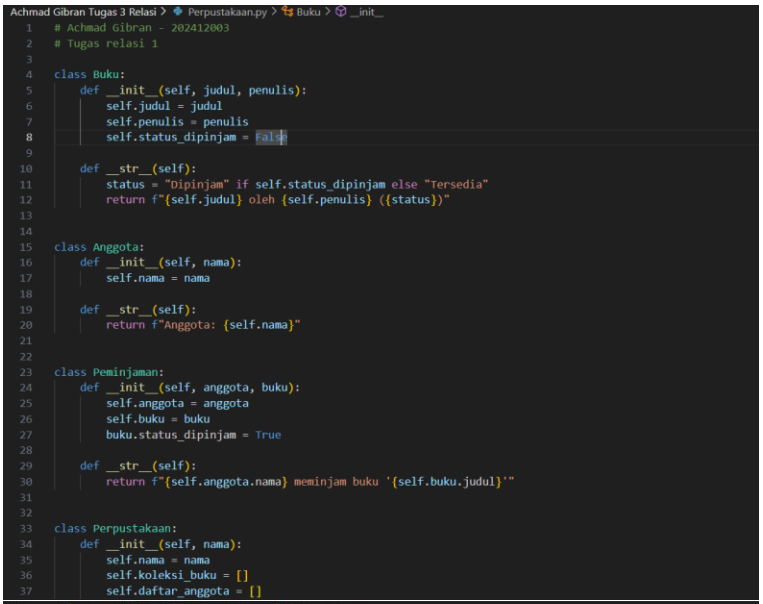
Nama: Achmad Gibran

Nim: 202412003

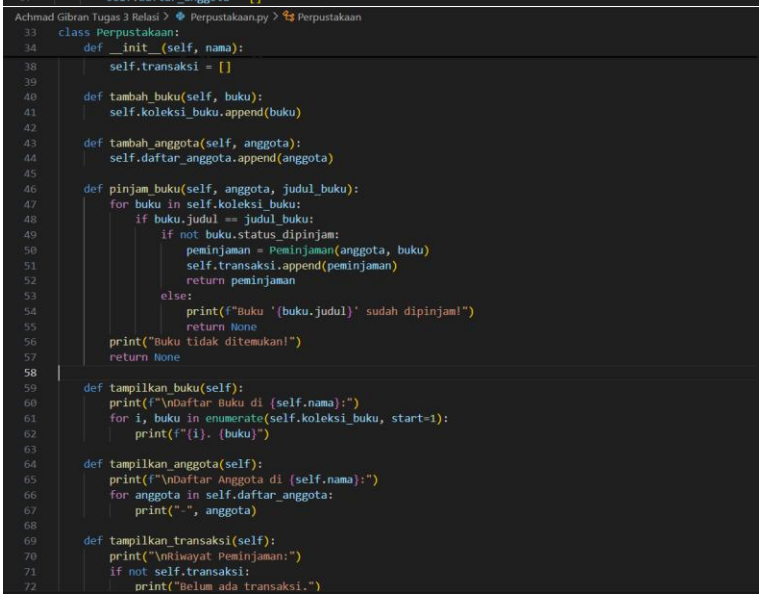
Mata Kuliah: Pemrograman Berorientasi Objek

Tugas Relasi

1. Perpustakaan
    a. SourceCode



```python
# Achmad Gibran - 202412003
# Tugas relasi 1

class Buku:
    def __init__(self, judul, penulis):
        self.judul = judul
        self.penulis = penulis
        self.status_dipinjam = False

    def __str__(self):
        status = "Dipinjam" if self.status_dipinjam else "Tersedia"
        return f"{self.judul} oleh {self.penulis} ({status})"


class Anggota:
    def __init__(self, nama):
        self.nama = nama

    def __str__(self):
        return f"Anggota: {self.nama}"


class Peminjaman:
    def __init__(self, anggota, buku):
        self.anggota = anggota
        self.buku = buku
        buku.status_dipinjam = True

    def __str__(self):
        return f"{self.anggota.nama} meminjam buku '{self.buku.judul}'"


class Perpustakaan:
    def __init__(self, nama):
        self.nama = nama
        self.koleksi_buku = []
        self.daftar_anggota = []
```



```python
class Perpustakaan:
    def __init__(self, nama):
        self.transaksi = []

    def tambah_buku(self, buku):
        self.koleksi_buku.append(buku)

    def tambah_anggota(self, anggota):
        self.daftar_anggota.append(anggota)

    def pinjam_buku(self, anggota, judul_buku):
        for buku in self.koleksi_buku:
            if buku.judul == judul_buku:
                if not buku.status_dipinjam:
                    peminjaman = Peminjaman(anggota, buku)
                    self.transaksi.append(peminjaman)
                    return peminjaman
                else:
                    print(f"Buku '{buku.judul}' sudah dipinjam!")
                    return None
        print("Buku tidak ditemukan!")
        return None

    def tampilkan_buku(self):
        print(f"\nDaftar Buku di {self.nama}:")
        for i, buku in enumerate(self.koleksi_buku, start=1):
            print(f"{i}. {buku}")

    def tampilkan_anggota(self):
        print(f"\nDaftar Anggota di {self.nama}:")
        for anggota in self.daftar_anggota:
            print("-", anggota)

    def tampilkan_transaksi(self):
        print("\nRiwayat Peminjaman:")
        if not self.transaksi:
            print("Belum ada transaksi.")
```

```
        for transaksi in self.transaksi:
            print("-", transaksi)


# PENGGUNAAN
perpus = Perpustakaan("Perpustakaan Kota")

# Tambah buku (2 pilihan saja)
perpus.tambah_buku(Buku("Laskar Pelangi", "Andrea Hirata"))
perpus.tambah_buku(Buku("Bumi Manusia", "Pramoedya Ananta Toer"))

# Tambah anggota (Bram dan Gibran)
bram = Anggota("Bram")
gibran = Anggota("Gibran")
perpus.tambah_anggota(bram)
perpus.tambah_anggota(gibran)

# Bram otomatis pinjam buku pertama
perpus.pinjam_buku(bram, "Laskar Pelangi")

# Tampilkan daftar buku sebelum Gibran memilih
perpus.tampilkan_buku()

# Gibran memilih buku yang tersisa
pilihan = int(input("\nGibran, pilih nomor buku yang ingin dipinjam: "))

if pilihan in [1, 2]:
    buku_dipilih = perpus.koleksi_buku[pilihan - 1].judul
    perpus.pinjam_buku(gibran, buku_dipilih)
else:
    print("Buku tidak tersedia.")

# Tampilkan hasil akhir
perpus.tampilkan_buku()
perpus.tampilkan_anggota()
perpus.tampilkan_transaksi()
```

b. Output

```
Daftar Buku di Perpustakaan Kota:
1. Laskar Pelangi oleh Andrea Hirata (Dipinjam)
2. Bumi Manusia oleh Pramoedya Ananta Toer (Tersedia)

Gibran, pilih nomor buku yang ingin dipinjam: 2

Daftar Buku di Perpustakaan Kota:
1. Laskar Pelangi oleh Andrea Hirata (Dipinjam)
2. Bumi Manusia oleh Pramoedya Ananta Toer (Dipinjam)

Daftar Anggota di Perpustakaan Kota:
- Anggota: Bram
- Anggota: Gibran

Riwayat Peminjaman:
- Bram meminjam buku 'Laskar Pelangi'
- Gibran meminjam buku 'Bumi Manusia'
```

c. Analisis

Kelas Perpustakaan menunjukkan agregasi, di mana ia mengelola koleksi independen dari objek Buku dan Anggota (koleksi ini tetap ada meskipun perpustakaan dibubarkan). Relasi Asosiasi diimplementasikan melalui kelas perantara Peminjaman, yang secara eksplisit menghubungkan satu objek Anggota dengan satu objek Buku, sekaligus memicu mutasi state pada objek Buku dengan mengubah status_dipinjam menjadi True. Struktur ini memastikan bahwa semua aktivitas utama sistem, mulai dari penambahan entitas (tambah_buku, tambah_anggota) hingga eksekusi transaksi (pinjam_buku), dikelola secara terpusat oleh objek Perpustakaan, menjaga integritas data status peminjaman pada setiap objek Buku yang terlibat.

2. E-Commerse

a. SourceCode

b. Output

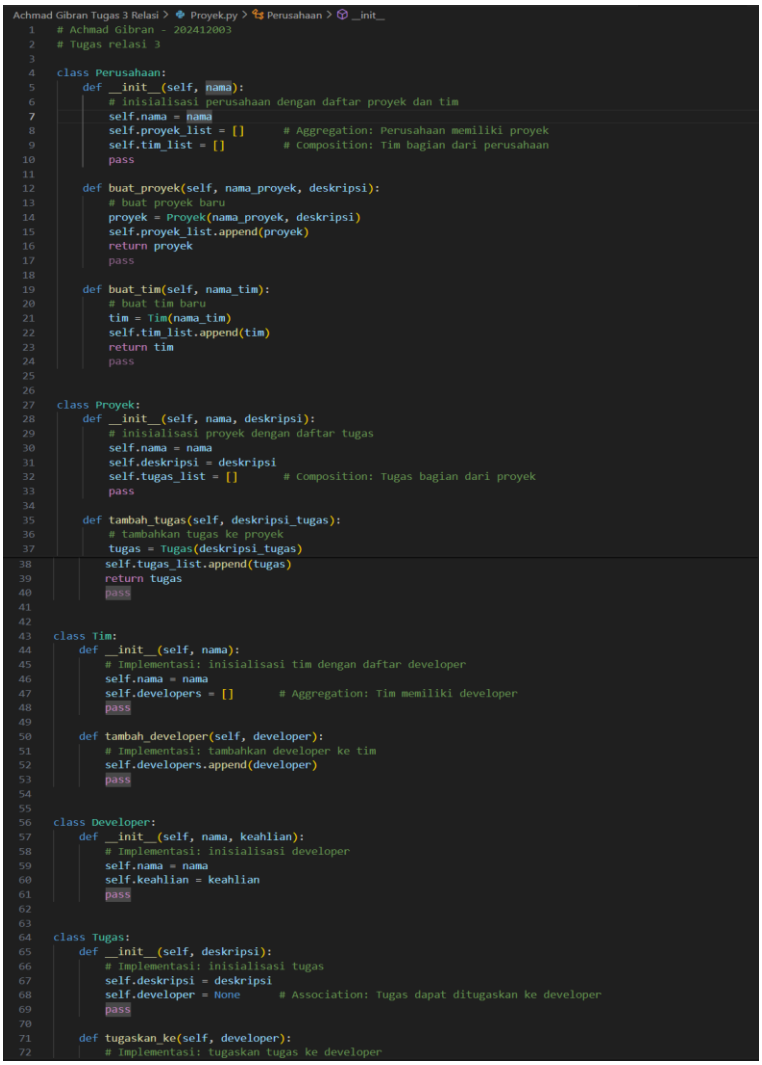
```
Pesanan dari Gibran berhasil dibuat.
Total Harga: Rp 110000
Stok Buku sekarang: 8
PS C:\Users\ACER NITRO\Documents\Achmad_Gibran_Tugas>
```

c. Analisis

Pelanggan memiliki Keranjang dan Pesanan terdiri dari ItemPesanan. Sementara itu, Asosiasi diimplementasikan melalui rujukan silang yang tidak terikat kepemilikan, di mana ItemPesanan merujuk pada Produk dan Pesanan merujuk balik pada Pelanggan. Metode kunci Pelanggan.buat_pesanan bertindak sebagai controller utama yang mengelola transisi state data: dari daftar sementara di Keranjang menjadi data permanen di objek Pesanan dan riwayat_pesanan, sambil secara esensial menjalankan proses bisnis dengan mengurangi stok pada objek Produk yang terasosiasi.

3. Proyek

a. SourceCode



```
        self.developer = developer
        pass


# ========================
# PROGRAM UTAMA (TESTING)
# ========================
if __name__ == "__main__":
    # 1. Membuat Perusahaan
    perusahaan = Perusahaan("Tech Innovators")

    # 2. Membuat tim dan menambah developer
    tim_backend = perusahaan.buat_tim("Backend")
    dev1 = Developer("Bram", "Backend Developer")
    dev2 = Developer("Zaldy", "Database Specialist")
    tim_backend.tambah_developer(dev1)
    tim_backend.tambah_developer(dev2)

    tim_frontend = perusahaan.buat_tim("Frontend")
    dev3 = Developer("Gibran", "Frontend Developer")
    tim_frontend.tambah_developer(dev3)

    # 3. Membuat proyek dan menambah tugas
    proyek1 = perusahaan.buat_proyek("Sistem E-Commerce", "Membuat platform belanja online")
    tugas1 = proyek1.tambah_tugas("Membuat API Produk")
    tugas2 = proyek1.tambah_tugas("Mendesain Halaman Utama")

    # 4. Menugaskan tugas ke developer
    tugas1.tugaskan_ke(dev1)
    tugas2.tugaskan_ke(dev3)

    # 5. Menampilkan status proyek dan tugas
    print(f"Perusahaan: {perusahaan.nama}")
    print("\nDaftar Tim dan Developer:")
    for tim in perusahaan.tim_list:
        print(f"- Tim {tim.nama}:")
        for dev in tim.developers:
            print(f"   • {dev.nama} ({dev.keahlian})")

    print("\nDaftar Proyek dan Tugas:")
    for proyek in perusahaan.proyek_list:
        print(f"- Proyek: {proyek.nama} ({proyek.deskripsi})")
        for tugas in proyek.tugas_list:
            if tugas.developer:
                print(f"   • Tugas: {tugas.deskripsi} → Dikerjakan oleh {tugas.developer.nama}")
            else:
                print(f"   • Tugas: {tugas.deskripsi} → Belum ditugaskan")

```

b. Output

```
Daftar Proyek dan Tugas:
- Proyek: Sistem E-Commerce (Membuat platform belanja online)
   • Tugas: Membuat API Produk → Dikerjakan oleh Bram
   • Tugas: Mendesain Halaman Utama → Dikerjakan oleh Gibran
```

c. Analisis

Tim dan Proyek terikat pada Perusahaan, dan Tugas terikat pada Proyek. Sementara itu, Agregasi diimplementasikan pada level yang lebih rendah, seperti Tim yang mengelola kumpulan Developer independen. Hubungan fungsional utama diwujudkan melalui Asosiasi dinamis, khususnya saat objek Tugas menetapkan referensi ke objek Developer melalui metode tugaskan_ke. Desain ini menjamin aliran kerja yang terpusat dan terstruktur, di mana controller utama (Perusahaan dan Proyek) mengelola siklus hidup elemen turunan (Tim, Tugas), sementara Developer dan Proyek berpartisipasi dalam relasi yang lebih fleksibel.