Pemrograman Berorientasi Objek

Buku Ajar Berbasis Outcome-Based Education (OBE)

Mata Kuliah: Pemrograman Berorientasi Objek

Kode: TIF-301

Program Studi Teknik Informatika

Fakultas Teknik

Universitas Lorem Ipsum

2026

Kata Pengantar

Puji syukur kehadirat Tuhan Yang Maha Esa atas terselesaikannya buku ajar *Pemrograman Berorientasi Objek* ini. Buku ini disusun dengan pendekatan **Outcome-Based Education (OBE)**, yang berfokus pada pencapaian kompetensi terukur sesuai dengan Capaian Pembelajaran Lulusan (CPL) dan Capaian Pembelajaran Mata Kuliah (CPMK).

Pemrograman Berorientasi Objek (OOP) merupakan paradigma pemrograman fundamental yang harus dikuasai oleh setiap mahasiswa Teknik Informatika. Paradigma ini tidak hanya mengajarkan cara menulis kode, tetapi juga cara berpikir dalam merancang solusi perangkat lunak yang modular, reusable, dan maintainable.

Buku ini dirancang untuk mendukung pembelajaran mahasiswa semester 3 dengan pendekatan student-centered learning. Setiap bab dilengkapi dengan:

- Sub-CPMK yang jelas dan terukur
- Materi pokok dengan contoh kode Java yang lengkap
- Aktivitas pembelajaran yang mendorong eksplorasi mandiri
- Latihan dan refleksi untuk penguatan pemahaman
- Asesmen untuk mengukur pencapaian kompetensi
- Checklist kompetensi untuk self-assessment

Kami berharap buku ini dapat menjadi panduan yang efektif dalam perjalanan pembelajaran Anda menguasai Pemrograman Berorientasi Objek.

Penyusun
Dr. Lorem Ipsum, M.Kom.

Cara Menggunakan Buku Ini

Buku ajar ini dirancang dengan pendekatan OBE untuk memaksimalkan pencapaian pembelajaran Anda. Berikut panduan penggunaan buku ini:

Struktur Buku

Bab I: Pendahuluan dan Orientasi
Memperkenalkan tujuan buku, keterkaitan dengan RPS, dan konteks kurikulum OBE.

Bab II: Landasan Teori
Menyajikan fondasi teoretis OOP yang menjadi basis pembelajaran seluruh bab berikutnya.

Bab III-XIII: Unit Materi Inti
Setiap bab mencakup satu topik utama OOP dengan struktur lengkap: Sub-CPMK, materi, aktivitas, latihan, asesmen, dan checklist.

Bab XIV: Evaluasi dan Integrasi
Berisi asesmen komprehensif dan panduan refleksi untuk mengukur pencapaian kompetensi secara menyeluruh.

Lampiran
Menyediakan rubrik penilaian, glosarium istilah OOP, dan referensi tambahan.

Komponen dalam Setiap Bab

1. **Sub-CPMK**: Baca dengan seksama untuk memahami kompetensi yang harus dicapai

2. **Materi Pokok**: Pelajari dengan cermat, jalankan semua contoh kode

3. **Aktivitas Pembelajaran**: Lakukan secara mandiri atau berkelompok

4. **Latihan**: Kerjakan untuk menguji pemahaman Anda

5. **Asesmen**: Gunakan untuk mengukur pencapaian Sub-CPMK

6. **Checklist**: Centang setelah yakin menguasai setiap indikator

Tips Belajar Efektif

- Jangan hanya membaca, praktikkan semua contoh kode
- Gunakan IDE (Eclipse, IntelliJ IDEA, atau NetBeans) untuk eksperimen
- Kerjakan latihan sebelum melihat solusi
- Diskusikan konsep yang sulit dengan teman atau dosen
- Manfaatkan checklist untuk self-assessment berkala
- Kerjakan proyek mini untuk mengintegrasikan konsep yang dipelajari

Identitas Mata Kuliah

Nama Program Studi	: Teknik Informatika
Nama Mata Kuliah	: Pemrograman Berorientasi Objek
Kode Mata Kuliah	: TIF-301
Semester	: 3 (Tiga)
SKS / Bobot Kredit	: 3 SKS (2 Teori, 1 Praktikum)
Dosen Pengampu	: Dr. Lorem Ipsum, M.Kom.
Tanggal Penyusunan	: 31 Januari 2026

Capaian Pembelajaran Lulusan (CPL)

CPL yang dibebankan pada mata kuliah ini mencakup kompetensi lulusan dalam aspek pengetahuan, keterampilan, dan sikap:

1. **CPL-1 (Pengetahuan):** Menguasai konsep teoretis bidang pengetahuan tertentu secara umum dan konsep teoretis bagian khusus dalam bidang pengetahuan tersebut secara mendalam, serta mampu memformulasikan penyelesaian masalah prosedural.
2. **CPL-2 (Keterampilan Umum):** Mampu menerapkan pemikiran logis, kritis, sistematis, dan inovatif dalam konteks pengembangan atau implementasi ilmu pengetahuan dan teknologi yang memperhatikan dan menerapkan nilai humaniora.
3. **CPL-3 (Keterampilan Khusus):** Mampu merancang, mengimplementasikan, dan mengevaluasi sistem perangkat lunak menggunakan paradigma berorientasi objek dengan mempertimbangkan standar kualitas perangkat lunak.
4. **CPL-4 (Sikap):** Menunjukkan sikap bertanggung jawab atas pekerjaan di bidang keahliannya secara mandiri dan mampu bekerja sama dalam tim.

Capaian Pembelajaran Mata Kuliah (CPMK)

Kemampuan atau kompetensi spesifik yang diharapkan mahasiswa kuasai setelah menyelesaikan mata kuliah:

1. **CPMK-1:** Mahasiswa mampu memahami dan menjelaskan konsep dasar pemrograman berorientasi objek (class, object, encapsulation, inheritance, polymorphism).

2. **CPMK-2:** Mahasiswa mampu merancang solusi permasalahan menggunakan diagram UML (Use Case, Class Diagram, Sequence Diagram).

3. **CPMK-3:** Mahasiswa mampu mengimplementasikan konsep OOP dalam bahasa pemrograman Java dengan menerapkan prinsip SOLID.

4. **CPMK-4:** Mahasiswa mampu menganalisis dan mengevaluasi kualitas kode program berorientasi objek berdasarkan best practices dan design patterns.

Matriks Keterkaitan CPL dan CPMK

CPL	**CPMK**	**Kontribusi**	**Keterangan**
CPL-1	CPMK-1	Tinggi	Penguasaan konsep teoretis OOP
CPL-1	CPMK-2	Tinggi	Kemampuan merancang solusi dengan UML
CPL-2	CPMK-3	Tinggi	Penerapan pemikiran sistematis dalam implementasi
CPL-2	CPMK-4	Sedang	Pemikiran kritis dalam evaluasi kode
CPL-3	CPMK-2	Tinggi	Perancangan sistem perangkat lunak
CPL-3	CPMK-3	Tinggi	Implementasi dengan standar kualitas
CPL-3	CPMK-4	Tinggi	Evaluasi kualitas perangkat lunak
CPL-4	CPMK-3	Sedang	Tanggung jawab dalam implementasi
CPL-4	CPMK-4	Sedang	Kerja sama dalam code review

Tabel 1: Matriks Keterkaitan CPL dan CPMK

Daftar Isi

Kata Pengantar iii

Cara Menggunakan Buku Ini v

Identitas Mata Kuliah vii

Daftar Isi ix

Daftar Gambar xv

Daftar Tabel xvii

Daftar Kode Program xix

1 Pendahuluan dan Orientasi Buku 1

1.1 Tujuan Buku Ajar . . . 1

1.2 Keterkaitan Buku Ajar dengan RPS Berbasis OBE . . . 2

1.2.1 Alignment dengan CPL dan CPMK . . . 2

1.2.2 Integrasi Metode Pembelajaran . . . 2

1.2.3 Sistem Penilaian Terintegrasi . . . 2

1.3 Petunjuk Penggunaan Buku Ajar . . . 3

1.3.1 Untuk Mahasiswa . . . 3

1.3.2 Untuk Dosen . . . 3

1.4 Konteks Kurikulum OBE . . . 4

1.4.1 Apa itu Outcome-Based Education? . . . 4

1.4.2 Implementasi OBE dalam Buku Ini . . . 4

1.4.3 Hierarki Capaian Pembelajaran . . . 4

1.5 Peta Konsep Pemrograman Berorientasi Objek . . . 5

2 Landasan Teori dan Konsep Dasar OOP 7

2.1 Sejarah dan Evolusi Paradigma Pemrograman 7
2.1.1 Timeline Paradigma Pemrograman 7
2.2 Paradigma Prosedural vs Berorientasi Objek 8
2.2.1 Pemrograman Prosedural 8
2.2.2 Pemrograman Berorientasi Objek 8
2.2.3 Perbandingan Prosedural vs OOP 9
2.3 Konsep Dasar OOP 9
2.3.1 1. Abstraksi (Abstraction) 10
2.3.2 2. Enkapsulasi (Encapsulation) 10
2.3.3 3. Pewarisan (Inheritance) 11
2.3.4 4. Polimorfisme (Polymorphism) 11
2.4 Keuntungan dan Tantangan OOP 11
2.4.1 Keuntungan OOP 11
2.4.2 Tantangan OOP 12
2.5 Peta Konsep OOP 12

3 Konsep Class dan Object **17**
3.1 Definisi Class dan Object 17
3.1.1 Apa itu Class? 17
3.1.2 Apa itu Object? 18
3.1.3 Hubungan Class dan Object 18
3.2 Attributes (Instance Variables) 19
3.2.1 Definisi Attributes 19
3.2.2 Jenis Attributes 19
3.3 Methods (Behavior) 19
3.3.1 Definisi Methods 19
3.3.2 Jenis Methods 20
3.4 Constructor 20
3.4.1 Apa itu Constructor? 20
3.4.2 Constructor Overloading 21
3.5 Keyword `this` 21
3.5.1 Penggunaan `this` 21
3.6 Object Creation dan Instantiation 22
3.6.1 Membuat Object 22
3.6.2 Memory Allocation 22

3.7 Static vs Instance Members 23
3.7.1 Instance Members 23
3.7.2 Static Members 23
3.8 Contoh Lengkap: Class Buku 24

4 Encapsulation dan Information Hiding 31
4.1 Konsep Encapsulation dan Access Modifier 31
4.1.1 Access Modifier di Java 31
4.1.2 Contoh Encapsulation Sederhana 31
4.2 Getter/Setter, Validasi, dan Immutability 33
4.2.1 Getter dan Setter 33
4.2.2 Immutability 33

5 Inheritance dan Hierarki Kelas 37
5.1 Konsep Inheritance dan Relasi "is-a" 37
5.1.1 Konsep Pewarisan 37
5.1.2 Keyword `extends` 37
5.1.3 Hierarki Kelas 38
5.2 Overriding, `super`, dan Desain Hierarki 38
5.2.1 Method Overriding 38
5.2.2 Penggunaan `super` 39
5.2.3 Composition vs Inheritance 39

6 Polymorphism dan Dynamic Binding 43
6.1 Polymorphism dan Method Overloading 43
6.1.1 Compile-time Polymorphism (Overloading) 43
6.1.2 Keuntungan Overloading 44
6.1.3 Perbandingan Overloading vs Overriding 44
6.2 Runtime Polymorphism dan Dynamic Dispatch 44
6.2.1 Overriding dan Dynamic Binding 44
6.2.2 Polymorphism pada Interface 45

7 Abstract Class dan Interface 49
7.1 Abstract Class dan Abstraksi 49
7.1.1 Karakteristik Abstract Class 49
7.1.2 Kapan Menggunakan Abstract Class 50

7.2 Interface dan Perbandingannya dengan Abstract Class 50
7.2.1 Konsep Interface 50
7.2.2 Perbandingan Abstract Class vs Interface 51

8 Perancangan dengan UML **55**
8.1 Use Case dan Class Diagram 55
8.1.1 Unified Modeling Language (UML) 55
8.1.2 Use Case Diagram 55
8.1.3 Class Diagram 55
8.2 Sequence Diagram dan Studi Kasus 56
8.2.1 Sequence Diagram 56
8.2.2 Contoh Studi Kasus 56

9 Exception Handling dan Keandalan Program **59**
9.1 Konsep Exception dan Try-Catch 59
9.1.1 Jenis Exception 59
9.1.2 Struktur Try-Catch 59
9.2 Custom Exception dan Best Practice 60
9.2.1 Membuat Custom Exception 60
9.2.2 Finally dan Try-with-Resources 60

10 Prinsip SOLID dalam Desain OOP **63**
10.1 SRP, OCP, dan LSP 63
10.1.1 Single Responsibility Principle (SRP) 63
10.1.2 Open/Closed Principle (OCP) 63
10.1.3 Liskov Substitution Principle (LSP) 63
10.2 ISP, DIP, dan Contoh Refactoring 64
10.2.1 Interface Segregation Principle (ISP) 64
10.2.2 Dependency Inversion Principle (DIP) 64

11 Design Patterns Dasar **67**
11.1 Design Patterns: Creational Patterns 67
11.1.1 Singleton Pattern 67
11.1.2 Factory Pattern 68
11.2 Behavioral Pattern: Observer 69
11.2.1 Observer Pattern 69

11.2.2 Kapan Menggunakan Design Pattern 69

12 Collections, Generics, dan File I/O **73**

12.1 Java Collections Framework dan Generics 73

12.1.1 Jenis Collections 73

12.1.2 Generics 73

12.2 File I/O dan Serialization 74

12.2.1 Membaca dan Menulis File 74

12.2.2 Serialization 74

13 Unit Testing, TDD, dan Refactoring **77**

13.1 Unit Testing dan Siklus TDD 77

13.1.1 Konsep Unit Testing 77

13.1.2 Test-Driven Development (TDD) 77

13.2 Code Quality dan Refactoring 78

13.2.1 Code Smell yang Umum 78

13.2.2 Teknik Refactoring 78

14 Evaluasi, Refleksi, dan Integrasi Kompetensi **81**

14.1 Asesmen Akhir Komprehensif 81

14.1.1 Petunjuk Umum 81

14.1.2 Bagian A: Pilihan Ganda (CPMK-1) 81

14.1.3 Bagian B: Essay (CPMK-1, CPMK-2) 83

14.1.4 Bagian C: Analisis Kode (CPMK-3, CPMK-4) 84

14.1.5 Bagian D: Coding Challenge (CPMK-2, CPMK-3, 4) 85

14.2 Rubrik Penilaian Komprehensif 86

14.2.1 Rubrik untuk Coding Challenge 86

14.2.2 Bobot Penilaian 87

14.3 Tinjauan Pencapaian Kompetensi Secara Menyeluruh 87

14.3.1 Pemetaan Asesmen ke CPMK 87

14.3.2 Self-Assessment Checklist 87

14.4 Rekomendasi Tindak Lanjut Pembelajaran 88

14.4.1 Untuk Mahasiswa yang Sudah Menguasai Semua CPMK 88

14.4.2 Untuk Mahasiswa yang Masih Perlu Perbaikan 88

14.4.3 Sumber Belajar Tambahan 89

Lampiran **91**

Daftar Pustaka **93**

Daftar Gambar

3.1 Konsep Class dan Object 17

5.1 Contoh Hierarki Kelas (Relasi is-a) 38

7.1 Hierarki Class Pegawai (Abstract) 50

11.1 Struktur Singleton Pattern 68

11.2 Struktur Factory Pattern 69

Daftar Tabel

1 Matriks Keterkaitan CPL dan CPMK viii

1.1 Implementasi OBE dalam Buku Ajar 4

2.1 Perbandingan Prosedural vs OOP 10

3.1 Hubungan Class dan Object 18

4.1 Access Modifier di Java 32

6.1 Perbandingan Overloading dan Overriding 44

7.1 Perbandingan Abstract Class dan Interface 51

14.1 Rubrik Penilaian Coding Challenge 86
14.2 Matriks Bobot Penilaian Akhir 87
14.3 Pemetaan Komponen Asesmen ke CPMK 87
14.4 Rubrik Penilaian Tugas Praktik 91

Daftar Kode Program

2.1 Class Mahasiswa dengan OOP 9
3.1 Struktur Dasar Class 18
3.2 Membuat Object 18
3.3 Contoh Attributes 19
3.4 Contoh Methods 19
3.5 Contoh Constructor 20
3.6 Penggunaan this 21
3.7 Membuat dan Menggunakan Object 22
3.8 Static vs Instance 23
3.9 Class Buku - Contoh Lengkap 24
3.10 Penggunaan Class Buku 26
4.1 Encapsulation pada class RekeningBank 32
4.2 Contoh Getter dan Setter dengan Validasi 33
5.1 Inheritance Sederhana 37
5.2 Contoh Method Overriding 38
5.3 Penggunaan super 39
6.1 Contoh Overloading 43
6.2 Runtime Polymorphism 44
7.1 Contoh Abstract Class 49
7.2 Contoh Interface 51
9.1 Try-Catch Sederhana 59
9.2 Custom Exception 60
10.1 Contoh DIP dengan Interface 64
11.1 Singleton Pattern 67
11.2 Factory Pattern 68
11.3 Observer Pattern Sederhana 69
12.1 Contoh Collections dan Generics 73

12.2 Contoh Menulis File 74
13.1 Contoh Unit Test JUnit 77
14.1 Kode untuk Dianalisis 84

Bab 1

Pendahuluan dan Orientasi Buku

1.1 Tujuan Buku Ajar

Buku ajar ini dirancang sebagai panduan komprehensif untuk menguasai Pemrograman Berorientasi Objek (OOP) menggunakan bahasa Java [8]. Fokus utama buku ini adalah pada implementasi prinsip-prinsip desain yang baik dan penerapan standar industri. Tujuan spesifik buku ini adalah:

1. Memberikan pemahaman mendalam tentang paradigma pemrograman berorientasi objek
2. Mengembangkan kemampuan merancang solusi perangkat lunak dengan prinsip OOP
3. Membangun keterampilan implementasi OOP dengan bahasa pemrograman Java
4. Menumbuhkan kemampuan analisis dan evaluasi kualitas kode
5. Memfasilitasi pencapaian CPL dan CPMK yang telah ditetapkan

Setelah mempelajari buku ini secara menyeluruh, mahasiswa diharapkan mampu:

- Menjelaskan konsep fundamental OOP (class, object, encapsulation, inheritance, polymorphism)
- Merancang sistem menggunakan UML diagrams
- Mengimplementasikan aplikasi Java dengan menerapkan prinsip SOLID
- Mengidentifikasi dan menerapkan design patterns yang tepat
- Menulis kode yang clean, maintainable, dan testable

1.2 Keterkaitan Buku Ajar dengan RPS Berbasis OBE

Buku ajar ini dirancang selaras dengan Rencana Pembelajaran Semester (RPS) mata kuliah Pemrograman Berorientasi Objek yang berbasis OBE, dengan mengadopsi prinsip perancangan sistem yang sistematis [6]. Keterkaitan ini diwujudkan melalui:

1.2.1 Alignment dengan CPL dan CPMK

Setiap bab dalam buku ini dipetakan secara eksplisit ke Sub-CPMK yang berkontribusi pada pencapaian CPMK dan CPL. Struktur ini memastikan bahwa:

- Materi pembelajaran fokus pada pencapaian kompetensi terukur
- Aktivitas pembelajaran mendukung pengembangan keterampilan yang diharapkan
- Asesmen mengukur pencapaian kompetensi secara objektif

1.2.2 Integrasi Metode Pembelajaran

Buku ini mengintegrasikan berbagai metode pembelajaran yang tercantum dalam RPS:

- **Problem-Based Learning**: Studi kasus nyata dalam setiap bab
- **Project-Based Learning**: Proyek mini untuk integrasi konsep
- **Peer Review**: Aktivitas code review antar mahasiswa
- **Flipped Classroom**: Materi untuk dipelajari mandiri sebelum kelas

1.2.3 Sistem Penilaian Terintegrasi

Komponen asesmen dalam buku ini sejalan dengan sistem penilaian RPS:

- Latihan untuk tugas individu (15%)
- Kuis untuk evaluasi formatif (10%)
- Aktivitas praktikum (15%)
- Proyek kelompok (20%)
- Persiapan UTS dan UAS (40%)

1.3 Petunjuk Penggunaan Buku Ajar

1.3.1 Untuk Mahasiswa

Sebelum Perkuliahan:

1. Baca Sub-CPMK di awal bab untuk memahami target pembelajaran
2. Pelajari materi pokok dengan seksama
3. Jalankan dan modifikasi semua contoh kode yang diberikan
4. Catat pertanyaan atau konsep yang belum dipahami

Selama Perkuliahan:

1. Diskusikan konsep yang sulit dengan dosen dan teman
2. Kerjakan aktivitas pembelajaran secara aktif
3. Tanyakan hal-hal yang belum jelas
4. Berpartisipasi dalam code review dan diskusi kelompok

Setelah Perkuliahan:

1. Kerjakan latihan dan refleksi
2. Lakukan asesmen mandiri
3. Centang checklist kompetensi yang telah dikuasai
4. Kerjakan proyek mini untuk memperdalam pemahaman

1.3.2 Untuk Dosen

Buku ini dapat digunakan sebagai:

- Bahan ajar utama untuk perkuliahan
- Sumber latihan dan tugas
- Referensi untuk menyusun soal ujian
- Panduan untuk merancang aktivitas pembelajaran
- Alat untuk mengukur pencapaian CPMK mahasiswa

1.4 Konteks Kurikulum OBE

1.4.1 Apa itu Outcome-Based Education?

Outcome-Based Education (OBE) adalah pendekatan pembelajaran yang berfokus pada pencapaian hasil (*outcomes*) yang terukur. Dalam OBE, proses pembelajaran dirancang secara sistematis untuk memastikan mahasiswa mencapai kompetensi yang telah ditetapkan.

Prinsip Utama OBE:

1. **Clarity of Focus**: Fokus jelas pada apa yang harus dicapai mahasiswa
2. **Designing Down**: Kurikulum dirancang mundur dari outcomes yang diinginkan
3. **High Expectations**: Ekspektasi tinggi untuk semua mahasiswa
4. **Expanded Opportunity**: Kesempatan beragam untuk mencapai outcomes

1.4.2 Implementasi OBE dalam Buku Ini

Buku ini mengimplementasikan OBE melalui:

Komponen OBE	**Implementasi dalam Buku**
Outcomes yang Jelas	Sub-CPMK eksplisit di setiap bab
Pembelajaran Terstruktur	Materi disusun dari dasar ke lanjut secara sistematis
Aktivitas Beragam	Latihan, studi kasus, proyek, code review
Asesmen Terukur	Rubrik penilaian yang jelas untuk setiap kompetensi
Feedback Berkelanjutan	Checklist untuk self-assessment

Tabel 1.1: Implementasi OBE dalam Buku Ajar

1.4.3 Hierarki Capaian Pembelajaran

Konsep Penting

Dalam kurikulum OBE, capaian pembelajaran tersusun dalam hierarki:
CPL (Capaian Pembelajaran Lulusan)
↓
CPMK (Capaian Pembelajaran Mata Kuliah)
↓
Sub-CPMK (Sub Capaian Pembelajaran Mata Kuliah)
↓
Indikator Pencapaian
Setiap level berkontribusi pada level di atasnya, memastikan bahwa pembelajaran di tingkat mikro (per bab) mendukung pencapaian kompetensi di tingkat makro (lulusan).

1.5 Peta Konsep Pemrograman Berorientasi Objek

Mata kuliah ini mencakup 14 topik utama yang saling terkait:

1. **Bab II**: Landasan Teori - Fondasi paradigma OOP
2. **Bab III**: Class dan Object - Building blocks OOP
3. **Bab IV**: Encapsulation - Information hiding
4. **Bab V**: Inheritance - Code reuse dan hierarki
5. **Bab VI**: Polymorphism - Fleksibilitas kode
6. **Bab VII**: Abstract Class & Interface - Abstraksi
7. **Bab VIII**: UML Diagrams - Perancangan visual
8. **Bab IX**: Exception Handling - Error management
9. **Bab X**: Prinsip SOLID - Design principles
10. **Bab XI**: Design Patterns - Solusi proven
11. **Bab XII**: Collections, Generics, & File I/O - Struktur data dan persistensi
12. **Bab XIII**: Unit Testing, TDD, & Refactoring - Quality assurance
13. **Bab XIV**: Evaluasi Komprehensif - Integrasi semua konsep

Alur Pembelajaran:

- Bab II-IV: Konsep fundamental OOP
- Bab V-VII: Konsep lanjut OOP
- Bab VIII: Perancangan sistem
- Bab IX-XIII: Praktik baik, kualitas, dan tooling
- Bab XIV: Evaluasi dan integrasi

Rangkuman

Bab ini memperkenalkan tujuan buku ajar, keterkaitan dengan RPS berbasis OBE, petunjuk penggunaan, dan konteks kurikulum OBE. Pemahaman yang baik tentang struktur dan pendekatan buku ini akan membantu Anda memaksimalkan pembelajaran Pemrograman Berorientasi Objek.

Poin Kunci:

- Buku ini dirancang dengan pendekatan OBE yang fokus pada pencapaian kompetensi terukur

- Setiap bab dipetakan ke Sub-CPMK yang berkontribusi pada CPL
- Gunakan komponen OBE (Sub-CPMK, aktivitas, latihan, asesmen, checklist) secara optimal
- Pembelajaran OOP tersusun sistematis dari konsep dasar hingga advanced

Bab 2

Landasan Teori dan Konsep Dasar OOP

Sub-CPMK yang Dicakup dalam Bab Ini:

- Sub-CPMK 1.1: Menjelaskan perbedaan antara pemrograman prosedural dan berorientasi objek serta mengidentifikasi karakteristik utama paradigma OOP

2.1 Sejarah dan Evolusi Paradigma Pemrograman

Pemrograman komputer telah mengalami evolusi paradigma sejak komputer pertama kali diciptakan [5]. Paradigma pemrograman adalah cara fundamental dalam mengorganisasi dan menulis program komputer.

2.1.1 Timeline Paradigma Pemrograman

1. **1950-an**: Machine Language dan Assembly
2. **1960-an**: Pemrograman Prosedural (FORTRAN, COBOL)
3. **1970-an**: Pemrograman Terstruktur (C, Pascal)
4. **1980-an**: Pemrograman Berorientasi Objek (Smalltalk, C++)
5. **1990-an**: Popularisasi OOP (Java, Python)
6. **2000-an**: Multi-paradigm Languages

2.2 Paradigma Prosedural vs Berorientasi Objek

2.2.1 Pemrograman Prosedural

Pemrograman prosedural adalah paradigma yang berfokus pada **prosedur** atau **fungsi** yang beroperasi pada data.

Karakteristik:

- Program terdiri dari serangkaian instruksi/prosedur
- Data dan fungsi terpisah
- Alur program linear (top-down)
- Fokus pada "bagaimana" melakukan sesuatu

Contoh

Contoh Kode Prosedural (Pseudo-code):

```
// Data terpisah
struct Mahasiswa {
    string nim;
    string nama;
    double ipk;
}

// Fungsi terpisah
function hitungPredikat(double ipk) {
    if (ipk >= 3.5) return "Cum Laude";
    else if (ipk >= 3.0) return "Sangat Memuaskan";
    else return "Memuaskan";
}

// Penggunaan
Mahasiswa mhs;
mhs.nim = "1234";
mhs.nama = "Budi";
mhs.ipk = 3.7;
string predikat = hitungPredikat(mhs.ipk);
```

Perhatikan bahwa data (`Mahasiswa`) dan fungsi (`hitungPredikat`) terpisah.

2.2.2 Pemrograman Berorientasi Objek

Pemrograman Berorientasi Objek (OOP) adalah paradigma yang berfokus pada **objek** yang mengenkapsulasi data dan perilaku.

Karakteristik:

- Program terdiri dari objek-objek yang berinteraksi
- Data dan fungsi digabung dalam satu unit (class)
- Fokus pada "apa" yang dimodelkan
- Lebih dekat dengan dunia nyata

Contoh

Contoh Kode OOP (Java):

```java
public class Mahasiswa {
    // Data (attributes)
    private String nim;
    private String nama;
    private double ipk;

    // Constructor
    public Mahasiswa(String nim, String nama) {
        this.nim = nim;
        this.nama = nama;
        this.ipk = 0.0;
    }

    // Behavior (methods)
    public void setIPK(double ipk) {
        this.ipk = ipk;
    }

    public String getPredikat() {
        if (ipk >= 3.5) return "Cum Laude";
        else if (ipk >= 3.0) return "Sangat Memuaskan";
        else return "Memuaskan";
    }
}

// Penggunaan
Mahasiswa mhs = new Mahasiswa("1234", "Budi");
mhs.setIPK(3.7);
String predikat = mhs.getPredikat();
```

Kode Program 2.1: Class Mahasiswa dengan OOP

Data dan fungsi menyatu dalam class **Mahasiswa**.

2.2.3 Perbandingan Prosedural vs OOP

2.3 Konsep Dasar OOP

OOP dibangun di atas empat pilar utama:

Aspek	**Prosedural**	**OOP**
Fokus	Fungsi/Prosedur	Objek
Struktur Data	Data terpisah dari fungsi	Data dan fungsi dalam class
Akses Data	Global atau parameter	Encapsulation (private/public)
Code Reuse	Function library	Inheritance
Maintainability	Sulit untuk program besar	Lebih mudah dengan modularitas
Real-world Modeling	Abstrak	Lebih natural
Contoh Bahasa	C, Pascal, FORTRAN	Java, C++, Python [5, 8]

Tabel 2.1: Perbandingan Prosedural vs OOP

2.3.1 1. Abstraksi (Abstraction)

Konsep Penting

Abstraksi adalah proses menyembunyikan detail implementasi dan hanya menampilkan fungsionalitas kepada pengguna.
Contoh: Ketika menggunakan mobil, Anda hanya perlu tahu cara menggunakan setir, pedal gas, dan rem. Anda tidak perlu tahu detail mesin internal.

Dalam OOP, abstraksi dicapai melalui:

- Abstract classes
- Interfaces
- Encapsulation

2.3.2 2. Enkapsulasi (Encapsulation)

Konsep Penting

Enkapsulasi adalah pembungkusan data dan method yang beroperasi pada data tersebut dalam satu unit (class), serta menyembunyikan detail internal dari luar.

Manfaat enkapsulasi:

- **Data Hiding**: Melindungi data dari akses tidak sah
- **Modularity**: Kode lebih terorganisir
- **Flexibility**: Mudah mengubah implementasi internal
- **Maintainability**: Lebih mudah dipelihara

2.3.3 3. Pewarisan (Inheritance)

Konsep Penting

Inheritance adalah mekanisme di mana class baru (subclass) dapat mewarisi properties dan methods dari class yang sudah ada (superclass).

Manfaat inheritance:

- **Code Reusability**: Tidak perlu menulis ulang kode yang sama
- **Hierarchical Classification**: Organisasi class yang terstruktur
- **Extensibility**: Mudah menambah fitur baru

2.3.4 4. Polimorfisme (Polymorphism)

Konsep Penting

Polimorfisme adalah kemampuan objek untuk mengambil banyak bentuk. Dalam OOP, ini berarti satu interface dapat digunakan untuk tipe data yang berbeda.

Jenis polimorfisme:

- **Compile-time Polymorphism**: Method overloading
- **Runtime Polymorphism**: Method overriding

2.4 Keuntungan dan Tantangan OOP

2.4.1 Keuntungan OOP

1. **Modularitas**: Kode terorganisir dalam class-class yang independen
2. **Reusability**: Code reuse melalui inheritance dan composition
3. **Maintainability**: Lebih mudah menemukan dan memperbaiki bug
4. **Scalability**: Mudah menambah fitur baru tanpa mengubah kode existing
5. **Real-world Modeling**: Lebih natural dalam memodelkan dunia nyata
6. **Data Security**: Enkapsulasi melindungi data
7. **Collaboration**: Tim dapat bekerja pada class yang berbeda secara paralel

2.4.2 Tantangan OOP

1. **Learning Curve**: Membutuhkan pemahaman konsep yang lebih dalam
2. **Overhead**: Bisa lebih lambat untuk program sederhana
3. **Complexity**: Bisa menjadi terlalu kompleks jika tidak dirancang dengan baik
4. **Design Effort**: Membutuhkan perencanaan dan desain yang matang

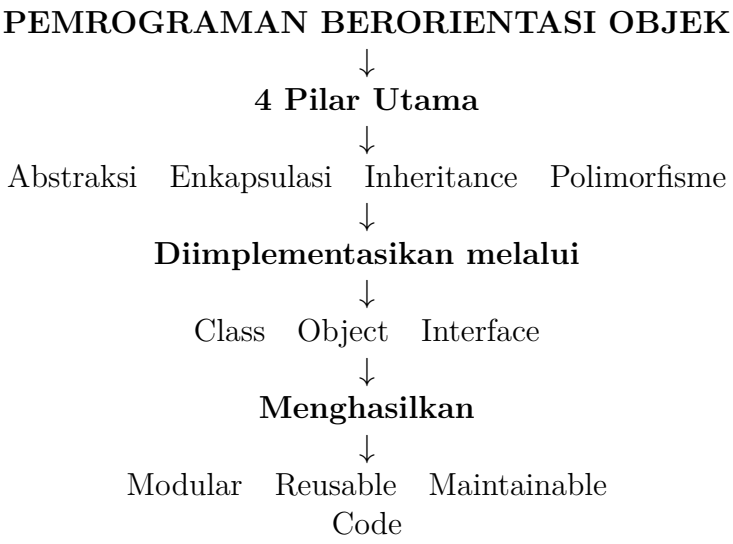
Catatan

OOP bukan solusi untuk semua masalah. Untuk program sederhana atau script kecil, pendekatan prosedural mungkin lebih efisien. Gunakan OOP ketika:

- Program cukup besar dan kompleks
- Membutuhkan code reuse yang tinggi
- Banyak developer bekerja pada proyek yang sama
- Sistem perlu mudah di-maintain dan di-extend

2.5 Peta Konsep OOP

Berikut adalah peta konsep yang menunjukkan hubungan antar konsep OOP:



Aktivitas Pembelajaran

1. **Diskusi Kelompok**: Identifikasi 5 contoh objek di dunia nyata (misalnya: mobil, HP, ATM) dan tentukan attributes dan behaviors yang bisa dimodelkan dalam OOP.

2. **Analisis Kode**: Diberikan kode prosedural untuk sistem perpustakaan sederhana, identifikasi bagian mana yang bisa diubah menjadi class dalam OOP.

3. **Perbandingan**: Tulis program sederhana (misalnya kalkulator) dalam dua versi: prosedural dan OOP. Bandingkan kelebihan dan kekurangan masing-masing.

4. **Mind Mapping**: Buat mind map yang menghubungkan 4 pilar OOP dengan contoh implementasi konkret dalam Java.

Latihan dan Refleksi

1. Jelaskan perbedaan utama antara pemrograman prosedural dan OOP! Berikan contoh kasus di mana OOP lebih cocok digunakan.

2. Sebutkan dan jelaskan 4 pilar OOP! Berikan contoh sederhana untuk masing-masing pilar.

3. Mengapa enkapsulasi penting dalam OOP? Apa yang terjadi jika semua data dalam class bersifat public?

4. Berikan 3 keuntungan dan 2 tantangan dalam menggunakan OOP.

5. Identifikasi objek-objek dalam sistem e-commerce (misalnya Tokopedia). Sebutkan minimal 5 class yang mungkin ada beserta attributes dan methods-nya.

6. Apakah semua program harus ditulis dengan OOP? Jelaskan kapan sebaiknya menggunakan OOP dan kapan tidak.

7. **Refleksi**: Bagaimana pemahaman Anda tentang OOP sebelum dan sesudah mempelajari bab ini? Konsep mana yang paling sulit dipahami?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 1.1

A. Pilihan Ganda

1. Manakah yang BUKAN merupakan pilar OOP?

   (a) Abstraksi
   (b) Enkapsulasi
   (c) Kompilasi
   (d) Polimorfisme

2. Dalam OOP, data dan method yang beroperasi pada data tersebut digabung dalam satu unit yang disebut:
   (a) Function
   (b) Module
   (c) Class
   (d) Library
3. Keuntungan utama enkapsulasi adalah:
   (a) Membuat program lebih cepat
   (b) Melindungi data dari akses tidak sah
   (c) Mengurangi ukuran file
   (d) Mempermudah kompilasi

B. Essay

1. Jelaskan dengan kata-kata Anda sendiri apa yang dimaksud dengan "objek" dalam OOP dan berikan 2 contoh objek dari dunia nyata beserta attributes dan behaviors-nya.
2. Bandingkan pendekatan prosedural dan OOP dalam menyelesaikan masalah pengelolaan data mahasiswa. Mana yang lebih cocok dan mengapa?

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya dapat menjelaskan perbedaan antara pemrograman prosedural dan OOP

☐ Saya dapat menyebutkan dan menjelaskan 4 pilar OOP

☐ Saya dapat memberikan contoh konkret untuk setiap pilar OOP

☐ Saya dapat mengidentifikasi kapan sebaiknya menggunakan OOP

☐ Saya memahami keuntungan dan tantangan dalam menggunakan OOP

☐ Saya dapat mengidentifikasi objek, attributes, dan methods dari dunia nyata

Rangkuman

Bab ini membahas landasan teori Pemrograman Berorientasi Objek, termasuk evolusi paradigma pemrograman, perbandingan antara pendekatan prosedural dan OOP, serta 4 pilar utama OOP.

Poin Kunci:

- OOP adalah paradigma yang berfokus pada objek yang mengenkapsulasi data dan perilaku
- 4 Pilar OOP: Abstraksi, Enkapsulasi, Inheritance, Polimorfisme
- OOP menawarkan modularitas, reusability, dan maintainability yang lebih baik
- OOP lebih cocok untuk sistem besar dan kompleks
- Pemahaman konsep dasar OOP adalah fondasi untuk mempelajari implementasi OOP dalam Java

Kata Kunci: ***OOP***, ***Class***, ***Object***, ***Abstraksi***, ***Enkapsulasi***, ***Inheritance***, ***Polimorfisme***, ***Modularitas***

Bab 3

Konsep Class dan Object

Sub-CPMK yang Dicakup dalam Bab Ini:

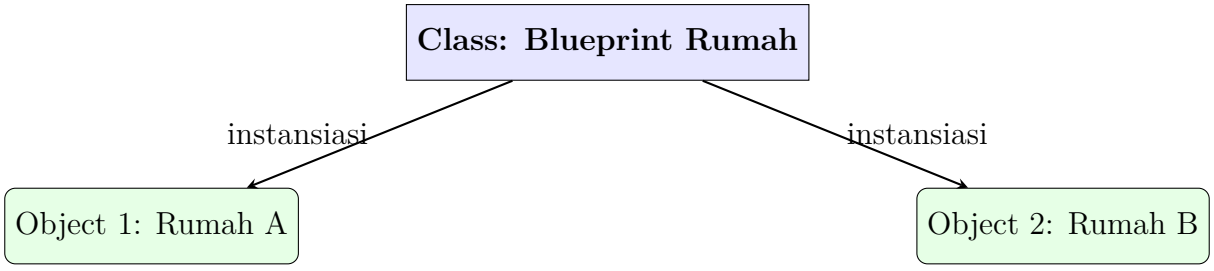
- Sub-CPMK 1.2: Mengidentifikasi class, object, dan attribute dalam studi kasus nyata, serta membuat class sederhana dengan attributes dan methods
- Sub-CPMK 1.2: Memahami perbedaan antara class dan object

3.1 Definisi Class dan Object

3.1.1 Apa itu Class?

Konsep Penting

Class adalah blueprint atau template yang mendefinisikan karakteristik dan perilaku dari suatu entitas [1]. Class mendefinisikan attributes (data) dan methods (behavior) yang akan dimiliki oleh objek.
Analogi: Class seperti cetak biru rumah, sedangkan object adalah rumah yang dibangun berdasarkan cetak biru tersebut.



Gambar 3.1: Konsep Class dan Object

Dalam Java, class didefinisikan dengan keyword `class`:

```java
public class NamaClass {
    // Attributes (instance variables)
    tipeData namaAttribute;

    // Constructor
    public NamaClass() {
        // Inisialisasi
    }

    // Methods
    public void namaMethod() {
        // Implementasi
    }
}
```

Kode Program 3.1: Struktur Dasar Class

3.1.2 Apa itu Object?

Konsep Penting

Object adalah instance (perwujudan) dari class. Object memiliki state (nilai attributes) dan behavior (methods) yang didefinisikan oleh class-nya.

Membuat object dalam Java menggunakan keyword **new**:

```java
NamaClass namaObject = new NamaClass();
```

Kode Program 3.2: Membuat Object

3.1.3 Hubungan Class dan Object

Aspek	Class	Object
Definisi	Template/Blueprint	Instance dari class
Keberadaan	Logical entity	Physical entity
Memory	Tidak menggunakan memory	Menggunakan memory
Jumlah	Satu class	Banyak object dari satu class
Keyword	`class`	`new`

Tabel 3.1: Hubungan Class dan Object

3.2 Attributes (Instance Variables)

3.2.1 Definisi Attributes

Attributes adalah variabel yang didefinisikan di dalam class untuk menyimpan state-/data dari object.

```
public class Mahasiswa {
    // Attributes
    String nim;
    String nama;
    String jurusan;
    int semester;
    double ipk;
} \cite{ref2, ref7}
```

Kode Program 3.3: Contoh Attributes

3.2.2 Jenis Attributes

1. **Instance Variables**: Unik untuk setiap object
2. **Static Variables**: Dibagi oleh semua object (akan dibahas nanti)

3.3 Methods (Behavior)

3.3.1 Definisi Methods

Methods adalah fungsi yang didefinisikan di dalam class untuk menentukan behavior dari object.

```
public class Mahasiswa {
    String nim;
    String nama;
    double ipk;

    // Method untuk menampilkan info
    public void tampilkanInfo() {
        System.out.println("NIM: " + nim);
        System.out.println("Nama: " + nama);
        System.out.println("IPK: " + ipk);
    }

    // Method untuk mengecek kelulusan
    public boolean lulus() {
        return ipk >= 2.0;
    }

    // Method untuk mendapatkan predikat
```

```
19      public String getPredikat() {
20          if (ipk >= 3.5) return "Cum Laude";
21          else if (ipk >= 3.0) return "Sangat Memuaskan";
22          else if (ipk >= 2.5) return "Memuaskan";
23          else return "Cukup";
24      }
25  }
```

Kode Program 3.4: Contoh Methods

3.3.2 Jenis Methods

1. **Accessor Methods (Getter)**: Mengambil nilai attribute
2. **Mutator Methods (Setter)**: Mengubah nilai attribute
3. **Utility Methods**: Melakukan operasi tertentu

3.4 Constructor

3.4.1 Apa itu Constructor?

Konsep Penting

Constructor adalah method khusus yang dipanggil saat object dibuat. Constructor digunakan untuk menginisialisasi attributes object.
Ciri-ciri constructor:

- Nama sama dengan nama class
- Tidak memiliki return type (bahkan bukan `void`)
- Dipanggil otomatis saat object dibuat dengan **`new`**

```
1   public class Mahasiswa {
2       String nim;
3       String nama;
4       double ipk;
5
6       // Constructor tanpa parameter (default constructor)
7       public Mahasiswa() {
8           nim = "0000";
9           nama = "Unknown";
10          ipk = 0.0;
11      }
12
13      // Constructor dengan parameter
14      public Mahasiswa(String nim, String nama) {
15          this.nim = nim;
```

```
16          this.nama = nama;
17          this.ipk = 0.0;
18      }
19
20      // Constructor dengan semua parameter
21      public Mahasiswa(String nim, String nama, double ipk) {
22          this.nim = nim;
23          this.nama = nama;
24          this.ipk = ipk;
25      }
26  }
```

Kode Program 3.5: Contoh Constructor

3.4.2 Constructor Overloading

Class dapat memiliki multiple constructors dengan parameter yang berbeda. Ini disebut ***constructor overloading***.

3.5 Keyword `this`

3.5.1 Penggunaan `this`

Keyword **this** merujuk pada object saat ini. Digunakan untuk:

1. Membedakan instance variable dengan parameter yang namanya sama
2. Memanggil constructor lain dalam class yang sama
3. Mengembalikan instance saat ini

```
1   public class Mahasiswa {
2       private String nama;
3       private double ipk;
4
5       // this untuk membedakan parameter dan instance variable
6       public Mahasiswa(String nama, double ipk) {
7           this.nama = nama;  // this.nama = instance variable
8           this.ipk = ipk;    // nama = parameter
9       }
10
11      // this untuk memanggil constructor lain
12      public Mahasiswa(String nama) {
13          this(nama, 0.0);  // Memanggil constructor di atas
14      }
15
16      // this untuk mengembalikan object saat ini
17      public Mahasiswa setNama(String nama) {
```

```
18          this.nama = nama;
19          return this;  // Method chaining
20      }
21  }
```

Kode Program 3.6: Penggunaan this

3.6 Object Creation dan Instantiation

3.6.1 Membuat Object

Proses membuat object disebut ***instantiation***. Dalam Java, gunakan keyword **new**:

```
1  public class Main {
2      public static void main(String[] args) {
3          // Membuat object dengan default constructor
4          Mahasiswa mhs1 = new Mahasiswa();
5
6          // Membuat object dengan parameterized constructor
7          Mahasiswa mhs2 = new Mahasiswa("123456", "Budi Santoso"
               );
8
9          // Mengakses attributes (jika public)
10         mhs2.ipk = 3.75;
11
12         // Memanggil methods
13         mhs2.tampilkanInfo();
14         System.out.println("Predikat: " + mhs2.getPredikat());
15
16         // Cek kelulusan
17         if (mhs2.lulus()) {
18             System.out.println(mhs2.nama + " dinyatakan LULUS")
                   ;
19         }
20     }
21 }
```

Kode Program 3.7: Membuat dan Menggunakan Object

3.6.2 Memory Allocation

Ketika object dibuat:

1. Memory dialokasikan di **heap**
2. Reference variable disimpan di **stack**
3. Constructor dipanggil untuk inisialisasi

Catatan

Jika tidak ada constructor yang didefinisikan, Java otomatis menyediakan **default constructor** tanpa parameter yang menginisialisasi attributes dengan nilai default (0, null, false).

3.7 Static vs Instance Members

3.7.1 Instance Members

Instance members (attributes dan methods) adalah milik object. Setiap object memiliki copy sendiri.

3.7.2 Static Members

Static members adalah milik class, bukan object. Dibagi oleh semua object.

```
1  public class Mahasiswa {
2      // Instance variable (setiap object punya copy sendiri)
3      private String nama;
4      private double ipk;
5
6      // Static variable (dibagi semua object)
7      private static int jumlahMahasiswa = 0;
8      private static String namaUniversitas = "Universitas Lorem
           Ipsum";
9
10     public Mahasiswa(String nama, double ipk) {
11         this.nama = nama;
12         this.ipk = ipk;
13         jumlahMahasiswa++;  // Increment setiap object dibuat
14     }
15
16     // Instance method
17     public void tampilkanInfo() {
18         System.out.println("Nama: " + nama);
19         System.out.println("IPK: " + ipk);
20     }
21
22     // Static method
23     public static int getJumlahMahasiswa() {
24         return jumlahMahasiswa;
25     }
26
27     public static String getNamaUniversitas() {
28         return namaUniversitas;
29     }
30 }
```

```
31
32 // Penggunaan
33 public class Main {
34     public static void main(String[] args) {
35         // Akses static member tanpa object
36         System.out.println("Universitas: " + Mahasiswa.
               getNamaUniversitas());
37
38         Mahasiswa mhs1 = new Mahasiswa("Budi", 3.5);
39         Mahasiswa mhs2 = new Mahasiswa("Ani", 3.8);
40
41         // Akses static member melalui class name
42         System.out.println("Total mahasiswa: " + Mahasiswa.
               getJumlahMahasiswa());
43         // Output: Total mahasiswa: 2
44     }
45 }
```

Kode Program 3.8: Static vs Instance

Catatan

Best Practice:

- Akses static members melalui class name, bukan object
- Static methods tidak bisa mengakses instance variables secara langsung
- Gunakan static untuk utility methods atau constants

3.8 Contoh Lengkap: Class Buku

```
1  public class Buku {
2      // Attributes
3      private String isbn;
4      private String judul;
5      private String penulis;
6      private int tahunTerbit;
7      private double harga;
8      private boolean tersedia;
9
10     // Static variable
11     private static int jumlahBuku = 0;
12
13     // Constructor
14     public Buku(String isbn, String judul, String penulis,
15                 int tahunTerbit, double harga) {
16         this.isbn = isbn;
17         this.judul = judul;
18         this.penulis = penulis;
```

```
19          this.tahunTerbit = tahunTerbit;
20          this.harga = harga;
21          this.tersedia = true;
22          jumlahBuku++;
23      }
24
25      // Getter methods
26      public String getISBN() {
27          return isbn;
28      }
29
30      public String getJudul() {
31          return judul;
32      }
33
34      public String getPenulis() {
35          return penulis;
36      }
37
38      public int getTahunTerbit() {
39          return tahunTerbit;
40      }
41
42      public double getHarga() {
43          return harga;
44      }
45
46      public boolean isTersedia() {
47          return tersedia;
48      }
49
50      // Setter methods
51      public void setHarga(double harga) {
52          if (harga > 0) {
53              this.harga = harga;
54          }
55      }
56
57      public void setTersedia(boolean tersedia) {
58          this.tersedia = tersedia;
59      }
60
61      // Utility methods
62      public void pinjam() {
63          if (tersedia) {
64              tersedia = false;
65              System.out.println("Buku '" + judul + "' berhasil
                    dipinjam");
66          } else {
67              System.out.println("Buku tidak tersedia");
68          }
```

```
69      }
70
71      public void kembalikan() {
72          tersedia = true;
73          System.out.println("Buku '" + judul + "' berhasil
               dikembalikan");
74      }
75
76      public void tampilkanInfo() {
77          System.out.println("=== Informasi Buku ===");
78          System.out.println("ISBN: " + isbn);
79          System.out.println("Judul: " + judul);
80          System.out.println("Penulis: " + penulis);
81          System.out.println("Tahun: " + tahunTerbit);
82          System.out.println("Harga: Rp " + harga);
83          System.out.println("Status: " + (tersedia ? "Tersedia"
               : "Dipinjam"));
84      }
85
86      // Static method
87      public static int getJumlahBuku() {
88          return jumlahBuku;
89      }
90  }
```

Kode Program 3.9: Class Buku - Contoh Lengkap

```
1   public class TestBuku {
2       public static void main(String[] args) {
3           // Membuat object buku
4           Buku buku1 = new Buku("978-0134685991", "Effective Java
                ",
5                                 "Joshua Bloch", 2018, 450000);
6
7           Buku buku2 = new Buku("978-0596009205", "Head First
                Java",
8                                 "Kathy Sierra", 2005, 350000);
9
10          // Menampilkan informasi
11          buku1.tampilkanInfo();
12          System.out.println();
13
14          // Meminjam buku
15          buku1.pinjam();
16          buku1.pinjam();  // Coba pinjam lagi
17
18          // Mengembalikan buku
19          buku1.kembalikan();
20
21          // Mengubah harga
22          buku2.setHarga(300000);
23
```

```
24          // Menampilkan jumlah buku
25          System.out.println("\nTotal buku: " + Buku.
               getJumlahBuku());
26      }
27  }
```

Kode Program 3.10: Penggunaan Class Buku

Aktivitas Pembelajaran

1. **Identifikasi Class**: Dari studi kasus sistem perpustakaan, identifikasi minimal 5 class yang diperlukan beserta attributes dan methods-nya.

2. **Implementasi Class**: Buat class `Mobil` dengan attributes: merk, model, tahun, warna, harga. Tambahkan constructor dan methods yang sesuai.

3. **Object Interaction**: Buat class `Dosen` dan `MataKuliah`. Buat program yang mendemonstrasikan interaksi antara object dosen dan mata kuliah.

4. **Static vs Instance**: Buat class `Counter` dengan static variable untuk menghitung jumlah object yang dibuat. Test dengan membuat beberapa object.

5. **Code Review**: Bertukar kode dengan teman, review class yang dibuat teman Anda. Berikan feedback tentang naming, encapsulation, dan design.

Latihan dan Refleksi

1. Jelaskan perbedaan antara class dan object! Berikan analogi dari dunia nyata.

2. Apa fungsi constructor? Apa yang terjadi jika tidak mendefinisikan constructor dalam class?

3. Jelaskan perbedaan antara instance variable dan static variable! Kapan sebaiknya menggunakan static variable?

4. Buat class `RekeningBank` dengan attributes: nomorRekening, namaPemilik, saldo. Tambahkan methods: setor(), tarik(), cekSaldo(). Implementasikan validasi yang sesuai.

5. Buat class `Lingkaran` dengan attribute radius. Tambahkan methods untuk menghitung luas dan keliling. Buat program untuk membuat beberapa object lingkaran dengan radius berbeda.

6. Modifikasi class `Mahasiswa` untuk menambahkan array nilai mata kuliah. Tambahkan method untuk menghitung IPK berdasarkan nilai-nilai tersebut.

7. Buat class `Produk` untuk sistem e-commerce dengan attributes yang sesuai. Implementasikan method untuk menghitung harga setelah diskon.

8. **Refleksi**: Apa tantangan terbesar yang Anda hadapi dalam memahami konsep class dan object? Bagaimana Anda mengatasinya?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 1.2

A. Pilihan Ganda

1. Keyword yang digunakan untuk membuat object dalam Java adalah:
   (a) create
   (b) new
   (c) object
   (d) instance

2. Manakah pernyataan yang BENAR tentang constructor?
   (a) Constructor harus memiliki return type
   (b) Nama constructor harus berbeda dengan nama class
   (c) Constructor dipanggil otomatis saat object dibuat
   (d) Satu class hanya boleh memiliki satu constructor

3. Static variable dalam class:
   (a) Unik untuk setiap object
   (b) Dibagi oleh semua object
   (c) Tidak bisa diakses dari luar class
   (d) Harus selalu private

4. Keyword `this` digunakan untuk:
   (a) Merujuk pada superclass
   (b) Merujuk pada object saat ini
   (c) Membuat object baru
   (d) Menghapus object

B. Essay

1. Jelaskan perbedaan antara instance method dan static method! Berikan contoh penggunaan masing-masing.

2. Mengapa encapsulation penting? Jelaskan menggunakan class **`RekeningBank`** sebagai contoh.

C. Coding Challenge

1. Buat class **`Karyawan`** dengan attributes: nik, nama, jabatan, gajiPokok. Tambahkan:
   - Constructor dengan parameter
   - Getter dan setter methods
   - Method hitungGajiBersih() yang menghitung gaji setelah potongan pajak 10%
   - Method tampilkanInfo()
2. Buat program main yang membuat minimal 3 object karyawan dan menampilkan informasi mereka.

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya dapat menjelaskan perbedaan antara class dan object

☐ Saya dapat membuat class dengan attributes dan methods

☐ Saya dapat membuat constructor dengan dan tanpa parameter

☐ Saya dapat membuat dan menggunakan object dari class

☐ Saya memahami penggunaan keyword `this`

☐ Saya dapat membedakan instance variable dan static variable

☐ Saya dapat mengidentifikasi class dan object dari studi kasus nyata

☐ Saya dapat mengimplementasikan getter dan setter methods

Rangkuman

Bab ini membahas konsep fundamental OOP: Class dan Object. Class adalah blueprint untuk membuat object, sedangkan object adalah instance dari class.

Poin Kunci:

- Class terdiri dari attributes (data) dan methods (behavior)

- Object dibuat menggunakan keyword `new`
- Constructor adalah method khusus untuk inisialisasi object
- Keyword `this` merujuk pada object saat ini
- Instance members milik object, static members milik class
- Encapsulation dicapai dengan access modifiers dan getter/setter

Kata Kunci: ***Class***, ***Object***, ***Attribute***, ***Method***, ***Constructor***, ***this***, ***static***, ***instance***, ***instantiation***

Bab 4

Encapsulation dan Information Hiding

4.1 Konsep Encapsulation dan Access Modifier

Sub-CPMK yang Dicakup dalam Bab Ini:

- Sub-CPMK 1.3: Mendemonstrasikan konsep encapsulation dengan access modifier

Konsep Penting

Encapsulation adalah prinsip OOP yang menggabungkan data (attributes) dan perilaku (methods) dalam satu class, sekaligus menyembunyikan detail internal dari akses langsung luar [1]. Tujuannya adalah menjaga *class invariant* agar data selalu dalam kondisi valid.

Dengan encapsulation, kita dapat:

- Melindungi data dari perubahan yang tidak sah
- Mengontrol akses melalui method publik
- Memudahkan perawatan dan perubahan implementasi internal

4.1.1 Access Modifier di Java

Java menyediakan access modifier untuk mengatur visibilitas anggota class:

4.1.2 Contoh Encapsulation Sederhana

Modifier	**Akses**	**Keterangan**
public	Semua class	Dapat diakses dari mana saja
protected	Package + subclass	Dapat diakses dari package yang sama dan subclass
(default)	Package	Tanpa keyword, hanya dalam package yang sama
private	Class sendiri	Hanya dapat diakses di dalam class

Tabel 4.1: Access Modifier di Java

```
1  public class RekeningBank {
2      private String nomorRekening;
3      private String namaPemilik;
4      private double saldo;
5
6      public RekeningBank(String nomorRekening, String
           namaPemilik) {
7          this.nomorRekening = nomorRekening;
8          this.namaPemilik = namaPemilik;
9          this.saldo = 0.0;
10     }
11
12     public void setor(double jumlah) {
13         if (jumlah <= 0) {
14             throw new IllegalArgumentException("Jumlah harus
                   positif");
15         }
16         saldo += jumlah;
17     }
18
19     public void tarik(double jumlah) {
20         if (jumlah <= 0 || jumlah > saldo) {
21             throw new IllegalArgumentException("Saldo tidak
                   mencukupi");
22         }
23         saldo -= jumlah;
24     }
25
26     public double getSaldo() {
27         return saldo;
28     }
29 }
```

Kode Program 4.1: Encapsulation pada class RekeningBank

Contoh di atas menunjukkan bahwa `saldo` hanya dapat diubah melalui method `setor()` dan `tarik()`, sehingga aturan bisnis dapat dijaga dengan konsisten.

4.2 Getter/Setter, Validasi, dan Immutability

4.2.1 Getter dan Setter

Getter dan setter adalah method publik yang mengontrol akses terhadap attribute private. Pola ini memungkinkan validasi dan konsistensi data.

```
public class Produk {
    private String kode;
    private String nama;
    private double harga;

    public Produk(String kode, String nama, double harga) {
        this.kode = kode;
        this.nama = nama;
        setHarga(harga);
    }

    public String getKode() {
        return kode;
    }

    public String getNama() {
        return nama;
    }

    public double getHarga() {
        return harga;
    }

    public void setHarga(double harga) {
        if (harga <= 0) {
            throw new IllegalArgumentException("Harga harus
                positif");
        }
        this.harga = harga;
    }
}
```

Kode Program 4.2: Contoh Getter dan Setter dengan Validasi

4.2.2 Immutability

Object *immutable* tidak dapat diubah setelah dibuat. Pendekatan ini meminimalkan bug dan meningkatkan keamanan data.

Catatan

Untuk membuat class immutable:

- Gunakan `private final` untuk attributes

- Tidak menyediakan setter
- Inisialisasi di constructor
- Jika menyimpan objek mutable, lakukan defensive copy

Aktivitas Pembelajaran

1. **Refactoring Encapsulation**: Diberikan class dengan attributes public, ubah menjadi private dan buat getter/setter yang tepat.
2. **Validasi Data**: Implementasikan validasi untuk class `Mahasiswa` (nilai 0-100, IPK 0.0-4.0).
3. **Design Review**: Diskusikan kapan sebaiknya sebuah attribute tidak diberi setter.
4. **Studi Kasus**: Buat class `BookingHotel` dengan aturan: tanggal check-out harus setelah check-in.

Latihan dan Refleksi

1. Jelaskan manfaat utama encapsulation dalam pengembangan software.
2. Apa perbedaan antara attribute `public` dan `private`? Berikan contoh.
3. Buat class `KartuMember` dengan attribute nomorMember, nama, poin. Terapkan encapsulation dan validasi poin tidak boleh negatif.
4. Kapan sebaiknya sebuah class dibuat immutable? Jelaskan dengan contoh.
5. **Refleksi**: Bagian apa dari encapsulation yang paling sering Anda lupakan saat menulis kode?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 1.3

A. Pilihan Ganda

1. Tujuan utama encapsulation adalah:
   (a) Mempercepat eksekusi program

(b) Menyembunyikan detail internal dan mengontrol akses data

(c) Mengurangi jumlah class

(d) Menghapus kebutuhan constructor

2. Keyword yang paling tepat untuk attribute yang hanya boleh diakses dalam class:

(a) public

(b) protected

(c) private

(d) default

B. Tugas Praktik

- Buat class **`AkunPengguna`** dengan validasi email dan password minimal 8 karakter.
- Sertakan getter yang diperlukan dan batasi setter pada data yang tidak boleh diubah.

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya dapat menjelaskan konsep encapsulation dan information hiding

☐ Saya dapat menggunakan access modifier dengan tepat

☐ Saya mampu membuat getter/setter dengan validasi

☐ Saya mengetahui kapan menggunakan object immutable

☐ Saya dapat menjaga class invariant dengan encapsulation

Rangkuman

Encapsulation menjaga data tetap aman dan konsisten melalui access modifier dan method pengontrol [1]. Getter/setter memungkinkan validasi, sedangkan immutability membantu mencegah perubahan yang tidak diinginkan.

Kata Kunci: ***Encapsulation***, ***Access Modifier***, ***Getter/Setter***, ***Immutable***, ***Validation***

Bab 5

Inheritance dan Hierarki Kelas

5.1 Konsep Inheritance dan Relasi "is-a"

Sub-CPMK yang Dicakup dalam Bab Ini:

- Sub-CPMK 3.1: Mengimplementasikan inheritance dan method overriding

5.1.1 Konsep Pewarisan

Konsep Penting

Inheritance memungkinkan kita untuk membuat class baru berdasarkan class yang sudah ada, mewarisi semua attributes dan methods-nya [5]. Relasi ini dikenal sebagai relasi *is-a*.

Keuntungan utama inheritance:

- **Code reuse**: mengurangi duplikasi kode
- **Extensibility**: menambah fitur dengan subclass
- **Polymorphism**: memungkinkan penggunaan reference superclass

5.1.2 Keyword `extends`

Dalam Java, inheritance dideklarasikan dengan keyword **`extends`**.

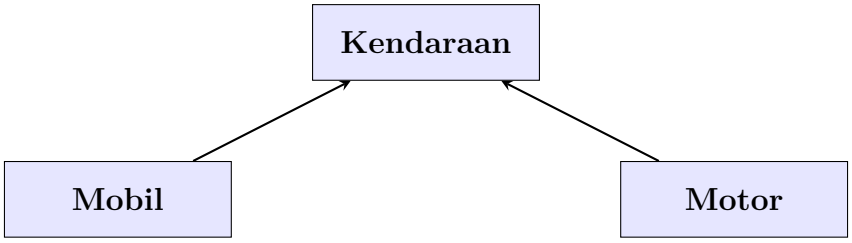
```
1 class Kendaraan {
2     protected String merk;
3
```

```
    public Kendaraan(String merk) {
        this.merk = merk;
    }

    public void nyalakan() {
        System.out.println("Kendaraan dinyalakan");
    }
}

class Mobil extends Kendaraan {
    private int jumlahPintu;

    public Mobil(String merk, int jumlahPintu) {
        super(merk);
        this.jumlahPintu = jumlahPintu;
    }
}
```

Kode Program 5.1: Inheritance Sederhana

5.1.3 Hierarki Kelas

Hierarki kelas membantu kita mengorganisasi konsep dalam struktur bertingkat:



Gambar 5.1: Contoh Hierarki Kelas (Relasi is-a)

5.2 Overriding, super, dan Desain Hierarki

5.2.1 Method Overriding

Subclass dapat mendefinisikan ulang method dari superclass. Gunakan anotasi `@Override` untuk memperjelas niat.

```
class Kendaraan {
    public void nyalakan() {
        System.out.println("Kendaraan dinyalakan");
    }
}

class Mobil extends Kendaraan {
    @Override
```

```
9       public void nyalakan() {
10          System.out.println("Mobil dinyalakan dengan kunci");
11      }
12  }
```

Kode Program 5.2: Contoh Method Overriding

5.2.2 Penggunaan super

`super` digunakan untuk memanggil constructor atau method dari superclass.

```
1   class Dosen {
2       protected String nama;
3
4       public Dosen(String nama) {
5           this.nama = nama;
6       }
7   }
8
9   class DosenTetap extends Dosen {
10      private String nidn;
11
12      public DosenTetap(String nama, String nidn) {
13          super(nama);
14          this.nidn = nidn;
15      }
16  }
```

Kode Program 5.3: Penggunaan super

5.2.3 Composition vs Inheritance

Gunakan inheritance jika relasi *is-a* kuat. Jika tidak, gunakan composition (*has-a*) untuk fleksibilitas.

Catatan

Contoh: **`Mobil`** *has-a* **`Mesin`**. Lebih tepat menggunakan composition.

Aktivitas Pembelajaran

1. **Hierarki Kelas**: Buat hierarki class untuk sistem akademik (Person → Mahasiswa-/Dosen).

2. **Overriding**: Implementasikan method `hitungGaji()` berbeda di class **`Pegawai`** dan **`Manajer`**.

3. **Diskusi**: Identifikasi contoh *is-a* dan *has-a* dalam aplikasi e-commerce.

4. **Code Review**: Cari potensi penyalahgunaan inheritance pada kode yang diberikan dosen.

Latihan dan Refleksi

1. Jelaskan perbedaan inheritance dan composition.

2. Buat class **`Hewan`** dan subclass **`Kucing`**. Tambahkan method `suara()` dan override di subclass.

3. Kapan penggunaan inheritance dapat menyebabkan masalah desain? Berikan contoh.

4. Buat class **`Pegawai`** dan subclass **`PegawaiKontrak`**. Tambahkan atribut dan method spesifik.

5. **Refleksi**: Apa tantangan terbesar Anda saat menyusun hierarki kelas?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 3.1

A. Pilihan Ganda

1. Keyword untuk melakukan inheritance di Java adalah:
   (a) inherit
   (b) extends
   (c) implements
   (d) override

2. Method overriding berarti:
   (a) Dua method dengan nama sama dalam class yang sama
   (b) Subclass mendefinisikan ulang method superclass
   (c) Method dipanggil berulang
   (d) Method private diakses dari luar class

B. Tugas Praktik

- Buat hierarki class **`Produk`** → **`ProdukDigital`** dan **`ProdukFisik`** dengan atribut dan method yang sesuai.

- Demonstrasikan overriding pada method `hitungHargaAkhir()`.

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya memahami konsep inheritance dan relasi *is-a*

☐ Saya dapat membuat subclass dengan **`extends`**

☐ Saya mampu menerapkan method overriding

☐ Saya memahami penggunaan `super`

☐ Saya dapat memilih inheritance atau composition dengan tepat

Rangkuman

Inheritance memungkinkan class baru mewarisi perilaku dan data dari class lain, mendukung code reuse dan desain hierarki [5]. Overriding dan `super` adalah kunci untuk menyesuaikan perilaku subclass.

Kata Kunci: ***Inheritance***, ***extends***, ***overriding***, ***super***, ***is-a***, ***composition***

Bab 6

Polymorphism dan Dynamic Binding

6.1 Polymorphism dan Method Overloading

Sub-CPMK yang Dicakup dalam Bab Ini:

- Sub-CPMK 3.2: Menerapkan polymorphism dalam desain aplikasi

Konsep Penting

Polymorphism adalah kemampuan suatu object untuk mengambil berbagai bentuk [5]. Dalam Java, polymorphism muncul dalam dua bentuk: compile-time (overloading) dan runtime (overriding).

6.1.1 Compile-time Polymorphism (Overloading)

Method overloading terjadi ketika terdapat beberapa method dengan nama sama, tetapi parameter berbeda.

```
class Kalkulator {
    public int tambah(int a, int b) {
        return a + b;
    }

    public double tambah(double a, double b) {
        return a + b;
    }

    public int tambah(int a, int b, int c) {
        return a + b + c;
    }
}
```

Kode Program 6.1: Contoh Overloading

6.1.2 Keuntungan Overloading

- Memudahkan penggunaan API karena nama method konsisten
- Mengurangi kompleksitas nama method
- Membuat kode lebih mudah dibaca

6.1.3 Perbandingan Overloading vs Overriding

Aspek	Overloading	Overriding
Binding Time	Compile-time (Static)	Runtime (Dynamic)
Class	Dalam class yang sama	Antara superclass dan subclass
Keyword	Tidak ada	`@Override` (Opsional tapi disarankan)
Parameter	Harus berbeda	Harus sama
Return Type	Boleh berbeda	Harus sama atau sub-type

Tabel 6.1: Perbandingan Overloading dan Overriding

6.2 Runtime Polymorphism dan Dynamic Dispatch

6.2.1 Overriding dan Dynamic Binding

Runtime polymorphism terjadi saat method yang dipanggil ditentukan pada saat runtime berdasarkan objek aktual.

```
1  class Bentuk {
2      public double luas() {
3          return 0.0;
4      }
5  }
6
7  class Lingkaran extends Bentuk {
8      private double r;
9
10     public Lingkaran(double r) {
11         this.r = r;
12     }
13
14     @Override
15     public double luas() {
```

```
16          return Math.PI * r * r;
17      }
18  }
19
20  class Persegi extends Bentuk {
21      private double s;
22
23      public Persegi(double s) {
24          this.s = s;
25      }
26
27      @Override
28      public double luas() {
29          return s * s;
30      }
31  }
32
33  public class Demo {
34      public static void main(String[] args) {
35          Bentuk b1 = new Lingkaran(7);
36          Bentuk b2 = new Persegi(4);
37          System.out.println(b1.luas());
38          System.out.println(b2.luas());
39      }
40  }
```

Kode Program 6.2: Runtime Polymorphism

6.2.2 Polymorphism pada Interface

Polymorphism juga terjadi ketika menggunakan reference bertipe interface.

Aktivitas Pembelajaran

1. **Eksperimen Overloading**: Buat class **`Printer`** dengan method `cetak()` untuk tipe data berbeda.

2. **Dynamic Dispatch**: Buat array bertipe superclass yang berisi beberapa subclass dan panggil method yang dioverride.

3. **Studi Kasus**: Implementasikan interface **`Pembayaran`** dengan beberapa jenis pembayaran.

4. **Diskusi**: Jelaskan manfaat polymorphism dalam pengembangan framework.

Latihan dan Refleksi

1. Apa perbedaan overloading dan overriding?
2. Buat class **`Hewan`** dengan method `suara()`. Buat subclass **`Anjing`** dan **`Kucing`** yang mengoverride method tersebut.
3. Buat contoh penggunaan interface yang menunjukkan polymorphism.
4. Mengapa polymorphism penting untuk extensibility aplikasi?
5. **Refleksi**: Dalam proyek Anda, bagian mana yang paling terbantu oleh polymorphism?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 3.2

A. Pilihan Ganda

1. Runtime polymorphism terjadi karena:
   (a) Overloading
   (b) Overriding
   (c) Constructor
   (d) Static method
2. Method yang dipanggil pada runtime ditentukan oleh:
   (a) Tipe reference
   (b) Tipe objek aktual
   (c) Nama method
   (d) Jumlah parameter

B. Tugas Praktik

- Buat hierarki class **`Notifikasi`** dengan subclass **`Email`** dan **`SMS`**. Demonstrasikan polymorphism saat mengirim notifikasi.

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya memahami konsep polymorphism

☐ Saya dapat membuat method overloading

☐ Saya dapat menerapkan method overriding

☐ Saya memahami dynamic dispatch pada runtime

☐ Saya dapat menggunakan polymorphism dengan interface

Rangkuman

Polymorphism memungkinkan satu interface digunakan untuk banyak bentuk objek [5].

Overloading terjadi di compile-time, sedangkan overriding terjadi di runtime melalui dynamic dispatch.

Kata Kunci: ***Polymorphism***, ***overloading***, ***overriding***, ***dynamic dispatch***, ***interface***

Bab 7

Abstract Class dan Interface

7.1 Abstract Class dan Abstraksi

Sub-CPMK yang Dicakup dalam Bab Ini:

- Sub-CPMK 3.2: Menerapkan polymorphism dalam desain aplikasi

Konsep Penting

Abstraksi melalui abstract class dan interface adalah mekanisme utama untuk mencapai desain yang fleksibel dalam Java [5]. Abstract class tidak dapat diinstansiasi dan dapat memiliki method abstrak serta konkret.

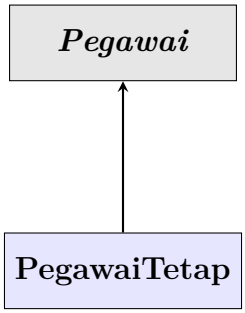
7.1.1 Karakteristik Abstract Class

- Dideklarasikan dengan keyword `abstract`
- Dapat memiliki atribut dan constructor
- Dapat memiliki method abstract dan non-abstract

```
abstract class Pegawai {
    protected String nama;

    public Pegawai(String nama) {
        this.nama = nama;
    }

    public abstract double hitungGaji();

    public void tampilkanInfo() {
```

```
        System.out.println("Nama: " + nama);
    }
}

class PegawaiTetap extends Pegawai {
    private double gajiBulanan;

    public PegawaiTetap(String nama, double gajiBulanan) {
        super(nama);
        this.gajiBulanan = gajiBulanan;
    }

    @Override
    public double hitungGaji() {
        return gajiBulanan;
    }
}
```

Kode Program 7.1: Contoh Abstract Class



Gambar 7.1: Hierarki Class Pegawai (Abstract)

7.1.2 Kapan Menggunakan Abstract Class

Gunakan abstract class ketika:

- Ada perilaku umum yang ingin diwariskan
- Ada method yang wajib diimplementasikan subclass
- Dibutuhkan constructor atau state bersama

7.2 Interface dan Perbandingannya dengan Abstract Class

7.2.1 Konsep Interface

Interface mendefinisikan kontrak yang harus diimplementasikan oleh class.

```java
interface Pembayaran {
    void proses();
    void verifikasi();
}

class PembayaranKartu implements Pembayaran {
    @Override
    public void proses() {
        System.out.println("Proses kartu kredit");
    }

    @Override
    public void verifikasi() {
        System.out.println("Verifikasi kartu kredit");
    }
}
```

Kode Program 7.2: Contoh Interface

7.2.2 Perbandingan Abstract Class vs Interface

Aspek	**Abstract Class**	**Interface**
Instansiasi	Tidak dapat diinstansiasi	Tidak dapat diinstansiasi
Inheritance	Single inheritance	Multiple inheritance of type
State	Dapat memiliki attribute	Hanya constant (public static final)
Method	Abstract dan konkret	Abstract, default, static

Tabel 7.1: Perbandingan Abstract Class dan Interface

Aktivitas Pembelajaran

1. **Diskusi**: Tentukan kapan menggunakan abstract class dan kapan interface.
2. **Implementasi**: Buat interface **`Notifikasi`**. Lalu buat class **`EmailNotifikasi`** dan **`SMSNotifikasi`**.
3. **Studi Kasus**: Rancang abstract class **`Bentuk`** dan implementasi **`Lingkaran`**, **`Segitiga`**.
4. **Review**: Analisis kode yang menggunakan inheritance berlebihan dan usulkan perbaikan dengan interface.

Latihan dan Refleksi

1. Jelaskan perbedaan utama antara abstract class dan interface.
2. Buat abstract class **`Transportasi`** dan subclass **`Kereta`**, **`Bus`**.
3. Buat interface **`Diskon`** dan implementasikan pada class **`Produk`**.
4. Jelaskan manfaat default method pada interface.
5. **Refleksi**: Bagaimana Anda menentukan pilihan desain antara abstract class dan interface?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 3.2

A. Pilihan Ganda

1. Interface di Java dapat diimplementasikan oleh:
   (a) Satu class saja
   (b) Banyak class
   (c) Hanya abstract class
   (d) Hanya class final
2. Abstract class dapat memiliki:
   (a) Hanya method abstract
   (b) Hanya method static
   (c) Method abstract dan method konkret
   (d) Hanya constructor private

B. Tugas Praktik

- Rancang interface **`Layanan`** dan dua implementasi berbeda. Tunjukkan penggunaan polymorphism.

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya memahami konsep abstract class

☐ Saya memahami konsep interface

☐ Saya dapat memilih antara abstract class dan interface

☐ Saya dapat mengimplementasikan interface di Java

☐ Saya dapat menggunakan polymorphism dengan abstract class dan interface

Rangkuman

Abstract class menyediakan dasar perilaku dan state bersama, sedangkan interface menyediakan kontrak untuk perilaku [5]. Keduanya mendukung polymorphism dan desain yang fleksibel.

Kata Kunci: ***Abstract Class***, ***Interface***, ***implements***, ***default method***, ***polymorphism***

Bab 8

Perancangan dengan UML

8.1 Use Case dan Class Diagram

Sub-CPMK yang Dicakup dalam Bab Ini:

- Sub-CPMK 2.1: Membuat use case diagram untuk sistem sederhana
- Sub-CPMK 2.2: Merancang class diagram dengan relasi yang tepat

8.1.1 Unified Modeling Language (UML)

Unified Modeling Language (UML) adalah bahasa standar untuk memvisualisasikan, menspesifikasikan, membangun, dan mendokumentasikan sistem perangkat lunak [6].

8.1.2 Use Case Diagram

Use case diagram menggambarkan interaksi aktor dengan sistem pada level fungsional.

Komponen utama:

- **Actor**: pihak yang berinteraksi dengan sistem
- **Use case**: layanan yang disediakan sistem
- **Boundary system**: batasan sistem

8.1.3 Class Diagram

Class diagram menggambarkan struktur statis sistem: class, attribute, method, dan relasi.

Relasi utama:

- **Association**: relasi umum antar class
- **Aggregation**: relasi has-a yang lemah
- **Composition**: relasi has-a yang kuat
- **Inheritance**: relasi is-a

Catatan

Class diagram yang baik membantu tim memahami struktur sistem sebelum implementasi kode.

8.2 Sequence Diagram dan Studi Kasus

8.2.1 Sequence Diagram

Sequence diagram menggambarkan alur komunikasi antar objek dalam urutan waktu.

Elemen penting:

- **Lifeline**: objek yang terlibat
- **Message**: pesan yang dikirim antar objek
- **Activation**: periode eksekusi suatu objek

8.2.2 Contoh Studi Kasus

Studi kasus peminjaman buku perpustakaan:

1. Mahasiswa memilih buku
2. Sistem mengecek ketersediaan
3. Sistem mencatat transaksi
4. Sistem memperbarui status buku

Aktivitas Pembelajaran

1. **Use Case**: Buat use case diagram untuk sistem pemesanan makanan online.
2. **Class Diagram**: Rancang class diagram untuk sistem perpustakaan dengan minimal 5 class.
3. **Sequence Diagram**: Buat sequence diagram untuk proses login aplikasi.

4. **Review**: Bandingkan class diagram Anda dengan rekan dan diskusikan perbaikannya.

Latihan dan Refleksi

1. Jelaskan perbedaan use case diagram dan class diagram.
2. Sebutkan minimal 3 relasi pada class diagram dan jelaskan masing-masing.
3. Buat use case diagram sederhana untuk sistem parkir kampus.
4. Buat sequence diagram untuk proses checkout e-commerce.
5. **Refleksi**: Bagian mana dari UML yang paling menantang bagi Anda?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 2.1 dan 2.2

A. Pilihan Ganda

1. Relasi *composition* pada class diagram menunjukkan:
   (a) Relasi is-a
   (b) Relasi has-a yang kuat
   (c) Relasi dependensi sementara
   (d) Relasi tanpa kepemilikan
2. Sequence diagram berfokus pada:
   (a) Struktur statis sistem
   (b) Urutan interaksi antar objek
   (c) Penjelasan kebutuhan pengguna
   (d) Deskripsi database

B. Tugas Praktik

- Rancang use case diagram dan class diagram untuk sistem peminjaman alat laboratorium.
- Tambahkan sequence diagram untuk proses peminjaman.

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya dapat membuat use case diagram untuk sistem sederhana

☐ Saya dapat merancang class diagram dengan relasi yang tepat

☐ Saya memahami fungsi sequence diagram

☐ Saya dapat menerjemahkan kebutuhan ke UML

☐ Saya mampu membaca dan meninjau diagram UML

Rangkuman

UML membantu memodelkan sistem secara visual sesuai dengan standar industri [6].

Use case diagram memetakan kebutuhan, class diagram menggambarkan struktur, dan sequence diagram menunjukkan alur interaksi.

Kata Kunci: ***UML***, ***Use Case***, ***Class Diagram***, ***Sequence Diagram***, ***Association***

Bab 9

Exception Handling dan Keandalan Program

9.1 Konsep Exception dan Try-Catch

Sub-CPMK yang Dicakup dalam Bab Ini:

- Sub-CPMK 3.1: Mengimplementasikan exception handling dalam program Java

Konsep Penting

Exception handling adalah mekanisme untuk menangani kondisi tidak terduga selama runtime, memastikan program tetap berjalan stabil [5]. Exception handling membantu program tetap stabil dan memberikan pesan kesalahan yang jelas.

9.1.1 Jenis Exception

- **Checked Exception**: wajib ditangani (contoh: `IOException`)
- **Unchecked Exception**: turunan `RuntimeException`
- **Error**: kesalahan serius yang biasanya tidak ditangani

9.1.2 Struktur Try-Catch

```
1 public class DemoException {
2     public static void main(String[] args) {
3         try {
4             int hasil = 10 / 0;
5             System.out.println(hasil);
```

```
        } catch (ArithmeticException e) {
            System.out.println("Terjadi kesalahan: " + e.
               getMessage());
        }
    }
}
```

Kode Program 9.1: Try-Catch Sederhana

9.2 Custom Exception dan Best Practice

9.2.1 Membuat Custom Exception

Custom exception digunakan untuk mewakili kesalahan yang spesifik pada domain aplikasi.

```
class SaldoTidakCukupException extends Exception {
    public SaldoTidakCukupException(String pesan) {
        super(pesan);
    }
}

class Rekening {
    private double saldo;

    public void tarik(double jumlah) throws
       SaldoTidakCukupException {
        if (jumlah > saldo) {
            throw new SaldoTidakCukupException("Saldo tidak
               mencukupi");
        }
        saldo -= jumlah;
    }
}
```

Kode Program 9.2: Custom Exception

9.2.2 Finally dan Try-with-Resources

Gunakan `finally` untuk memastikan resource ditutup. Di Java modern, gunakan try-with-resources.

Aktivitas Pembelajaran

1. **Eksperimen**: Buat program yang memicu `NumberFormatException` dan tangani dengan try-catch.

2. **Custom Exception**: Buat exception `StokHabisException` pada sistem inventori.

3. **Diskusi**: Kapan sebaiknya melempar exception dan kapan mengembalikan nilai default?

4. **Refactoring**: Ganti penanganan error manual dengan try-with-resources pada contoh file I/O.

Latihan dan Refleksi

1. Jelaskan perbedaan checked dan unchecked exception.

2. Buat program input angka yang aman dari kesalahan format input.

3. Buat custom exception untuk validasi umur minimal 17 tahun.

4. Jelaskan kapan `finally` tetap dieksekusi.

5. **Refleksi**: Bagaimana exception handling membantu kualitas aplikasi Anda?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 3.1

A. Pilihan Ganda

1. Exception yang wajib ditangani adalah:
   (a) Checked exception
   (b) Runtime exception
   (c) Error
   (d) Logical error

2. Try-with-resources digunakan untuk:
   (a) Menghindari overloading
   (b) Menutup resource secara otomatis
   (c) Menghapus object
   (d) Mempercepat program

B. Tugas Praktik

- Buat class `Login` yang melempar exception jika password salah tiga kali.

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya memahami konsep exception dan error

☐ Saya dapat menggunakan try-catch dengan benar

☐ Saya mampu membuat custom exception

☐ Saya memahami penggunaan finally dan try-with-resources

☐ Saya dapat menentukan kapan melempar exception

Rangkuman

Exception handling menjaga program tetap stabil saat terjadi kesalahan sesuai spesifikasi Java [8]. Dengan custom exception dan try-with-resources, aplikasi menjadi lebih robust dan mudah dipelihara.

Kata Kunci: ***Exception***, ***try-catch***, ***checked***, ***unchecked***, ***try-with-resources***

Bab 10

Prinsip SOLID dalam Desain OOP

10.1 SRP, OCP, dan LSP

Sub-CPMK yang Dicakup dalam Bab Ini:

- Sub-CPMK 4.1: Menganalisis code smell dan melakukan refactoring

Konsep Penting

SOLID adalah lima prinsip desain yang membuat perangkat lunak lebih mudah dipahami, fleksibel, dan maintainable [7].

10.1.1 Single Responsibility Principle (SRP)

Satu class sebaiknya memiliki satu alasan untuk berubah.

10.1.2 Open/Closed Principle (OCP)

Class harus terbuka untuk extension tetapi tertutup untuk modification.

10.1.3 Liskov Substitution Principle (LSP)

Objek subclass harus dapat menggantikan objek superclass tanpa merusak perilaku program.

Catatan

Jika class sering berubah karena banyak alasan, itu adalah tanda pelanggaran SRP.

10.2 ISP, DIP, dan Contoh Refactoring

10.2.1 Interface Segregation Principle (ISP)

Gunakan interface kecil dan spesifik daripada interface besar yang memaksa class mengimplementasikan method yang tidak dibutuhkan.

10.2.2 Dependency Inversion Principle (DIP)

High-level module tidak boleh bergantung pada low-level module. Keduanya harus bergantung pada abstraksi.

```
interface Notifier {
    void kirim(String pesan);
}

class EmailNotifier implements Notifier {
    public void kirim(String pesan) {
        System.out.println("Email: " + pesan);
    }
}

class LayananNotifikasi {
    private Notifier notifier;

    public LayananNotifikasi(Notifier notifier) {
        this.notifier = notifier;
    }

    public void kirimPromo() {
        notifier.kirim("Promo baru!");
    }
}
```

Kode Program 10.1: Contoh DIP dengan Interface

Aktivitas Pembelajaran

1. **Identifikasi Code Smell**: Temukan pelanggaran SRP pada class layanan.
2. **Refactoring**: Ubah class yang terlalu besar menjadi beberapa class kecil.
3. **Diskusi**: Contoh pelanggaran LSP pada hierarki class.
4. **Praktik**: Terapkan DIP pada modul pengiriman notifikasi.

Latihan dan Refleksi

1. Jelaskan masing-masing prinsip SOLID dengan contoh singkat.
2. Berikan contoh pelanggaran OCP dan cara memperbaikinya.
3. Mengapa ISP penting saat mendesain API?
4. Buat desain sederhana yang menerapkan DIP untuk layanan pembayaran.
5. **Refleksi**: Prinsip SOLID mana yang paling sulit Anda terapkan?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 4.1

A. Pilihan Ganda

1. Prinsip yang menyatakan class memiliki satu alasan untuk berubah adalah:
   (a) SRP
   (b) OCP
   (c) LSP
   (d) DIP
2. DIP menekankan ketergantungan pada:
   (a) Implementasi konkret
   (b) Abstraksi
   (c) Class final
   (d) Static method

B. Tugas Praktik

- Refactor class `OrderService` agar mengikuti SRP dan OCP.

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya memahami prinsip SRP, OCP, dan LSP

☐ Saya memahami prinsip ISP dan DIP

☐ Saya dapat mengidentifikasi pelanggaran SOLID

☐ Saya dapat melakukan refactoring dasar sesuai SOLID

☐ Saya dapat menerapkan SOLID pada desain sederhana

Rangkuman

Prinsip SOLID membantu mengurangi ketergantungan, meningkatkan modularitas, dan memudahkan perubahan, sebagaimana dijelaskan dalam konsep arsitektur bersih [7]. Dengan menerapkannya, kode menjadi lebih bersih dan tahan terhadap perubahan.

Kata Kunci: ***SOLID***, ***SRP***, ***OCP***, ***LSP***, ***ISP***, ***DIP***

Bab 11

Design Patterns Dasar

11.1 Design Patterns: Creational Patterns

Sub-CPMK yang Dicakup dalam Bab Ini:

- Sub-CPMK 4.2: Mengimplementasikan minimal 3 design patterns (Singleton, Factory, Observer)

Konsep Penting

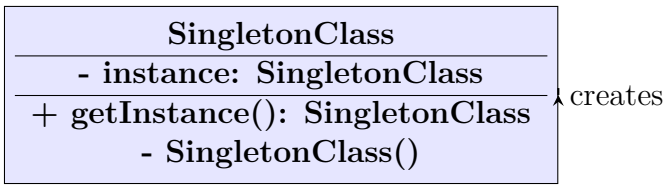
Design Pattern adalah solusi umum untuk masalah yang sering muncul dalam konteks desain perangkat lunak [4]. Pattern bukanlah desain akhir yang dapat langsung diubah menjadi kode, melainkan template tentang cara menyelesaikan masalah.

11.1.1 Singleton Pattern

Menjamin hanya ada satu instance dari class.

```
public class DatabaseConnection {
    private static DatabaseConnection instance;

    private DatabaseConnection() {}

    public static DatabaseConnection getInstance() {
        if (instance == null) {
            instance = new DatabaseConnection();
        }
        return instance;
    }
}
```

Kode Program 11.1: Singleton Pattern



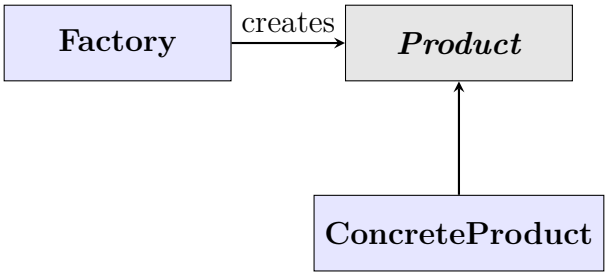
Gambar 11.1: Struktur Singleton Pattern

11.1.2 Factory Pattern

Menyediakan cara membuat objek tanpa mengekspos logika instansiasi.

```
interface Kendaraan {
    void jalan();
}

class Mobil implements Kendaraan {
    public void jalan() {
        System.out.println("Mobil berjalan");
    }
}

class Motor implements Kendaraan {
    public void jalan() {
        System.out.println("Motor berjalan");
    }
}

class KendaraanFactory {
    public static Kendaraan buat(String jenis) {
        if ("mobil".equalsIgnoreCase(jenis)) return new Mobil();
        if ("motor".equalsIgnoreCase(jenis)) return new Motor();
        return null;
    }
}
```

Kode Program 11.2: Factory Pattern



Gambar 11.2: Struktur Factory Pattern

11.2 Behavioral Pattern: Observer

11.2.1 Observer Pattern

Observer digunakan ketika satu objek (subject) perlu memberi tahu banyak objek lain (observer) saat ada perubahan.

```
interface Observer {
    void update(String pesan);
}

class Subscriber implements Observer {
    private String nama;

    public Subscriber(String nama) {
        this.nama = nama;
    }

    public void update(String pesan) {
        System.out.println(nama + " menerima: " + pesan);
    }
}
```

Kode Program 11.3: Observer Pattern Sederhana

11.2.2 Kapan Menggunakan Design Pattern

Gunakan pattern saat:

- Solusi berulang muncul di banyak bagian aplikasi
- Dibutuhkan struktur yang konsisten
- Tim perlu pola komunikasi yang jelas

Aktivitas Pembelajaran

1. **Implementasi Singleton**: Buat class `Logger` sebagai singleton.
2. **Factory**: Buat factory untuk membuat objek pembayaran.
3. **Observer**: Buat sistem notifikasi berita dengan subject dan observer.
4. **Diskusi**: Identifikasi pattern pada framework Java (misalnya Spring).

Latihan dan Refleksi

1. Jelaskan perbedaan Factory dan Singleton.
2. Buat contoh kasus yang cocok menggunakan Observer.
3. Apa resiko jika Singleton digunakan berlebihan?
4. Tambahkan pattern Strategy pada studi kasus diskon.
5. **Refleksi**: Pattern mana yang paling membantu desain Anda?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 4.2

A. Pilihan Ganda

1. Pattern yang menjamin hanya satu instance adalah:
   (a) Factory
   (b) Singleton
   (c) Observer
   (d) Strategy
2. Observer pattern cocok digunakan untuk:
   (a) Membuat banyak objek
   (b) Mengelola notifikasi perubahan
   (c) Menghitung data
   (d) Menyimpan konfigurasi

B. Tugas Praktik

- Implementasikan Singleton, Factory, dan Observer pada satu studi kasus sederhana.

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya memahami konsep design patterns

☐ Saya dapat menerapkan Singleton

☐ Saya dapat menerapkan Factory

☐ Saya dapat menerapkan Observer

☐ Saya dapat memilih pattern sesuai kebutuhan

Rangkuman

Design patterns menyediakan solusi teruji untuk masalah desain umum [4, 3]. Singleton, Factory, dan Observer adalah pola dasar yang sering digunakan di aplikasi nyata.

Kata Kunci: ***Design Pattern***, ***Singleton***, ***Factory***, ***Observer***, ***Strategy***

Bab 12

Collections, Generics, dan File I/O

12.1 Java Collections Framework dan Generics

Sub-CPMK yang Dicakup dalam Bab Ini:

- Sub-CPMK 3.1: Mengimplementasikan struktur data dengan Collections dan Generics

Java Collections Framework adalah arsitektur terpadu untuk merepresentasikan dan memanipulasi koleksi object [5].

12.1.1 Jenis Collections

- **List**: urutan terjaga, boleh duplikat (ArrayList, LinkedList)
- **Set**: tidak boleh duplikat (HashSet, TreeSet)
- **Map**: pasangan key-value (HashMap, TreeMap)
- **Queue**: antrian (PriorityQueue)

12.1.2 Generics

Generics membuat koleksi lebih aman terhadap tipe data dan mengurangi casting.

```
import java.util.ArrayList;
import java.util.List;

public class DemoList {
    public static void main(String[] args) {
        List<String> nama = new ArrayList<>();
        nama.add("Ayu");
        nama.add("Bima");
```

```
        for (String n : nama) {
            System.out.println(n);
        }
    }
}
```

Kode Program 12.1: Contoh Collections dan Generics

12.2 File I/O dan Serialization

12.2.1 Membaca dan Menulis File

File I/O memungkinkan aplikasi menyimpan data secara permanen.

```
import java.io.FileWriter;
import java.io.IOException;

public class TulisFile {
    public static void main(String[] args) throws IOException {
        try (FileWriter writer = new FileWriter("catatan.txt"))
            {
            writer.write("Belajar File I/O");
        }
    }
}
```

Kode Program 12.2: Contoh Menulis File

12.2.2 Serialization

Serialization mengubah objek menjadi byte stream agar dapat disimpan atau dikirim.

Aktivitas Pembelajaran

1. **Collections**: Buat program manajemen daftar tugas dengan ArrayList.
2. **Map**: Simpan pasangan NIM dan nama mahasiswa menggunakan HashMap.
3. **File I/O**: Simpan data mahasiswa ke file teks dan baca kembali.
4. **Serialization**: Simpan objek `Mahasiswa` ke file menggunakan `ObjectOutputStream`.

Latihan dan Refleksi

1. Jelaskan perbedaan List, Set, dan Map.
2. Buat contoh penggunaan generics pada class `Kotak<T>`.
3. Buat program yang membaca file CSV sederhana.
4. Jelaskan manfaat serialization dan kapan sebaiknya digunakan.
5. **Refleksi**: Bagaimana Collections membantu Anda menulis kode yang lebih rapi?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 3.1

A. Pilihan Ganda

1. Struktur data yang tidak mengizinkan duplikat adalah:
   (a) List
   (b) Set
   (c) Map
   (d) Queue
2. Try-with-resources berguna untuk:
   (a) Menghapus file
   (b) Menutup resource otomatis
   (c) Mengubah format file
   (d) Mempercepat loop

B. Tugas Praktik

- Buat aplikasi pencatatan transaksi sederhana menggunakan List dan file teks.

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya dapat membedakan List, Set, dan Map

☐ Saya dapat menggunakan generics untuk keamanan tipe

☐ Saya dapat membaca dan menulis file sederhana

☐ Saya memahami konsep serialization

☐ Saya dapat memilih koleksi yang sesuai dengan kebutuhan

Rangkuman

Collections memudahkan pengelolaan data dinamis, generics meningkatkan keamanan tipe, dan file I/O memungkinkan data disimpan secara permanen sesuai standar Java [8]. Serialization membantu penyimpanan objek secara langsung.

Kata Kunci: ***Collections***, ***Generics***, ***List***, ***Map***, ***File I/O***, ***Serialization***

Bab 13

Unit Testing, TDD, dan Refactoring

13.1 Unit Testing dan Siklus TDD

Sub-CPMK yang Dicakup dalam Bab Ini:

- Sub-CPMK 4.1: Menganalisis code smell dan melakukan refactoring

13.1.1 Konsep Unit Testing

Unit testing memastikan setiap method bekerja sesuai harapan. Pengujian yang baik mempercepat deteksi bug.

13.1.2 Test-Driven Development (TDD)

Siklus TDD: **Red** (tulis test gagal), **Green** (buat test lulus), **Refactor** (perbaiki struktur).

Refactoring adalah proses mengubah sistem perangkat lunak sedemikian rupa sehingga tidak mengubah perilaku eksternal kode namun memperbaiki struktur internalnya [2].

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;

class KalkulatorTest {
    @Test
    void testTambah() {
        Kalkulator k = new Kalkulator();
        assertEquals(5, k.tambah(2, 3));
    }
}
```

Kode Program 13.1: Contoh Unit Test JUnit

13.2 Code Quality dan Refactoring

13.2.1 Code Smell yang Umum

Beberapa contoh code smell:

- Method terlalu panjang
- Duplikasi kode
- Nama variabel tidak jelas
- Kelas melakukan terlalu banyak tugas

13.2.2 Teknik Refactoring

Refactoring adalah proses memperbaiki struktur kode tanpa mengubah perilaku.

- Extract Method
- Rename Variable
- Replace Magic Number with Constant
- Move Method

Aktivitas Pembelajaran

1. **Unit Test**: Buat unit test untuk class `RekeningBank`.
2. **TDD**: Terapkan siklus Red-Green-Refactor pada fitur login sederhana.
3. **Refactoring**: Perbaiki class dengan method yang terlalu panjang.
4. **Diskusi**: Identifikasi code smell pada proyek mini.

Latihan dan Refleksi

1. Jelaskan manfaat unit testing bagi tim pengembang.
2. Buat test case untuk method `hitungDiskon()` pada class **`Produk`**.
3. Sebutkan minimal 3 contoh refactoring dan kapan digunakan.
4. Bagaimana TDD membantu desain yang lebih baik?
5. **Refleksi**: Apa code smell yang paling sering Anda temui?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 4.1

A. Pilihan Ganda

1. Siklus TDD yang benar adalah:
   (a) Green-Red-Refactor
   (b) Red-Green-Refactor
   (c) Refactor-Red-Green
   (d) Red-Refactor-Green
2. Refactoring bertujuan untuk:
   (a) Mengubah perilaku program
   (b) Memperbaiki struktur tanpa mengubah perilaku
   (c) Menambah fitur baru
   (d) Menghapus semua test

B. Tugas Praktik

- Lakukan refactoring pada class **`Order`** agar mengikuti SRP dan tambahkan unit test.

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya memahami konsep unit testing

☐ Saya memahami siklus TDD

☐ Saya dapat mengidentifikasi code smell

☐ Saya dapat melakukan refactoring dasar

☐ Saya dapat menulis test untuk method penting

Rangkuman

Unit testing dan TDD meningkatkan kualitas software. Refactoring membantu menjaga struktur kode tetap bersih dan mudah dipelihara, sebagaimana didokumentasikan dalam katalog refactoring [2].

Kata Kunci: ***Unit Testing***, ***TDD***, ***Refactoring***, ***Code Smell***, ***JUnit***

Bab 14

Evaluasi, Refleksi, dan Integrasi Kompetensi

Bab ini berisi asesmen komprehensif untuk mengukur pencapaian seluruh CPMK yang telah dipelajari sepanjang mata kuliah Pemrograman Berorientasi Objek.

14.1 Asesmen Akhir Komprehensif

14.1.1 Petunjuk Umum

Asesmen ini dirancang untuk mengukur pencapaian CPMK-1, CPMK-2, CPMK-3, dan CPMK-4 secara menyeluruh [6]. Kerjakan dengan jujur dan mandiri.

Alokasi Waktu:

- Bagian A (Pilihan Ganda): 30 menit
- Bagian B (Essay): 45 menit
- Bagian C (Analisis Kode): 45 menit
- Bagian D (Coding Challenge): 90 menit

14.1.2 Bagian A: Pilihan Ganda (CPMK-1)

Petunjuk: Pilih satu jawaban yang paling tepat.

1. Manakah yang BUKAN merupakan pilar OOP?
   (a) Abstraksi
   (b) Enkapsulasi
   (c) Kompilasi
   (d) Polimorfisme

2. Dalam Java, keyword untuk mewarisi class adalah:
   (a) inherits
   (b) extends
   (c) implements
   (d) super
3. Method overriding terjadi ketika:
   (a) Dua method dalam class yang sama memiliki nama sama
   (b) Subclass mendefinisikan ulang method dari superclass
   (c) Method memiliki parameter yang berbeda
   (d) Method dipanggil berkali-kali
4. Interface dalam Java:
   (a) Dapat memiliki instance variables
   (b) Semua methods harus abstract (sebelum Java 8)
   (c) Dapat di-instantiate
   (d) Hanya boleh memiliki satu method
5. Prinsip SOLID yang menyatakan "class harus terbuka untuk extension tapi tertutup untuk modification" adalah:
   (a) Single Responsibility Principle
   (b) Open/Closed Principle
   (c) Liskov Substitution Principle
   (d) Dependency Inversion Principle
6. Design pattern yang memastikan hanya ada satu instance dari class adalah:
   (a) Factory Pattern
   (b) Singleton Pattern
   (c) Observer Pattern
   (d) Strategy Pattern
7. Dalam Java Collections, struktur data yang tidak mengizinkan duplikat adalah:
   (a) List
   (b) Set
   (c) Map
   (d) Queue
8. Exception yang harus di-handle dengan try-catch disebut:
   (a) RuntimeException

(b) Checked Exception

(c) Unchecked Exception

(d) Error

9. Dalam UML Class Diagram, relasi "has-a" digambarkan dengan:

   (a) Generalization

   (b) Association/Aggregation/Composition

   (c) Dependency

   (d) Realization

10. Kelas yang digunakan untuk menulis objek ke file secara serialization adalah:

    (a) FileWriter

    (b) ObjectOutputStream

    (c) BufferedReader

    (d) Scanner

11. Annotation untuk menandai method sebagai test case dalam JUnit adalah:

    (a) @Test

    (b) @TestCase

    (c) @Unit

    (d) @Assert

12. Refactoring yang memecah method panjang menjadi beberapa method kecil disebut:

    (a) Extract Method

    (b) Inline Method

    (c) Replace Conditionals with Polymorphism

    (d) Move Method

14.1.3 Bagian B: Essay (CPMK-1, CPMK-2)

Petunjuk: Jawab dengan jelas dan lengkap.

1. Jelaskan perbedaan antara abstract class dan interface dalam Java! Kapan sebaiknya menggunakan abstract class dan kapan menggunakan interface? Berikan contoh kasus untuk masing-masing.

2. Jelaskan konsep polymorphism dalam OOP! Berikan contoh kode Java yang mendemonstrasikan compile-time polymorphism dan runtime polymorphism.

3. Jelaskan 5 prinsip SOLID! Pilih salah satu prinsip dan berikan contoh kode yang melanggar prinsip tersebut, kemudian perbaiki kode tersebut.

4. Gambarkan Class Diagram untuk sistem perpustakaan sederhana yang memiliki minimal 5 class dengan relasi yang tepat (association, aggregation, composition, inheritance).

5. Jelaskan konsep Test-Driven Development (TDD)! Apa keuntungan menggunakan TDD dalam pengembangan software?

6. Jelaskan contoh refactoring sederhana yang dapat Anda lakukan untuk memperbaiki code smell "Long Method".

14.1.4 Bagian C: Analisis Kode (CPMK-3, CPMK-4)

Petunjuk: Analisis kode berikut dan jawab pertanyaan.

```
public class BankAccount {
    public String accountNumber;
    public double balance;
    public String ownerName;

    public void deposit(double amount) {
        balance = balance + amount;
    }

    public void withdraw(double amount) {
        balance = balance - amount;
    }

    public double getBalance() {
        return balance;
    }
}

public class SavingsAccount extends BankAccount {
    public double interestRate;

    public void addInterest() {
        balance = balance + (balance * interestRate);
    }
}
```

Kode Program 14.1: Kode untuk Dianalisis

Pertanyaan:

1. Identifikasi minimal 5 masalah dalam kode di atas terkait dengan prinsip OOP dan best practices.

2. Perbaiki kode tersebut dengan menerapkan encapsulation yang tepat.

3. Tambahkan validasi yang diperlukan pada method `deposit()` dan `withdraw()`.

4. Implementasikan constructor yang sesuai untuk kedua class.

5. Apakah kode ini melanggar prinsip SOLID? Jelaskan!

14.1.5 Bagian D: Coding Challenge (CPMK-2, CPMK-3, 4)

Petunjuk: Implementasikan sistem berikut dengan menerapkan semua konsep OOP yang telah dipelajari.

Studi Kasus: Sistem Manajemen Toko Buku Online

Buat sistem manajemen toko buku online dengan requirements berikut:

Requirements:

1. **Class `Buku`**:
   - Attributes: isbn, judul, penulis, penerbit, tahunTerbit, harga, stok
   - Methods: getInfo(), updateStok(), hitungDiskon()
2. **Class `BukuFisik`** (extends **`Buku`**):
   - Attribute tambahan: berat, dimensi
   - Method: hitungOngkir()
3. **Class `Ebook`** (extends **`Buku`**):
   - Attribute tambahan: ukuranFile, format
   - Method: download()
4. **Class `Pelanggan`**:
   - Attributes: idPelanggan, nama, email, alamat
   - Methods: register(), updateProfile()
5. **Class `Keranjang`**:
   - Attributes: daftarBuku (ArrayList), pelanggan
   - Methods: tambahBuku(), hapusBuku(), hitungTotal(), checkout()
6. **Interface `Pembayaran`**:
   - Methods: prosesPembayaran(), verifikasiPembayaran()
7. **Class `PembayaranKartuKredit`** dan **`PembayaranTransfer`** (implements **`Pembayaran`**)

Kriteria Penilaian:

- Penerapan encapsulation (private attributes, getter/setter)
- Penggunaan inheritance yang tepat
- Implementasi polymorphism
- Penggunaan interface

- Validasi data yang sesuai
- Exception handling
- Code quality (naming, comments, structure)
- Unit testing untuk minimal 3 methods

14.2 Rubrik Penilaian Komprehensif

14.2.1 Rubrik untuk Coding Challenge

Kriteria	**Excellent (4)**	**Good (3)**	**Fair (2)**	**Poor (1)**
Encapsulation	Semua attributes private dengan getter/setter yang tepat	Sebagian besar private	Beberapa public	Semua public
Inheritance	Hierarki class tepat, code reuse optimal	Inheritance digunakan dengan baik	Inheritance kurang optimal	Tidak menggunakan inheritance
Polymorphism	Men-de-mon-stra-si-kan compile-time dan runtime polymorphism	Menggunakan salah satu jenis polymorphism	Polymorphism minimal	Tidak ada polymorphism
Interface	Interface digunakan dengan tepat	Interface digunakan tapi kurang optimal	Interface ada tapi tidak efektif	Tidak menggunakan interface
Exception Handling	Comprehensive error handling	Error handling untuk kasus utama	Minimal error handling	Tidak ada error handling
Code Quality	Clean code, well-documented, follows conventions	Good structure, adequate comments	Basic structure, minimal comments	Poor structure, no comments
Unit Testing	Comprehensive tests, good coverage	Tests untuk fungsi utama	Minimal testing	No testing

Tabel 14.1: Rubrik Penilaian Coding Challenge

Komponen	**Bobot**
Bagian A: Pilihan Ganda	20%
Bagian B: Essay	20%
Bagian C: Analisis Kode	20%
Bagian D: Coding Challenge	40%
Total	**100%**

Tabel 14.2: Matriks Bobot Penilaian Akhir

14.2.2 Bobot Penilaian

14.3 Tinjauan Pencapaian Kompetensi Secara Menyeluruh

14.3.1 Pemetaan Asesmen ke CPMK

CPMK	**Diukur Melalui**
CPMK-1: Memahami konsep dasar OOP	Pilihan Ganda (1-4), Essay (1-2)
CPMK-2: Merancang solusi dengan UML	Essay (4), Coding Challenge (Design)
CPMK-3: Mengimplementasikan OOP dengan SOLID	Analisis Kode, Coding Challenge (Implementation)
CPMK-4: Menganalisis dan mengevaluasi kualitas kode	Analisis Kode, Coding Challenge (Quality)

Tabel 14.3: Pemetaan Komponen Asesmen ke CPMK

14.3.2 Self-Assessment Checklist

Sebelum mengerjakan asesmen akhir, pastikan Anda telah menguasai:

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Konsep dasar OOP (class, object, encapsulation, inheritance, polymorphism)

☐ Perbedaan abstract class dan interface

☐ Cara membuat dan menggunakan UML diagrams

☐ Implementasi inheritance dan polymorphism dalam Java

☐ Exception handling yang tepat

☐ Prinsip-prinsip SOLID

☐ Design patterns dasar (Singleton, Factory, Observer)

☐ Java Collections Framework

☐ File I/O dan serialization

☐ Unit testing dengan JUnit

☐ Refactoring dan identifikasi code smell

☐ Best practices dalam penulisan kode Java

14.4 Rekomendasi Tindak Lanjut Pembelajaran

14.4.1 Untuk Mahasiswa yang Sudah Menguasai Semua CPMK

Jika Anda telah menguasai semua materi dengan baik, pertimbangkan untuk:

1. **Proyek Portofolio**: Buat aplikasi Java lengkap yang mendemonstrasikan semua konsep OOP
2. **Eksplorasi Framework**: Pelajari Spring Framework atau Jakarta EE
3. **Design Patterns Lanjut**: Pelajari 23 GoF Design Patterns secara mendalam
4. **Clean Code**: Baca buku "Clean Code" oleh Robert C. Martin
5. **Kontribusi Open Source**: Berkontribusi pada proyek Java open source
6. **Sertifikasi**: Persiapkan Oracle Certified Professional Java Programmer

14.4.2 Untuk Mahasiswa yang Masih Perlu Perbaikan

Jika ada CPMK yang belum dikuasai dengan baik:

1. **Review Materi**: Baca ulang bab terkait dengan lebih teliti
2. **Praktik Lebih Banyak**: Kerjakan lebih banyak latihan coding
3. **Konsultasi**: Diskusikan dengan dosen atau asisten
4. **Study Group**: Bentuk kelompok belajar dengan teman
5. **Online Resources**: Manfaatkan tutorial online (Coursera, Udemy, YouTube)
6. **Coding Practice**: Gunakan platform seperti HackerRank, LeetCode untuk latihan

14.4.3 Sumber Belajar Tambahan

Buku:

- "Effective Java" - Joshua Bloch [1]
- "Head First Design Patterns" - Freeman & Robson [3]
- "Clean Code" - Robert C. Martin [7]
- "Refactoring" - Martin Fowler [2]
- "Design Patterns" - Gamma et al. [4]

Online Courses:

- Java Programming and Software Engineering Fundamentals (Coursera)
- Object Oriented Programming in Java (Udemy)
- Design Patterns in Java (Pluralsight)

Practice Platforms:

- HackerRank (Java track)
- LeetCode (OOP problems)
- Codewars
- Exercism (Java track)

Rangkuman

Selamat! Anda telah menyelesaikan perjalanan pembelajaran Pemrograman Berorientasi Objek. Dari memahami konsep dasar hingga menerapkan design patterns dan best practices, Anda telah membangun fondasi yang kuat dalam pengembangan software modern.

Apa yang Telah Anda Pelajari:

- Paradigma OOP dan 4 pilar utama
- Class, Object, Constructor, dan Method
- Encapsulation dan Information Hiding
- Inheritance dan Polymorphism
- Abstract Class dan Interface
- UML untuk perancangan sistem
- Exception Handling

- Prinsip SOLID
- Design Patterns
- Collections dan Generics
- File I/O dan Serialization
- Unit Testing dan TDD
- Refactoring dan Code Quality

Langkah Selanjutnya: Terus praktik, buat proyek nyata, dan jangan berhenti belajar. OOP adalah fondasi, tetapi masih banyak yang bisa dipelajari dalam dunia pengembangan software!

Lampiran

Lampiran A: Rubrik Penilaian Tugas Praktik

Kriteria	**Sangat Baik**	**Baik**	**Perlu Perbaikan**
Desain Class	Struktur jelas, relasi tepat	Struktur cukup jelas	Struktur belum tepat
Encapsulation	Semua data terlindungi	Sebagian data terlindungi	Banyak data terbuka
Implementasi OOP	Inheritance dan polymorphism tepat	Ada kekurangan kecil	Banyak kesalahan konsep
Code Quality	Naming rapi, tidak ada duplikasi	Ada sedikit duplikasi	Banyak duplikasi dan nama tidak jelas
Testing	Test lengkap dan relevan	Test sebagian	Tidak ada test

Tabel 14.4: Rubrik Penilaian Tugas Praktik

Lampiran B: Contoh Template Laporan Tugas

1. Judul dan Identitas Mahasiswa
2. Deskripsi Masalah
3. Desain Class dan UML
4. Implementasi (cuplikan kode penting)
5. Hasil Pengujian
6. Refleksi dan Kesimpulan

Lampiran C: Glosarium Istilah OOP

- **Class**: Blueprint untuk membuat objek
- **Object**: Instance dari class
- **Encapsulation**: Penyembunyian data melalui akses terkontrol

- **Inheritance**: Pewarisan atribut dan method
- **Polymorphism**: Satu interface untuk banyak bentuk
- **Abstraction**: Menyembunyikan detail implementasi
- **Interface**: Kontrak perilaku tanpa implementasi
- **Refactoring**: Perbaikan struktur kode tanpa mengubah perilaku

Lampiran D: Referensi Tambahan

- Oracle Java Documentation: https://docs.oracle.com/javase/
- JUnit 5 User Guide: https://junit.org/junit5/docs/current/user-guide/
- Refactoring Catalog: https://refactoring.com/catalog/

Daftar Pustaka

[1] Joshua Bloch. *Effective Java*. Addison-Wesley Professional, 3rd edition, 2018.

[2] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 2nd edition, 2018.

[3] Eric Freeman and Elisabeth Robson. *Head First Design Patterns*. O'Reilly Media, 2nd edition, 2020.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[5] Cay S. Horstmann. *Core Java Volume I–Fundamentals*. Prentice Hall, 11th edition, 2019.

[6] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*. Prentice Hall, 3rd edition, 2004.

[7] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.

[8] Oracle. The java tutorials. https://docs.oracle.com/javase/tutorial/, 2024.