# Class Diagram #1

**Rekayasa Perangkat Lunak**

Informatika FTI UKDW

Genap TA 2024/2025

# Software Design

# MENGAPA MERANCANG?

Mari kita asumsikan sudah memiliki daftar spesifikasi kebutuhan yang tetap. **Apakah kita dapat langsung coding?**

Jika proyek tersebut adalah single-person project, ya betul kita langsung dapat coding.

> Anda mungkin mendapati hal ini tidak benar di waktu awal. Namun Anda dapat terus memodifikasi kode sampai Anda mencapai suatu rancangan yang membuat Anda senang.
>
> Pendekatan ini bekerja baik terutama jika Anda programmer yang baik. Anda dapat mendokumentasikan rancangan di akhir proyek sehingga orang lain dapat melanjutkan proyek tersebut.

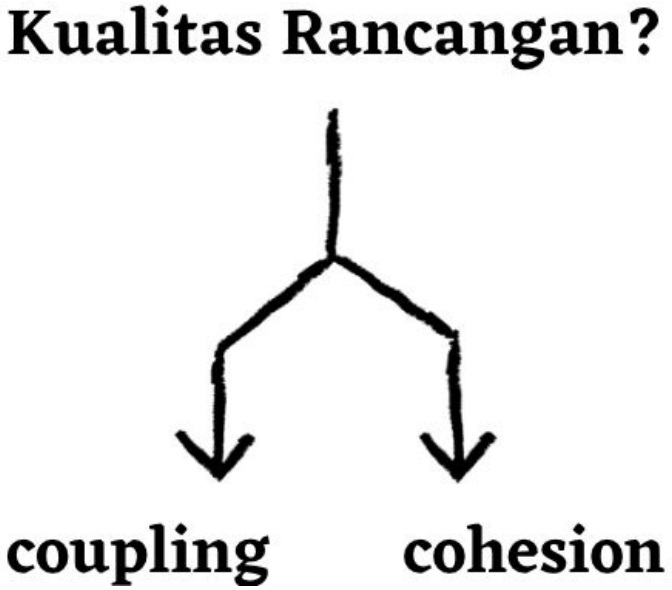
Namun, apakah tidak lebih baik jika kita rancang terlebih dahulu? Salah satu alasannya adalah setiap orang mengetahui struktur dari sistem. Sehingga Anda tahu bagian mana yang akan Anda bangun.

# MENGAPA MERANCANG?

- Namun berapa banyak struktur yang perlu Anda ketahui untuk memutuskan bagian mana yang akan dikerjakan?
- Jika Anda adalah anggota tim pengembangan Minesweeper, apakah Anda perlu tahu method-method apa yang harus dimiliki oleh Cell?
    - Tidak kecuali jika Anda memprogram secara langsung class Cell
    - Anda tidak ingin terbebani dengan rancangan detail yang tidak Anda perlukan.
- Rancangan detail dari suatu sistem besar dan komplek akan menjadi sangat komplek juga (bayangkan berapa banyak class yang dimiliki).
    - Bagaimana kita menjaga rancangan yang komplek?
    - Kapan Anda membangun suatu produk komplek dalam suatu tim,
    - Anda perlu mengabaikan information overload.
    - Tidak mungkin setiap anggota tim perlu tahu setiap sistem detail.
- Bagaimana Anda mencegah information overloaded?
    - Senjata utama melawan information overload (atau kompleksitas) adalah **abstraction**.

# Kualitas Rancangan

# Bagaimana caranya mengetahui seberapa bagus kualitas rancangan kita?



# Coupling

- Coupling adalah derajat dimana komponen bergantung pada komponen lainnya.
- Coupling secara langsung terkait dengan dependencies.
- Berdasar definisi, dependencies meningkatkan coupling.
- Sebuah dashed arrow dapat digunakan untuk menunjukkan dependency.

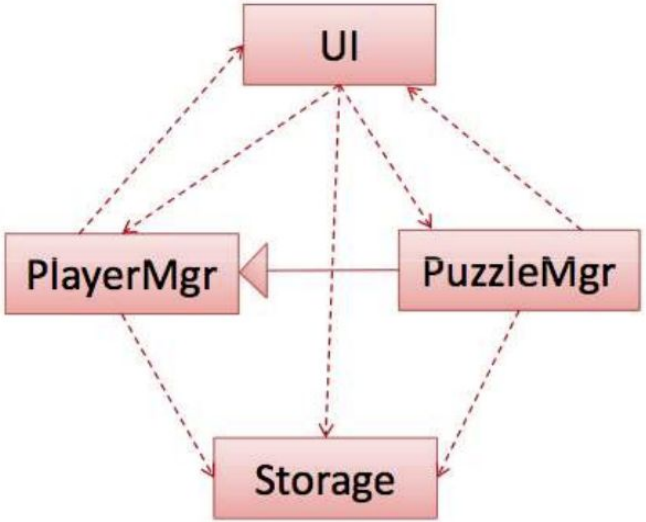
# Coupling

- Low-coupling berarti sebuah komponen kurang bergantung pada komponen lain.
- Ketika terdapat high-coupling, sebuah perubahan dalam satu komponen sering memerlukan perubahan di pasangan komponen lainnya. Oleh karena itu kita dituntut untuk mencapai low coupling dalam desain kita.

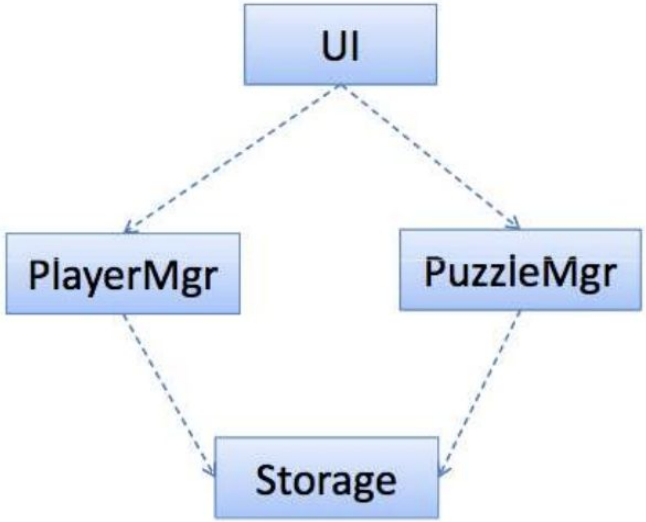
- Beberapa hal dapat menyebabkan coupling. Contoh:
  - Jika komponen A memiliki akses ke struktur internal komponen B (high level of coupling)
  - Jika komponen A dan B bergantung pada variabel global yang sama
  - Jika komponen A memanggil komponen B
  - Jika komponen A menerima sebuah objek komponen B sebagai sebuah parameter atau return value
  - Jika komponen A diturunkan dari komponen B
  - Jika komponen A dan B harus mengikuti format data atau protokol komunikasi yang sama.

# Latihan

Rancangan mana yang memiliki coupling lebih tinggi?




Rancangan A




Rancangan B

# Latihan

Dari pilihan berikut, mana yang menimbulkan less coupling antar class Employee dan class PayCalculator

**PayCalculator API**

Pilihan (a)

**Operasi:** `calculateBonus(Employee emp): int`

**Keterangan:**
Menghitung bonus employee, berdasar tanggal awal, posisi, jenjang kinerja.

Pilihan (b)

**Operasi:**
```
calculateBonus(Date
start_date, POSITION
position, GRADE
performance_grade): int
```

**Keterangan:**
Menghitung bonus employee, berdasar tanggal awal, posisi, jenjang kinerja.

# Cohesion

- Cohesion adalah suatu ukuran dari seberapa kuat suatu relasi dan terfokuskan pada berbagai responsibility suatu komponen.
- Sebuah komponen yang memiliki cohesion tinggi terus menjaga sesuatu yang terkait satu sama lain bersama-sama sambil mencegah hal lainnya.
- Kita menuntut high cohesion karena menjaga sesuatu yang terkait bersama-sama membuat pengkodean lebih mudah untuk pemeliharaan dan pemakaian ulang.

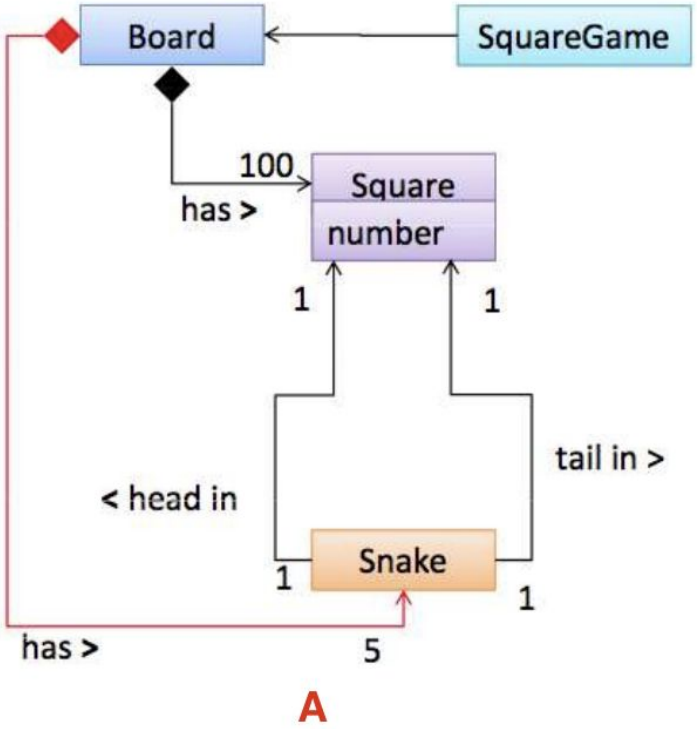
# Cohesion

Cohesion dapat dinyatakan dalam beberapa bentuk. Contoh:

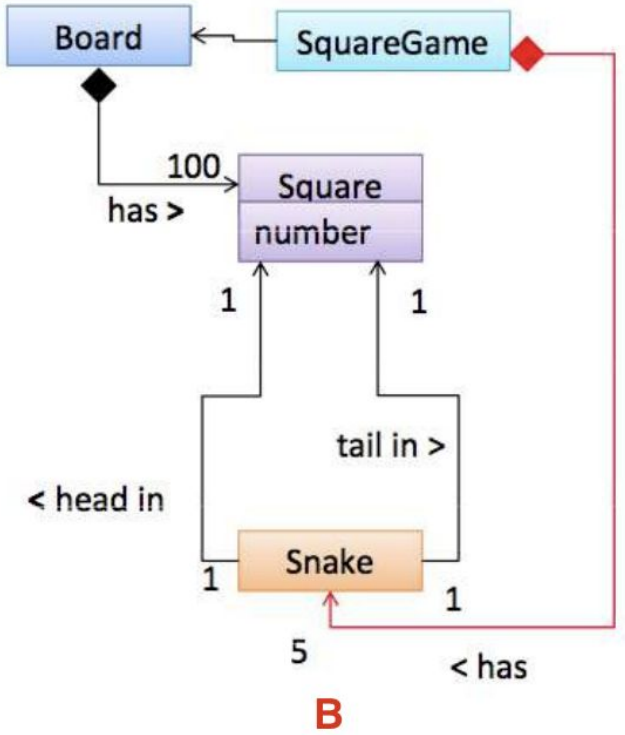
- Kode yang terkait dengan sebuah konsep tunggal dijaga bersama-sama. Contoh: class Student menangani segala sesuatu yang terkait dengan siswa.
- Kode yang dipanggil dalam kurun waktu berdekatan disimpan dijaga bersama-sama. Contoh: semua kode terkait dengan inisialisasi sistem dijaga bersama-sama.
- Kode yang memanipulasi struktur data yang sama dijaga bersama-sama. Contoh: GameArchive menangani segala sesuatu terkait dengan pengarsipan dan penerimaan sesi game.

# Latihan Kohesi

Pilihan mana yang lebih cohesive?




A




B

# Low Level Design

- The focus is more on designing each component in detail such as what **classes** are needed, what **abstractions** to use, how **object creation** should happen, how **data** flows between different objects, etc.
- LLD converts the high-level design into detailed design (ready to code) components.
- Writing code is the easiest part of building software if we have the designs already in place.
- Maintaining and Extending software will be easy if designs are well thought of.

# Low Level Design

- Software Developers need to learn Low-Level Design not only to crack the Interviews but also to build modular, extensible, reusable, and maintainable software. It is the thought process that developers need to develop to effectively build the software from a set of requirements.
- **Steps in for understand how to design any low-level:**
  - **Object-oriented Principles** (Inheritance, encapsulation, polymorphism, and abstraction)
- **Design Patterns** (Factory Design Pattern, Abstract factory Pattern, Singleton Pattern, Observer Pattern)
- **UML Diagram**
  - **Structure Diagrams**: Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Deployment Diagram, Package Diagram, Profile Diagram
  - **Behavior Diagrams**: Use Case Diagram, Activity Diagram, State Machine Diagram
  - **Interaction Diagrams**: Sequence Diagram, Communication Diagram, Interaction Overview Diagram, Timing Diagram

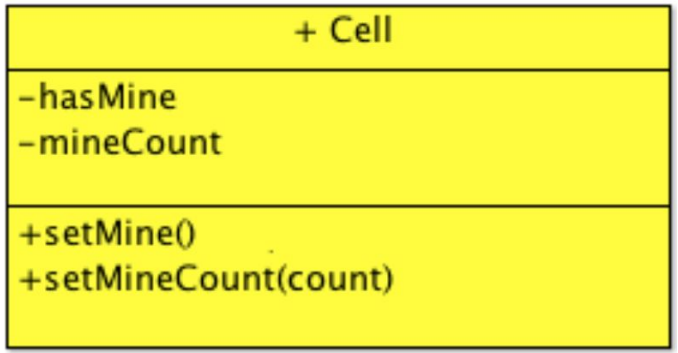
# Class Diagram

# Class Diagram

**Bagaimana Menunjukkan Visibility?**

- Kita juga dapat menunjukkan visibilitas dari atribut dan operation. Nama digunakan untuk visibility dan arti pastinya tergantung pada bahasa pemrograman yang digunakan.
- Berikut beberapa visibility yang umum digunakan dan bagaimana mereka dinyatakan dalam class diagram.

  + public
  - private
  # protected
  ~ package



# Class Diagram

- Berikut beberapa format detil untuk operation dan attributes.
  - Hanya nama adalah wajib.

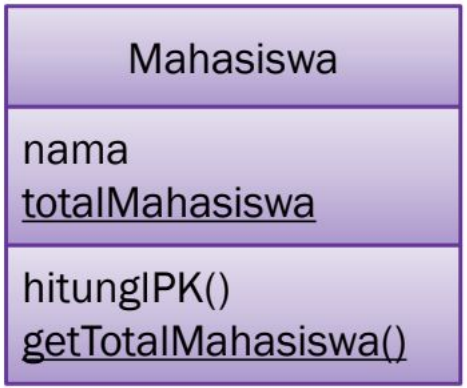
visibility name : type multiplicity = default-value

visibility name (parameter-list) : return-type

| Minefield |
| --- |
| - H : int = 10<br>-W : int = 10 |
| + Minefield(String config_string): void<br>+ newGame(): void<br>+ getWidth() : int<br>+ getHeight() : int<br>+ clearCellAt(x,y) : void<br>+ markCellAt(x,y) : void<br>+ getAppearanceOfCellAt (x,y) : CELL_APPEARANCE |

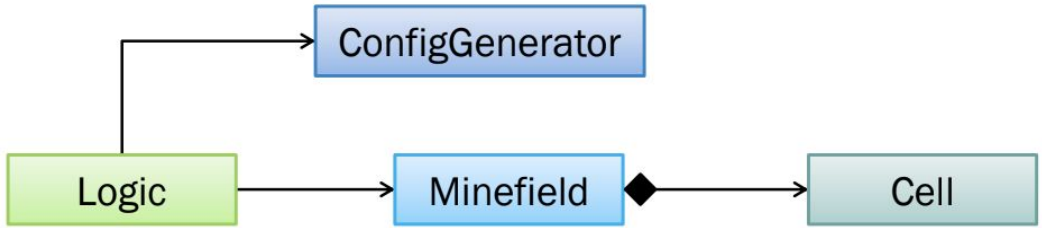
# Member STATIC

- Attribute dan operation pada level class
  - seperti variabel/fungsi 'global' pada class
  - contoh: variable totalMahasiswa pada class Mahasiswa dideklarasikan static sebab dimiliki bersama oleh semua obyek Mahasiswa. Demikian pula operasi getTotalMahasiswa
  - variable nama tidak dideklarasikan static sebab setiap obyek Mahasiswa memiliki namanya sendiri
- Attribute dan operation pada level class(static) di UML ditandai dengan garis bawah



# Navigability

- Navigability adalah relasi yang menandai bahwa suatu class yang terlibat dalam asosiasi "mengetahui" keberadaan class yang lain
  - ditandai tanda panah pd association link

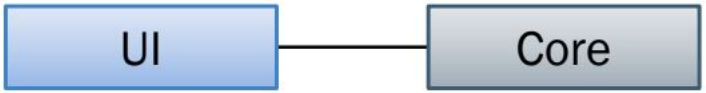



- Logic mengetahui Minefield, namun tidak sebaliknya

# Uni/Bidirectional

- Arah navigability adalah unidirectional atau bidirectional
  - unidirectional, hanya satu class yang mengetahui yang lain
  - bidirectional, setiap class mengetahui yang lain


- bidirectional dpt disederhanakan dengan tanpa panah,




- walau juga bisa berarti unspecified (tidak dispesifikasi)

# TIPS UNTUK CLASS DIAGRAM