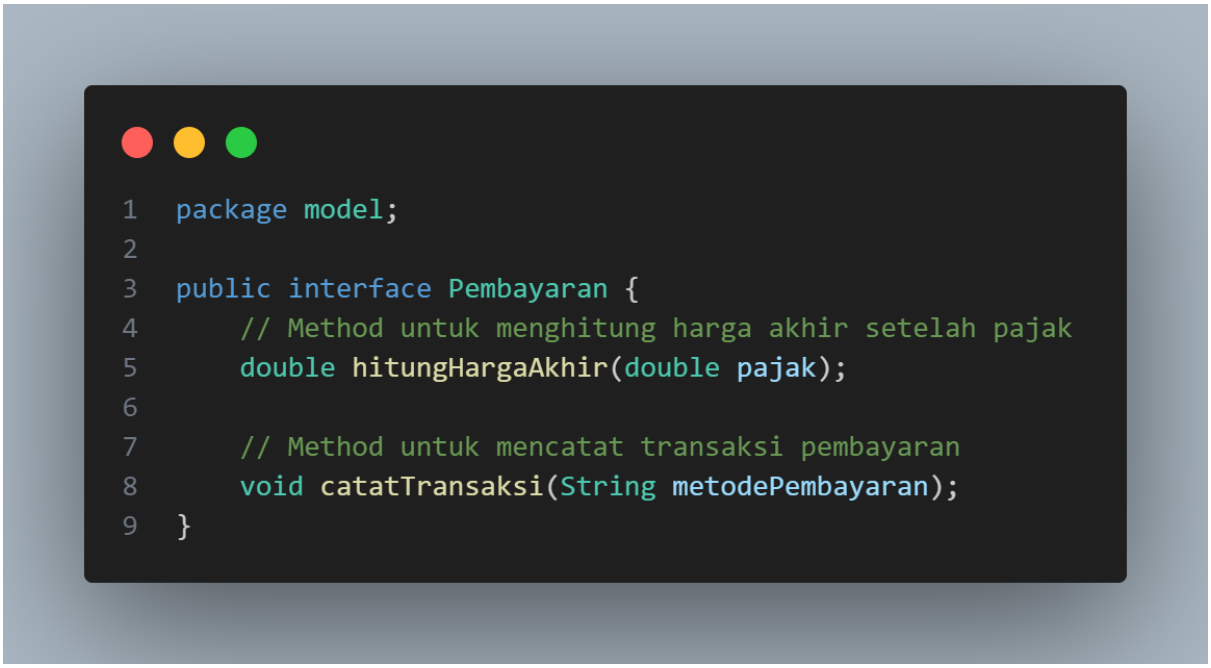

```java
package model;

public interface Pembayaran {
    // Method untuk menghitung harga akhir setelah pajak
    double hitungHargaAkhir(double pajak);

    // Method untuk mencatat transaksi pembayaran
    void catatTransaksi(String metodePembayaran);
}
```

```java
package model;

public class PesananBoots extends Pesanan {
    private boolean perawatanKulit;

    public PesananBoots(int id, String namaPelanggan, String jenisSepatu, int jumlahSepatu,
                        JenisPencucian jenisPencucian, boolean perawatanKulit) {
        super(id, namaPelanggan, jenisSepatu, jumlahSepatu, jenisPencucian);
        this.perawatanKulit = perawatanKulit;
        this.hargaDasar = 35000.0;
    }

    // overloading dengan harga dasar kustom
    public PesananBoots(int id, String namaPelanggan, String jenisSepatu, int jumlahSepatu,
                        JenisPencucian jenisPencucian, boolean perawatanKulit, double hargaDasar) {
        super(id, namaPelanggan, jenisSepatu, jumlahSepatu, jenisPencucian, hargaDasar);
        this.perawatanKulit = perawatanKulit;
    }

    public boolean isPerawatanKulit() {
        return perawatanKulit;
    }

    public void setPerawatanKulit(boolean perawatanKulit) {
        this.perawatanKulit = perawatanKulit;
    }

    @Override
    public double hitungHarga() {
        double harga = super.hitungHarga(); // Panggil metode parent

        // Tambahan biaya untuk perawatan kulit
        if (perawatanKulit) {
            harga += 20000.0 * jumlahSepatu;
        }

        return harga;
    }

    // Tidak dapat mengganti hitungHarga(double diskonPersen) karena sudah final di kelas induk

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Perawatan Kulit: " + (perawatanKulit ? "Ya" : "Tidak"));
    }

    @Override
    public void displayInfo(boolean showPrice) {
        super.displayInfo(showPrice);
        System.out.println("Perawatan Kulit: " + (perawatanKulit ? "Ya" : "Tidak"));
    }

    @Override
    public String getServiceDetails() {
        return "Boots - " + (perawatanKulit ? "With Leather Treatment" : "Standard Cleaning");
    }

    // Override metode dari interface untuk kasus khusus boots
    @Override
    public double hitungHargaAkhir(double pajak) {
        try {
            double harga = super.hitungHargaAkhir(pajak);
            // Boots mendapat diskon pajak 2% jika ada perawatan kulit
            if (perawatanKulit) {
                harga -= (harga * 0.02);
            }
            return harga;
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
            return hitungHarga(); // Return harga tanpa pajak jika ada error
        }
    }
}
```

```java
package model;

public enum JenisPencucian {
    REGULER(1.0),
    EXPRESS(1.5);

    private final double multiplier;

    // Constructor enum
    JenisPencucian(double multiplier) {
        this.multiplier = multiplier;
    }

    // Method untuk mendapatkan multiplier
    public double getMultiplier() {
        return this.multiplier;
    }

    // Static method untuk mendapatkan JenisPencucian dari string
    public static JenisPencucian fromString(String text) {
        try {
            return valueOf(text.toUpperCase());
        } catch (IllegalArgumentException e) {
            throw new IllegalArgumentException("Jenis pencucian tidak valid: " + text +
                                               ". Pilih antara Reguler atau Express.");
        }
    }
}
```

```java
package model;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public abstract class Pesanan implements Pembayaran {
    protected int id;
    protected String namaPelanggan;
    protected String jenisSepatu;
    protected int jumlahSepatu;
    protected JenisPencucian jenisPencucian;
    protected final double DISKON_MAKS = 80.0; // final attribute
    protected double hargaDasar = 25000.0; // harga dasar untuk semua jenis pesanan

    // Static counter untuk menghitung total pesanan
    private static int totalPesanan = 0;
    // Static formatter untuk tanggal
    private static final DateTimeFormatter DATE_FORMATTER = DateTimeFormatter.ofPattern("dd MM yyyy HH:mm:ss");

    // Constructor yang sudah ada
    public Pesanan(int id, String namaPelanggan, String jenisSepatu, int jumlahSepatu, JenisPencucian jenisPencucian) {
        this.id = id;
        this.namaPelanggan = namaPelanggan;
        this.jenisSepatu = jenisSepatu;
        this.jumlahSepatu = jumlahSepatu;
        this.jenisPencucian = jenisPencucian;
        totalPesanan++; // Increment counter setiap kali objek dibuat
    }

    // Constructor overloading - versi dengan harga dasar custom
    public Pesanan(int id, String namaPelanggan, String jenisSepatu, int jumlahSepatu,
                   JenisPencucian jenisPencucian, double hargaDasar) {
        this.id = id;
        this.namaPelanggan = namaPelanggan;
        this.jenisSepatu = jenisSepatu;
        this.jumlahSepatu = jumlahSepatu;
        this.jenisPencucian = jenisPencucian;
        this.hargaDasar = hargaDasar;
        totalPesanan++; // Increment counter setiap kali objek dibuat
    }

    public int getId() {
        return id;
    }

    public String getNamaPelanggan() {
        return namaPelanggan;
    }

    public void setNamaPelanggan(String namaPelanggan) {
        this.namaPelanggan = namaPelanggan;
    }

    public String getJenisSepatu() {
        return jenisSepatu;
    }

    public void setJenisSepatu(String jenisSepatu) {
        this.jenisSepatu = jenisSepatu;
    }

    public int getJumlahSepatu() {
        return jumlahSepatu;
    }

    public void setJumlahSepatu(int jumlahSepatu) {
        if (jumlahSepatu > 0) {
            this.jumlahSepatu = jumlahSepatu;
        } else {
            throw new IllegalArgumentException("Jumlah sepatu harus lebih dari 0.");
        }
    }

    public JenisPencucian getJenisPencucian() {
        return jenisPencucian;
    }

    public void setJenisPencucian(JenisPencucian jenisPencucian) {
        this.jenisPencucian = jenisPencucian;
    }

    // Static method untuk mendapatkan total pesanan
    public static int getTotalPesanan() {
        return totalPesanan;
    }

    // Static method untuk reset counter (untuk keperluan testing)
    public static void resetTotalPesanan() {
        totalPesanan = 0;
    }

    // Static method untuk memformat tanggal
    public static String formatTanggal(LocalDateTime dateTime) {
        return dateTime.format(DATE_FORMATTER);
    }

    // Method untuk menghitung harga
    public double hitungHarga() {
        double harga = hargaDasar * jumlahSepatu;

        // Harga tambahan untuk pencucian express
        if (jenisPencucian == JenisPencucian.EXPRESS) {
            harga *= 1.5; // Tambahan 50% untuk express
        }

        return harga;
    }

    // Method overloading untuk hitungHarga dengan parameter diskon
    public final double hitungHarga(double diskonPersen) { // Final method
        if (diskonPersen > DISKON_MAKS) {
            diskonPersen = DISKON_MAKS; // Batasi diskon maksimum
        }

        double hargaAwal = hitungHarga();
        double diskon = hargaAwal * (diskonPersen / 100.0);
        return hargaAwal - diskon;
    }

    public void displayInfo() {
        System.out.println("ID: " + id + ", Nama: " + namaPelanggan + ", Sepatu: " + jenisSepatu +
                           ", Jumlah: " + jumlahSepatu + ", Pencucian: " + jenisPencucian +
                           ", Harga: Rp" + hitungHarga());
    }

    // Method overloading untuk displayInfo
    public void displayInfo(boolean showPrice) {
        System.out.println("ID: " + id + ", Nama: " + namaPelanggan + ", Sepatu: " + jenisSepatu +
                           ", Jumlah: " + jumlahSepatu + ", Pencucian: " + jenisPencucian);
        if (showPrice) {
            System.out.println("Harga: Rp" + hitungHarga());
        }
    }

    // Abstract method that must be implemented by subclasses
    public abstract String getServiceDetails();

    // Implementasi dari interface Pembayaran
    @Override
    public double hitungHargaAkhir(double pajak) {
        if (pajak < 0) {
            throw new IllegalArgumentException("Pajak tidak boleh negatif");
        }
        double harga = hitungHarga();
        return harga + (harga * pajak / 100);
    }

    @Override
    public void catatTransaksi(String metodePembayaran) {
        LocalDateTime now = LocalDateTime.now();
        String timestamp = formatTanggal(now);
        System.out.println("------ TRANSAKSI DICATAT ------");
        System.out.println("Tanggal: " + timestamp);
        System.out.println("ID Pesanan: " + id);
        System.out.println("Nama Pelanggan: " + namaPelanggan);
        System.out.println("Metode Pembayaran: " + metodePembayaran);
        System.out.println("Total Pembayaran: Rp" + hitungHarga());
        System.out.println("==============================");
    }
}
```

```java
package model;

public class PesananSandal extends Pesanan {
    private boolean repairSole;

    public PesananSandal(int id, String namaPelanggan, String jenisSepatu, int jumlahSepatu,
                         JenisPencucian jenisPencucian, boolean repairSole) {
        super(id, namaPelanggan, jenisSepatu, jumlahSepatu, jenisPencucian);
        this.repairSole = repairSole;
        this.hargaDasar = 20000.0;
    }

    // overloading dengan harga dasar kustom
    public PesananSandal(int id, String namaPelanggan, String jenisSepatu, int jumlahSepatu,
                         JenisPencucian jenisPencucian, boolean repairSole, double hargaDasar) {
        super(id, namaPelanggan, jenisSepatu, jumlahSepatu, jenisPencucian, hargaDasar);
        this.repairSole = repairSole;
    }

    public boolean isRepairSole() {
        return repairSole;
    }

    public void setRepairSole(boolean repairSole) {
        this.repairSole = repairSole;
    }

    @Override
    public double hitungHarga() {
        double harga = super.hitungHarga(); // Panggil metode parent

        // Tambahan biaya perbaikan sol
        if (repairSole) {
            harga += 10000.0 * jumlahSepatu;
        }

        return harga;
    }

    // Tidak dapat mengganti hitungHarga(double diskonPersen) karena sudah final di kelas induk

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Perbaikan Sol: " + (repairSole ? "Ya" : "Tidak"));
    }

    @Override
    public void displayInfo(boolean showPrice) {
        super.displayInfo(showPrice);
        System.out.println("Perbaikan Sol: " + (repairSole ? "Ya" : "Tidak"));
    }

    @Override
    public String getServiceDetails() {
        return "Sandal - " + (repairSole ? "With Sole Repair" : "Standard Cleaning");
    }

    // Override metode dari interface untuk kasus khusus sandal
    @Override
    public double hitungHargaAkhir(double pajak) {
        try {
            double harga = super.hitungHargaAkhir(pajak);
            // Sandal memiliki pajak yang lebih rendah jika ada perbaikan sol
            if (repairSole) {
                harga -= (harga * 0.015);
            }
            return harga;
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
            return hitungHarga(); // Return harga tanpa pajak jika ada error
        }
    }
}
```

```java
package model;

public class PesananSneakers extends Pesanan {
    private boolean deepCleaning;

    public PesananSneakers(int id, String namaPelanggan, String jenisSepatu, int jumlahSepatu,
                           JenisPencucian jenisPencucian, boolean deepCleaning) {
        super(id, namaPelanggan, jenisSepatu, jumlahSepatu, jenisPencucian);
        this.deepCleaning = deepCleaning;
        this.hargaDasar = 30000.0;
    }

    // Constructor overloading dengan harga dasar kustom
    public PesananSneakers(int id, String namaPelanggan, String jenisSepatu, int jumlahSepatu,
                           JenisPencucian jenisPencucian, boolean deepCleaning, double hargaDasar) {
        super(id, namaPelanggan, jenisSepatu, jumlahSepatu, jenisPencucian, hargaDasar);
        this.deepCleaning = deepCleaning;
    }

    public boolean isDeepCleaning() {
        return deepCleaning;
    }

    public void setDeepCleaning(boolean deepCleaning) {
        this.deepCleaning = deepCleaning;
    }

    @Override
    public double hitungHarga() {
        double harga = super.hitungHarga(); // Panggil metode parent

        // Tambahan biaya untuk deep cleaning
        if (deepCleaning) {
            harga += 15000.0 * jumlahSepatu;
        }

        return harga;
    }

    // Tidak dapat mengganti hitungHarga(double diskonPersen) karena sudah final di kelas induk

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Deep Cleaning: " + (deepCleaning ? "Ya" : "Tidak"));
    }

    @Override
    public void displayInfo(boolean showPrice) {
        super.displayInfo(showPrice);
        System.out.println("Deep Cleaning: " + (deepCleaning ? "Ya" : "Tidak"));
    }

    @Override
    public String getServiceDetails() {
        return "Sneakers - " + (deepCleaning ? "With Deep Cleaning" : "Standard Cleaning");
    }

    // Override metode dari interface untuk kasus khusus sneakers
    @Override
    public double hitungHargaAkhir(double pajak) {
        try {
            double harga = super.hitungHargaAkhir(pajak);
            // Sneakers memiliki tambahan pajak 1% untuk deep cleaning
            if (deepCleaning) {
                harga += (harga * 0.01);
            }
            return harga;
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
            return hitungHarga(); // Return harga tanpa pajak jika ada error
        }
    }
}
```

```java
package model;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class TransaksiManager {
    // Static variables
    private static List<String> logTransaksi = new ArrayList<>();
    private static final String LOG_FILE = "transaksi_log.txt";

    // Static method untuk mencatat transaksi
    public static void catatTransaksi(Pesanan pesanan, String metodePembayaran, double pajak) {
        try {
            if (metodePembayaran == null || metodePembayaran.trim().isEmpty()) {
                throw new IllegalArgumentException("Metode pembayaran tidak boleh kosong");
            }

            // Hitung total pembayaran dengan pajak
            double totalPembayaran = pesanan.hitungHargaAkhir(pajak);

            // Format log entry
            String logEntry = String.format(
                "ID: %d, Pelanggan: %s, Total: Rp%.2f, Metode: %s, Waktu: %s",
                pesanan.getId(),
                pesanan.getNamaPelanggan(),
                totalPembayaran,
                metodePembayaran,
                Pesanan.formatTanggal(LocalDateTime.now())
            );

            // Tambahkan ke list log
            logTransaksi.add(logEntry);

            // Catat transaksi menggunakan method dari interface
            pesanan.catatTransaksi(metodePembayaran);

            // Simpan ke file
            simpanKeFile(logEntry);

        } catch (IllegalArgumentException e) {
            System.out.println("Error mencatat transaksi: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Terjadi kesalahan saat mencatat transaksi: " + e.getMessage());
        }
    }

    // Static method untuk menyimpan transaksi ke file
    private static void simpanKeFile(String logEntry) {
        try (PrintWriter out = new PrintWriter(new FileWriter(LOG_FILE, true))) {
            out.println(logEntry);
        } catch (IOException e) {
            System.out.println("Error menyimpan log transaksi ke file: " + e.getMessage());
        }
    }

    // Static method untuk mendapatkan semua log transaksi
    public static List<String> getLogTransaksi() {
        return new ArrayList<>(logTransaksi); // Return copy untuk keamanan
    }

    // Static method untuk menghapus semua log (untuk testing)
    public static void resetLog() {
        logTransaksi.clear();
    }
}
```