LAPORAN PRAKTIKUM PEMOGRAMAN BERORIENTASI OBYEK

Nama : **ILMA MA'RIFATUL QURANA**

NPM : **2213020195**

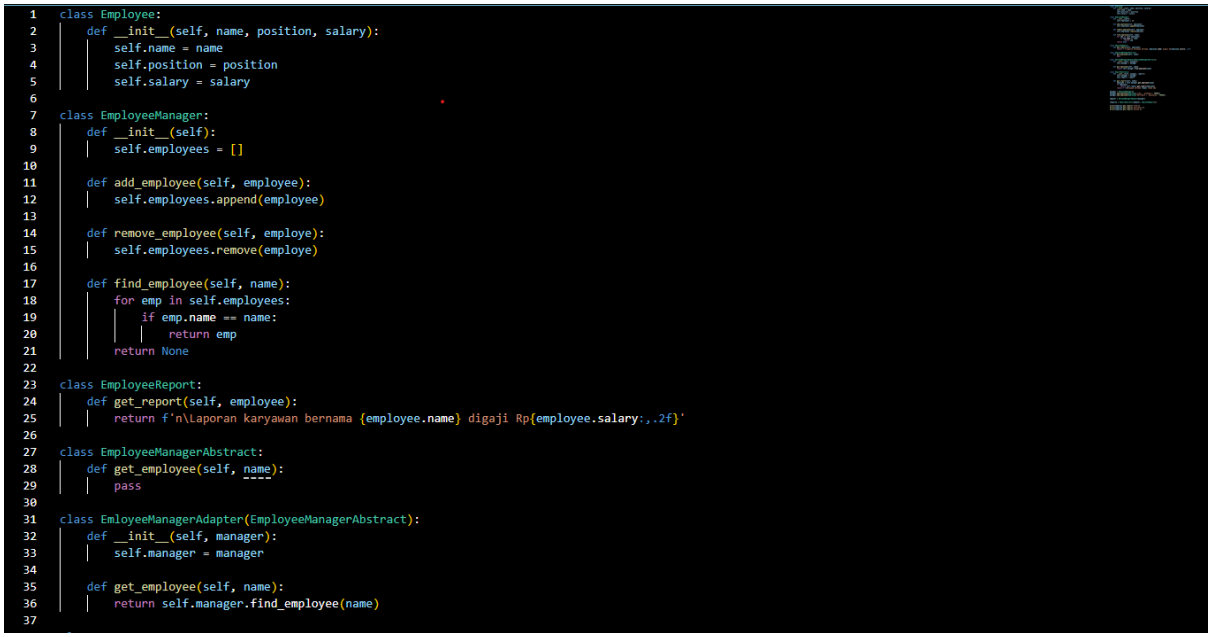
Program Studi : **TEKNIK INFORMATIKA**

FAKULTAS TEKNIK PRODI INFORMATIKA

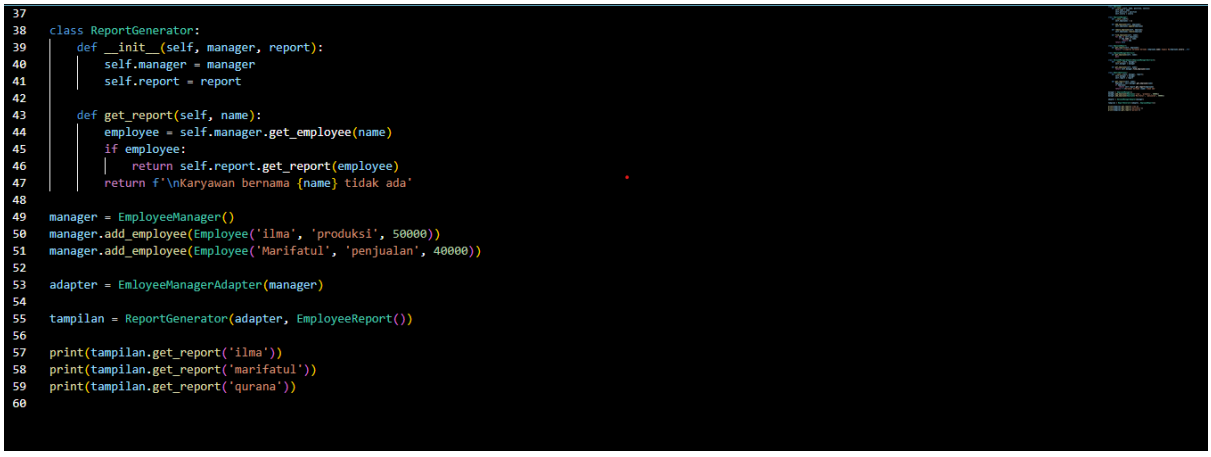
UNIVERSITAS NUSANTARA KEDIRI

2023

Penerapan S.O.L.I.D



```python
class Employee:
    def __init__(self, name, position, salary):
        self.name = name
        self.position = position
        self.salary = salary

class EmployeeManager:
    def __init__(self):
        self.employees = []

    def add_employee(self, employee):
        self.employees.append(employee)

    def remove_employee(self, employe):
        self.employees.remove(employe)

    def find_employee(self, name):
        for emp in self.employees:
            if emp.name == name:
                return emp
        return None

class EmployeeReport:
    def get_report(self, employee):
        return f'n\Laporan karyawan bernama {employee.name} digaji Rp{employee.salary:,.2f}'

class EmployeeManagerAbstract:
    def get_employee(self, name):
        pass

class EmloyeeManagerAdapter(EmployeeManagerAbstract):
    def __init__(self, manager):
        self.manager = manager

    def get_employee(self, name):
        return self.manager.find_employee(name)

```



```python

class ReportGenerator:
    def __init__(self, manager, report):
        self.manager = manager
        self.report = report

    def get_report(self, name):
        employee = self.manager.get_employee(name)
        if employee:
            return self.report.get_report(employee)
        return f'\nKaryawan bernama {name} tidak ada'

manager = EmployeeManager()
manager.add_employee(Employee('ilma', 'produksi', 50000))
manager.add_employee(Employee('Marifatul', 'penjualan', 40000))

adapter = EmloyeeManagerAdapter(manager)

tampilan = ReportGenerator(adapter, EmployeeReport())

print(tampilan.get_report('ilma'))
print(tampilan.get_report('marifatul'))
print(tampilan.get_report('qurana'))

```

Implentasi prinsip S.O.L.I.D
1. Single Responsibility Principle (SRP):
Setiap kelas tampak memiliki satu tanggung jawab, seperti `Employee` yang hanya merepresentasikan karyawan dan `EmployeeManager` yang mengelola karyawan.

2. Open/Closed Principle (OCP):
Kelas-kelas terlihat dapat diperluas tanpa harus mengubah kode sumber yang sudah ada. Misalnya, Anda bisa membuat tipe laporan baru tanpa mengubah kode yang sudah ada.

3. Liskov Substitution Principle (LSP):
Penggantian antara `EmployeeReport` dan `SalaryReport` dalam `ReportGenerator` tampak sesuai, karena keduanya merupakan turunan dari `EmployeeReport`.

4. Interface Segregation Principle (ISP):
Meskipun tidak ada antarmuka eksplisit, setiap kelas tampak fokus pada tanggung jawabnya masing-masing, sesuai dengan prinsip ISP.

5. Dependency Inversion Principle (DIP):
`EmployeeManagerAdapter` mengikuti DIP dengan bergantung pada abstraksi (`EmployeeManagerAbstrak`) daripada implementasi konkret.

Tujuan dari setiap class
`class Employee:`: Kelas ini dibuat untuk menjelaskan objek karyawan. Tiap objek dari kelas ini menyimpan informasi tentang satu karyawan

`class EmployeeManager:`: Kelas ini berfungsi sebagai pengelola karyawan. Dengan membuat objek dari kelas ini, kita dapat menambahkan, menghapus, dan mencari karyawan dalam daftar. Ini membantu dalam pengelolaan koleksi karyawan secara efektif

`class EmployeeReport:`: Kelas ini berfungsi untuk menghasilkan laporan karyawan. Meskipun contohnya hanya menunjukkan laporan dasar, ide di baliknya adalah untuk memisahkan fungsi pembuatan laporan dari kelas utama `Employee`. Ini mengikuti prinsip desain perangkat lunak yang disebut "single responsibility," di mana setiap kelas sebaiknya memiliki satu tanggung jawab utama.

`class SalaryReport(EmployeeReport):`: Kelas ini mewarisi fungsionalitas dari `EmployeeReport` dan memberikan implementasi yang lebih khusus untuk laporan gaji.

`class EmployeeManagerAbstrak:`: Kelas ini mendefinisikan antarmuka atau kontrak umum untuk manajer karyawan.

`class EmployeeManagerAdapter:`: Kelas ini berfungsi sebagai adaptor untuk mengesekusi antarmuka manajer karyawan yang sebenarnya (`EmployeeManager`) ke dalam antarmuka yang diharapkan (`EmployeeManagerAbstrak`). Ini memungkinkan penggunaan kelas yang berbeda dengan antarmuka yang seragam.

`class ReportGenerator:`: Kelas ini bertanggung jawab untuk menghasilkan laporan menggunakan objek `EmployeeReport` atau subclassnya

penjelasan tiap line

- Line 1 : Ini adalah penggunaan kelas dalam Python. Kelas ini digunakan untuk membuat objek yang merepresentasikan seorang karyawan. Sebuah objek karyawan yang memiliki atribut seperti nama (`name`), posisi (`position`), dan gaji (`salary`)

- Line 2: Metode `__init__` adalah konstruktor kelas. Ketika objek karyawan dibuat (`Employee('Yoga', 'Produksi', 50000)`), metode ini secara otomatis dipanggil untuk menginisialisasi atribut-atribut objek

- Line 3 : menetapkan nilai yang diterima sebagai `name` ke atribut `name` dari objek karyawan.

- Line 4 : Sama seperti yang sebelumnya, ini menetapkan nilai `position` ke atribut `position`

- Line 5 : menetapkan nilai `salary` ke atribut `salary`

- Line 7 : definisi dari kelas lain yang bertanggung jawab untuk mengelola objek-objek karyawan.

- Line 8 : Konstruktor untuk `EmployeeManager`. Saat objek manajer dibuat (`manager = EmployeeManager()`), daftar `employee` (akan menyimpan objek-objek karyawan) dibuat

- Line 9 : atribut daftar kosong untuk menyimpan objek-objek karyawan.

- Line 11 : Metode ini memungkinkan untuk menambahkan objek karyawan ke dalam daftar `employee`

- Line 14 : Metode ini memungkinkan untuk menghapus objek karyawan dari daftar `employee`

- Line 15 : Melakukan penghapusan objek karyawan tertentu dari daftar

- • Line 17 : Metode ini memungkinkan untuk menemukan objek karyawan berdasarkan nama dalam daftar `employee`

- Line 18 : Melakukan iterasi melalui setiap objek karyawan dalam daftar.
- Line 19 : Memeriksa apakah atribut `name` dari objek saat ini cocok dengan nama
- yang dicari.
- Line 20 : Jika ditemukan, metode mengembalikan objek karyawan yang cocok
- Line 21 : Jika tidak ada yang cocok, metode mengembalikan `None`
- Line 23 : definisi kelas untuk membuat laporan karyawan.
- Line 24 : Metode ini menghasilkan string laporan yang diformat berdasarkan objek karyawan yang diberikan
- Line 25 : Mengembalikan string laporan dengan informasi nama, posisi, dan gaji karyawan
- Line 27: merupakan kelas abstrak yang terkait dengan pengelolaan karyawan
- Line 28:merupakan bagian dari suatu kelas untuk mengembalikan informasi karyawan
- Line 29: dilewati

- Line 31: Mendefinisikan kelas `EmployeeManagerAdapter` yang mengimplementasikan metode abstrak `getEmployee` dari kelas `EmployeeManagerAbstrak`
- Line 32: Konstruktor untuk `EmployeeManagerAdapter` yang menerima instance dari `EmployeeManager` sebagai parameter
- Line 33: Menetapkan instance `EmployeeManager` ke atribut `manager` dalam `EmployeeManagerAdapter`
- Line 35: Implementasi metode `getEmployee` yang menggunakan metode `find` dari objek `manager` (instance dari `EmployeeManager`) untuk mencari objek karyawan berdasarkan nama
- Line 36: Mengembalikan hasil penemuan objek `Employee` berdasarkan nama
- Line 38: Mendefinisikan kelas `ReportGenerator` yang bertanggung jawab menghasilkan laporan berdasarkan objek `EmployeeManager` dan `EmployeeReport`.
- Line 39: Konstruktor untuk `ReportGenerator` yang menerima instance dari `EmployeeManager` dan `EmployeeReport` sebagai parameter.
- Line 40: : Menetapkan instance `EmployeeManager` ke atribut `manager` dalam `ReportGenerator`
- Line 41: Menetapkan instance `EmployeeReport` ke atribut `report` dalam `ReportGenerator`
- Line 43: Metode untuk menghasilkan laporan berdasarkan nama karyawan.
- Line 44: Menggunakan metode `getEmployee` dari objek `manager` untuk mendapatkan objek `Employee` berdasarkan nama.
- Line 45: Memeriksa apakah objek `Employee` berhasil ditemukan.
- Line 46: Menghasilkan laporan menggunakan metode `getReport` dari objek `report`.
- Line 47: : Mengembalikan pesan jika karyawan tidak ditemukan.
- Line 49: : Membuat instance dari `EmployeeManager` dengan nama `manager`.
- Line 50: Menambahkan objek `Employee` ke dalam `manager`.
- Line 51: Menambahkan objek `Employee` lainnya ke dalam `manager`
- Line 53: Membuat instance dari `EmployeeManagerAdapter` dengan menggunakan `manager` yang sudah ada
- Line 55: menampilan reportGenerator
- Line 57: mencetak laporan untuk karyawan yang Bernama ‘yoga’
- Line 58: mencetak laporan untuk karyawan yang bernama’firnanda’
- Line 59: mencetak laporan untuk karyawan yang bernama’wicaksono’ atau pesan bahwa karyawan tidak ditemukan jika tidak ada yang cocok.

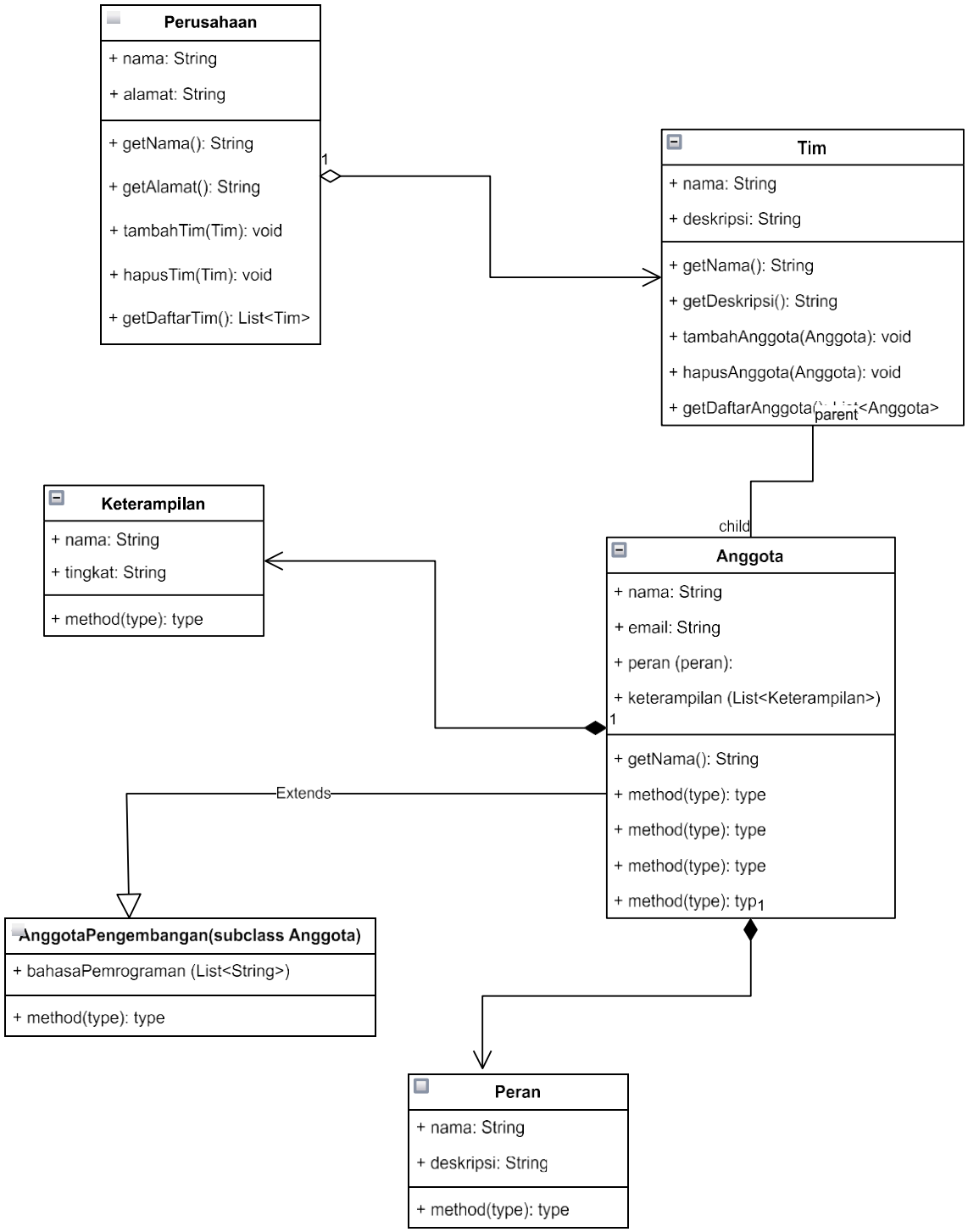
Outputan

Laporan Karyawan Bernama ilma Rp.50,000.00

Laporan Karyawan Bernama marifatul Rp.40,000.00

Laporan Karyawan Bernama qurana Tidak Ada

CLASS DIAGRAM



Penjabaran tiap kelas

- Perusahaan: Perusahaan memiliki nama dan alamat. Perusahaan juga memiliki beberapa tim.
- Tim: Setiap tim memiliki nama dan deskripsi. Tim juga memiliki beberapa anggota
- Anggota: Setiap anggota memiliki nama dan email. Anggota juga memiliki peran dalam tim dan beberapa keterampilan.
- Peran: Setiap peran memiliki nama dan deskripsi.
- Keterampilan: Setiap keterampilan memiliki nama dan tingkat (misalnya, pemula, menengah, mahir).

Hubungan antar kelas

- Asosiasi antara Tim dan Anggota (setiap tim memiliki beberapa anggota dan setiap anggota adalah bagian dari satu tim)
- Agregasi antara Perusahaan dan Tim (setiap perusahaan memiliki beberapa tim, tetapi tim dapat ada tanpa Perusahaan)
- Komposisi antara Anggota dan Peran (setiap anggota memiliki satu peran dan peran tidak dapat ada tanpa anggota).
- Pewarisan antara kelas baru "AnggotaPengembangan" dan "Anggota" (AnggotaPengembangan adalah jenis khusus dari Anggota)