RESPONSI

PEMROGRAMAN BERORIENTASI OBJEK PRAKTIK

Kelas VIII

T.A. Semester Genap 2024/2025

Disusun Oleh :

Raihan Ramadhan Indratmo

5230411282

PROGRAM STUDI INFORMATIKA

FAKULTAS SAINS & TEKNOLOGI

UNIVERSITAS TEKNOLOGI YOGYAKARTA

YOGYAKARTA

2024

1. Jelaskan Perbedaan Use Case Diagram dan Class Diagram

USE CASE DIAGRAM

- Tujuan
  Menggambarkan hubungan antara user (aktor) dengan sistem agar mencapai tujuan tertentu.
- Elemen Utama
  Use case diagram memiliki beberapa elemen utama diantaranya :
  - Aktor
    Representasi pengguna atau sistem lain yang berinteraksi dengan sistem utama. Aktor dapat berupa pengguna manusia atau sistem eksternal.
  - Use Case
    Representasi dari fungsi atau layanan yang diberikan sistem kepada aktor untuk mencapai tujuan tertentu.
  - Hubungan
    Menghubungkan aktor dengan *use case* melalui hubungan *association*, *include*, dan *extend*.
- Contoh Penggunaan
  Menggambarkan alur skenario seperti "Pengguna Login ke Sistem," "Memesan Produk," atau "Mengelola Profil".
- Hasil
  Memberikan gambaran umum tentang apa yang bisa dilakukan oleh sistem (fitur dan layanan) dan siapa yang menggunakannya. Biasanya digunakan dalam tahap awal analisis untuk mendefinisikan batasan sistem.

CLASS DIAGRAM

- Tujuan
  Menggambarkan struktur statis dari sistem, termasuk class-class yang ada dalam sistem dan hubungan antar class tersebut
- Elemen Utama
  Class diagram memiliki beberapa elemen utama diantaranya :
  - Class
    Menunjukkan objek atau entitas yang ada pada sistem, lengkap dengan atribut dan method (fungsi).
  - Atribut
    Data yang dimiliki oleh class
  - Method
    Fungsi yang bisa dilakukan oleh class

- Hubungan
  Menunjukkan hubungan seperti association, inheritance, aggregation, dan composition antar class

- Contoh Penggunaan
  Menggambarkan class-class seperti “User,” “Product,” dan “Order,” lengkap dengan atributnya seperti “username” atau “harga,” dan method seperti “login()” atau “tambahItem()”
- Hasil
  Memberikan gambaran mendetail tentang struktur data dalam sistem

2. Jelaskan Jenis – Jenis Dependesi

Dalam pemodelan UML, terutama pada Use Case Diagram dan Class Diagram, dependensi mengacu pada hubungan antar elemen yang menunjukkan ketergantungan atau hubungan antar satu elemen dengan elemen lainnya. Berikut beberapa jenis dependensi :

a) Dependency
   Menunjukkan bahwa satu elemen bergantung pada elemen lain untuk menjalankan atau mengimplementasikan fungsinya. Dependensi ini bersifat umum dan sering digunakan untuk menggambarkan bahwa perubahan pada satu elemen dapat memengaruhi elemen yang bergantung padanya. Disimbolkan dengan garis putus-putus dengan panah di salah satu ujungnya
b) Inklusi
   Jenis dependensi ini digunakan dalam Use Case Diagram untuk menunjukkan bahwa sebuah use case membutuhkan use case lain sebagai bagian dari langkah-langkahnya. Dengan kata lain, satu use case mencakup use case lain agar dapat berjalan dengan lengkap. Disimbolkan dengan garis putus-putus dengan label "<<include>>"
c) Extend
   Jenis dependensi ini digunakan untuk menggambarkan kondisi opsional atau tambahan. Jika kondisi tertentu dipenuhi, maka sebuah use case akan “memperluas” fungsionalitas use case utama. Berbeda dengan include yang wajib, extend hanya terjadi dalam situasi tertentu. Disimbolkan dengan garis putus-putus dengan label "<<extend>>"
d) Asosiasi
   Menunjukkan bahwa satu elemen berhubungan dengan elemen lain. Dalam konteks Class Diagram, asosiasi menunjukkan hubungan antara dua class di mana satu objek class dapat berinteraksi dengan objek class lainnya. Dependensi ini bisa bersifat satu arah atau dua arah. Disimbolkan dengan panah (jika asosiasinya satu arah)

e) Agregasi

Jenis asosiasi khusus yang menunjukkan hubungan “memiliki” tetapi bersifat lemah. Class yang mengandung tidak memiliki sepenuhnya objek class lain, sehingga objek tersebut dapat eksis sendiri meskipun objek yang mengandung dihapus. Disimbolkan dengan bentuk belah ketupat kosong di sisi class yang mengandung

3. Apa Perbedaan Pemrograman Terstruktur Dengan Berorientasi Objek, Jelaskan ?

Aspek	**Pemrograman Terstruktur**	**Pemrograman Berorientasi Objek**
Konsep	Berbasis prosedur, membagi program menjadi fungsi atau prosedur yang lebih kecil.	Berbasis objek, membagi program menjadi objek-objek yang memiliki atribut dan method
Fokus Utama	Fokus pada tugas atau fungsi yang harus dilakukan	Fokus pada objek yang merepresentasikan atribut dan method
Struktur Program	Program dibagi menjadi fungsi atau prosedur yang terpisah	Program dibagi menjadi class - class yang merepresentasikan objek
Enkapsulasi	Tidak mendukung enkapsulasi secara langsung (data dan fungsi tidak selalu terikat)	Mendukung enkapsulasi, menggabungkan data dan metode dalam satu class
Pewarisan (Inheritance)	Tidak mendukung pewarisan (setiap fungsi/prosedur harus ditulis ulang)	Mendukung pewarisan, memungkinkan class baru mewarisi sifat dari class yang ada
Contoh Bahasa Pemrograman	C, Pascal, Fortran, COBOL	Java, C++, Python, Ruby, C#
Kelebihan	Lebih sederhana dan cocok untuk program kecil dan sederhana.	Lebih modular, mudah untuk dikembangkan dan dipelihara, terutama untuk program besar.
Kekurangan	Sulit dipelihara dan dikembangkan untuk proyek besar, karena minimnya modularitas.	Lebih kompleks dan membutuhkan pemahaman konsep OOP (object oriented program)

4. Jelaskan Konsep Objek Dan Beri Contohnya ?

Konsep Objek

Konsep objek dalam pemrograman berorientasi objek (Object-Oriented Programming/OOP) adalah elemen atau entitas yang merepresentasikan sesuatu di dunia nyata atau konsep abstrak dalam bentuk yang dapat dioperasikan oleh program. Objek adalah instansiasi dari class yang berisi atribut dan method. Berikut merupakan konsep dasar objek :

1. Atribut
   Atribut adalah data atau karakteristik yang dimiliki oleh objek
2. Method
   Method adalah fungsi yang dapat dilakukan oleh objek
3. Enkapsulasi
   Data dan method disatukan dalam objek, sehingga data hanya bisa diakses atau diubah melalui method yang disediakan.
4. Identitas
   Setiap objek memiliki identitas unik, meskipun atribut dan methodnya mungkin sama. Identitas ini membedakan satu objek dengan objek lainnya.

Contoh

Saya memakai contoh buku. Disini langkah awal membuat cetakan (class) buku untuk objek-objek buku

```
class Buku:
    def __init__(self, judul, penulis, tahun_terbit, genre):
        self.judul = judul              # Atribut judul buku
        self.penulis = penulis          # Atribut penulis buku
        self.tahun_terbit = tahun_terbit  # Atribut tahun terbit buku
        self.genre = genre              # Atribut genre buku

    def info_buku(self):
        print(
                        f"Judul: {self.judul}, Penulis: {self.penulis}, Tahun: {self.tahun_terbit}, Genre: {self.genre}")

    def pinjam(self):
        print(f"Buku '{self.judul}' sedang dipinjam.")
```

Setelah membuat cetakannya, kemudia saya mengisikan beberapa objek buku yang berbeda

```
buku1 = Buku("Laskar Pelangi", "Andrea Hirata", 2005, "Novel")
```

```
buku2 = Buku("Mahir OOP", "Raihan Ramadhan Indratmo", 2024, "Pendidikan")
buku3 = Buku("Lets Go King", "King Zharif", 2021, "Pengembangan Diri")
```

Mencoba mengakses dan melakukan aksi menggunakan method (info_buku) dan (pinjam).

```
buku2.info_buku()
buku2.pinjam()
```

Akan menghasilkan output seperti berikut.

```
Judul: Mahir OOP, Penulis: Raihan Ramadhan Indratmo, Tahun: 2024, Genre: Pendidikan
Buku 'Mahir OOP' sedang dipinjam.
```

5. Jelaskan Jenis – Jenis Access Modifier Beri Contohnya Dalam Baris Pemrograman ?

Dalam pemrograman berorientasi objek, Access Modifier adalah kata kunci yang digunakan untuk menentukan aksesibilitas atau tingkat visibilitas anggota class (seperti atribut dan method) dari luar class tersebut. Access modifier menentukan apakah bagian dari kode tersebut dapat diakses oleh class lain atau hanya oleh class itu sendiri. Berikut adalah tiga jenis access modifier yang umum digunakan :

- Public

  Anggota yang dideklarasikan sebagai public dapat diakses dari mana saja, baik di dalam class itu sendiri maupun dari luar class. Penggunaannya atribut atau metode yang tidak diawali dengan karakter khusus dianggap sebagai *public*.

```
class Mobil:
def __init__(self, merk, model):
self.merk = merk            # Public attribute
self.model = model          # Public attribute

def info_mobil(self):           # Public method
print(f"Mobil: {self.merk} {self.model}")

# Membuat objek dari class Mobil
mobil1 = Mobil("Toyota", "Camry")
print(mobil1.merk)              # Akses public attribute dari luar class
mobil1.info_mobil()             # Akses public method dari luar class
```

```
Toyota
Mobil: Toyota Camry
```

- Protected

  Anggota yang dideklarasikan sebagai protected hanya dapat diakses di dalam class itu sendiri dan class turunannya (class anak atau subclass). Penggunaannya atribut atau metode yang diawali dengan satu underscore (_) dianggap sebagai protected.

```
class Mobil:
    def __init__(self, merk, model):
        self._kecepatan = 0          # Protected attribute
        self.merk = merk
        self.model = model

    def _ubah_kecepatan(self, kecepatan):  # Protected method
        self._kecepatan = kecepatan
            print(f"Kecepatan diubah menjadi {self._kecepatan} km/jam")

# Membuat objek dari class Mobil
mobil1 = Mobil("Toyota", "Camry")
mobil1._ubah_kecepatan(60)          # Akses protected method dari luar class (bisa, tapi tidak disarankan)
```

```
Kecepatan diubah menjadi 60 km/jam
```

- Private

  Anggota yang dideklarasikan sebagai private hanya dapat diakses di dalam class itu sendiri. Anggota ini tidak dapat diakses dari luar class maupun oleh class turunannya. Penggunaannya atribut atau metode yang diawali dengan dua underscore (__) dianggap sebagai private.

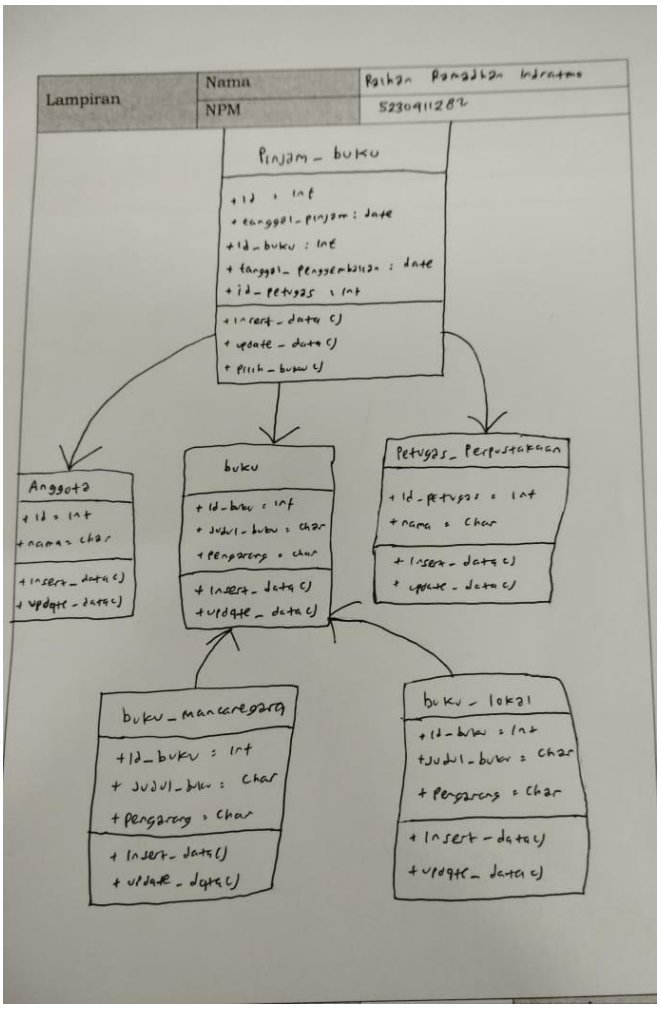
```
class Mobil:
    def __init__(self, merk, model):
        self.__nomor_mesin = "12345ABC"  # Private attribute

    def __info_mesin(self):              # Private method
        print(f"Nomor mesin: {self.__nomor_mesin}")

# Membuat objek dari class Mobil
mobil1 = Mobil("Toyota", "Camry")
# print(mobil1.__nomor_mesin)              # Error: tidak bisa diakses dari luar
# mobil1.__info_mesin()                # Error: tidak bisa diakses dari luar
```

  Akan terjadi error karena tidak bisa mengakses dari luar.

6. Gambarkan Contoh Pewarisan Dalam Diagram Class ?



Penjelasan

1. Struktur Utama:

- Ada 6 class utama yang merepresentasikan entitas dalam sistem perpustakaan
- Setiap class memiliki atribut (properties) dan method (fungsi)

2. Detail Setiap class :

a) Class Anggota:

- Atribut:
  * id (tipe: integer) - untuk identifikasi unik anggota
  * nama (tipe: char) - menyimpan nama anggota
- Method:
  * insert_data() - untuk menambah data anggota baru

  * update_data() - untuk memperbarui data anggota

b) Class buku (Parent Class):

- Atribut:
  * id_buku (tipe: integer)
  * judul_buku (tipe: char)
  * pengarang (tipe: char)
- Method:
  * insert_data()
  * update_data()

c) Class buku_mancanegara (Child Class dari buku):

- Mewarisi semua atribut dan method dari class buku
- Digunakan untuk mengelola buku-buku berbahasa asing

d) Class buku_lokal (Child Class dari buku):

- Mewarisi semua atribut dan method dari class buku
- Digunakan untuk mengelola buku-buku lokal

e) class pinjam_buku:

- Atribut:
  * id (tipe: integer)
  * tanggal_pinjam (tipe: date)
  * id_buku (tipe: integer)
  * tanggal_pengembalian (tipe: date)
  * id_petugas (tipe: integer)
- Method:
  * insert_data()

* update_data()

* pilih_buku()

f) class petugas_perpustakaan:

- Atribut:

* id_petugas (tipe: integer)

* nama (tipe: char)

- Method:

* insert_data()

* update_data()

3. Hubungan Antar Class:

- Inheritance (Pewarisan):

* buku_mancanegara dan buku_lokal mewarisi sifat dari class buku

- Association (Asosiasi):

* class pinjam_buku terhubung dengan:

- class buku (untuk mencatat buku yang dipinjam)

- class petugas_perpustakaan (untuk mencatat petugas yang melayani)

- class anggota (untuk mencatat peminjam)

4. Fungsionalitas:

- Sistem dapat mengelola data anggota perpustakaan

- Dapat mencatat peminjaman buku

- Membedakan antara buku lokal dan buku asing

- Mencatat aktivitas petugas perpustakaan

- Memungkinkan untuk melakukan operasi dasar (insert dan update) pada setiap entitas.

SOAL PRAKTIK

```
class Pegawai:
    def __init__(self, id_pegawai, nama, alamat):
        self.id_pegawai = id_pegawai
        self.nama = nama
        self.alamat = alamat

    def tampil_info(self):
        print(f"ID Pegawai: {self.id_pegawai}")
        print(f"Nama: {self.nama}")
        print(f"Alamat: {self.alamat}")

class Produk:
    def __init__(self, kode_produk, nama_produk, jenis_produk):
        self.kode_produk = kode_produk
        self.nama_produk = nama_produk
        self.jenis_produk = jenis_produk
        self.harga = 0
        self.stok = 0

    def tampil_info(self):
        print(f"Kode Produk: {self.kode_produk}")
        print(f"Nama Produk: {self.nama_produk}")
        print(f"Jenis Produk: {self.jenis_produk}")
        print(f"Harga: Rp {self.harga:,}")
        print(f"Stok: {self.stok}")

class Snack(Produk):
    def __init__(self, kode_produk, nama_snack, harga, stok):
        super().__init__(kode_produk, nama_snack, "Snack")
        self.harga = harga
        self.stok = stok

    def tampil_info(self):
        super().tampil_info()
        print("Kategori: Snack")

class Makanan(Produk):
    def __init__(self, kode_produk, nama_makanan, harga, stok):
        super().__init__(kode_produk, nama_makanan, "Makanan")
        self.harga = harga
```

```
        self.stok = stok

    def tampil_info(self):
        super().tampil_info()
        print("Kategori: Makanan")

class Minuman(Produk):
    def __init__(self, kode_produk, nama_minuman, harga, stok):
        super().__init__(kode_produk, nama_minuman, "Minuman")
        self.harga = harga
        self.stok = stok

    def tampil_info(self):
        super().tampil_info()
        print("Kategori: Minuman")

class Transaksi:
    def __init__(self, no_transaksi):
        self.no_transaksi = no_transaksi
        self.detail_transaksi = []
        self.total_harga = 0

    def tambah_item(self, produk, jumlah):
        if produk.stok >= jumlah:
            self.detail_transaksi.append({
                'produk': produk,
                'jumlah': jumlah,
                'subtotal': produk.harga * jumlah
            })
            self.total_harga += produk.harga * jumlah
            produk.stok -= jumlah
            return True
        return False

    def tampil_struk(self):
        print("\n===== STRUK TRANSAKSI =====")
        print(f"No Transaksi: {self.no_transaksi}")
        print("\nDetail Pembelian:")
        print("-" * 50)
        for item in self.detail_transaksi:
            print(f"Produk: {item['produk'].nama_produk}")
            print(f"Jumlah: {item['jumlah']}")
```

```
            print(f"Harga Satuan: Rp {item['produk'].harga:,}")
            print(f"Subtotal: Rp {item['subtotal']:,}")
            print("-" * 50)
        print(f"\nTotal Harga: Rp {self.total_harga:,}")
        print("==========================")

class SistemManajemen:
    def __init__(self):
        self.daftar_pegawai = []
        self.daftar_produk = []
        self.daftar_transaksi = []
        self.counter_transaksi = 1

    def menu_pegawai(self):
        while True:
            print("\n=== MENU MANAJEMEN PEGAWAI ===")
            print("1. Tambah Pegawai")
            print("2. Tampil Daftar Pegawai")
            print("3. Pecat Pegawai")
            print("4. Kembali ke Menu Utama")

            pilihan = input("\nPilih menu (1-4): ")

            if pilihan == "1":
                id_pegawai = input("Masukkan ID Pegawai: ")
                nama = input("Masukkan Nama Pegawai: ")
                alamat = input("Masukkan Alamat: ")
                self.tambah_pegawai(id_pegawai, nama, alamat)

            elif pilihan == "2":
                self.tampil_daftar_pegawai()

            elif pilihan == "3":
                self.tampil_daftar_pegawai()
                id_pegawai = input("Masukkan ID Pegawai yang akan dipecat: ")
                self.pecat_pegawai(id_pegawai)

            elif pilihan == "4":
                break

            else:
                print("Pilihan tidak valid")
```

```
def menu_produk(self):
    while True:
        print("\n=== MENU MANAJEMEN PRODUK ===")
        print("1. Tambah Produk")
        print("2. Tampil Daftar Produk")
        print("3. Kembali ke Menu Utama")

        pilihan = input("\nPilih menu (1-3): ")

        if pilihan == "1":
            print("\nJenis Produk:")
            print("1. Snack")
            print("2. Makanan")
            print("3. Minuman")
            jenis = input("Pilih jenis produk (1-3): ")

            jenis_map = {"1": "snack", "2": "makanan", "3": "minuman"}
            if jenis in jenis_map:
                kode = input("Masukkan Kode Produk: ")
                nama = input("Masukkan Nama Produk: ")
                try:
                    harga = int(input("Masukkan Harga: "))
                    stok = int(input("Masukkan Jumlah Stok: "))
                    self.tambah_produk(jenis_map[jenis], kode, nama, harga, stok)
                except ValueError:
                    print("Harga dan stok harus berupa angka")
            else:
                print("Pilihan tidak valid")

        elif pilihan == "2":
            self.tampil_daftar_produk()

        elif pilihan == "3":
            break

        else:
            print("Pilihan tidak valid")

def menu_transaksi(self):
    while True:
        print("\n=== MENU TRANSAKSI ===")
```

```
            print("1. Buat Transaksi Baru")
            print("2. Kembali ke Menu Utama")

            pilihan = input("\nPilih menu (1-2): ")

            if pilihan == "1":
                transaksi = self.buat_transaksi()
                while True:
                    self.tampil_daftar_produk()
                    kode_produk = input("\nMasukkan kode produk (atau 'selesai' untuk
mengakhiri): ")

                    if kode_produk.lower() == 'selesai':
                        break

                    produk = self.cari_produk(kode_produk)
                    if produk:
                        try:
                            jumlah = int(input("Masukkan jumlah: "))
                            if jumlah <= 0:
                                print("Jumlah harus lebih dari 0")
                                continue

                            if produk.stok < jumlah:
                                print(f"Stok tidak mencukupi. Stok tersedia: {produk.stok}")
                                continue

                            if transaksi.tambah_item(produk, jumlah):
                                print("Produk berhasil ditambahkan ke transaksi")
                            else:
                                print("Gagal menambahkan produk")
                        except ValueError:
                            print("Jumlah harus berupa angka")
                    else:
                        print("Produk tidak ditemukan")

                transaksi.tampil_struk()

            elif pilihan == "2":
                break

            else:
```

```
            print("Pilihan tidak valid")

    # Method lainnya tetap sama
    def tambah_pegawai(self, id_pegawai, nama, alamat):
        pegawai = Pegawai(id_pegawai, nama, alamat)
        self.daftar_pegawai.append(pegawai)
        print(f"Pegawai {nama} berhasil ditambahkan")

    def tampil_daftar_pegawai(self):
        print("\n=== DAFTAR PEGAWAI ===")
        if not self.daftar_pegawai:
            print("Belum ada pegawai terdaftar")
            return
        for pegawai in self.daftar_pegawai:
            pegawai.tampil_info()
            print()

    def pecat_pegawai(self, id_pegawai):
        for pegawai in self.daftar_pegawai:
            if pegawai.id_pegawai == id_pegawai:
                self.daftar_pegawai.remove(pegawai)
                print(f"Pegawai dengan ID {id_pegawai} berhasil dipecat")
                return
        print(f"Pegawai dengan ID {id_pegawai} tidak ditemukan")

    def tambah_produk(self, jenis, kode_produk, nama, harga, stok):
        if jenis.lower() == "snack":
            produk = Snack(kode_produk, nama, harga, stok)
        elif jenis.lower() == "makanan":
            produk = Makanan(kode_produk, nama, harga, stok)
        elif jenis.lower() == "minuman":
            produk = Minuman(kode_produk, nama, harga, stok)
        else:
            print("Jenis produk tidak valid")
            return

        self.daftar_produk.append(produk)
        print(f"{jenis.capitalize()} {nama} berhasil ditambahkan")

    def buat_transaksi(self):
        transaksi = Transaksi(f"TRX{self.counter_transaksi:03d}")
        self.counter_transaksi += 1
```

```
        self.daftar_transaksi.append(transaksi)
        return transaksi

    def tampil_daftar_produk(self):
        print("\n=== DAFTAR PRODUK ===")
        if not self.daftar_produk:
            print("Belum ada produk terdaftar")
            return
        for produk in self.daftar_produk:
            produk.tampil_info()
            print()

    def cari_produk(self, kode_produk):
        for produk in self.daftar_produk:
            if produk.kode_produk == kode_produk:
                return produk
        return None

def main():
    sistem = SistemManajemen()

    # Menambahkan beberapa data awal
    sistem.tambah_pegawai("P001", "Joko Sigma Sejati", "Jakarta")
    sistem.tambah_produk("snack", "SNK-001", "Potato Chips", 8000, 50)
    sistem.tambah_produk("makanan", "MKN-001", "Nasi Goreng", 15000, 30)
    sistem.tambah_produk("minuman", "MNM-001", "Es Teh", 5000, 100)

    while True:
        print("\n=== SISTEM MANAJEMEN PRODUK DAN TRANSAKSI ===")
        print("1. Manajemen Pegawai")
        print("2. Manajemen Produk")
        print("3. Manajemen Transaksi")
        print("4. Keluar")

        pilihan = input("\nPilih menu (1-4): ")

        if pilihan == "1":
            sistem.menu_pegawai()
        elif pilihan == "2":
            sistem.menu_produk()
        elif pilihan == "3":
            sistem.menu_transaksi()
```

```
        elif pilihan == "4":
            print("Terima kasih telah menggunakan sistem ini")
            break
        else:
            print("Pilihan tidak valid")

if __name__ == "__main__":
    main()
```