

## **LAPORAN UJIAN AKHIR SEMESTER**

### **“APLIKASI GAME BLACKJACK”**

Laporan ini disusun untuk memenuhi Tugas Ujian Akhir Semester (UAS) Mata Kuliah

Pemrograman Berorientasi Objek

Dosen Pengampu : Taufik Ridwan, S.T., M.T.



#### **Kelas 4C**

Disusun oleh Kelompok 5:

Dwi Septian	(2310631250048)
Fachry Firdaus Avicenna	(2310631250014)
Galih Yusuf Ghifari	(2310631250059)
Muhammad Alpha Athallah Nugroho	(2310631250067)

**PROGRAM STUDI SISTEM INFORMASI**

**FAKULTAS ILMU KOMPUTER**

**UNIVERSITAS SINGAPERBANGSA KARAWANG**

**2025**

## DAFTAR ISI

DAFTAR ISI.....	i
BAB I PENDAHULUAN.....	1
1.1 Latar Belakang.....	1
1.2 Rumusan Masalah.....	1
1.3 Tujuan.....	2
BAB II PEMBAHASAN.....	3
2.1 Fitur-Fitur Sistem.....	3
2.2 Konsep OOP yang digunakan.....	4
2.3 Perancangan UML.....	5
2.4 Implementasi Program.....	8
2.5 Pengujian Program.....	20
BAB III PENUTUP.....	22
3.1 Kesimpulan.....	22
3.2 Saran.....	22

# **BAB I**

## **PENDAHULUAN**

### **1.1 Latar Belakang**

Perkembangan teknologi informasi telah membawa perubahan signifikan dalam berbagai aspek kehidupan, termasuk industri hiburan dan permainan. Permainan digital terus berevolusi, menawarkan pengalaman yang lebih imersif dan interaktif kepada pengguna. Di sisi lain, konsep pemrograman berorientasi objek (OOP) telah menjadi paradigma standar dalam pengembangan perangkat lunak modern karena kemampuannya dalam menyediakan struktur kode yang modular, mudah dipelihara, dan dapat diperluas. Java, sebagai salah satu bahasa pemrograman berorientasi objek yang populer, menawarkan berbagai fitur untuk membangun aplikasi yang kompleks dan berskala, termasuk antarmuka pengguna grafis (GUI) dan konektivitas database.

Proyek ini berfokus pada pengembangan sebuah permainan kartu Blackjack sederhana berbasis Java GUI. Penting untuk digaris bawahi bahwa pemilihan Blackjack sebagai studi kasus ini semata-mata didasarkan pada popularitasnya serta aturan permainannya yang cukup ringkas, sehingga ideal untuk mendemonstrasikan implementasi konsep OOP dalam konteks yang menyenangkan dan mudah dipahami, bukan untuk mempromosikan atau mengajarkan praktik perjudian. Tujuan utama adalah mengemas pembelajaran OOP dengan pendekatan yang lebih menarik melalui sebuah permainan. Permainan ini memiliki potensi besar untuk menerapkan berbagai prinsip OOP secara komprehensif. Tantangan tambahan muncul dalam kebutuhan untuk menyimpan dan melacak statistik pemain (seperti jumlah kemenangan dan kekalahan) secara permanen, sehingga pemain dapat melanjutkan progres mereka di sesi bermain yang berbeda. Hal ini mendorong integrasi sistem persistensi data menggunakan database relasional MySQL, yang diakses melalui Java Database Connectivity (JDBC).

Melalui pengembangan aplikasi ini, diharapkan dapat memberikan pemahaman praktis yang mendalam mengenai penerapan prinsip-prinsip OOP, perancangan GUI yang interaktif, serta implementasi konektivitas database dalam sebuah proyek nyata. Proyek ini juga akan menyoroti pentingnya desain yang baik dalam menciptakan aplikasi yang efisien, mudah dikelola, dan memberikan pengalaman pengguna yang lebih baik.

### **1.2 Rumusan Masalah**

Berdasarkan latar belakang di atas, rumusan masalah yang akan dibahas dalam proyek ini adalah sebagai berikut:

1. Bagaimana cara merancang dan mengimplementasikan struktur kelas menggunakan konsep OOP (Encapsulation, Inheritance, Polymorphism, Abstraction) untuk memodelkan entitas dalam permainan Blackjack (kartu, dek, pemain, dealer)?
2. Bagaimana cara membangun antarmuka pengguna grafis (GUI) yang intuitif dan interaktif untuk permainan Blackjack menggunakan Java Swing?
3. Bagaimana cara mengintegrasikan sistem persistensi data menggunakan database MySQL (melalui JDBC) untuk menyimpan dan memuat statistik kemenangan dan kekalahan pemain?
4. Bagaimana cara mengimplementasikan alur verifikasi pemain (login) yang memungkinkan pemain untuk melanjutkan progres permainan mereka dari sesi sebelumnya?

### **1.3 Tujuan**

Berdasarkan latar belakang di atas, rumusan masalah yang akan dibahas dalam proyek ini adalah sebagai berikut:

1. Bagaimana cara merancang dan mengimplementasikan struktur kelas menggunakan konsep OOP (Encapsulation, Inheritance, Polymorphism, Abstraction) untuk memodelkan entitas dalam permainan Blackjack (kartu, dek, pemain, dealer)?
2. Bagaimana cara membangun antarmuka pengguna grafis (GUI) yang intuitif dan interaktif untuk permainan Blackjack menggunakan Java Swing?
3. Bagaimana cara mengintegrasikan sistem persistensi data menggunakan database MySQL (melalui JDBC) untuk menyimpan dan memuat statistik kemenangan dan kekalahan pemain?
4. Bagaimana cara mengimplementasikan alur verifikasi pemain (login) yang memungkinkan pemain untuk melanjutkan progres permainan mereka dari sesi sebelumnya?

## **BAB II**

### **PEMBAHASAN**

#### **2.1 Fitur-Fitur Sistem**

Aplikasi game Blackjack GUI yang dikembangkan ini menawarkan serangkaian fitur untuk memberikan pengalaman bermain yang interaktif dan personal:

##### **2.1.1 Sistem Login/Registrasi Pemain**

Saat memulai aplikasi, pemain akan disajikan dengan layar login di mana mereka dapat memasukkan nama mereka. Sistem ini cerdas dalam mengelola data pemain: jika nama pemain sudah ada dalam database, data kemenangan dan kekalahan mereka akan dimuat. Jika nama pemain baru, sistem akan mendaftarkan mereka sebagai pemain baru. Penting untuk dicatat bahwa data pemain secara otomatis disimpan atau diperbarui ke database saat aplikasi permainan ditutup, memastikan kemajuan pemain tidak hilang.

##### **2.1.2 Permainan Blackjack Dasar**

Inti dari game ini adalah simulasi Blackjack klasik. Permainan dimulai dengan dek standar 52 kartu yang dikocok sebelum setiap babak baru. Pemain dan Dealer masing-masing menerima dua kartu awal, dengan satu kartu Dealer disembunyikan dari pandangan pemain sampai gilirannya tiba. Pemain memiliki opsi untuk "Hit" (mengambil kartu tambahan) untuk mencoba mendekati total 21 poin, atau "Stand" (berhenti mengambil kartu) jika mereka puas dengan tangan mereka. Poin kartu dihitung secara standar: kartu angka sesuai nilai nominalnya, Jack, Queen, dan King bernilai 10, sementara As secara fleksibel bernilai 11 atau 1 untuk mencegah "bust" (melebihi 21 poin). Dealer memiliki perilaku otomatis untuk "Hit" sampai total poinnya setidaknya 17.

##### **2.1.3 Penentuan Pemenang**

Setelah semua giliran selesai, permainan menentukan pemenang berdasarkan aturan Blackjack. Jika total poin pemain atau dealer melebihi 21, mereka "bust" dan kalah. Kondisi "Blackjack" (tepat 21 poin dengan dua kartu awal) secara otomatis memenangkan ronde. Jika tidak ada yang "bust", tangan dengan total poin tertinggi yang tidak melebihi 21 adalah pemenangnya. Permainan ini juga menangani kasus "seri" (push) dimana pemain dan dealer memiliki total poin yang sama.

##### **2.1.4 Pelacakan Skor Pemain**

Game ini melacak kinerja pemain dari waktu ke waktu dengan mencatat jumlah kemenangan dan kekalahan mereka. Skor ini terus diperbarui dan ditampilkan di

antarmuka permainan, memberikan pemain gambaran yang jelas tentang rekor mereka. Yang terpenting, skor ini disimpan secara persisten dalam database, yang berarti riwayat kemenangan dan kekalahan pemain akan dipertahankan di antara sesi permainan. Proyek pengembangan aplikasi game Blackjack berbasis Java GUI dengan integrasi database MySQL telah berhasil diselesaikan dengan baik dan sesuai dengan tujuan yang dirancang sejak awal. Selama proses pengembangan, berbagai konsep penting dalam Pemrograman Berorientasi Objek (OOP) seperti enkapsulasi, pewarisan, polimorfisme, dan abstraksi dapat diterapkan secara nyata dalam memodelkan elemen-elemen inti permainan, seperti kartu, dek, pemain, dan dealer. Hal ini membantu menciptakan struktur kode yang rapi, fleksibel, dan mudah dikembangkan lebih lanjut.

## **2.2 Konsep OOP yang digunakan**

Pengembangan sistem ini mengadopsi empat pilar utama Pemrograman Berorientasi Objek (OOP) untuk mencapai kode yang modular, efisien, dan mudah dipelihara:

- 2.2.1 Encapsulation (Enkapsulasi): Memastikan bahwa data sensitif suatu objek tersembunyi dari akses eksternal langsung. Contohnya, atribut jenis, nilai, dan poin pada kelas Kartu bersifat *private* dan hanya bisa diakses melalui metode getter (`getJenis()`, `getNilai()`, `getPoin()`). Demikian pula, detail koneksi dan query SQL pada kelas `DatabaseManager` disembunyikan di dalam kelas tersebut, sehingga kelas lain hanya perlu memanggil metode `muatPemain()` atau `simpanAtauPerbaruiPemain()` tanpa perlu tahu implementasi internalnya.
- 2.2.2 Inheritance (Pewarisan): Mekanisme di mana sebuah kelas (subkelas) dapat mewarisi atribut dan perilaku dari kelas lain (superkelas). Kelas `EntitasPermainan` bertindak sebagai superkelas yang mendefinisikan properti umum (`nama`, `tanganKartu`) dan metode (`tambahKartuKeTangan()`, `hitungTotalPoin()`) yang dimiliki oleh setiap partisipan game. Kelas `Pemain` dan `Dealer` kemudian mewarisi dari `EntitasPermainan`, mengurangi duplikasi kode dan membangun hierarki yang logis.
- 2.2.3 Polymorphism (Polimorfisme): Kemampuan objek yang berbeda untuk merespons perintah yang sama dengan cara yang sesuai dengan jenisnya masing-masing. Metode `harusHit()` dideklarasikan sebagai *abstract* di `EntitasPermainan`. Kelas `Pemain` mengimplementasikannya untuk menunggu input dari tombol GUI, sementara kelas `Dealer` mengimplementasikannya dengan logika otomatis (*hit* jika total < 17). Saat `GameBlackjackGUI` memanggil

dealer.harusHit(), Java secara otomatis menjalankan implementasi yang spesifik untuk Dealer.

2.2.4 Abstraction (Abstraksi): Proses menyembunyikan detail implementasi yang tidak perlu dan hanya menampilkan fungsionalitas esensial. Kelas EntitasPermainan adalah abstract class yang menentukan apa yang harus dilakukan (misalnya, harusHit()) tanpa menjelaskan bagaimana hal itu dilakukan. Pengguna DatabaseManager hanya perlu tahu apa yang bisa dilakukan (memuat/menyimpan data) tanpa perlu memahami bagaimana koneksi JDBC atau query SQL bekerja. Ini menyederhanakan kompleksitas sistem.

## 2.3 Perancangan UML

### 2.3.1 Use Case

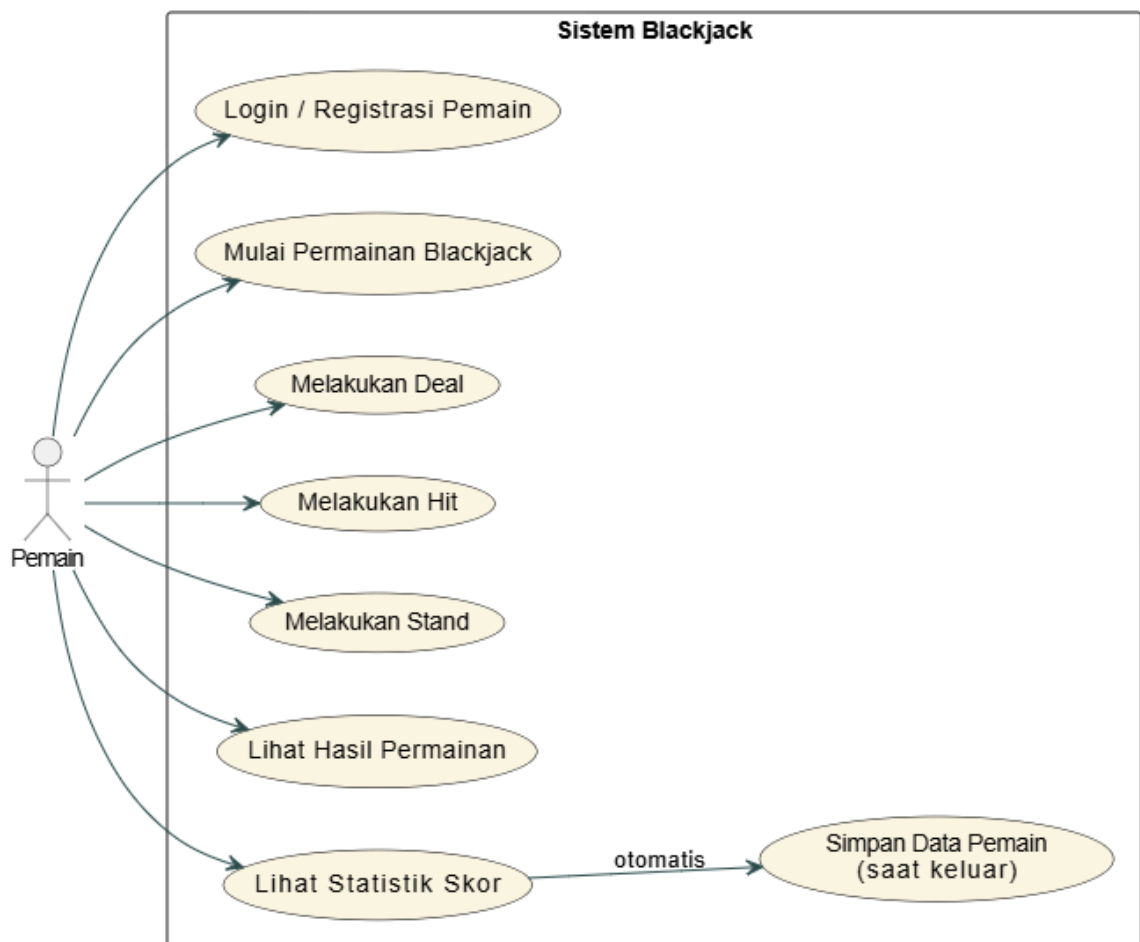


Diagram ini menunjukkan bahwa Pemain adalah aktor utama yang dapat melakukan serangkaian tindakan, dimulai dari Login / Registrasi Pemain untuk mengakses sistem. Setelah masuk, Pemain dapat Mulai Permainan Blackjack, kemudian berinteraksi langsung dengan permainan melalui aksi Melakukan Deal

(untuk memulai putaran), Melakukan Hit (mengambil kartu tambahan), dan Melakukan Stand (berhenti mengambil kartu).

Setelah putaran selesai, Pemain dapat Lihat Hasil Permainan dan juga Lihat Statistik Skor pribadi mereka. Fungsionalitas Simpan Data Pemain (saat keluar) adalah proses otomatis yang terjadi di latar belakang ketika Pemain menyudahi sesi, memastikan semua progres dan statistik tersimpan untuk sesi berikutnya.

### 2.3.2 Class Diagram

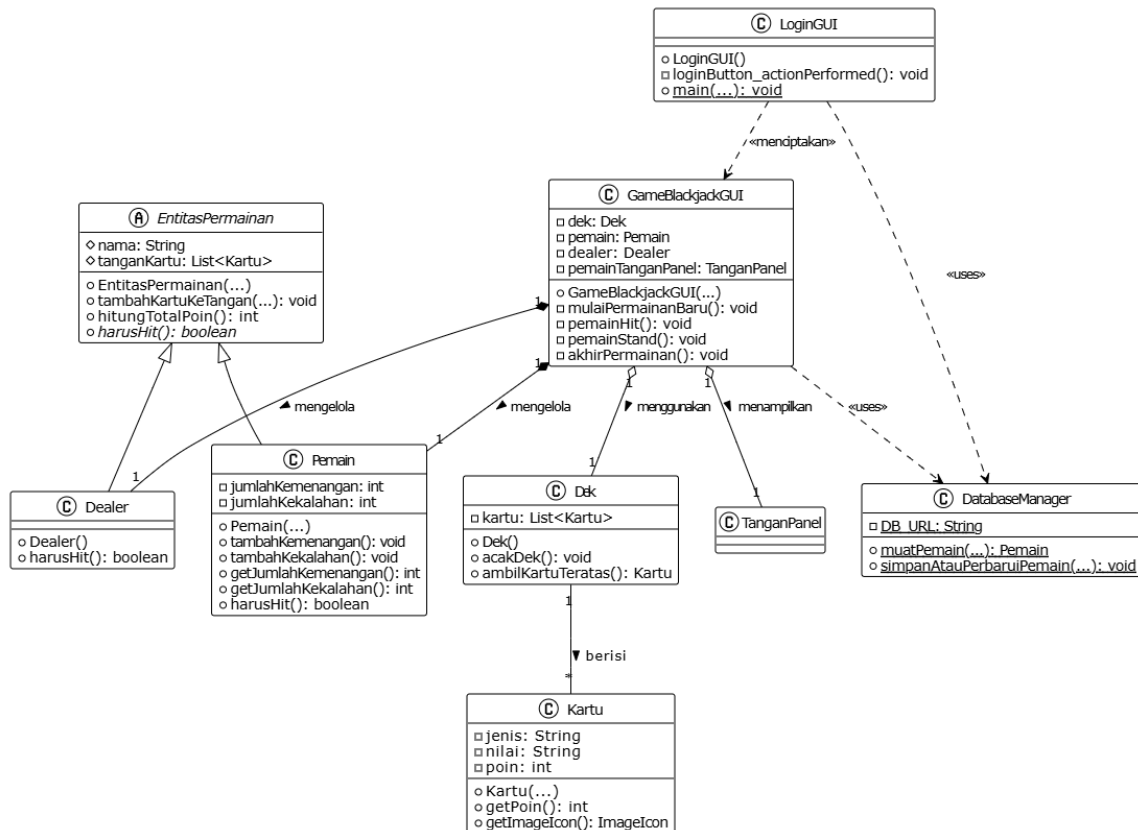


Diagram ini dirancang untuk memberikan gambaran arsitektur sistem secara ringkas dan mudah dipahami:

#### 1. Entitas Permainan Inti:

- Kartu adalah representasi dasar dari setiap kartu dengan jenis, nilai, dan poinnya.
- Dek bertanggung jawab untuk mengelola kumpulan objek Kartu, termasuk mengocok dan membagikannya. Dek memiliki banyak Kartu (asosiasi komposisi).
- EntitasPermainan adalah kelas abstrak yang menjadi fondasi umum untuk semua partisipan dalam permainan. Ia mendefinisikan sifat dan perilaku dasar seperti nama, tangan kartu, dan kemampuan menghitung total poin.



d. Pemain dan Dealer adalah subkelas dari EntitasPermainan (hubungan pewarisan), yang berarti mereka mewarisi semua sifat dasar dan memiliki perilaku khusus (seperti harusHit()) yang diimplementasikan secara berbeda sesuai perannya.

2. Komponen Antarmuka Pengguna (GUI):

- a. KartuPanel bertanggung jawab untuk menampilkan satu objek Kartu secara visual di GUI.
- b. TanganPanel mengelola dan menampilkan koleksi KartuPanel untuk merepresentasikan tangan pemain atau dealer.
- c. GameBlackjackGUI adalah jendela utama permainan. Ia memiliki (o--) dan mengelola (\*--) objek dari Dek, Pemain, dan Dealer, serta menampilkan TanganPanel dan komponen GUI lainnya. Ini adalah pusat orkestrasi permainan.
- d. LoginGUI adalah jendela terpisah yang menjadi titik masuk aplikasi. Ia bertanggung jawab untuk proses autentikasi/registrasi pemain awal sebelum memulai permainan utama.

3. Manajemen Data (Persistensi):

- a. DatabaseManager adalah kelas statis yang khusus menangani semua interaksi dengan database MySQL. Kelas ini bertanggung jawab untuk memuat data pemain (muatPemain()) dan menyimpan/memperbarui data pemain (simpanAtauPerbaruiPemain()).
- b. Hubungan ketergantungan (..> <<uses>>) antara LoginGUI dan GameBlackjackGUI dengan DatabaseManager menunjukkan bahwa kedua kelas GUI ini menggunakan layanan dari DatabaseManager untuk operasi data, tanpa perlu tahu detail implementasi SQL-nya.

Secara keseluruhan, Class Diagram ini menunjukkan bagaimana setiap bagian sistem terdefinisi dengan jelas sebagai objek atau kelas yang memiliki tanggung jawab spesifik, berinteraksi secara terstruktur, dan memanfaatkan prinsip-prinsip OOP untuk mencapai modularitas dan pemeliharaan kode yang lebih baik.

## 2.4 Implementasi Program

### 2.4.1 Pengelolaan Data Pemain (DatabaseManager.java)

```
// File: DatabaseManager.java

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class DatabaseManager {
    // Konfigurasi Database (Ganti sesuai konfigurasi Anda)
    private static final String DB_URL = "jdbc:mysql://localhost:3306/blackjack_db";
    private static final String USER = "root";
    private static final String PASS = "";

    // Metode untuk mendapatkan koneksi ke database
    private static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(DB_URL, USER, PASS);
    }

    /**
     * Memuat data pemain dari database berdasarkan nama pemain.
     * * Jika pemain tidak ada, akan membuat objek Pemain baru.
     * * Akan namaPemain Nama pemain yang akan dicari.
     * * Return Objek Pemain dengan data dari DB, atau Pemain baru jika tidak ditemukan.
     */
    public static Pemain muatPemain(String namaPemain) {
        Pemain pemain = null;
        String sql = "SELECT nama, jumlah_kemenangan, jumlah_kekalahan FROM pemain WHERE nama = ?";

        try (Connection conn = getConnection();
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, namaPemain);
            ResultSet rs = pstmt.executeQuery();

            if (rs.next()) {
                // Pemain ditemukan di database
                String nama = rs.getString("nama");
                int kemenangan = rs.getInt("jumlah_kemenangan");
                int kekalahan = rs.getInt("jumlah_kekalahan");
                pemain = new Pemain(nama, kemenangan, kekalahan);
                System.out.println("Pemain " + namaPemain + " dimuat dari database.");
            } else {
                // Pemain tidak ditemukan, buat objek Pemain baru
                pemain = new Pemain(namaPemain);
                System.out.println("Pemain " + namaPemain + " tidak ditemukan, membuat pemain baru.");
                // Dan simpan ke DB untuk pertama kalinya
                simpanAtauPerbaruiPemain(pemain);
            }
        } catch (SQLException e) {
            System.err.println("Error saat memuat pemain dari database: " + e.getMessage());
            e.printStackTrace();
            // Jika ada error DB, kembalikan pemain baru sebagai fallback
            pemain = new Pemain(namaPemain);
        }
        return pemain;
    }

    /**
     * Menyimpan data pemain ke database. Jika pemain sudah ada, akan diperbarui.
     * * Jika belum ada, akan ditambahkan sebagai record baru.
     * * Akan pemain Objek Pemain yang akan disimpan/diperbarui.
     */
    public static void simpanAtauPerbaruiPemain(Pemain pemain) {
        // Cek apakah pemain sudah ada di DB
        String checkSql = "SELECT COUNT(*) FROM pemain WHERE nama = ?";
        String updateSql = "UPDATE pemain SET jumlah_kemenangan = ?, jumlah_kekalahan = ? WHERE nama = ?";
        String insertSql = "INSERT INTO pemain (nama, jumlah_kemenangan, jumlah_kekalahan) VALUES (?, ?, ?)";

        try (Connection conn = getConnection();
            PreparedStatement checkStmt = conn.prepareStatement(checkSql)) {
            checkStmt.setString(1, pemain.getNama());
            ResultSet rs = checkStmt.executeQuery();
            rs.next();
            int count = rs.getInt(1);

            if (count > 0) {
                // Pemain sudah ada, lakukan UPDATE
                try (PreparedStatement updateStmt = conn.prepareStatement(updateSql)) {
                    updateStmt.setInt(1, pemain.getJumlahKemenangan());
                    updateStmt.setInt(2, pemain.getJumlahKekalahan());
                    updateStmt.setString(3, pemain.getNama());
                    updateStmt.executeUpdate();
                    System.out.println("Data pemain " + pemain.getNama() + " berhasil diperbarui.");
                }
            } else {
                // Pemain belum ada, lakukan INSERT
                try (PreparedStatement insertStmt = conn.prepareStatement(insertSql)) {
                    insertStmt.setString(1, pemain.getNama());
                    insertStmt.setInt(2, pemain.getJumlahKemenangan());
                    insertStmt.setInt(3, pemain.getJumlahKekalahan());
                    insertStmt.executeUpdate();
                    System.out.println("Pemain " + pemain.getNama() + " baru ditambahkan ke database.");
                }
            }
        } catch (SQLException e) {
            System.err.println("Error saat menyimpan/memperbarui pemain: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

[sourcecode](#)

Kelas DatabaseManager berfungsi sebagai jembatan utama antara aplikasi game Blackjack dan *database* MySQL. Tugas utamanya adalah mengelola data statistik pemain (jumlah kemenangan dan kekalahan) secara persisten. Kelas ini

menerapkan konsep Abstraksi dengan menyembunyikan detail kompleks koneksi dan operasi *database* di balik antarmuka yang sederhana.

Fungsi Kunci:

1. Konfigurasi Koneksi: Menentukan detail koneksi *database* MySQL (URL, *username*, *password*).
2. Membangun Koneksi: Metode *private* getConnection() membuat koneksi JDBC ke *database*, mengamankan detail internalnya sebagai bagian dari Enkapsulasi.
3. Memuat Data Pemain (muatPemain):
  - a. Mencari pemain berdasarkan nama di *database*.
  - b. Jika ditemukan, data pemain (nama, kemenangan, kekalahan) dimuat ke objek Pemain yang sudah ada.
  - c. Jika tidak ditemukan, objek Pemain baru akan dibuat dan langsung disimpan ke *database* untuk pertama kalinya (fitur *login/registrasi* otomatis).
  - d. Dilengkapi penanganan kesalahan untuk memastikan stabilitas aplikasi.
4. Menyimpan/Memperbarui Data Pemain (simpanAtauPerbaruiPemain):
  - a. Memeriksa apakah pemain sudah ada di *database*.
  - b. Jika sudah ada, data kemenangan dan kekalahan pemain diperbarui (*UPDATE*).
  - c. Jika belum ada, pemain baru ditambahkan (*INSERT*) sebagai *record* baru.
  - d. Juga dilengkapi penanganan kesalahan untuk menjaga integritas data.

### 2.4.2 EntitasPermainan (Kelas Abstrak)

```
// File: EntitasPermainan.java

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

public abstract class EntitasPermainan {
    protected String nama;
    protected List<Kartu> tanganKartu;

    public EntitasPermainan(String nama) {
        this.nama = nama;
        this.tanganKartu = new ArrayList<>();
    }

    public void tambahKartuKeTangan(Kartu kartu) {
        this.tanganKartu.add(kartu);
    }

    public int hitungTotalPoin() {
        int totalPoin = 0;
        int jumlahAs = 0;

        for (Kartu kartu : tanganKartu) {
            totalPoin += kartu.getPoin();
            if (kartu.getNilai().equals("As")) {
                jumlahAs++;
            }
        }

        while (totalPoin > 21 && jumlahAs > 0) {
            totalPoin -= 10;
            jumlahAs--;
        }

        return totalPoin;
    }

    // Untuk GUI, kita tidak akan lagi menggunakan System.out.print di sini
    // Public getter untuk mendapatkan tangan kartu
    public List<Kartu> getTanganKartu() {
        return tanganKartu;
    }

    public String getName() {
        return nama;
    }

    public abstract boolean harusHit();

    // Metode untuk mengosongkan tangan kartu
    public void kosongkanTangan() {
        this.tanganKartu.clear();
    }
}
```

codesnap.dev

Kelas EntitasPermainan adalah kelas abstrak yang berfungsi sebagai *blueprint* atau kerangka dasar untuk semua entitas yang dapat memegang kartu dalam permainan, seperti pemain dan *dealer*. Ini adalah contoh Abstraksi, di mana perilaku umum didefinisikan tanpa implementasi spesifik.

- Setiap entitas memiliki nama dan tanganKartu (list objek Kartu). Atribut ini dilindungi menggunakan `protected` sebagai bagian dari Enkapsulasi, hanya dapat diakses melalui metode yang disediakan.
- Metode `tambahKartuKeTangan()` memungkinkan penambahan kartu ke tangan.

- c. Metode `hitungTotalPoin()` menghitung total poin dari kartu di tangan, dengan logika khusus untuk kartu As (bernilai 11 atau 1) untuk mencegah *bust*.
- d. Metode abstrak `harusHit()` harus diimplementasikan oleh setiap kelas anak, menunjukkan Polimorfisme karena perilaku "harus *hit*" akan berbeda antara pemain dan *dealer*.
- e. Metode `kosongkanTangan()` digunakan untuk mereset kartu di tangan pada awal ronde baru.

#### 2.4.3 Pemain.java (Subkelas EntitasPermainan)

```
// File: Pemain.java (KERBALI KE VERSI AWAL, HANYA TAMBAH ATRIBUT SKOR)
// Tidak perlu lagi import java.io.Serializable
import java.util.ArrayList;
import java.util.List;

public class Pemain extends EntitasPermainan {
    private int jumlahKemenangan;
    private int jumlahKekalahan;

    public Pemain(String nama) {
        super(nama);
        this.jumlahKemenangan = 0;
        this.jumlahKekalahan = 0;
    }

    // Constructor tambahan untuk memuat pemain dari database
    public Pemain(String nama, int kemenangan, int kekalahan) {
        super(nama);
        this.jumlahKemenangan = kemenangan;
        this.jumlahKekalahan = kekalahan;
    }

    @Override
    public boolean harusHit() {
        return false;
    }

    public int getJumlahKemenangan() {
        return jumlahKemenangan;
    }

    public void tambahKemenangan() {
        this.jumlahKemenangan++;
    }

    public int getJumlahKekalahan() {
        return jumlahKekalahan;
    }

    public void tambahKekalahan() {
        this.jumlahKekalahan++;
    }

    public void setJumlahKemenangan(int jumlahKemenangan) { // Tambahkan setter untuk inisialisasi dari DB
        this.jumlahKemenangan = jumlahKemenangan;
    }

    public void setJumlahKekalahan(int jumlahKekalahan) { // Tambahkan setter untuk inisialisasi dari DB
        this.jumlahKekalahan = jumlahKekalahan;
    }

    public void resetSkor() {
        this.jumlahKemenangan = 0;
        this.jumlahKekalahan = 0;
    }
}
```

Kelas `Pemain` adalah subkelas dari `EntitasPermainan`, yang merepresentasikan pemain manusia dalam game. Ini adalah contoh Pewarisan (Inheritance), di mana `Pemain` mewarisi properti (nama, tanganKartu) dan perilaku dasar (tambahKartuKeTangan, `hitungTotalPoin`, `kosongkanTangan`) dari `EntitasPermainan`.

Kelas ini menambahkan atribut `jumlahKemenangan` dan `jumlahKekalahan` yang merupakan bagian dari Enkapsulasi, dengan metode getter (`getJumlahKemenangan()`, `getJumlahKekalahan()`) dan setter

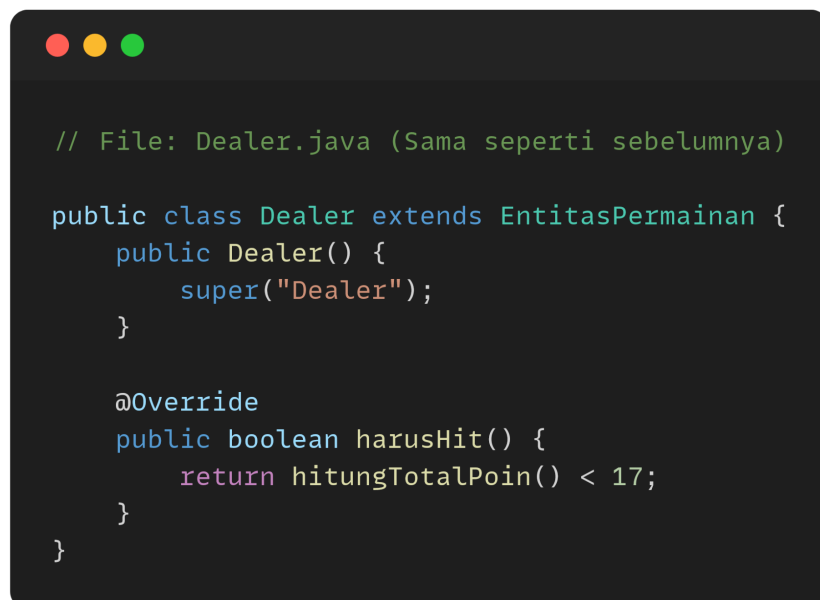
(setJumlahKemenangan(), setJumlahKekalahan()) untuk mengakses dan memodifikasinya.

Dua konstruktor disediakan: satu untuk pemain baru, dan satu lagi untuk memuat pemain yang sudah ada dari database dengan data skornya.

Metode tambahKemenangan() dan tambahKekalahan() memperbarui skor pemain.

Metode harusHit() di-override untuk selalu mengembalikan false, karena keputusan untuk hit atau stand pemain dilakukan secara manual melalui tombol GUI, bukan otomatis.

#### 2.4.4 Dealer.java (Subkelas EntitasPermainan)



```
// File: Dealer.java (Sama seperti sebelumnya)

public class Dealer extends EntitasPermainan {
    public Dealer() {
        super("Dealer");
    }

    @Override
    public boolean harusHit() {
        return hitungTotalPoin() < 17;
    }
}
```

codesnap.dev

Sama seperti Pemain, kelas Dealer juga merupakan subkelas dari EntitasPermainan, mewarisi fungsionalitas dasar dari kelas induknya. Ini juga menunjukkan contoh Pewarisan.

Metode harusHit() di-override untuk mengimplementasikan aturan spesifik *dealer* Blackjack: *dealer* secara otomatis akan terus "Hit" (mengambil kartu) selama total poin di tangannya kurang dari 17. Ini adalah contoh konkret dari Polimorfisme, di mana harusHit() memiliki implementasi yang berbeda di Dealer dibandingkan dengan Pemain.

### 2.4.5 Dek.java (Pengelolaan Kumpulan Kartu)

```
// File: Dek.java

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Dek {
    private List<Kartu> kartu;

    public Dek() {
        this.kartu = new ArrayList<>();
        String[] jenisKartu = {"Mati", "Wajik", "Keriting", "Sekop"};
        String[] nilaiKartu = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "As"};

        for (String jenis : jenisKartu) {
            for (String nilai : nilaiKartu) {
                this.kartu.add(new Kartu(jenis, nilai));
            }
        }
        acakDek();
    }

    public void acakDek() {
        Collections.shuffle(this.kartu);
        System.out.println("Dek telah dikocok.");
    }

    public Kartu ambilKartuTeratas() {
        if (kartu.isEmpty()) {
            // Untuk GUI, kita bisa memutuskan untuk membuat dek baru atau menangani skenario ini.
            // Untuk sementara, kita buat dek baru jika habis.
            System.out.println("Dek kosong! Membuat dek baru dan mengocok.");
            this.kartu = new Dek().kartu; // Buat dek baru secara rekursif (bukan cara terbaik untuk game sungguhan)
            acakDek();
        }
        return kartu.remove(0);
    }

    public int getJumlahKartu() {
        return kartu.size();
    }
}
```

codesnap.dev

Kelas Dek bertanggung jawab untuk merepresentasikan dan mengelola kumpulan 52 kartu standar dalam permainan.

1. Atribut kartu (sebuah List dari objek Kartu) dijaga secara Enkapsulasi dan diinisialisasi dalam konstruktor dengan 52 kartu lengkap (empat jenis dan tiga belas nilai).
2. Metode `acakDek()` menggunakan `Collections.shuffle()` untuk mengocok kartu, mensimulasikan pengocokan dek fisik.
3. Metode `ambilKartuTeratas()` mengambil kartu dari bagian atas dek (indeks 0). Jika dek kosong, ia akan membuat dek baru dan mengocoknya lagi, memastikan permainan dapat terus berlanjut.

#### 2.4.6 KartuPanel.java (Representasi Visual Kartu)

```
// File: KartuPanel.java

import javax.swing.*;
import java.awt.*;

public class KartuPanel extends JPanel {
    private Kartu kartu;
    private boolean tersembunyi; // Untuk kartu tertutup (dealer)

    public KartuPanel(Kartu kartu, boolean tersembunyi) {
        this.kartu = kartu;
        this.tersembunyi = tersembunyi;
        setPreferredSize(new Dimension(72, 96)); // Ukuran standar kartu remi
        // Set background, border, dll. jika diperlukan
        // setBorder(BorderFactory.createLineBorder(Color.BLACK));
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Image img;
        if (tersembunyi) {
            img = Kartu.getBackImageIcon().getImage(); // Ambil gambar belakang kartu
        } else {
            img = kartu.getImageIcon().getImage(); // Ambil gambar kartu asli
        }
        if (img != null) {
            // Gambar kartu agar sesuai dengan ukuran panel
            g.drawImage(img, 0, 0, getWidth(), getHeight(), this);
        } else {
            // Gambar placeholder jika gambar tidak ditemukan
            g.setColor(Color.RED);
            g.fillRect(0, 0, getWidth(), getHeight());
            g.setColor(Color.WHITE);
            g.drawString("?", getWidth() / 2 - 5, getHeight() / 2 + 5);
        }
    }

    // Metode untuk mengubah status tersembunyi
    public void setTersembunyi(boolean tersembunyi) {
        this.tersembunyi = tersembunyi;
        repaint(); // Gambar ulang panel
    }
}
```

codesnap.dev

Kelas `KartuPanel` adalah komponen GUI (`JPanel`) yang bertugas menampilkan satu kartu individu di layar.

1. Ini menyimpan objek `Kartu` yang akan ditampilkan dan status `tersembunyi` (boolean) untuk kartu *dealer* yang tertutup.
2. Metode `paintComponent()` di-*override* untuk menggambar gambar kartu yang sesuai. Jika `tersembunyi` adalah `true`, gambar belakang kartu akan ditampilkan; jika tidak, gambar depan kartu asli akan muncul. Ini memungkinkan tampilan kartu *dealer* yang terbalik.
3. Metode `setTersembunyi()` memungkinkan status tampilan kartu diubah secara dinamis, lalu memperbarui tampilan kartu.



#### 2.4.7 TanganPanel.java (Panel untuk Tangan Kartu)

```
// File: TanganPanel.java

import javax.swing.*;
import java.awt.*;
import java.util.List;

public class TanganPanel extends JPanel {
    public TanganPanel() {
        setLayout(new FlowLayout(FlowLayout.LEFT, 5, 0)); // Tata letak horizontal untuk kartu
        setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10)); // Sedikit padding
    }

    // Metode untuk memperbarui tampilan tangan kartu
    public void perbaruiTangan(List<Kartu> tangan, boolean isDealer, boolean sembunyikanKartuPertamaDealer) {
        removeAll(); // Hapus semua kartu yang ada
        for (int i = 0; i < tangan.size(); i++) {
            Kartu kartu = tangan.get(i);
            boolean tersembunyiUntukPanelIni = sembunyikanKartuPertamaDealer && isDealer && i == 0;
            KartuPanel kartuPanel = new KartuPanel(kartu, tersembunyiUntukPanelIni);
            add(kartuPanel);
        }
        revalidate(); // Validasi ulang layout
        repaint();    // Gambar ulang komponen
    }

    // Metode untuk mengosongkan panel tangan
    public void kosongkan() {
        removeAll();
        revalidate();
        repaint();
    }
}
```



Kelas TanganPanel adalah komponen GUI (JPanel) yang berfungsi sebagai kontainer untuk menampilkan kumpulan kartu di tangan pemain atau *dealer*.

1. Menggunakan FlowLayout untuk menata kartu secara horizontal.
2. Metode perbaruiTangan() adalah inti dari panel ini. Ia menghapus semua KartuPanel yang ada, kemudian menambahkan KartuPanel baru untuk setiap kartu di tangan yang diberikan. Logika khusus diterapkan untuk menyembunyikan kartu pertama *dealer* pada awal permainan, kemudian menampilkannya saat giliran *dealer*.
3. Metode kosongkan() digunakan untuk membersihkan tampilan tangan pada awal ronde baru.

## 2.4.8 Kartu.java ( Representasi Kartu Permainan)

```
// File: Kartu.java
import javax.swing.ImageIcon;
import java.net.URL;

public class Kartu {
    private String jenis; // Contoh: "Sekop", "Hati", "Keriting", "Wajik"
    private String nilai; // Contoh: "2", "3", ..., "10", "Jack", "Queen", "King", "As"
    private int poin; // Nilai poin Blackjack dari kartu tersebut

    public Kartu(String jenis, String nilai) {
        this.jenis = jenis;
        this.nilai = nilai;
        this.poin = hitungPoinKartu(nilai);
    }

    private int hitungPoinKartu(String nilaiKartu) {
        try {
            return Integer.parseInt(nilaiKartu);
        } catch (NumberFormatException e) {
            switch (nilaiKartu) {
                case "Jack":
                case "Queen":
                case "King":
                    return 10;
                case "As":
                    return 11; // Untuk sementara As selalu 11.
                default:
                    return 0;
            }
        }
    }

    public String getJenis() {
        return jenis;
    }

    public String getNilai() {
        return nilai;
    }

    public int getPoin() {
        return poin;
    }

    @Override
    public String toString() {
        return nilai + " " + jenis;
    }

    /**
     * Mengembalikan ImageIcon dari gambar kartu.
     * Asumsikan gambar berada di folder 'kartu_gambar/' di root classpath.
     * Nama file: nilai_jenis.png (contoh: 'as_sekop.png', '10_hati.png')
     * atau back.png untuk kartu tertutup.
     */
    public ImageIcon getImageIcon() {
        String namaFile = nilai.toLowerCase() + "_" + jenis.toLowerCase() + ".png";
        URL imgURL = getClass().getResource("/kartu_gambar/" + namaFile); // Path relatif dari classpath
        if (imgURL != null) {
            return new ImageIcon(imgURL);
        } else {
            System.err.println("Gambar kartu tidak ditemukan: " + namaFile);
            return null; // Atau kembalikan placeholder
        }
    }

    /**
     * Mengembalikan ImageIcon untuk bagian belakang kartu.
     */
    public static ImageIcon getBackImageIcon() {
        URL imgURL = Kartu.class.getResource("/kartu_gambar/back_blue.png");
        if (imgURL != null) {
            return new ImageIcon(imgURL);
        } else {
            System.err.println("Gambar belakang kartu tidak ditemukan: back.png");
            return null;
        }
    }
}
```

Kelas Kartu merepresentasikan setiap kartu remi dalam permainan Blackjack. Kelas ini mengimplementasikan Enkapsulasi dengan menyimpan atribut jenis (misal "Sekop"), nilai (misal "As"), dan poin (nilai Blackjack) secara *private*, diakses hanya melalui metode *getter*.

Fungsi Kunci:

1. Penentuan Poin Kartu: Konstruktor kelas Kartu secara otomatis menghitung poin Blackjack yang sesuai (angka 2-10, Jack/Queen/King bernilai 10, As bernilai 11) saat objek kartu dibuat.

2. Akses Informasi: Metode *getter* menyediakan akses ke jenis, nilai, dan poin kartu.

#### 2.4.9 LoginGUI.java (Antarmuka Login Pemain)

```
// File: LoginGUI.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class LoginGUI extends JFrame {
    private JTextField usernameField;
    private JButton loginButton;
    private JLabel statusLabel;

    public LoginGUI() {
        super("BlackJack Login");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 200);
        setLocationRelativeTo(null); // Pusatkan jendela
        setUpUI();
    }

    private void setUpUI() {
        JPanel panel = new JPanel();
        panel.setLayout(new GridBagLayout());
        panel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(5, 5, 5, 5); // Padding

        // Username Label
        gbc.gridx = 0;
        gbc.gridy = 0;
        panel.add(new JLabel("Nama Pemain:"), gbc);

        // Username Field
        gbc.gridx = 1;
        gbc.gridy = 0;
        gbc.fill = GridBagConstraints.HORIZONTAL; // Mengisi ruang horizontal
        usernameField = new JTextField(15); // Lebar field
        panel.add(usernameField, gbc);

        // Login Button
        gbc.gridx = 0;
        gbc.gridy = 1;
        gbc.gridwidth = 2; // Membentang 2 kolom
        gbc.fill = GridBagConstraints.NONE; // Tidak mengisi ruang
        loginButton = new JButton("Masuk / Daftar");
        panel.add(loginButton, gbc);

        // Status Label
        gbc.gridx = 0;
        gbc.gridy = 2;
        statusLabel = new JLabel("Masukkan nama pemain Anda.", SwingConstants.CENTER);
        statusLabel.setForeground(Color.BLUE);
        panel.add(statusLabel, gbc);

        add(panel, BorderLayout.CENTER);

        // Tambahkan aksi untuk tombol login
        loginButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String username = usernameField.getText().trim();
                if (username.isEmpty()) {
                    statusLabel.setText("Nama pemain tidak boleh kosong!");
                    statusLabel.setForeground(Color.RED);
                } else {
                    statusLabel.setText("Memuat pemain...");
                    statusLabel.setForeground(Color.BLACK);
                    // Panggil DatabaseManager untuk memuat/mendaftar pemain
                    Pemain pemain = DatabaseManager.muatPemain(username);
                    if (pemain != null) {
                        statusLabel.setText("Berhasil masuk sebagai " + pemain.getNama() + "!");
                        statusLabel.setForeground(Color.GREEN.darker());

                        // Sembunyikan jendela login dan tampilkan jendela permainan
                        dispose(); // Tutup jendela login
                        SwingUtilities.invokeLater(() -> {
                            new GameBlackJackGUI(pemain).setVisible(true); // Kirim objek Pemain ke game
                        });
                    } else {
                        statusLabel.setText("Gagal memuat/mendaftar pemain. Coba lagi.");
                        statusLabel.setForeground(Color.RED);
                    }
                }
            }
        });
    }

    public static void main(String[] args) {
        // Jalankan GUI di Event Dispatch Thread (EDT)
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new LoginGUI().setVisible(true);
            }
        });
    }
}
```

Kelas LoginGUI adalah jendela (JFrame) pertama yang muncul saat aplikasi dijalankan, berfungsi sebagai antarmuka untuk pemain masuk atau mendaftar.

1. Menggunakan komponen Swing seperti JTextField untuk input nama pemain, JButton untuk aksi *login*, dan JLabel untuk menampilkan pesan status.
2. Tombol "Masuk / Daftar" memiliki *ActionListener* yang akan memanggil metode DatabaseManager.muatPemain(). Ini menunjukkan integrasi antara GUI dan modul persistensi data.

3. Berdasarkan hasil `muatPemain()`, pesan status akan diperbarui, dan jika berhasil, jendela `LoginGUI` akan ditutup, lalu jendela `GameBlackjackGUI` akan dibuka, mengirimkan objek `Pemain` yang telah dimuat/dibuat. Ini memastikan sesi permainan dimulai dengan data pemain yang benar.
4. Metode `main` aplikasi terletak di kelas ini, yang bertanggung jawab untuk memulai `LoginGUI` pada *Event Dispatch Thread* (EDT) untuk memastikan keamanan *thread* pada `Swing`.

#### 2.4.10 GameBlackjackGUI.java (Antarmuka Permainan Utama)



Kelas `GameBlackjackGUI` adalah jendela utama (`JFrame`) tempat permainan `Blackjack` berlangsung.

1. Inisialisasi: Konstruktor menerima objek `Pemain` yang sudah dimuat/dibuat dari `LoginGUI`, menunjukkan bagaimana data pemain diwariskan dari satu modul GUI ke modul lainnya. Objek `Dek` dan `Dealer` juga diinisialisasi di sini.

2. Tata Letak GUI (`setupUI()`): Mengatur tata letak menggunakan `BorderLayout` dan menempatkan panel-panel untuk tangan *dealer*, tangan pemain, label skor, label status, dan tombol kontrol. `TanganPanel` digunakan untuk menampilkan kartu pemain dan *dealer*.
3. Pengelolaan Tombol: Tombol "Deal", "Hit", dan "Stand" disediakan dengan `ActionListener` masing-masing yang memicu logika permainan.
4. Alur Permainan (`mulaiPermainanBaru()`, `pemainHit()`, `pemainStand()`, `dealerMain()`, `akhirPermainan()`):
  - a. `mulaiPermainanBaru()`: Mereset dek dan tangan kartu, mengocok dek, membagikan dua kartu awal ke pemain dan *dealer*, memperbarui tampilan, dan mengaktifkan tombol "Hit" dan "Stand".
  - b. `pemainHit()`: Menambahkan kartu baru ke tangan pemain, memperbarui skor, dan memeriksa apakah pemain *bust* atau mencapai 21.
  - c. `pemainStand()`: Mengakhiri giliran pemain dan menyerahkan kontrol ke *dealer*.
  - d. `dealerMain()`: Mengungkap kartu tersembunyi *dealer* dan secara otomatis menarik kartu berdasarkan aturan `dealer.harusHit()` (menunjukkan Polimorfisme yang digunakan).
  - e. `akhirPermainan()`: Menentukan pemenang berdasarkan poin pemain dan *dealer*, memperbarui status game, dan mengupdate skor kemenangan/kekalahan pemain.
5. Pembaruan Skor: Metode `updateWinLossLabel()` secara dinamis memperbarui tampilan jumlah kemenangan dan kekalahan pemain di GUI.
6. Persistensi Saat Keluar: Sebuah `WindowListener` ditambahkan untuk mendeteksi penutupan jendela. Saat aplikasi ditutup, `DatabaseManager.simpan AtauPerbaruiPemain (pemain)` dipanggil untuk menyimpan skor akhir pemain ke *database*, memastikan data tidak hilang.

#### 2.4.11 Aset Gambar Kartu (`kartu_gambar/`)

Untuk mendukung representasi visual kartu dalam antarmuka grafis, aplikasi menyimpan koleksi gambar kartu dalam folder `kartu_gambar/` yang ditempatkan di *classpath* proyek. Folder ini berisi total 53 *file* gambar berformat .png, yang terdiri dari 52 Gambar Kartu Remi Standar: Setiap gambar dinamai sesuai format `nilai_jenis.png` (misal: `as_hati.png`, `10_sekop.png`, `king_wajik.png`) untuk merepresentasikan semua kombinasi nilai dan jenis kartu. Penamaan yang

konsisten ini memudahkan kelas Kartu.java untuk secara dinamis memuat gambar yang tepat.

## 2.5 Pengujian Program

Pengujian program dilakukan menggunakan metode Black Box Testing, di mana fungsionalitas sistem diuji berdasarkan spesifikasi kebutuhan tanpa melihat struktur kode internal. Pengujian ini berfokus pada validasi *input* dan *output* sistem.

No.	Modul yang Diuji	Skenario Pengujian	Input	Harapan Output Sistem	Status	Keterangan
1.	Login	Memasukkan nama pemain baru (pertama kali login)	usernameField: "PemainBaru"	- Layar login tertutup - Layar game Blackjack muncul - Skor "Kemenangan: 0"	Kekalahan: 0"	Lulus
2.	Login	Memasukkan nama pemain yang sudah ada (lanjutan sesi)	usernameField: "PemainBaru" (setelah bermain)	- Layar login tertutup . - Layar game Blackjack muncul. - Skor sesuai dengan terakhir disimpan	Lulus	Sistem berhasil memuat data pemain dari sesi sebelumnya.
3.	Login	Memasukkan nama pemain kosong	usernameField: "" (kosong)	- Status label menampilkan "Nama pemain tidak boleh kosong!" (merah) . - Layar login tetap terbuka	Lulus	Validasi input nama pemain berhasil.
4.	Permainan	Memulai permainan baru (Deal)	Klik tombol "Deal"	- 2 kartu untuk pemain . - 2 kartu untuk dealer (1 tersembunyi) . - Skor awal diperbarui . - Tombol "Hit", "Stand" aktif	Lulus	Pembagian kartu awal dan pembaruan UI berhasil.
5.	Permainan	Pemain memilih Hit dan tidak Bust	Klik tombol "Hit"	- Pemain menerima 1 kartu tambahan . - Tangan pemain diperbarui . - Skor pemain diperbarui . - Tombol "Hit", "Stand" tetap aktif	Lulus	Pemain berhasil mengambil kartu.
6.	Permainan	Pemain memilih Hit dan Bust (>21)	Klik tombol "Hit" (hingga skor >21)	- Status label menampilkan "Anda BUST! Dealer menang." . - Skor kekalahan pemain bertambah . - Tombol "Deal" aktif, lainnya non-aktif	Lulus	Logika Bust pemain dan pembaruan skor kekalahan berhasil.

7.	Permainan	Pemain memilih Stand	Klik tombol "Stand"	<ul style="list-style-type: none"> <li>- Giliran dealer dimulai .</li> <li>- Dealer membuka kartu tersembunyi .</li> <li>- Dealer mengambil kartu sesuai aturan</li> </ul>	Lulus	Transisi giliran dan logika dealer otomatis berhasil.
8.	Permainan	Dealer Bust (>21)	(Otomatis setelah Stand pemain)	<ul style="list-style-type: none"> <li>- Status label menampilkan "Dealer BUST! Anda menang!" .</li> <li>- Skor kemenangan pemain bertambah</li> </ul>	Lulus	Logika Bust dealer dan pembaruan skor kemenangan berhasil.
9.	Permainan	Pemain menang (skor > dealer, &lt;=21)	(Otomatis setelah Stand pemain)	<ul style="list-style-type: none"> <li>- Status label menampilkan "Anda Menang! [Skor Pemain] vs [Skor Dealer]" .</li> <li>- Skor kemenangan pemain bertambah</li> </ul>	Lulus	Penentuan pemenang (pemain) dan pembaruan skor berhasil.
10.	Permainan	Dealer menang (skor > pemain, &lt;=21)	(Otomatis setelah Stand pemain)	<ul style="list-style-type: none"> <li>- Status label menampilkan "Dealer Menang! [Skor Dealer] vs [Skor Pemain]" .</li> <li>- Skor kekalahan pemain bertambah</li> </ul>	Lulus	Penentuan pemenang (dealer) dan pembaruan skor berhasil.
11.	Permainan	Permainan Seri (skor pemain = dealer, &lt;=21)	(Otomatis setelah Stand pemain)	<ul style="list-style-type: none"> <li>- Status label menampilkan "Seri! [Skor Pemain] vs [Skor Dealer]" .</li> <li>- Tidak ada perubahan skor kemenangan/kekalahan</li> </ul>	Lulus	Penentuan seri berhasil.
12.	Persistensi	Menutup aplikasi setelah bermain dan memenangkan 1 ronde	Tutup jendela GUI dengan tombol 'X'	<ul style="list-style-type: none"> <li>- Aplikasi tertutup .</li> <li>- Data "PemainBaru" di database memiliki Kemenangan: 1, Kekalahan: 0</li> </ul>	Lulus	Data kemenangan berhasil disimpan ke database.
13.	Persistensi	Menutup aplikasi setelah bermain dan kalah 1 ronde	Tutup jendela GUI dengan tombol 'X'	<ul style="list-style-type: none"> <li>- Aplikasi tertutup .</li> <li>- Data "PemainBaru" di database memiliki Kemenangan: X, Kekalahan: 1</li> </ul>	Lulus	Data kekalahan berhasil disimpan ke database.

## **BAB III**

### **PENUTUP**

#### **3.1 Kesimpulan**

Antarmuka grafis dibangun menggunakan Java Swing, yang memungkinkan interaksi pengguna dilakukan dengan cara yang intuitif dan responsif. Selain itu, sistem juga dilengkapi dengan fitur penyimpanan data menggunakan koneksi JDBC ke MySQL, sehingga statistik permainan seperti jumlah kemenangan dan kekalahan pemain dapat tersimpan secara otomatis. Fitur login sederhana turut ditambahkan agar setiap pemain dapat masuk ke dalam profil masing-masing, dan melanjutkan permainan dari sesi sebelumnya.

Secara keseluruhan, proyek ini tidak hanya menghasilkan aplikasi game yang berjalan dengan lancar dan menyenangkan, tetapi juga memberikan pengalaman praktis yang berharga dalam penerapan prinsip-prinsip OOP, pengembangan antarmuka grafis, dan pengelolaan basis data. Hasilnya adalah sebuah aplikasi yang tidak hanya fungsional, tetapi juga memiliki pondasi yang kuat untuk dikembangkan lebih lanjut di masa mendatang.

#### **3.2 Saran**

Berdasarkan hasil pengembangan dan pengujian aplikasi game Blackjack ini, terdapat sejumlah saran yang dapat dipertimbangkan untuk peningkatan lebih lanjut, baik dari segi fungsionalitas maupun pengalaman pengguna:

1. Validasi Input yang Lebih Ketat

Validasi input pada form login, khususnya nama pemain, saat ini masih bersifat dasar. Kedepannya, sistem dapat diperkuat dengan aturan validasi yang lebih komprehensif, seperti membatasi jumlah karakter, mencegah penggunaan simbol tertentu, serta memfilter kata-kata yang tidak pantas.

2. Logika Penanganan Kartu As yang Lebih Cerdas

Saat ini, kartu As akan bernilai 11, dan baru dikurangi menjadi 1 jika total poin pemain melebihi 21. Untuk meningkatkan akurasi simulasi permainan Blackjack, logika ini bisa dikembangkan agar sistem secara dinamis menentukan nilai As (1 atau 11) berdasarkan kombinasi kartu yang paling menguntungkan bagi pemain.

3. Fitur Logout dan Manajemen Sesi

Aplikasi saat ini belum menyediakan mekanisme logout atau pergantian akun secara eksplisit. Menambahkan fitur logout dan pengelolaan sesi pemain akan memberikan fleksibilitas lebih, terutama jika aplikasi digunakan oleh lebih dari satu pengguna pada perangkat yang sama.



#### 4. Penambahan Fitur Permainan Blackjack Lanjutan

Untuk meningkatkan kompleksitas dan keseruan permainan, fitur-fitur lanjutan seperti Split, Double Down, dan Insurance dapat diimplementasikan. Kehadiran fitur ini akan memperkaya strategi permainan sekaligus menantang pengembang dalam membangun logika permainan yang lebih kompleks.

#### 5. Peningkatan Antarmuka Pengguna

Meskipun tampilan antarmuka sudah berjalan dengan baik secara fungsional, peningkatan dari sisi estetika akan membuat permainan lebih menarik. Penggunaan tema GUI yang lebih modern, animasi interaktif untuk kartu, ikon yang lebih menarik, serta efek suara, dapat menciptakan pengalaman bermain yang lebih imersif.

#### 6. Penanganan Error yang Lebih Ramah Pengguna

Saat ini, kesalahan terkait koneksi database atau operasi data hanya dicetak di konsol. Untuk meningkatkan kenyamanan pengguna, pesan-pesan error sebaiknya ditampilkan langsung di antarmuka (misalnya menggunakan JOptionPane), sehingga pengguna mendapatkan informasi yang lebih jelas jika terjadi kendala.

#### 7. Dukungan Multi-Pemain

Pengembangan sistem yang memungkinkan lebih dari satu pemain manusia bermain secara bersamaan, baik dalam mode lokal maupun jaringan, akan menjadi tantangan yang menarik. Fitur ini dapat membuka jalan bagi mode kompetitif atau kolaboratif, sekaligus meningkatkan daya tarik aplikasi secara keseluruhan.