



Konsep Inheritance

IF2210 – Semester II 2021/2022

Inheritance dalam OOP

- › Kemampuan kelas untuk menurunkan atribut dan method dari kelas lain disebut dengan ***inheritance***.
- › *Inheritance* adalah salah satu konsep dalam OOP yang membedakan dari memrogram ADT.
- › Istilah:
 - › ***Subclass/derived class***: kelas “anak”, yang menurunkan atribut & method dari kelas lain.
 - › ***Superclass/base class***: kelas “induk”, yang atribut & method-nya diturunkan ke kelas lain.

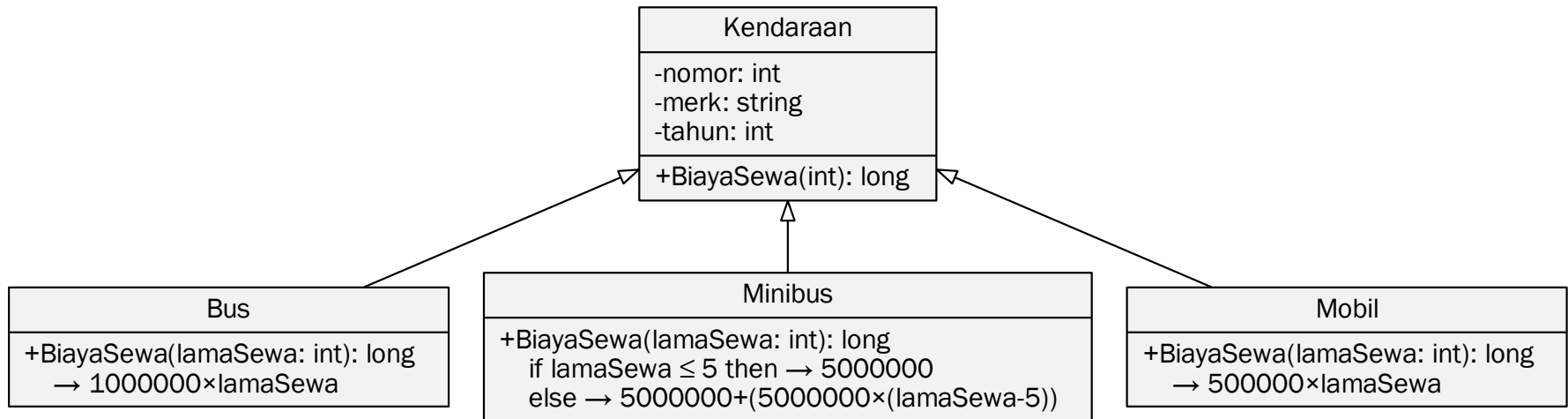


Mengapa inheritance?

- › Tinjau contoh kelas Kendaraan pada latihan soal sebelumnya.
- › Pada implementasi method `BiayaSewa(int)` terdapat kondisional:
 - › Jika kategori="bus" maka biaya sewa = $1000000 \times \text{lama sewa}$
 - › Jika kategori="minibus" maka biaya sewa = ...
 - › Jika kategori="mobil" maka biaya sewa = ...
- › Ketiga kategori kendaraan memiliki implementasi yang berbeda-beda untuk method `BiayaSewa()`.
- › Kendaraan dengan kategori yang berbeda-beda dapat dirancang sebagai subclass dari kelas Kendaraan.



Ilustrasi



Notasi: diagram kelas (UML)
yang dimodifikasi untuk
memudahkan ilustrasi
(jangan ditiru)

- Tidak ada lagi atribut kategori.
- `BiayaSewa()` diimplementasikan di *subclass*.
- Efek: jika ada jenis kendaraan baru, dapat membuat *subclass* baru tanpa mengubah kelas **Kendaraan**.

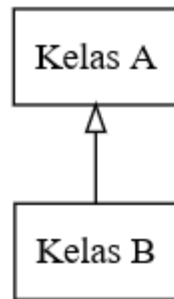


Kelas abstrak

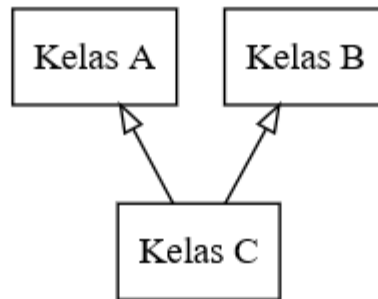
- › Pada contoh sebelumnya, kelas Kendaraan adalah kelas abstrak.
 - › Ada method yang tidak diimplementasikan (`BiayaSewa(int)`).
- › Kelas abstrak tidak dapat langsung diinstansiasi.
- › Objek yang diciptakan adalah instansiasi dari subclass-nya yang tidak abstrak.
- › Contoh lain: kelas Shape adalah abstrak, tidak bisa digambar (method `Draw()` tidak dapat diimplementasi) jika tidak tahu bentuk “nyata”-nya.
 - › Subclass `Rectangle`, `Circle` diketahui bagaimana cara menggambarinya (mengimplementasikan `Draw()` sesuai atribut yang dimiliki: `Rectangle` memiliki panjang dan lebar, `Circle` memiliki titik pusat dan radius).



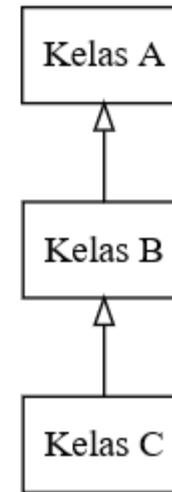
Jenis-jenis *inheritance* (1)



Single inheritance

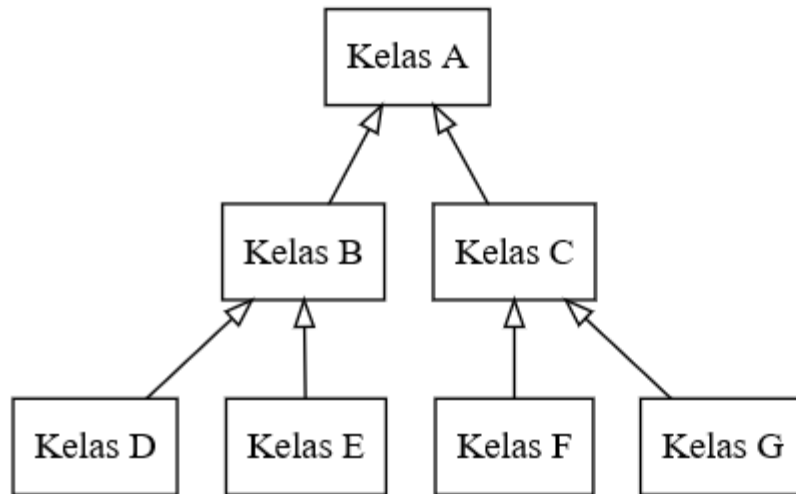


Multiple inheritance

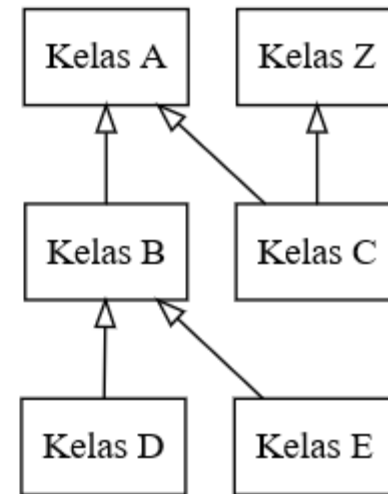


Multilevel inheritance

Jenis-jenis *inheritance* (2)



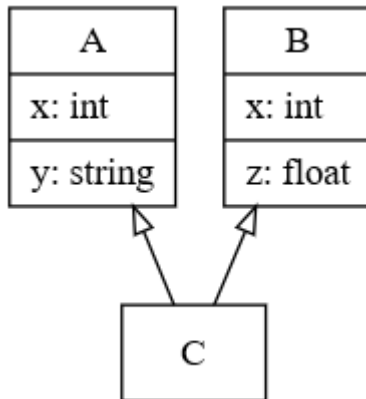
Hierarchical inheritance



Hybrid/virtual inheritance

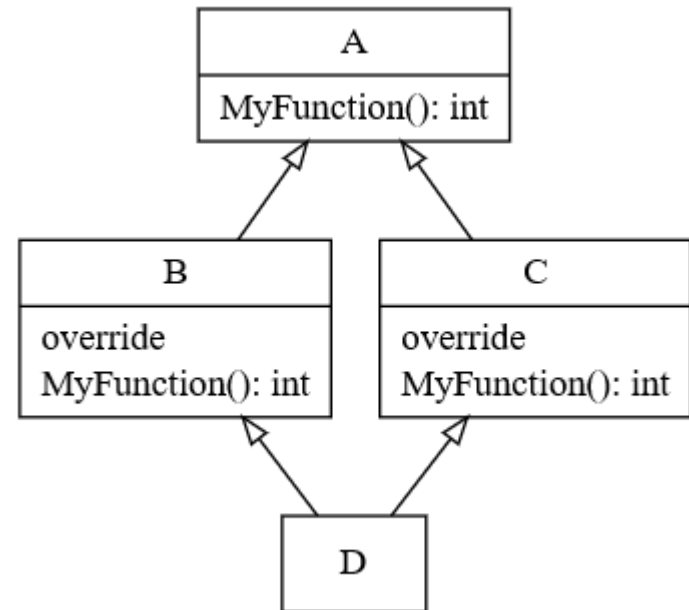
Masalah pada multiple inheritance (1)

- Beberapa base class memiliki nama anggota yang sama.



- c.x yang mana?

- Diamond problem/deadly diamond of death (DDD).



- d.MyFunction() yang mana?

Masalah pada multiple inheritance (2)

- › Untuk mengatasi masalah-masalah tersebut, setiap bahasa OO memiliki caranya sendiri, misalnya:
 - › Rename, redefinition
 - › C++: untuk mencegah DDD, kelas B dan C harus dideklarasikan virtual
 - › Java, Ruby, Smalltalk: tidak memperbolehkan *multiple inheritance*
 - › dll.



Root object/class

- › Jika sebuah kelas dapat merupakan turunan dari kelas lain, adakah “*mother of all classes*”?
- › Pada beberapa bahasa OO, semua kelas, baik yang disediakan oleh bahasa/*framework* maupun yang dibuat sendiri oleh pemrogram, adalah *subclass* dari sebuah *root class* meskipun tidak dituliskan secara eksplisit.
 - › C#: kelas `System.Object`
 - › Java: kelas `java.lang.Object`
 - › Python: kelas `object`



Polymorphism

- › Objek-objek dari kelas turunan memiliki sifat sebagai kelas tersebut dan sekaligus kelas dasarnya.
 - › *polymorphism* (*poly* = banyak, *morph* = bentuk).
- › Sebuah “variabel” dapat dideklarasikan sebagai kelas Kendaraan, namun pada *run time* dapat merupakan instansiasi dari kelas Mobil atau Bus (subclass dari Kendaraan).
- › Sebaliknya, sebuah “variabel” bertipe Mobil dapat diperlakukan sebagai Kendaraan.
 - › i.e. *message* yang dapat diterima Kendaraan juga dapat diterima oleh Mobil.
- › Hubungan “is-a”: Mobil is a Kendaraan, Bus is a Kendaraan.



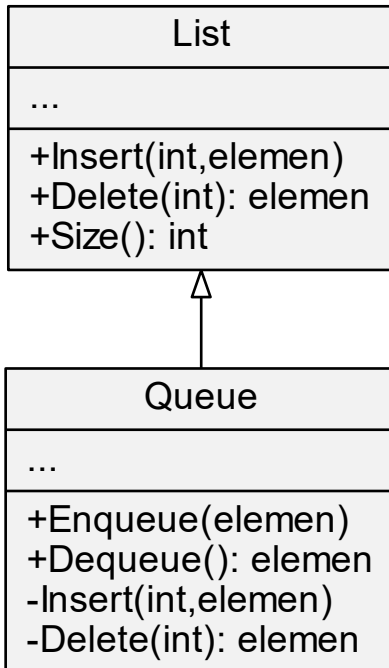
Inheritance vs. Composition

- › *Inheritance*: hubungan *is-a*, *composition*: hubungan *has-a*.
 - › Sedan *is a* mobil, SUV *is a* mobil, dst.
 - › Mobil *has a* mesin, mesin *has a* set of busi, dst.
- › Misalkan kita memiliki sebuah kelas `List` yang bisa ditambah/kurangi elemennya di posisi manapun
 - › Kita dapat memanfaatkan kelas tersebut untuk membuat kelas `Queue`:
 - › Komposisi? (*Queue has a List*)
 - › Inheritance? (*Queue is a List*)
 - › Queue harus bisa melakukan apa pun yang bisa dilakukan `List` (seharusnya tidak)
 - › Beberapa bahasa mendukung “penyembunyian” method dari superclass yang `public` menjadi `private` di subclass

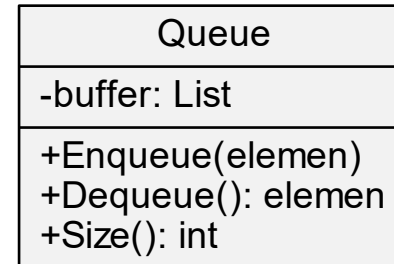


Queue (is a | has a) List

Queue 'is a' List



Queue 'has a' List



- Queue harus menyembunyikan `Insert(int, elemen)` dan `Delete(int)`

- `Enqueue(e)` memanggil `buffer.Insert(last_idx, e)`
- `Dequeue()` mengembalikan `buffer.Delete(0)`
- `Size()` membungkus `buffer.Size()`



Kapan menggunakan *inheritance*?

- › Inheritance should only be used when:
 - › Both classes are in the same logical domain
 - › The subclass is a proper subtype of the superclass
 - › The superclass's implementation is necessary or appropriate for the subclass
 - › The enhancements made by the subclass are primarily additive.

(<https://www.thoughtworks.com/insights/blog/composition-vs-inheritance-how-choose>)

