# LAPORAN AKHIR PRAKTIKUM

| | |
|---|---|
| Mata Praktikum | : Rekayasa Perangkat Lunak 2 |
| Kelas | : 4IA18 |
| Praktikum ke- | : 2 |
| Tanggal | : 25 Oktober 2025 |
| Materi | : Konsep OOP |
| NPM | : 50422215 |
| Nama | : Andi Purnama |
| Ketua Asisten | : Haikal Abizar |
| Paraf Asisten | : - |
| Nama Asisten | : 1. |
| | 2. |
| | 3. |
| Jumlah Lembar | : 5 Lembar |


**LABORATORIUM INFORMATIKA**

**UNIVERSITAS GUNADARMA**

**2025**

**Soal**

---

**1. Jelaskan konsep-konsep utama dalam Object-Oriented Programming (OOP) seperti Encapsulation, Inheritance, dan Polymorphism. Berikan contoh penerapan setiap konsep ini dalam program Java.**

**Jawab:**

**1. Encapsulation (Enkapsulasi)**

Encapsulation atau enkapsulasi merupakan konsep dasar dalam OOP yang berfungsi untuk menyembunyikan data internal suatu objek dari akses langsung oleh pihak luar. Dengan kata lain, atribut atau variabel dalam sebuah kelas dibuat bersifat private, sehingga tidak dapat diakses langsung dari luar kelas. Sebagai gantinya, kelas menyediakan method publik seperti *getter* dan *setter* untuk membaca dan mengubah nilai atribut dengan cara yang aman dan terkontrol. Tujuan dari enkapsulasi adalah untuk menjaga integritas data agar tidak diubah secara sembarangan dan untuk mencegah kesalahan yang dapat terjadi akibat akses langsung.
Sebagai contoh, dalam program Java dapat dibuat kelas Mahasiswa yang memiliki atribut nama dan umur dengan modifier private. Untuk mengakses atau mengubah nilai atribut tersebut, digunakan method getNama(), setNama(), getUmur(), dan setUmur(). Dengan cara ini, data mahasiswa hanya dapat dimodifikasi melalui method yang telah ditentukan, bukan secara langsung, sehingga data menjadi lebih aman dan terjaga.

**2. Inheritance (Pewarisan)**

Inheritance atau pewarisan adalah konsep dalam OOP yang memungkinkan sebuah kelas untuk mewarisi atribut dan method dari kelas lain. Kelas yang mewarisi disebut subclass atau child class, sedangkan kelas yang diwarisi disebut superclass atau parent class. Konsep ini memudahkan pengembang dalam mengelola dan mengorganisasi kode, karena dapat menghindari duplikasi serta meningkatkan kemampuan untuk melakukan ekspansi fungsi tanpa harus menulis ulang kode dari awal.
Contohnya, dalam Java dapat dibuat kelas Hewan yang memiliki atribut nama dan method makan(). Kemudian dibuat kelas Kucing yang mewarisi dari Hewan menggunakan keyword extends. Dengan pewarisan ini, Kucing otomatis memiliki atribut dan method dari Hewan, serta dapat menambahkan perilaku baru seperti suaraKucing(). Dengan inheritance, hubungan "is-a" terbentuk — artinya Kucing adalah Hewan dengan karakteristik tambahan.

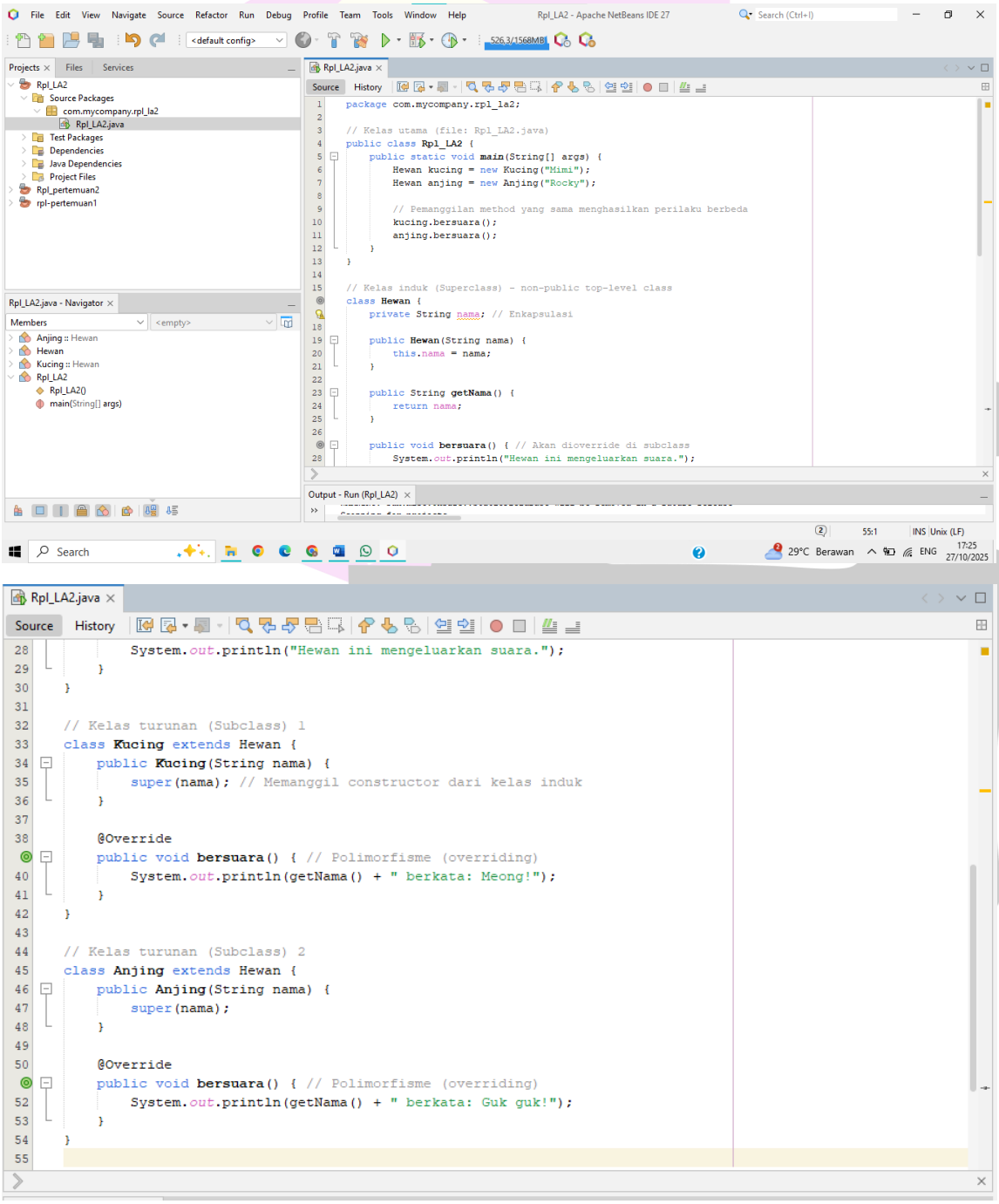
**3. Polymorphism (Polimorfisme)**

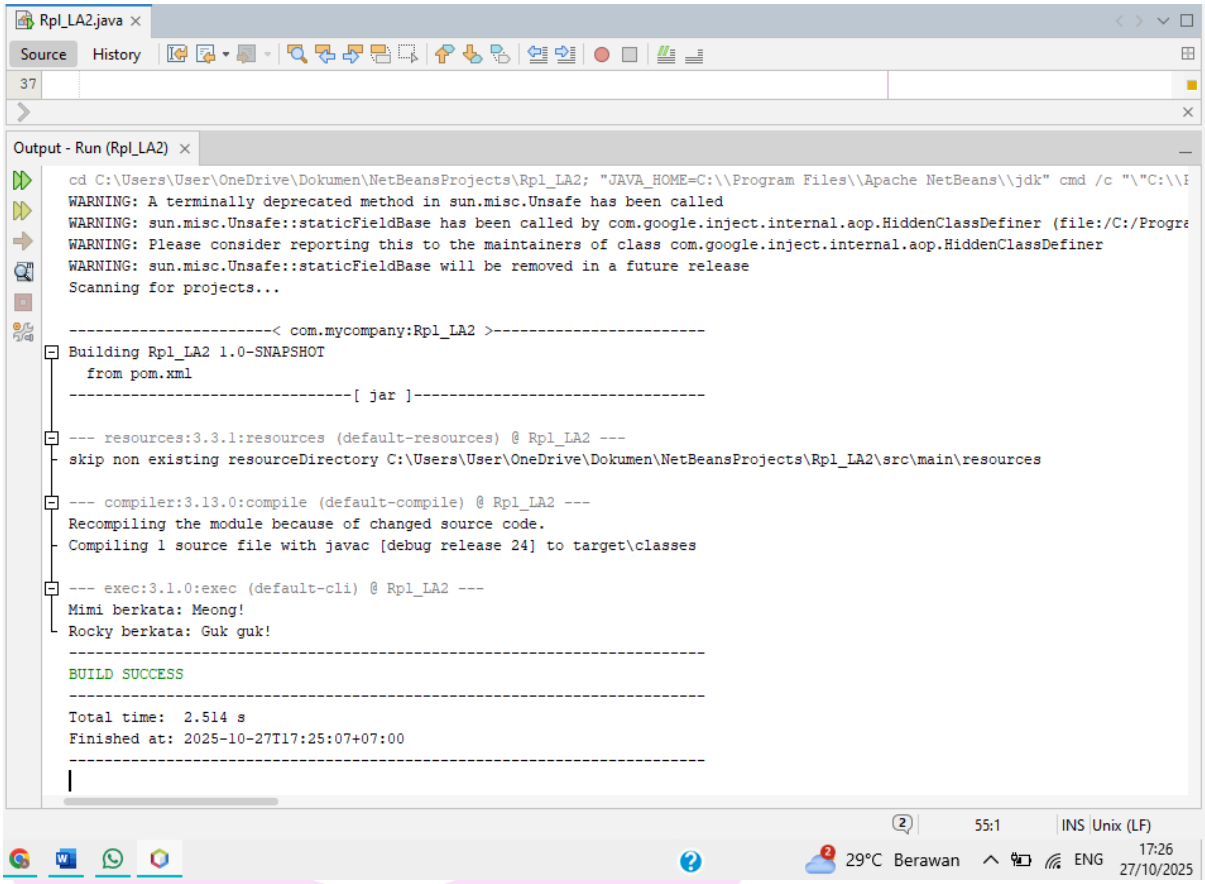
Polymorphism atau polimorfisme adalah kemampuan suatu objek untuk berperilaku berbeda meskipun memiliki antarmuka yang sama. Dalam OOP, polimorfisme memungkinkan satu method atau objek memiliki banyak bentuk tergantung konteks penggunaannya. Konsep ini sering diterapkan melalui overriding dan overloading. Overriding terjadi ketika subclass menulis ulang method yang sudah ada di superclass untuk menyesuaikan perilaku tertentu, sedangkan overloading berarti mendefinisikan beberapa method dengan nama yang sama tetapi parameter berbeda dalam satu kelas.
Sebagai contoh, kelas Hewan dapat memiliki method bersuara(), sementara subclass Kucing dan Anjing masing-masing melakukan *override* pada method tersebut agar menghasilkan suara berbeda, seperti "Meong" dan "Guk". Ketika program memanggil bersuara() pada objek yang

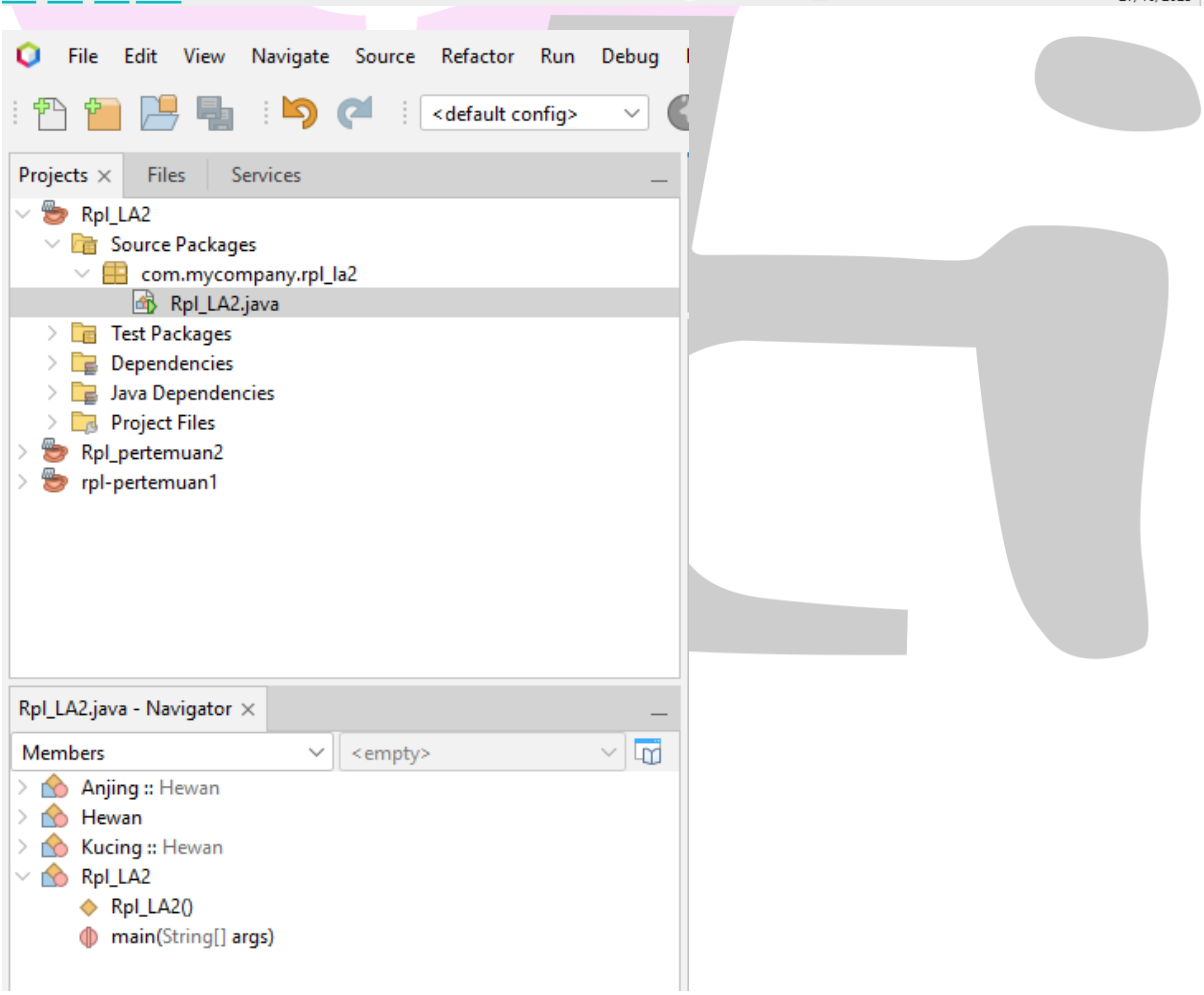
bertipe Hewan, suara yang dihasilkan bergantung pada jenis hewan yang sebenarnya. Inilah esensi polimorfisme — satu antarmuka, banyak perilaku.

Secara keseluruhan, enkapsulasi menjaga data agar aman, inheritance memudahkan penggunaan kembali kode dan menciptakan hierarki antar kelas, sedangkan polymorphism memungkinkan fleksibilitas perilaku antar objek. Ketiga konsep ini membentuk fondasi utama dari pemrograman berorientasi objek yang membuat kode lebih terstruktur, efisien, dan mudah dikembangkan.

**2. Buatlah sebuah program yang memanfaatkan konsep-konsep OOP dan berikan penjelasannya.**




Rpl_LA2.java
Source
History
37
Output - Run (Rpl_LA2)
cd C:\Users\User\OneDrive\Dokumen\NetBeansProjects\Rpl_LA2; "JAVA_HOME=C:\\Program Files\\Apache NetBeans\\jdk" cmd /c "\"C:\\
WARNING: A terminally deprecated method in sun.misc.Unsafe has been called
WARNING: sun.misc.Unsafe::staticFieldBase has been called by com.google.inject.internal.aop.HiddenClassDefiner (file:/C:/Progra
WARNING: Please consider reporting this to the maintainers of class com.google.inject.internal.aop.HiddenClassDefiner
WARNING: sun.misc.Unsafe::staticFieldBase will be removed in a future release
Scanning for projects...
----------------------< com.mycompany:Rpl_LA2 >-----------------------
Building Rpl_LA2 1.0-SNAPSHOT
from pom.xml
--------------------------------[ jar ]---------------------------------
--- resources:3.3.1:resources (default-resources) @ Rpl_LA2 ---
skip non existing resourceDirectory C:\Users\User\OneDrive\Dokumen\NetBeansProjects\Rpl_LA2\src\main\resources
--- compiler:3.13.0:compile (default-compile) @ Rpl_LA2 ---
Recompiling the module because of changed source code.
Compiling 1 source file with javac [debug release 24] to target\classes
--- exec:3.1.0:exec (default-cli) @ Rpl_LA2 ---
Mimi berkata: Meong!
Rocky berkata: Guk guk!
BUILD SUCCESS
Total time: 2.514 s
Finished at: 2025-10-27T17:25:07+07:00
55:1
INS Unix (LF)
29°C Berawan
ENG
17:26
27/10/2025


File Edit View Navigate Source Refactor Run Debug
<default config>
Projects
Files
Services
Rpl_LA2
Source Packages
com.mycompany.rpl_la2
Rpl_LA2.java
Test Packages
Dependencies
Java Dependencies
Project Files
Rpl_pertemuan2
rpl-pertemuan1
Rpl_LA2.java - Navigator
Members
<empty>
Anjing :: Hewan
Hewan
Kucing :: Hewan
Rpl_LA2
Rpl_LA2()
main(String[] args)

**Penjelasan:**

Program tersebut merupakan implementasi dari tiga konsep utama **Object-Oriented Programming (OOP)**, yaitu **Encapsulation**, **Inheritance**, dan **Polymorphism**, yang diterapkan dalam konteks sistem sederhana tentang hewan.

Pertama, konsep **Encapsulation (enkapsulasi)** diterapkan pada kelas Hewan. Kelas ini memiliki atribut nama yang dibuat bersifat private, artinya data tersebut tidak dapat diakses langsung dari luar kelas. Untuk menjaga keamanan data, disediakan method getNama() sebagai cara resmi untuk mengakses nilai atribut tersebut. Dengan pendekatan ini, data internal objek terlindungi dari perubahan yang tidak diinginkan, dan setiap manipulasi nilai atribut hanya dapat dilakukan melalui method yang telah ditentukan.

Selanjutnya, program menerapkan **Inheritance (pewarisan)** dengan membuat dua kelas turunan, yaitu Kucing dan Anjing, yang mewarisi atribut dan method dari kelas Hewan menggunakan keyword extends. Melalui pewarisan ini, kedua kelas tersebut otomatis memiliki sifat dan perilaku dasar yang sama dengan kelas induknya, seperti atribut nama dan method bersuara(). Namun, masing-masing kelas turunan dapat menambahkan atau memodifikasi perilaku tertentu sesuai kebutuhan. Dalam hal ini, baik Kucing maupun Anjing menyesuaikan perilaku method bersuara() agar menampilkan suara khas hewan masing-masing.

Konsep terakhir yang digunakan adalah **Polymorphism (polimorfisme)**. Dalam program ini, objek bertipe Hewan dapat merujuk pada objek Kucing atau Anjing. Ketika method bersuara() dipanggil, hasilnya berbeda tergantung jenis objek yang sebenarnya — Kucing mengeluarkan suara “Meong!” sedangkan Anjing mengeluarkan suara “Guk guk!”. Meskipun kedua objek dipanggil melalui referensi bertipe Hewan, perilakunya berbeda sesuai dengan kelas aslinya. Hal ini menunjukkan bagaimana polimorfisme memungkinkan satu method dengan nama yang sama menghasilkan perilaku berbeda pada objek yang berbeda.

Secara keseluruhan, program ini menunjukkan hubungan erat antara tiga pilar utama OOP. **Enkapsulasi** menjaga keamanan dan integritas data, **inheritance** memungkinkan penggunaan kembali dan perluasan kode, sementara **polimorfisme** memberikan fleksibilitas perilaku antar objek. Kombinasi ketiganya menghasilkan struktur program yang efisien, modular, dan mudah dikembangkan — prinsip dasar dalam perancangan perangkat lunak berorientasi objek.