NAMA : Muhammad Irfan Baihaqi

NPM : 5230411327

Kelas : VIII

Prodi : Informatika

RESPONSI 1

1. Perbedaan use case diagram dan Class diagram:
   - Dari segi tujuan:
     Use case diagram menggambarkan interaksi antara aktor dengan system. Sedangkan, class diagram menggambarkan struktur statis dalam system, kelas, atribut dan metode beserta interaksi antar kelas-kelas di dalamnya.
   - Komponen Utama
     Use case memiliki komponen utama , seperti Aktor, Use Case, dan hubungan di antara mereka (association, include, dan extend.). Sedangkan, class diagram berisi kelas , atribut, method, dan interaksi antar kelas (association, aggregation, composition, dan inheritance).
   - Manfaat
     Use case membantu memahami kebutuhan fungsional dari sudut pandang pengguna. Diagram ini cocok untuk komunikasi awal dengan stakeholder karena lebih fokus pada “apa” yang harus dilakukan system sesuai kebutuhan pengguna. Sedangkan class diagram, Membantu desainer dan pengembang memahami struktur data dan keterkaitan antar komponen dalam sistem. Diagram ini lebih berfokus pada “bagaimana” sistem dibangun secara teknis.
2. Jenis – Jenis dependency:
   - Usage
     Menunjukkan bahwa satu elemen menggunakan atau membutuhkan elemen lain untuk melaksanakan suatu tugas atau fungsionalitas. Biasanya digunakan ketika satu kelas menggunakan metode dari kelas lain. Contoh, Kelas A menggunakan metode dari kelas B, maka kelas A bergantung pada kelas B untuk melakukan tugas tersebut.
   - Abstaraction dependency
     Menunjukkan bahwa satu elemen merupakan abstraksi atau generalisasi dari elemen lain. Hubungan ini sering muncul dalam relasi antara model konseptual dan im
     plementasi. Contohnya, kelas Interface yang digunakan untuk mengabstraksi kelas implementasi tertentu.
   - Realization
     Realization dalam UML menunjukkan bahwa sebuah elemen, seperti kelas atau komponen, memenuhi kontrak yang ditentukan oleh elemen lain, biasanya sebuah antarmuka atau spesifikasi. Ini berarti kelas atau komponen

tersebut mengimplementasikan semua operasi yang didefinisikan oleh antarmuka, memastikan fungsionalitas yang dijanjikan dapat dijalankan. Misalnya, jika ada antarmuka "Kendaraan" dengan metode "berjalan", kelas "Mobil" yang merealisasikan antarmuka ini harus mengimplementasikan metode "berjalan" tersebut, menjamin bahwa "Mobil" benar-benar bisa berjalan sesuai kontrak yang ditentukan oleh "Kendaraan".

- Refinement Dependancy
  Refinement dependency dalam UML menggambarkan bagaimana suatu elemen model, seperti use case atau class, merupakan penyempurnaan atau detail lebih lanjut dari elemen lain. Bayangkan ketika punya desain awal sistem yang masih kasar, seperti sketsa. Refinement ini mirip dengan menggambar ulang sketsa itu dengan lebih banyak detail, menunjukkan bagaimana elemen-elemen di dalamnya bekerja lebih spesifik. Misalnya, dari konsep umum "kelola data mahasiswa", refinement bisa menjadi spesifik, seperti "input data mahasiswa", "update data", dan "hapus data". Dengan begitu, kita bisa melihat perkembangan detail dari ide besar ke implementasi yang lebih konkret.
- Trace dependency
  Trace dependency dalam UML menunjukkan bagaimana elemen-elemen model dari fase pengembangan yang berbeda saling berhubungan. Seperti saat membuat diagram alur dari kebutuhan awal sistem, lalu di akhir proyek kita punya implementasi kode. Trace dependency membantu menelusuri bagaimana kebutuhan awal itu diubah menjadi implementasi konkret di kode. Jadi, kita bisa melihat jejak atau "trace" dari ide awal hingga jadi program yang berjalan, memastikan semua kebutuhan awal terakomodasi dengan baik di hasil akhir.
- Permission Dependancy
  Permission dependency dalam UML menunjukkan bahwa satu paket atau modul dalam sistem diperbolehkan untuk mengakses elemen-elemen dari paket atau modul lain. Seperti ketika ada dua folder dalam proyek, satu berisi fungsi-fungsi untuk memanipulasi data dan satu lagi berisi antarmuka pengguna. Permission dependency memastikan bahwa fungsi-fungsi di folder antarmuka pengguna bisa mengakses dan menggunakan fungsi-fungsi di folder manipulasi data. Dengan cara ini, dependency ini membantu mengatur akses antar komponen dalam proyek besar, memastikan bahwa semua bagian bekerja sama dengan benar.
- Binding Dependancy
  Binding dependency dalam UML menggambarkan bagaimana satu elemen atau modul terkait erat dengan elemen lain yang hanya bisa diakses atau diketahui pada saat runtime. Bayangkan saat membuat program yang memanggil fungsi dari plugin eksternal. Binding dependency memastikan bahwa fungsi-fungsi ini hanya dapat dihubungkan ketika program dijalankan, bukan selama proses kompilasi. Ini seperti memastikan bahwa dua potongan puzzle hanya bisa saling terhubung ketika semua bagian sudah ada di tempatnya.

3. Pemrograman terstruktur adalah pendekatan pemrograman yang fokus pada pembagian program menjadi blok-blok fungsional yang lebih kecil, seperti prosedur atau fungsi. Setiap blok ini menangani tugas tertentu dan bisa dipanggil kapan saja dalam program, yang membantu dalam mengorganisir kode dan mengurangi kompleksitas. Dalam pemrograman terstruktur, alur program biasanya mengikuti urutan logis dari atas ke bawah, dengan kontrol alur yang jelas menggunakan pernyataan seperti if-else, loops, dan lain-lain. Ini memudahkan debugging dan pemeliharaan karena kode yang terstruktur dengan baik lebih mudah dipahami dan diikuti.

   Sementara itu, pemrograman berorientasi objek (OOP) memfokuskan pada konsep objek, yang merupakan representasi dari entitas dunia nyata dengan atribut dan perilaku. Dalam OOP, kode dibagi menjadi kelas-kelas yang mendefinisikan objek, dengan data (atribut) dan metode (fungsi) yang spesifik untuk masing-masing kelas. Pendekatan ini membantu dalam mengelola kompleksitas dengan memungkinkan penggunaan kembali kode melalui konsep pewarisan (inheritance) dan polimorfisme (polymorphism). OOP juga mempromosikan enkapsulasi, yang menyembunyikan detail internal objek dari luar dan hanya menyediakan antarmuka untuk berinteraksi dengan objek tersebut.

4. Konsep objek dalam pemrograman berorientasi objek (OOP) adalah entitas yang merepresentasikan sesuatu fakta dari dunia nyata (objek , benda, manusia, peristiwa, hewan), dengan atribut (data) yang mendeskripsikan objek tersebut dan metode (fungsi) yang merupakan behavior atau apa yang dapat dilakukan objek tersebut. Misalnya . dalam dunia nyata objeknya adalah Mahasiswa , jika diimplementasikan dalam OOP, Mahasiswa akan menjadi objek yang akan memiliki atribut dan method. Atribut akan mendeskripsikan mahasiwa tersebut dalam ruang lingkup system, missal atribut Nama_Mahasiswa, NPM, Semester. Method adalah behavior atau apa yang dapat dilakukan Mahasiswa dalam ruang lingkup system yang ditentukan, misal UpdateData(), InputKRS(), dll.
5. Access modifier:
   - Publik

     Access modifier public memberikan akses terbuka terhadap komponen kelas baik itu atribut atau metode dari mana saja dalam program. Ini berarti bahwa variabel atau metode yang dideklarasikan sebagai public dapat diakses dari kelas lain, baik dalam paket yang sama maupun berbeda. Public sering digunakan untuk metode yang perlu diakses secara luas atau atribut yang sengaja dibuat agar dapat diakses dari luar kelas tersebut

     Contoh kodingan :

```python
class Buku:
    daftarBuku = []

    def __init__(self,judulBuku, pengarang,jumlahbuku) -> None:
        self.__id =  Buku.generate_id()
        self.judulbuku = judulBuku
        self.pengarang = pengarang
        self.jumlahbuku = jumlahbuku
```

Atribut judulbuku, pengarang dan jumlahbuku merupakan atribut public.

- Protected

  Acces modifier protected artinya komponen class tersebut (atribut, method) hanya dapat diakses oleh class itu sendiri dan juga subclassnya (turunannya). Ini berguna jika ada class yang mewarisi class induknya , maka ia akan dapat memodifikasi atribut tersebut dari dalam kelasnya.

  Contoh kodingan :

```python
class Barang:
    def __init__(self, nama, harga) -> None:
        global sku
        self._sku = sku
        self.namaBarang = nama
        self.harga = harga
        daftar_barang.append(self)
        sku += 1
```

  Self._sku merupakan atribut protected .

- Privat

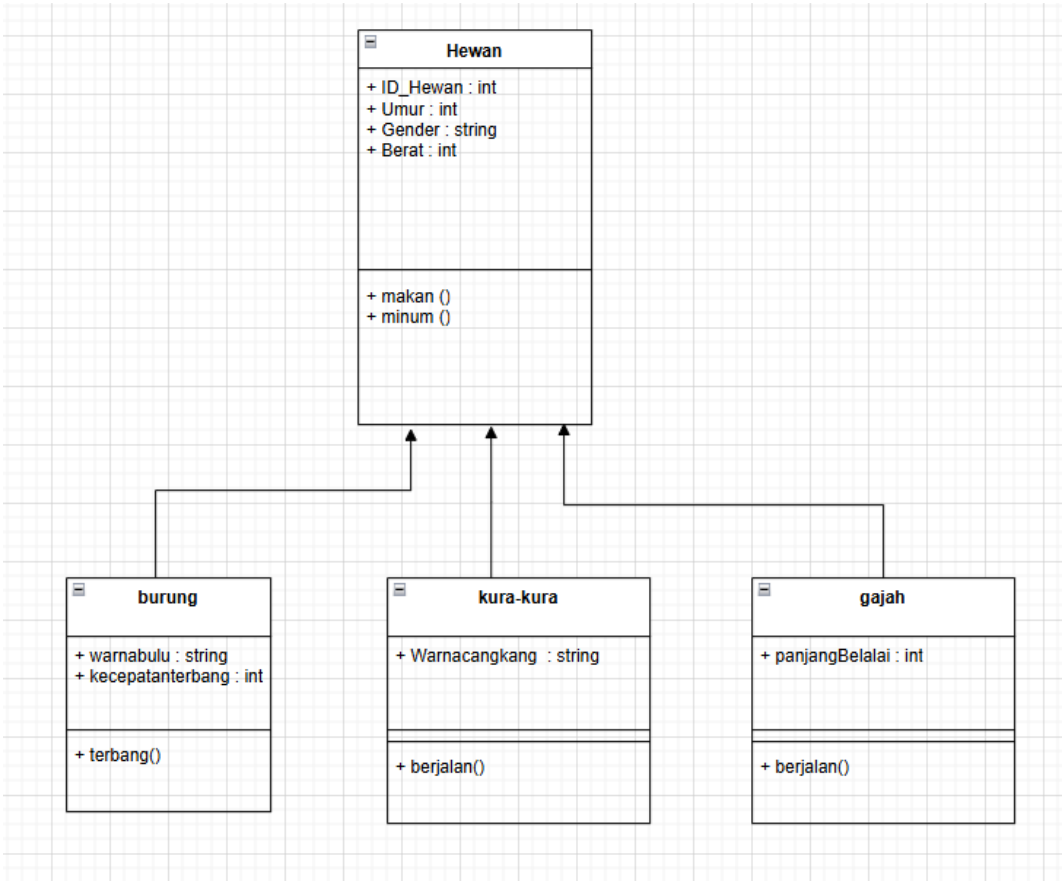
  Access modifier privat artinya komponen class tersebut (atribut, method) hanya dapat diakses oleh class itu sendiri. Atribut atau metode private tidak dapat diakses atau diubah langsung dari kelas lain, bahkan jika kelas tersebut berada dalam paket yang sama. Ini mendukung prinsip enkapsulasi, di mana data penting disembunyikan dari luar dan hanya dapat diakses melalui method khusus yang biasanya akan mereturnkan nilai dari atribut private tersebut

  Contoh kodingan :

```python
class User:
    totalUser = 0
    daftarUser = []
    def __init__(self,name,role) -> None:
        self.name = name
        self.__role = role
        User.totalUser += 1
```

Self.__role merupakan atribut privat.

6. Contoh pewarisan dalam diagram kelas:



Hewan merupakan parent kelas atau kelas abstraksi. Sedangkan, burung kura-kura dan gajah merupakan child class yang akan mewarisi method dan atribut dari kelas Hewan, yang mana masing-masing child class juga memiliki atribut dan methodnya tambahannya sendiri.