira();
        Sedih objSedih = new Sedih();
        Marah objMarah = new Marah();

        EkspresiWajah[] arrEkspresi = new EkspresiWajah[4];
        arrEkspresi[0] = objEkspresi;
        arrEkspresi[1] = objGembira;
        arrEkspresi[2] = objSedih;
        arrEkspresi[3] = objMarah;

        System.out.println("Ekspresi[0] : "+arrEkspresi[0].respons());
        System.out.println("Ekspresi[1] : "+arrEkspresi[1].respons());
        System.out.println("Ekspresi[2] : "+arrEkspresi[2].respons());
        System.out.println("Ekspresi[3] : "+arrEkspresi[3].respons());
    }
}
```

Example 2:

```
public class Employee {
    private String name;
    private double salary;
    protected Employee(String n, double s) {
        name = n;
        salary = s;
    }
    protected String getDetails() {
        return "Name : "+name+ "\nSalary : "+salary;
    }
    public void cetak() {
        System.out.println("coba di Employee");
    }
}
public class Manager extends Employee {
    private String department;
    public Manager(String nama, double salary, String dep) {
        super(nama, salary);
        department = dep;
    }
```

```
        public String getDepartment() {
            return department;
        }
        public String getDetails() {
            return super.getDetails()+ "\nDepartment : "+getDepartment();
        }
        public void cetak() {
            System.out.println("Coba di Manager");
        }
    }

    public class View {
        public static void main(String[] args) {
           Employee e = new Employee("Ali",1200000);
           Employee em = new Manager("Ali",1200000,"Informatika");
           System.out.println("Data employee :\n"+e.getDetails());
           System.out.println("Data manager :\n"+em.getDetails());
           em.cetak();
        }
    }
```

Note:

If cetak() method exists in the Employee class and Manager class then the one that running is the method in Manager class. The priority will be Manager class then Employee class.

5.3 TOOLS AND MATERIALS

Tools and materials used in this meeting:

1. Computer.
2. Intellij IDEA Community
3. Java SDK

5.4 INSTRUCTIONS

1. Create a new project named prak5.
2. Try to code the two examples above.
3. Create a new POJO class named Mahasiswa as follows:

```
public class Mahasiswa {
    private String nim;
    private String nama;

    public Mahasiswa() {
    }
    public Mahasiswa(String nim, String nama) {
        this.nim = nim;
        this.nama = nama;
    }
    public String getNama() {
        return nama;
    }
    public void setNama(String nama) {
        this.nama = nama;
    }
    public String getNim() {
        return nim;
    }
    public void setNim(String nim) {
        this.nim = nim;
    }
}
```

4. Create a class named HM which is a class that gain inheritance from Mahasiswa class and create the necessary methods for the HM class.

5. Create a class named KelompokStudi which is a class that gain inheritance from Mahasiswa class and create the necessary methods for the KelompokStudi class.
6. Create one object to instantiate the HM class and demonstrate the implementation of the methods that you have created before in the HM class.
7. Call the objects that you have instantiated from the KelompokStudi and HM classes to demonstrate the work to show the effect of polymorphism.

5.5 ASSIGNMENT

Build a program from the illustration in figure 5.1 including the implementation of the methods and demonstrate the implementation in a case.

MEETING 6: CLASS RELATION

Meeting : 6

Time Allocation : 90 minutes
- Pre-Test : 15 minutes
- Practicum : 55 minutes
- Post-Test : 20 minutes

Scoring : 100%
- Pre-Test : 20 %
- Practicum : 40 %
- Post-Test : 40 %

6.1 GOALS AND LEARNING OUTCOMES

By following this session, students are expected to be able to:
1. Analyze the relation between classes.
2. Implement the relation between classes into a program.

Learning outcomes, students are expected to have the ability to:
1. Implement the relation between classes using Java.
2. Explain the relation between classes from the code in a program.

6.2 LITERATURE STUDY

Introduction to Array

We will review about array before we talk about class. Array is used to group a data or object that has the same type. Array is declared using primitive data type or object. For example:

```
char c[];
titik t[]; // t is a class
```

In Java, array considered as object eventhough the element of array consists of primitive data type. When declaring an array, the object array is not immediately instantiated. Array declaration only created a reference that can be used to refer an object array. The capacity of memory that will be used by elements in an array will be allocated dynamically using new statement or array initializer.

The example of array declaration which uses square bracket after the name of variable, is a standard array declaration in C, C++ and Java. In Java, there is an alternative to write array decalaration using square bracket on the left of the name of variable as shown in the example below:

```
char [] c;
titik [] t; // titik is a class
```

When declaring an array with square bracket on the left of the variable name, the square bracket is valid for every variable on the right

Creating an Array

Array can be made using keyword “new”. For example:

```
c = new char[26];
```

Index of array is started from 0. The valid range of index of array is from 0 to the number of element in an array - 1. If the program tries to access array outside the valid index, then the runtime exception occurred. To create array of object, we can use:

```
t = new titik[5];
```

That statement is not yet created five objects of Titik. To create the Titik object, we can use:

```
t[0] = new titik(1, 5);
t[1] = new titik (2, 4);
```

Array Initialization

Java has two ways to create an array of object with initialization:

```
String names[] = {"Anni","Syauqi","Savitri"};
```

That code is similar to this:

```
String names[];
names[0] = "Anni";
names[1] = "Syauqi";
names[2] = "Savitri";
```

Multi Dimensional Array

In Java, an array can be created from array hence the array called multi-dimensional array. For example:

```
int twoDim [][] = new int[4][];
twoDim[0] = new int[5];
twoDim[1] = new int[5];
```

Array Boundary

In Java, the index of array started from 0. The number of element in an array is stored as a part of array object in the length attribute. This attribute can be used to perform iteration of array as shown in the example below:

```
int list[] = new int[10];
for(int i = 0; i < list.length; i++) {
System.out.println(list[i]);
}
```

Using the length attribute, the maintenance of the code become easier because the program does not need to know the number of element in an array when compiling the code.

Changing the Size of Array

After creating an array object, the size of array is fixed. However, we can use similar reference variable to refer the new array object as shown in the example below:

```
int myArray[] = new int[6];
myArray = new int[10];
```

Copying Array

Java provides a special method to copy an array using arrayCopy() method from System class as shown in the example below:

```
// array semula
int myArray[] = {1, 2, 3, 4, 5, 6};
// array engan elemen yang lebih banyak
int hold[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
//menyalin semua elemen myArray ke hold
// dimulai dengan indeks ke 0
System.arraycopy(myArray, 0, hold, 0,myArray.length);
```

After the copying process is done, the element of hold array become 1, 2, 3, 4, 5, 6, 4, 3, 2, 1.

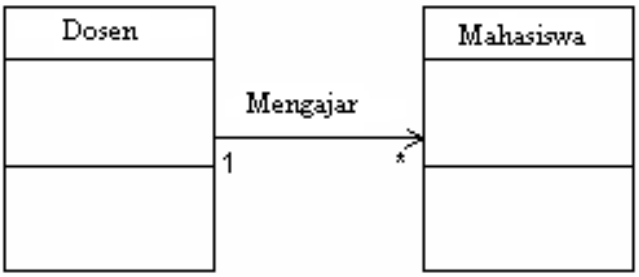
Relation Between Classes

In OOP, the classes may have a relation between them. The relation depends on the classes condition. There are some kinds of relation between classes, for example:

1. Association
2. Agregation
3. Composition
4. Generalization (related to inheritance)
5. Spesialization (related to inheritance)

1. Association

Association is a structural relation between classes that illustrates a set of link between objects. For example:



The implementation of association diagram can be seen in the Example 1:

Example 1:

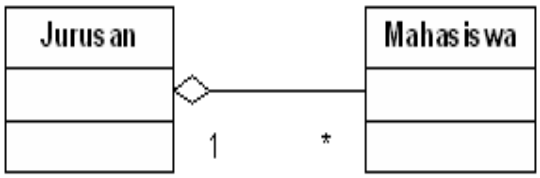
```
class Mahasiswa {
      private String nim;
      private String nama;

      public Mahasiswa(String nim, String nama)
      {
            this.nim=nim;
            this.nama=nama;
      }
      public void setnama (String nama) {
      this.nama = nama;
      }
```

```
	public void setnim (String nim) {
	this.nim = nim;
	}
	public String getNim() {
	return this.nim;
	}
	public String getNama () {
	return this.nama;
	}
}

class Dosen {
	private String Kddosen;
	private String[] nimMHS=new String[5];
	private int jmlMahasiswa=0;

	public Dosen(String kode)
	{
		this.Kddosen=kode;
	}
	public void setKddosen (String Kddosen) {
		this.Kddosen = Kddosen;
	}
	public void setNimMahasiswa (String nim) {
		nimMHS[jmlMahasiswa]=nim;
		jmlMahasiswa++;
	}
	public int getJmlMahasiswa () {
	return this.jmlMahasiswa;
	}
	public String getKddosen () {
	return this.Kddosen;
	}
	public void daftarMahasiswa() {
		System.out.println("Kode Dosen "+Kddosen);
		System.out.println("Daftar Mahasiswa");
		for (int i=0;i<jmlMahasiswa;i++)
		{
			System.out.println(nimMHS[i]);
		}

	}
}
```

```
class MahasiswaDosen
{
	public static void main(String[] args)
	{
	Mahasiswa mhs1 = new Mahasiswa("30107998","Abdul Kadir");
	Mahasiswa mhs2 = new Mahasiswa("30107999","Asep Sumarta");
		Dosen dsn = new Dosen("SKS");
		dsn.setNimMahasiswa(mhs1.getNim());
		dsn.setNimMahasiswa(mhs2.getNim());
		dsn.daftarMahasiswa();
	}
}
```

2. Aggregation

Aggregation is a relation between two classes where one of the class is a part of another class but both of the classes can stand independently. Aggregation often called “part of” relation or “whole-part” relation. For example:



The implementation of aggregation diagram can be seen in the example 2:

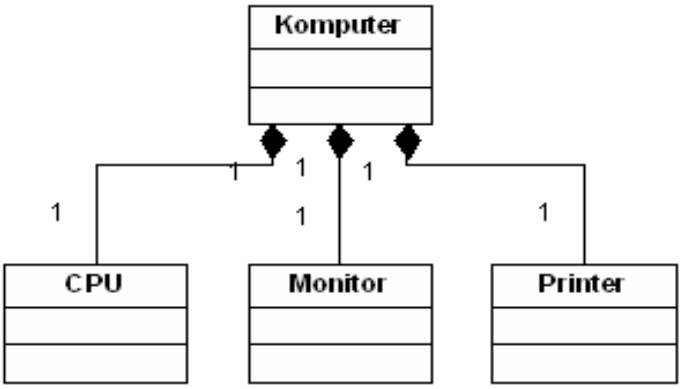
Example 2:

```
	//mahasiswa.java
	public class mahasiswa {
		private String NIM, Nama;
	public mahasiswa(String no, String nm) {
			this.NIM = no;
			this.Nama = nm;
		}
	public String GetNIM() {
			return (NIM);
		}
		public String GetNama() {
			return (Nama);
		}
	}
	//jurusan.java
	public class Jurusan {
	     private String KodeJurusan, NamaJurusan;
	     private Mahasiswa[] Daftar = new Mahasiswa[10];
	public Jurusan(String kode, String nama) {
			this.KodeJurusan = kode;
```

```
            this.NamaJurusan = nama;
        }
        private static int JmlMhs = 0;
        public void AddMahasiswa(Mahasiswa m) {
            this.Daftar[JmlMhs] = m;
            this.JmlMhs++;
        }
    public void DisplayMahasiswa() {
        int i;
        System.out.println("Kode Jurusan : "+this.KodeJurusan);
        System.out.println("Nama Jurusan : "+this.NamaJurusan);
        System.out.println("Daftar Mahasiswa :");
        for (i=0;i<JmlMhs;i++)
    System.out.println(Daftar[i].GetNIM()+" "+Daftar[i].GetNama());
        }
    }
```

3. Composition

Composition is a special form of aggregation where the new part class can be created after the whole class is made and when the whole class is destroyed then the new part class also get destroyed. For example:



The implementation of composition diagram in Java is shown in the code below:

```
//CPU.java
public class CPU {
private String Merk;
private int Kecepatan;
public CPU(String m, int k) {
this.Merk = m;
this.Kecepatan = k;
}
public void DisplaySpecCPU() {
System.out.println(this.Merk + ", " + this.Kecepatan);
    }
}
```

```
//Monitor.java
public class Monitor {
private String Merk;
public Monitor(String m) {
this.Merk = m;
      }
public void DisplaySpecMonitor() {
            System.out.println(this.Merk);
      }
}

//Printer.java
public class Printer {
private String Merk, Type;
public Printer(String m, String t) {
            this.Merk = m;
            this.Type = t;
      }
public void DisplaySpecPrinter() {
            System.out.println(this.Merk + ", " + this.Type);
      }
}

//Komputer.java
public class Komputer {
private String Kode;
private long Harga;
      private CPU Proc;
      private Monitor Mon;
      private Printer Prn ;

public Komputer(String k, long h) {
      this.Kode = k;
            this.Harga = h;
      if (k == "PC-01") {
      Proc = new CPU("Pentium IV", 500);
      Mon = new Monitor("Sony Multiscan 15sf");
Prn = new Printer("Canon BJC-210SP", "Color");
    }
}
      public void DisplaySpec() {
```

```
                System.out.println("Kode      : " + this.Kode);
            System.out.print("Processor: ");
            Proc.DisplaySpecCPU();
                System.out.print("Monitor   : ");
            Mon.DisplaySpecMonitor();
        System.out.print("Printer   : ");
                        Prn.DisplaySpecPrinter();
                        System.out.println("Harga : Rp. " + this.Harga);
            }
        }
```

6.3 TOOLS AND MATERIALS

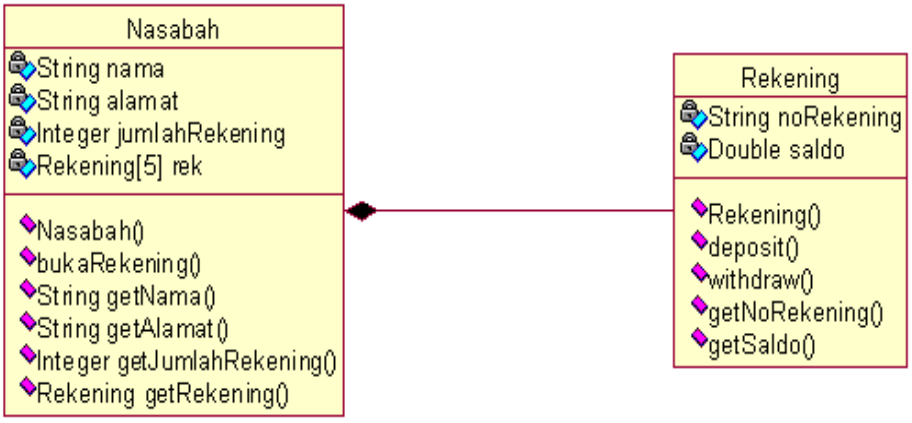
Tools and materials used in this meeting:

1. Computer.
2. Intellij IDEA Community
3. Java SDK

6.4 INSTRUCTIONS

Implement and run the code of class relation such as association, aggregation and composition in the above example. Give a comment where is the part that implement the relation between classes in the code!

6.5 ASSIGNMENT



Gambar 6.4. Account Holder Diagram

The class diagram in Figure 6.4 shows the relation between Nasabah class and Rekening class. An account holder can create maximal 5 accounts in a bank.
In the Nasabah class:

- There are some attributes, “nama” to store the name of the account holder, “alamat” to store the address of account holder, “jumlahRekening” to store the balance, and “rek” to store the account data.
- There is a constructor to set the account holder name and address.
- There is a method named “bukaRekening” to create a new account.

In the Rekening class:

- There are some attributes, “noRekening” to store the account number and “saldo” to store the account balance.
- There is a method named “deposit” to add some money to the account balance.
- There is a method named “withdraw” to withdraw some money from the account balance.

Question:

- Explain the relation type of class diagram in Figure 6.4.!
- Create a Java code to implement the class diagram in Figure 6.4!
- Create a tester class to show the scenario of an account holder and his account! (initiated the necessary data by yourself)

MEETING 7: ABSTRACT CLASS

Meeting : 7

Time Allocation : 90 minutes
- Pre-Test : 15 minutes
- Practicum : 55 minutes
- Post-Test : 20 minutes

Scoring : 100%
- Pre-Test : 20 %
- Practicum : 40 %
- Post-Test : 40 %

7.1 GOALS AND LEARNING OUTCOMES

By following this session, students are expected to be able to:
1. Explain abstract class.
2. Create an example and implementation of abstract class.

Learning outcomes, students are expected to have the ability to:
1. Implement the abstract class.
2. Code abstract class.

7.2 LITERATURE STUDY

Abstract Class is a class that has the highest hierarchy among the class hierarchy. As the name imply, abstract class can be defined by the class itself. For example:

```
abstract class NamaKelas{
  //the content of abstract class
}
Example:
abstract class Manusia{
  //the content of abstract class
}
```

In abstract class, abstract method also allowed but its optional. Abstract method does not need a statement in the method. For example:

```
abstract class NamaKelas{
  //for a procedure
 [access_modifier] abstract void [namaMethod]();

  //for a function
 [access_modifier] abstract [tipe_data] [namaMethod]();
 }
 Example:
 abstract class Manusia{
  //for a procedure
 public abstract void setNama();
```

```
    //for a function
    public abstract String getNama();
    }
```

Note:

1. If there is exist an abstract method in the abstract class and the class is inherited to the children class, the method must be overrided with a statement in the method content.
2. If the class is an abstract class, then the class may have an abstract method or not (optional). If a class has an abstract method then the class must be abstract class.

Keyword "final"

Keyword "final" is used to avoid inheritance of a class or to avoid overriding a method. For example:

```
final class NamaKelas{
// the content of the class
}
```

Example

```
final class Manusia{
      //the content of the class
}
```

To declare "final" in the method:

```
final class NamaKelas{
//for a procedure
}
```

Example:

```
final class Manusia{
      //for a procedure
      public final void cetakBeratBadanIdeal(){
            //the content of the method
      }
      //for a function
      public final double hitungBeratBadanIdeal(){
            //the content of the method
            return 0;
      }

}
```

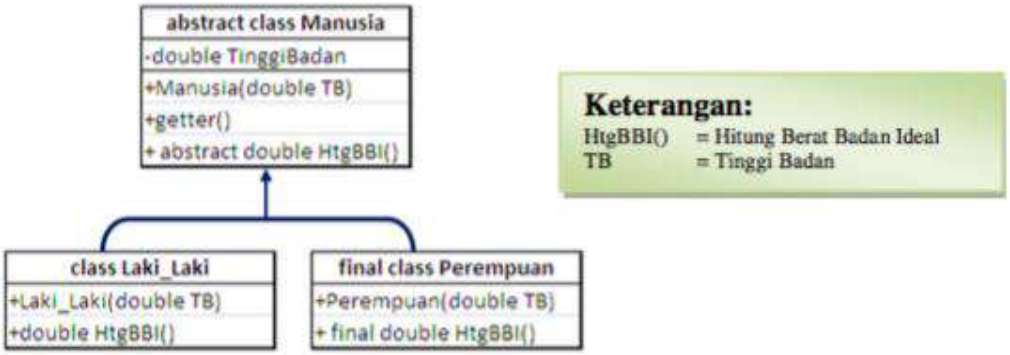
7.3 TOOLS AND MATERIALS

Tools and materials used in this meeting:

1. Computer.
2. Intellij IDEA Community
3. Java SDK.

7.4 INSTRUCTIONS

1. Create a new project named prak8
2. Take a look at this class diagram:



To compute ideal weight:

a. Male = (height(cm)-100) kg X 90%
b. Female = (height(cm)-100) kg X 80%

3. Create a POJO class named Manusia.

```
public class Manusia {
    private double tinggiBadan;

    public Manusia () {
    }
    public Manusia (double tb) {
        this.tinggiBadan = tb;
    }
    public double getTinggi() {
        return tinggiBadan;
    }
    public abstract doule hitungBeratBadanIdeal();
}
```

4. Create a class named Lakilaki.

```
public class Lakilaki extends Manusia{

    public Lakilaki (double tinggiBadan) {
        super(tinggiBadan);
    }
    public doule hitungBeratBadanIdeal(){
        return (super.getTinggiBadan()-100)*0.9;
    }
}
```

5. Create a class named Perempuan.

```
final public class Perempuan extends Manusia {
    public Perempuan (double tinggiBadan) {
        super(tinggiBadan);
    }
    public final double hitungBeratBadanIdeal(){
        return (super.getTinggiBadan()-100)*0.8;
}
```

The keyword “final” in the Perempuan class is used to avoid creating a new inheritance class from the Perempuan class. The keyword “final” in the method is used to avoid overriding a method in the inherited class.

7.5 ASSIGNMENT

1. Create a tester class from the above example.
2. Instantiate an object from Manusia object then initiate a value for the height and compute the ideal weight of that Manusia object.

3. Instantiate an object from Lakilaki object then initiate a value for the height and compute the ideal weight of that Lakilaki object.
4. Instantiate an object from Perempuan object then initiate a value for the height and compute the ideal weight of that Perempuan object.
5. Create a new class named PerempuanDewasa as an inheritance from Perempuan class and perform overriding method hitungBeratBadanIDeal with formula to compute the ideal weight = (height(cm)-100) kg X 85%.
6. Instantiate an object from PerempuanDewasa object then initiate a value for the height and compute the ideal weight of that PerempuanDewasa object.

MEETING 8: INTERFACE

Meeting : 8

Time Allocation : 90 minutes
- Pre-Test : 15 minutes
- Practicum : 55 minutes
- Post-Test : 20 minutes

Scoring : 100%
- Pre-Test : 20 %
- Practicum : 40 %
- Post-Test : 40 %

8.1 GOALS AND LEARNING OUTCOMES

By following this session, students are expected to be able to: Create and declare a class interface.

Learning outcomes, students are expected to have the ability to:
1. Implement class interface.
2. Code class interface.

8.2 LITERATURE STUDY

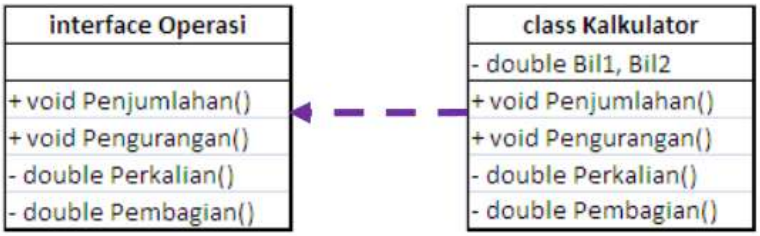
Interface is a method consists of declaration and structure without the details of implementation. For example:

```
interface Nama_Interface
{
    //deklarasi variabel dan/atau method
}
```

Example:

```
interface Operasi
{
    //deklarasi variabel dan/atau method
    public void Penjumlahan();
    public void Pengurangan();
    public double Perkalian();
    public double Pembagian();
}
```

In the above example, the declaration of interface does not contain any statement, neither a formula or return value. Interface only consists of constants or methods without the information about the body method. Interface is not a class and classes can only implement interfaces. Therefore, interface is not a superclass. The implementation of interface in a class can be seen through an OOP scheme below:



The dashed arrow illustrates that Kalkulator class implements interface Operasi, where the methods in an interface Operasi must be overrided from the Kalkulator class. Interface is symbolized using dashed arrow while inheritance is symbolized using straight arrow.

In OOP, the implementation of interface uses keyword “implements”. For example:

```
class Nama_Kelas implements Nama_Interface
{
    //isi kelas
}
```

The implementation of interface:

```
class Kalkulator implements Operasi
{
    public void Penjumlahan()
    {
        //isi method Penjumlahan()
    };

    public void Pengurangan()
    {
        //isi method Pengurangan()
    };

    public double Perkalian()
    {
        //isi method Perkalian()
        return 0;
    };

    public double Pembagian()
    {
        //isi method Pembagian()
        return 0;
    };
};
```

8.3 TOOLS AND MATERIALS

Tools and materials used in this meeting:

1. Computer.
2. Intellij IDEA Community
3. Java SDK

8.4 INSTRUCTIONS

1. Create a new project named prak8.
2. Write the code below and observe the result!

```
/*interface berisi abstract method atau method header*/
interface MyComparable {
boolean greaterThan(Object obj); boolean lessThan(Object obj);
boolean equal(Object obj);
}
/*interface berisi konstanta*/
interface Constants {
int min = 1000;
int max = 9999;
}
/*class mengimplementasikan dua interface*/
class FourDigitsNumber implements Constants, MyComparable {
private int value;
public FourDigitsNumber(int initValue) {
/*latihan 1: atur agar nilai yang diinputkan dalam constructor hanya
berada di antara min - max
*/
}
/*latihan 2: tambahkan method get untuk mengakses value*/
/*overriding method greaterThan*/
public boolean greaterThan(Object obj) {
/*casting from superclass to subclass*/ FourDigitsNumber number =
(FourDigitsNumber)obj; return ( value > number.getValue() );
}
/*latihan 3: lengkapi overriding method interface*/
}
/*Contoh penggunaan class*/
class ExInterface {
public static void main(String [] args) { FourDigitsNumber number1 =
new FourDigitsNumber(700); FourDigitsNumber number2 = new
FourDigitsNumber(1700); FourDigitsNumber number3 = new
FourDigitsNumber(70000); System.out.println(number1.getValue());
System.out.println(number2.getValue());
System.out.println(number3.getValue());
System.out.println(number1.greaterThan(number2));
System.out.println(number1.lessThan(number2));
System.out.println(number1.equal(number2));
}}
```

3. Do the exercise in the above example!

8.5 ASSIGNMENT

Create a new program that illustrate the implementation of interface Operasi below into the Kalculator class that can compute two numbers Bil1 and Bil2 using a method in the interface Operasi!

```
1  interface Operasi
2  {
3      //deklarasi variabel dan/atau method
4      public void Penjumlahan();
5      public void Pengurangan();
6      public double Perkalian();
7      public double Pembagian();
8  }
```

MEETING 9: EXCEPTION HANDLING AND I/O

Meeting : 9

Time Allocation : 90 minutes
- Pre-Test : 15 minutes
- Practicum : 55 minutes
- Post-Test : 20 minutes

Scoring : 100%
- Pre-Test : 20 %
- Practicum : 40 %
- Post-Test : 40 %

9.1 GOALS AND LEARNING OUTCOMES

By following this session, students are expected to be able to implement exception handling and I/O. **Learning outcomes**, students are expected to have the ability to implement exception handling and I/O.

9.2 LITERATURE STUDY

The programs that need to acquire external data will use an input and ouput (I/O) process. Java supports I/O process in the java.io package. The package contains many classes and interfaces that ready to be used to ease the programmer when dealing with process of acquiring or storing information from/to another media e.g. file.

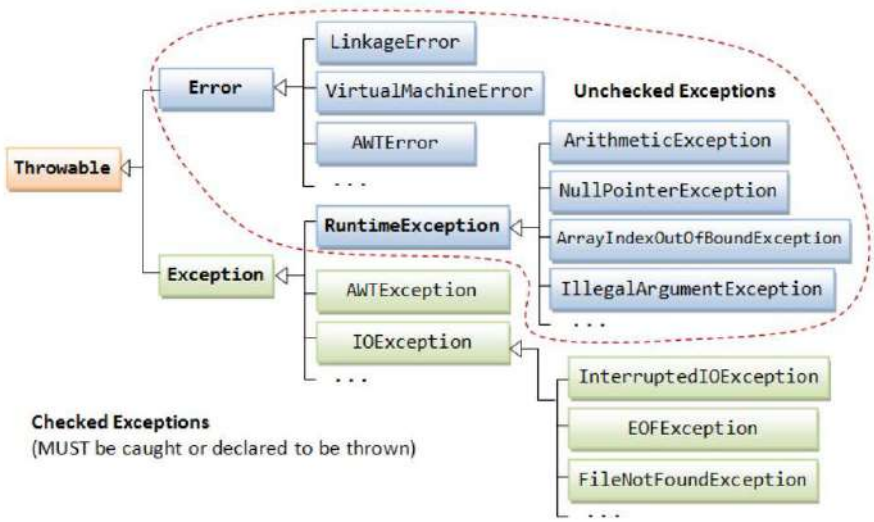
Java program performs I/O process through stream, which is an abstraction that gives or gets information. Stream can be connected to a physical equipment that listed in Java I/O system e.g. keyboard, file, console screen, network socket, etc. To write a text in a console screen or to a file, we can use the same class and method of Stream. Stream consists of two types, byte stream and character stream. Byte stream is used to give or store information in the form of byte. For example, to write and read a binary file. Stream character process data in the form of character. For example, process to read/write to a text file using Unicode character. Stream is defined using four abstract classes which are InputStream and OutputStream, as a superclass for a classes in the byte stream category and abstract class Reader and Writer for character stream category. Through an inheritance process, every inherited class from InputStream or Reader will have a method read() that can be used to read a data. To write a data, a method write() can be used in every class that inherited from OutputStream or Writer.

Exception is a runtime error. Exception can occur if the program is not run as its must. The flow of program will change and the application will stop with error. Exception occurs because there is no anticipation for an error in the program. For example, program is designed to compute number that inputted by the user. If the user enters a character that can not be calculated for example a letter, then the exception will occur. In Java, there is a mechanism to deal with runtime error which is exception handling and special class for exception. The hierarchy of this class is shown in Figure 9.1. There are two exceptions in Java:

1. *Checked Exception*. Is an exception that must be catched or throwed explicitly. If there is no

method that configured to catch this exception, then the program can not be compiled.

2. *Unchecked Exception*. Is an exception that can be made to make a reliable program.



Gambar 9.1 Hierarchy of exception class (Source: https://www.ntu.edu.sg/home/ehchua/programming/java/J5a_ExceptionAssert.html)

9.3 TOOLS AND MATERIALS

Tools and materials used in this meeting:

1. Computer.
2. Intellij IDEA Community
3. Java SDK

9.4 INSTRUCTIONS

1. Create a new project named Prak9.
2. Create a POJO class named ContohOutput as shown in Figure 9.2 and observe the syntax.

```
1  package IO;
2
3  public class ContohOutput {
4      public static void main(String[] args) {
5          int varInt = 45;
6          double varDouble = 7.98845;
7          float varFloat = 6.97f;
8          char varChar = 'X';
9          String varString = "Ini sebuah String";
10
11         System.out.printf("%d\n", varInt);
12         System.out.printf("%e\n", varDouble);
13         System.out.printf("%f\n", varFloat);
14         System.out.printf("%.2f\n", varFloat);
15         System.out.printf("%c\n", varChar);
16         System.out.printf("%.3s\n", varString);
17         System.out.printf("%s\n", varString);
18
19         System.out.println("=========================");
20         System.out.print("Ini adalah ");
21         System.out.print("contoh penggunaan print");
22
23         System.out.println();
24         System.out.println("=========================");
25         System.out.println("Ini adalah ");
26         System.out.println("contoh penggunaan println");
27     }
28 }
```

```
Output - Prak9 (run)
run:
45
7.988450e+00
6.970000
6.97
X
Ini
Ini sebuah String
=========================
Ini adalah contoh penggunaan print
=========================
Ini adalah
contoh penggunaan println
```

Gambar 9.2 The result of example 1.

3. Create a POJO class named ContohInput as shown in Figure 9.3 and observe the syntax!

```
1   package IO;
2
3   import java.io.BufferedReader;
4   import java.io.IOException;
5   import java.io.InputStreamReader;
6
7   public class ContohInput {
8       public static void main(String[] args) {
9           String nama = "";
10          BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
11          System.out.println("Nama : ");
12          try {
13              nama = input.readLine();
14          } catch(IOException e) {
15              e.printStackTrace();
16          }
17          System.out.println("Nama anda : " + nama);
18      }
19  }
```

```
Output - Prak9 (run)
run:
Nama :
Herman
Nama anda : Herman
```

Gambar 9.3 The result of example 2.

4. Create a POJO class named ContohInputScanner as shown in Figure 9.4 and observe the syntax.

```
1   package IO;
2
3   import java.util.Scanner;
4
5   public class ContohInputScanner {
6       public static void main(String[] args) {
7           Scanner input = new Scanner(System.in);
8           System.out.println("Nama : ");
9           String nama = input.nextLine();
10          System.out.println("Nama anda : " + nama);
11      }
12  }
```

```
Output - Prak9 (run)
run:
Nama :
Herman
Nama anda : Herman
```

Gambar 9.4 The result of example 3.

5. Create a POJO class named ContohInputJOptionPane as shown in Figure 9.5 and observe the syntax.

```
1   package IO;
2
3   import javax.swing.JOptionPane;
4
5   public class ContohInputJOptionPane {
6       public static void main(String[] args) {
7           String nama = JOptionPane.showInputDialog(null, "Nama : ");
8           JOptionPane.showMessageDialog(null, "Nama anda : " + nama);
9       }
10  }
11
12
13
14
```

```
Input
Nama :
Herman
OK  Cancel
```

```
Message
Nama anda : Herman
OK
```

Gambar 9.5 The result of example 4.

6. Create a POJO class named ContohClassFile as shown in Figure 9.6 and observe the syntax.

```java
package IO;

public class ContohClassFile {
    public static void main(String[] args) {
        java.io.File file = new java.io.File("teks-ku.txt");
        try {
            if (file.createNewFile())
                System.out.println("File Berhasil Dibuat");
            else
                System.out.println("File Gagal Dibuat");
        } catch (Exception e) {
            System.out.println("Error");
        }

        System.out.println("Apakah file ada? " + file.exists());
        System.out.println("Apakah bisa dibaca? " + file.canRead());
        System.out.println("Apakah bisa ditulis? " + file.canWrite());
        System.out.println("Apakah berupa direktori? " + file.isDirectory());
        System.out.println("Apakah berupa file? " + file.isFile());
        System.out.println("Apa namanya? " + file.getName());
        System.out.println("Dimana lokasinya? " + file.getPath());
        System.out.println("Dimana lokasi lengkapnya? " + file.getAbsolutePath());
    }
}
```

```
Output - Prak9 (run)
run:
File Berhasil Dibuat
Apakah file ada? true
Apakah bisa dibaca? true
Apakah bisa ditulis? true
Apakah berupa direktori? false
Apakah berupa file? true
Apa namanya? teks-ku.txt
Dimana lokasinya? teks-ku.txt
Dimana lokasi lengkapnya? D:\Developer\Netbeans\Projects\Prak9\teks-ku.txt
```

Gambar 9.6 The result of example 5.

7. Create a POJO class named ContohTulisFile as shown in Figure 9.7 and observe the syntax.

```java
package IO;

import java.io.FileNotFoundException;

public class ContohTulisFile {
    public static void main(String[] args) {

        java.io.File file = new java.io.File("teks-ku.txt");

        try {
            java.io.PrintWriter output = new java.io.PrintWriter(file);
            output.println("Herman Yuliansyah, S.T., M.Eng.");
            output.println("Pemrograman Berorientasi Obyek");
            output.println("Universitas Ahmad Dahlan");
            output.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

```
teks-ku.txt - Notep...
File Edit Format View Help
Herman Yuliansyah, S.T., M.Eng.
Pemrograman Berorientasi Obyek
Universitas Ahmad Dahlan
```

Gambar 9.7 The result of example 6.

8. Create a POJO class named ContohBacaFile as shown in Figure 9.2 and observe the syntax.

```java
package IO;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class ContohBacaFile {
    public static void main(String[] args) {

        java.io.File file = new java.io.File("teks-ku.txt");
        try {
            Scanner input = new Scanner(file);
            input.useDelimiter("\n");
            while(input.hasNext()) {
                String nama = input.next();
                String mk = input.next();
                String pt = input.next();
                System.out.println("Nama : " + nama);
                System.out.println("Mata Kuliah : " + mk);
                System.out.println("Perguruan Tinggi : " + pt);
            }
            input.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

```
Output - Prak9 (run) - E...
Output - Prak9 (run)
run:
Nama : Herman Yuliansyah, S.T., M.Eng.
Mata Kuliah : Pemrograman Berorientasi Obyek
Perguruan Tinggi : Universitas Ahmad Dahlan
```

Gambar 9.8 The result of example 7.

9.5 ASSIGNMENT

In a wood cutter shop, a User want to buy a block of wood with size that will be inputted by the User itself. However, the shop only stocks a block of wood that has length greater than the width and the width is greater than the height.

1. Compute the volume of block of wood that the User want to buy then show the result on the screen and save it to a file.
2. Give an *exception handling* for the input to check the mistake when inputting the value of length, width and height.

MEETING 10: GAME

Meeting : 10

Time Allocation : 90 minutes
- Pre-Test : 15 minutes
- Practicum : 55 minutes
- Post-Test : 20 minutes

Scoring : 100%
- Pre-Test : 20 %
- Practicum : 40 %
- Post-Test : 40 %

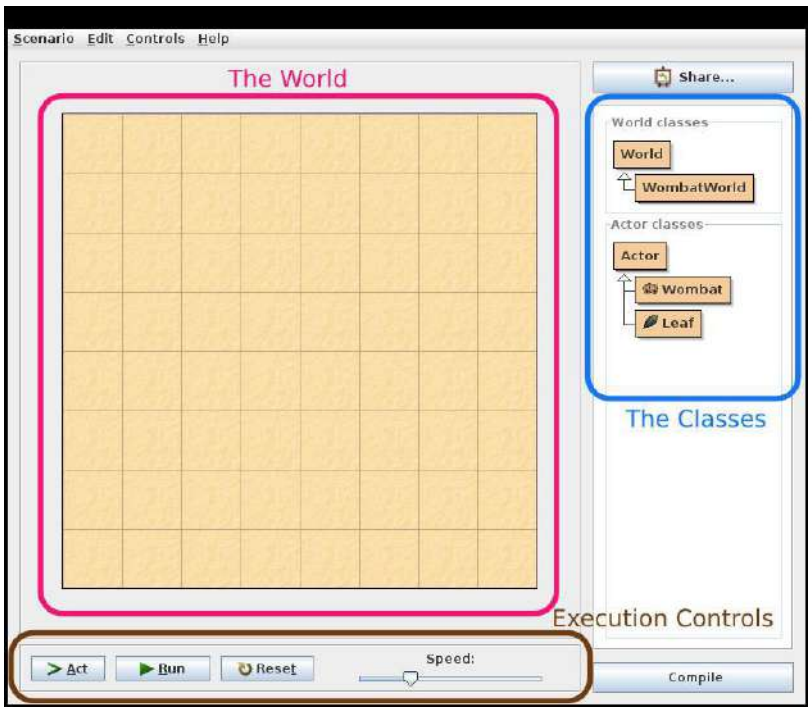
10.1 GOALS AND LEARNING OUTCOMES

By following this session, students are expected to be able to implement OOP model in an application. **Learning outcomes**, students are expected to have the ability to implement OOP model in an application.

10.2 LITERATURE STUDY

Greenfoot is a Java interactive environment designed for education in high school or university. Greenfoot allows an easy development of 2D graphical application such as simulation and game. Greenfoot is developed and maintenance by the University of Kent and supported by Oracle. It is a free software under GPL license. Greenfoot is available for Windows, OS X, Linux, Solaris and every JVM. Greenfoot project is founded by Michael Kölling in 2003 and the first prototype was built by Poul Henriksen (master student) and Michael Kölling (supervisor) in 2003/2004. In 2005, the development is continued and involving the members of BlueJ Group from University of Kent and Deakin University. The Greenfoot programming model consists of World class (represented by the rectangle area as a screen) and some numbers of Actor object which present in World and can be programmed to act independently. World and Actor are represented by Java object and defined by the Java class. Greenfoot offers a method to easily program the Actors including to program the movement, rotation, changing the appearance, collision, etc. Basic programming Greenfoot will be subclassing two built-in class, World and Actor as shown in Figure 10.1. World subclass epresent the World that Greenfoot object are excecuted. Actor subclass is an object that can exist and act in the World. World subclass is automatically created by the environment.

The excecution in Greenfoot consists of built-in main loop that repeatedly called an act method from the Actor. The programming scenario implements the act method from the Actor. The implementation is done with standard Java. Greenfoot offer an API method for a common task such as animation, sound, randomize and image manipulation. All of the Java standard library can be used in Greenfoot.



Gambar 10.1 Interface Greenfoot.

Greenfoot has purpose to motivate educator and student by providing easy access to graphical animation, sound, and interaction. The environment is very interactive and support exploration and experiment. Pedagogic design based on constructive approach. The environment designed to illustrate the importance of abstraction and object oriented programming. The class/object relation, method, parameter, and object interaction that presented via visualization and guided interaction. The purpose of Greenfoot is to build and support the right model of modern object oriented programming.

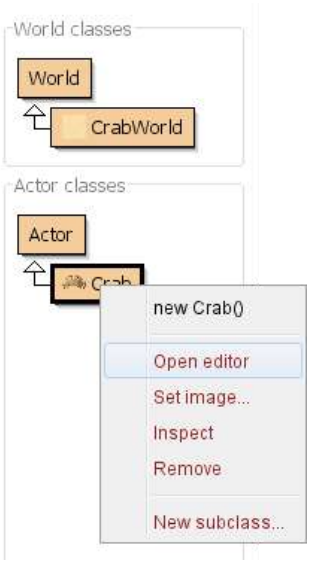
10.3 TOOLS AND MATERIALS

Tools and materials used in this meeting:

1. Computer.
2. Greenfoot https://www.greenfoot.org.

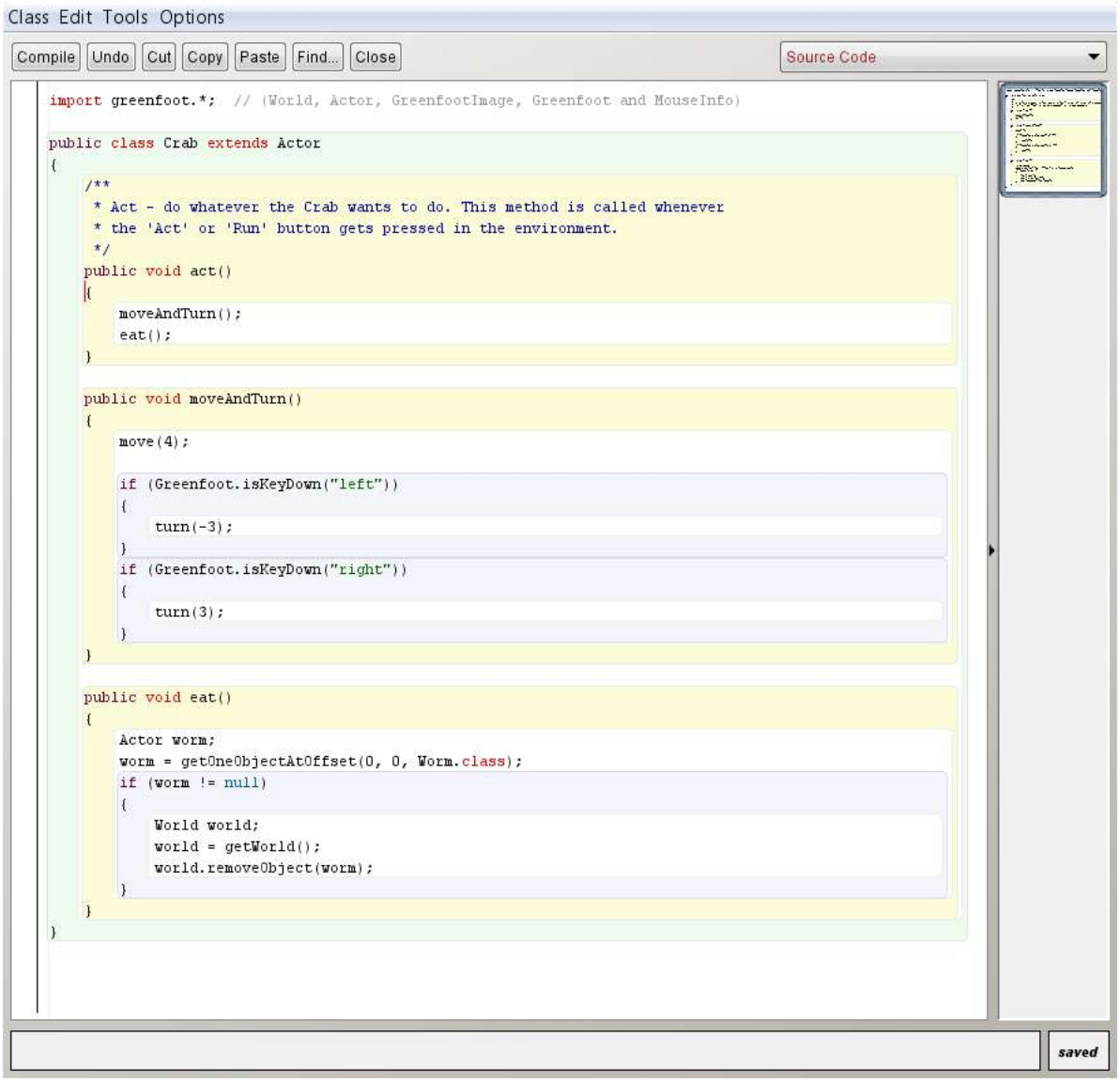
10.4 INSTRUCTIONS

1. Download the example of Greenfoot scenario in url: http://www.greenfoot.org/tutorial-files/modern-crab.zip
2. Create a Crab Actor with the right click then open the editor as shown in Figure 10.2.



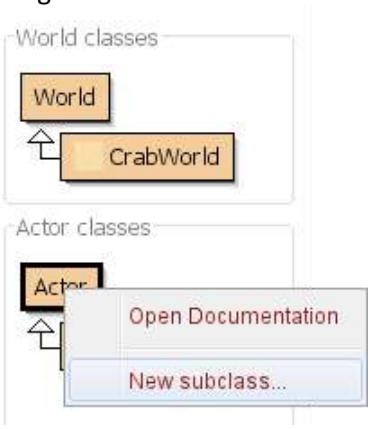
Gambar 10.2 Open the editor of the Crab Actor.

3. Complete the Crab class code as shown in Figure 10.3.



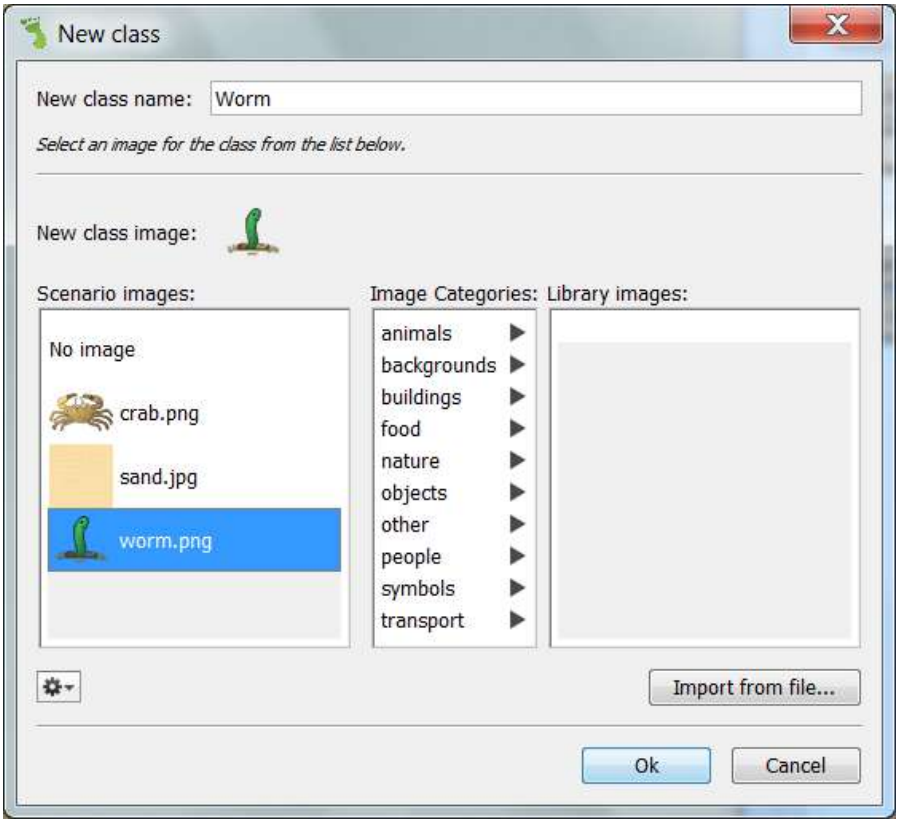
Gambar 10.3 Behavior code of Crab Actor.

4. Create a class diagram named Worm by pressing the right click button on the actor class and select New subclass as shown in Figure 10.4.



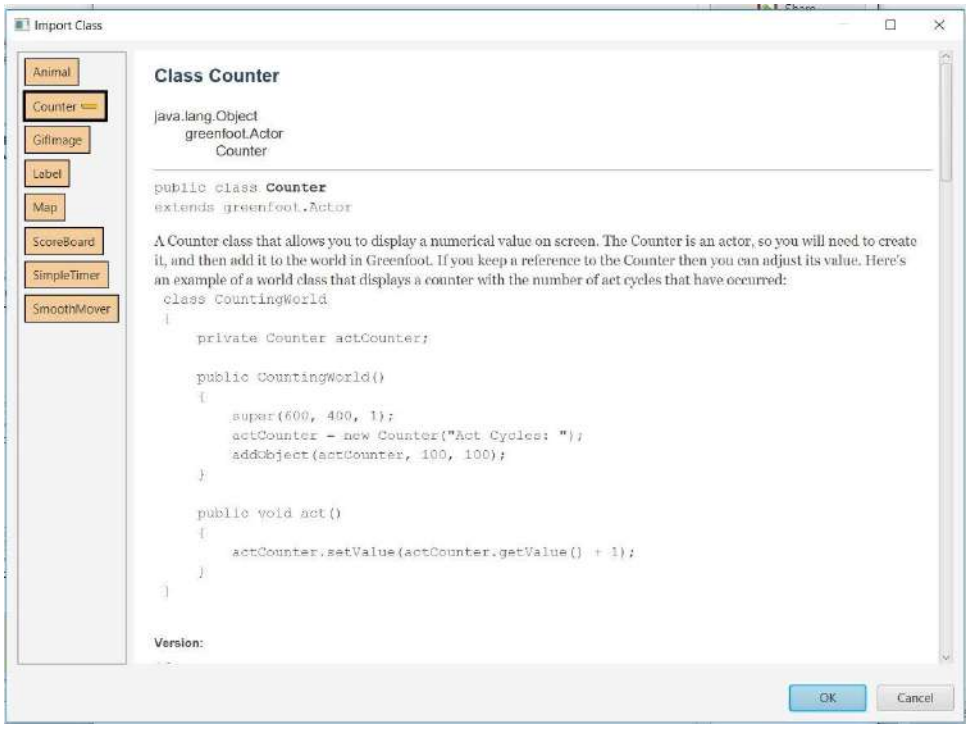
Gambar 10.4 Buat Aktor Worm.

5. Select the worm image to represent the worm visualization then click the OK button as shown in Figure 10.5.



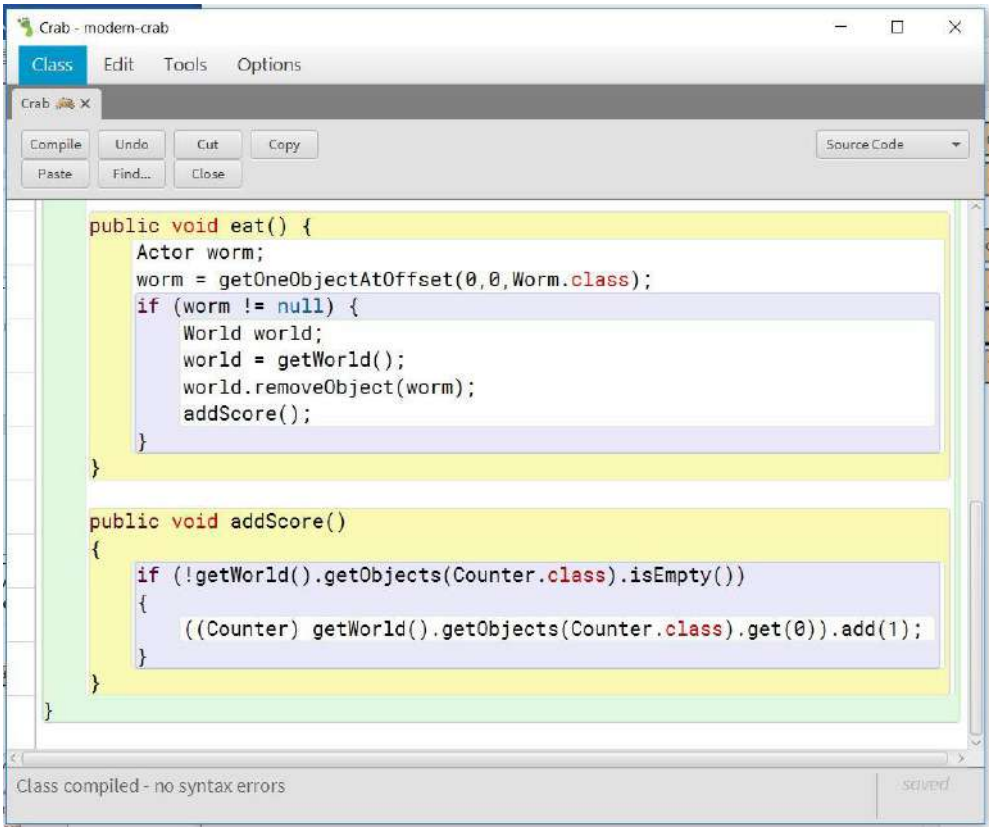
Gambar 10.5 Select the visualization of Worm object.

6. Create a counter class to show the score. Click Edit – Import class – Counter – OK as shown in Figure 10.6.



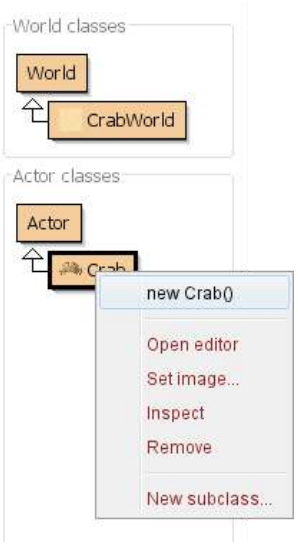
Gambar 10.6 Import class counter.

7. Right click the Crab Actor then add the addScore() code as shown in Figure 10.7 to update the score in the Counter.



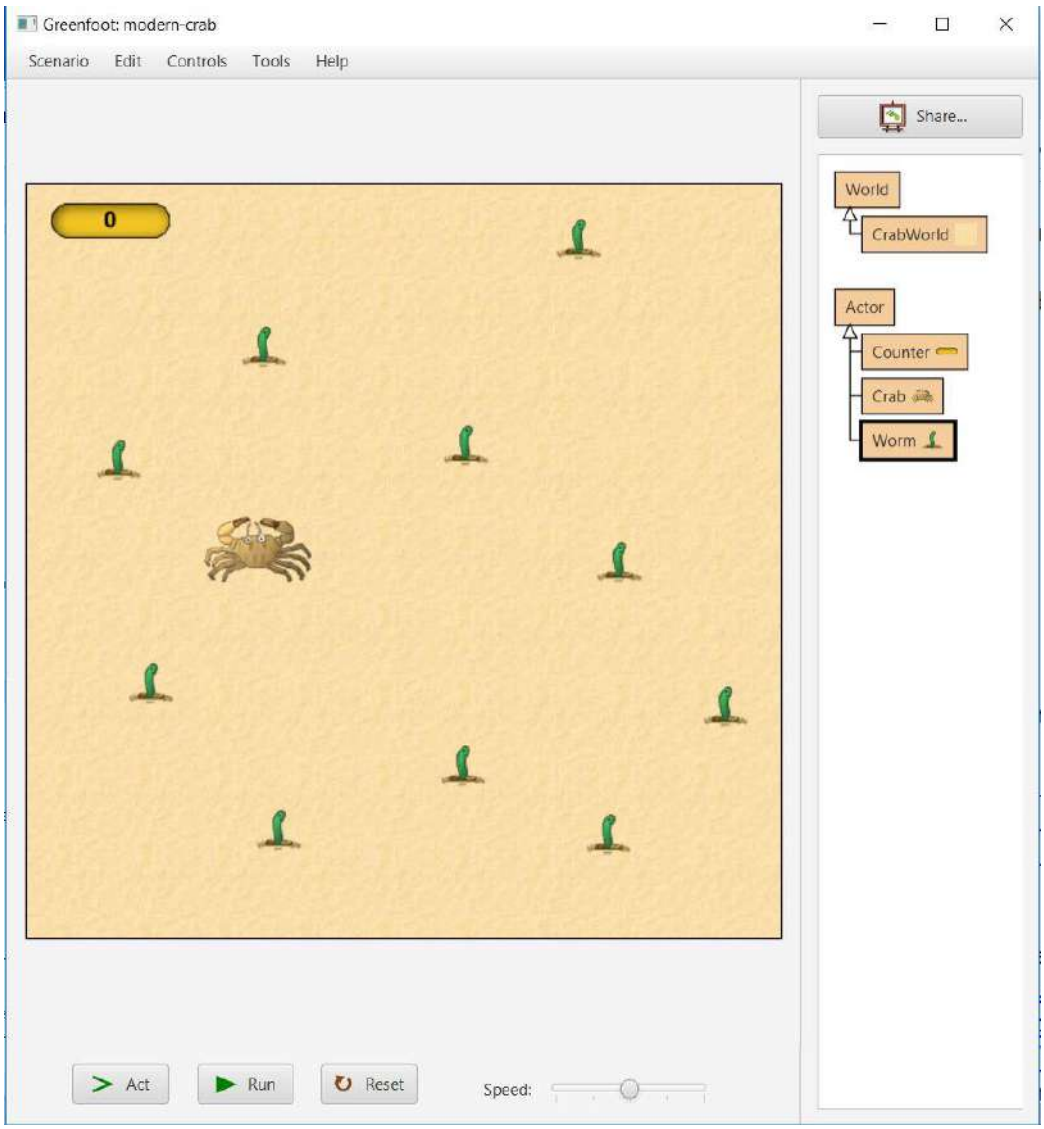
Gambar 10.7 Code to update Counter.

8. Create some Worm objects, one Crab object and one Counter object then put them in the World by pressing the right click button on the Worm class and Crab class. Select new Crab() to create Crab object, select new Worm() to create Worm object and select new Counter() to create Counter object as shown in Figure 10.8.



Gambar 10.8 Put the Crab actor in the World.

9. Put some Worms, one Counter, one Crab as shown in Figure 10.9.



Gambar 10.9 Visualization of Crab and Worms in the World.

10. Right click on the World then select Save the World to save the arrangement of objects.
11. Select Run button to run the Greenfoot and use the arrow from the keyboard to control the movement of the Crab.

10.5 ASSIGNMENT

Add the scenario to the game using this behavior:

1. Crab can move very fast up to three times as its normal movement (dash) if the Up button is pressed.
2. Worm that approached by the Crab will blink randomly if the Crab is in the radius 50 m from the Worm. Use **getObjectsInRange()** to check the distance between the Crab and the Worm and use **Greenfoot.getRandomNumber()** to generate random number to be used by the Crab in the teleportation direction.
3. There is only one way the Crab can eat the Worm which is by moving very fast (Dash) in the Worm direction before the Worm blink to another place.

PRE-TEST / POST-TEST ANSWER SHEET

Name : **NIM :**	**Assistant:** **Assistant Signature:**	**Date:** **Score:**

REFERENCES

Brown, B. (2005). *A Guide to Programming in Java*. Lawrenceville Press.

Deitel, P., & Deitel, H. (2012). *Java How to Program 9th*. *Journal of Chemical Information and Modeling* (Vol. 53). Pearson.

Schildt, H. (2007). *The Complete Reference Java Seventh Edition*. *Mc Graw Hill*. https://doi.org/10.1192/bjp.112.483.211-a


LABORATORIUM
TEKNIK INFORMATIKA
UNIVERSITAS AHMAD DAHLAN


UNIVERSITAS AHMAD DAHLAN