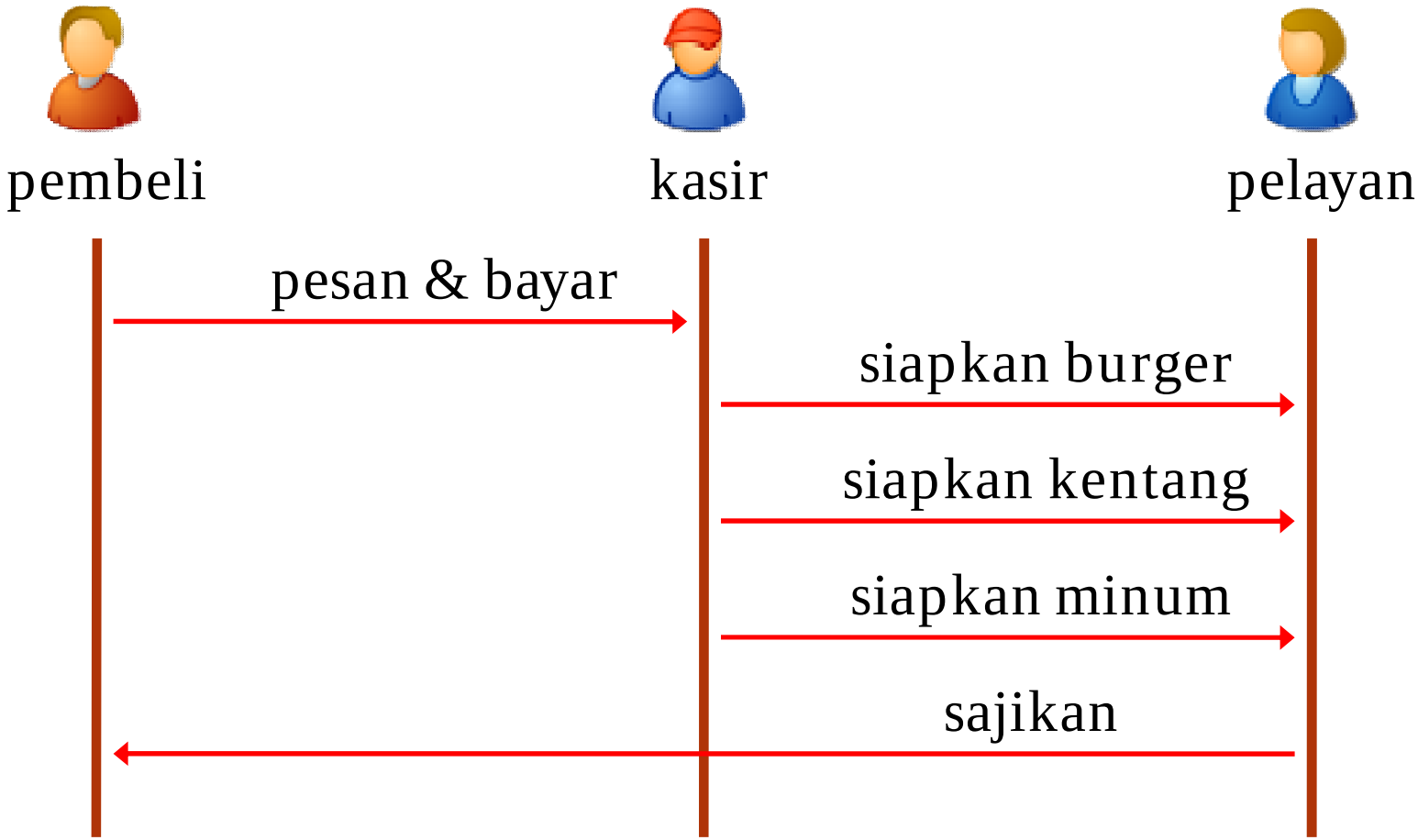
11. Konsep Berbasis Object (Object-Oriented Concept)

S. Indriani L, M.T

Konsep Object

- Dalam paradigma berbasis object, “object” adalah representasi sebuah entitas yang memiliki makna tertentu yang menjadi perhatian si pemandang → object memiliki abstraksi tertentu bagi si pemandang
  - Contoh: dalam sistem akademik di kampus, beberapa kandidat object adalah: mahasiswa, dosen, kelas, mata kuliah, kurikulum, praktikum, laboratorium, perpustakaan, KRS, KHS, …
- Object dapat menerima pesan *(message)*, mengolah data, dan mengirimkan pesan ke object lain → membentuk interaksi antar object
- Object bersifat independen: tiap object dapat dipandang sebagai sebuah entitas mandiri yang memiliki peran atau tanggung jawab tertentu

Object dan Interaksinya



Skenario: restoran cepat-saji

Object dan Representasi Lingkup Permasalahan

- Object dan interaksinya dapat digunakan untuk merepresentasikan lingkup permasalahan
  - Object merepresentasikan entitas-entitas yang dianggap penting dalam skenario permasalahan
  - Interaksi antar object menggambarkan bentuk-bentuk relasi antar entitas di dalam skenario permasalahan
- OO sebagai *tool* untuk memodelkan sistem di dunia nyata (tujuan bahasa Simula-67) → setiap sistem selalu dapat digambarkan melalui object-object penyusunnya dan bagaimana object-object tersebut saling berinteraksi
- Dalam software engineering: OO analysis → OO design → OO programming

Kelebihan OO Sebagai Model Representasi

- Natural: mengikuti cara berpikir manusia (manusia memandang dunianya sebagai kumpulan object yang berinteraksi)
- Abstraksi: menjelaskan makna sebuah entitas secara cepat dan mudah
- Enkapsulasi: dapat menyembunyikan detil yang tidak perlu
- Modular: object adalah entitas yang independen

Pemrograman dan Bahasa Pemrograman Berbasis OO

- Merealisasikan entitas-entitas object dalam desain software (OOD) dalam bahasa pemrograman
- Bahasa pemrograman berbasis object menyediakan mekanisme untuk bekerja dengan:
  - kelas dan object
  - methods
  - Pewarisan (*inheritance*)
  - Polimorfisme (*polymorphism*)
  - reusability

Pemrograman dan Bahasa Pemrograman Berbasis OO..

Ragam bahasa pemrograman berbasis OO

- Bahasa OO "murni" → semua diperlakukan secara konsisten mengikuti teori OO dan "memaksa" pemrogram mengikutinya.

  Contoh: Smalltalk, Eiffel, Ruby
- Bahasa yang dirancang untuk OOP, tetapi dengan beberapa elemen prosedural.

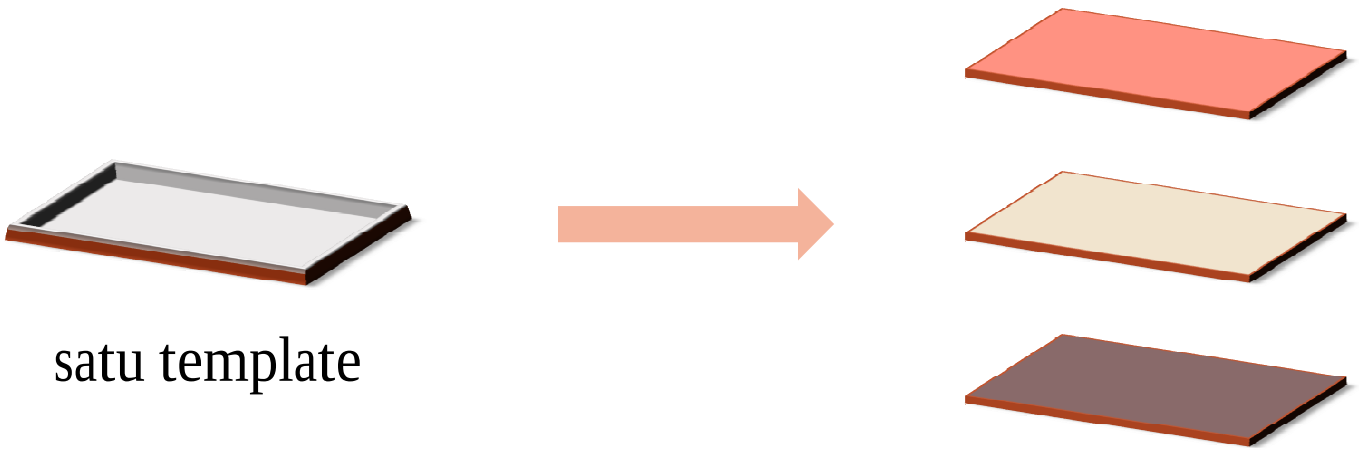
  Contoh: Java (program utama / entry point)
- Bahasa yang aslinya prosedural, tetapi kemudian ditambah fitur-fitur OO. Contoh: C++, Perl, PHP

Konsep-Konsep dalam OOP: Kelas (Class)

- Mendefinisikan karakteristik abstrak dari sebuah entitas
  - Property (statis) – fields, atribut
  - Behaviour (dinamika) – fitur, methods
  - Contoh: abstraksi "sepeda" memiliki property "punya roda", "punya rantai", dsb., dan behaviour "bisa berjalan", "bisa berbelok", dsb.
- Sebagai abstraksi dari sebuah entitas, definisi kelas harus mencerminkan karakteristik sebenarnya dari entitas tsb.

Kelas

- Kelas merupakan “cetakan” (template) untuk *instance* (wujud nyata) entitas-entitas yang direpresentasikannya
  - Sebuah kelas dapat melahirkan lebih dari satu instance

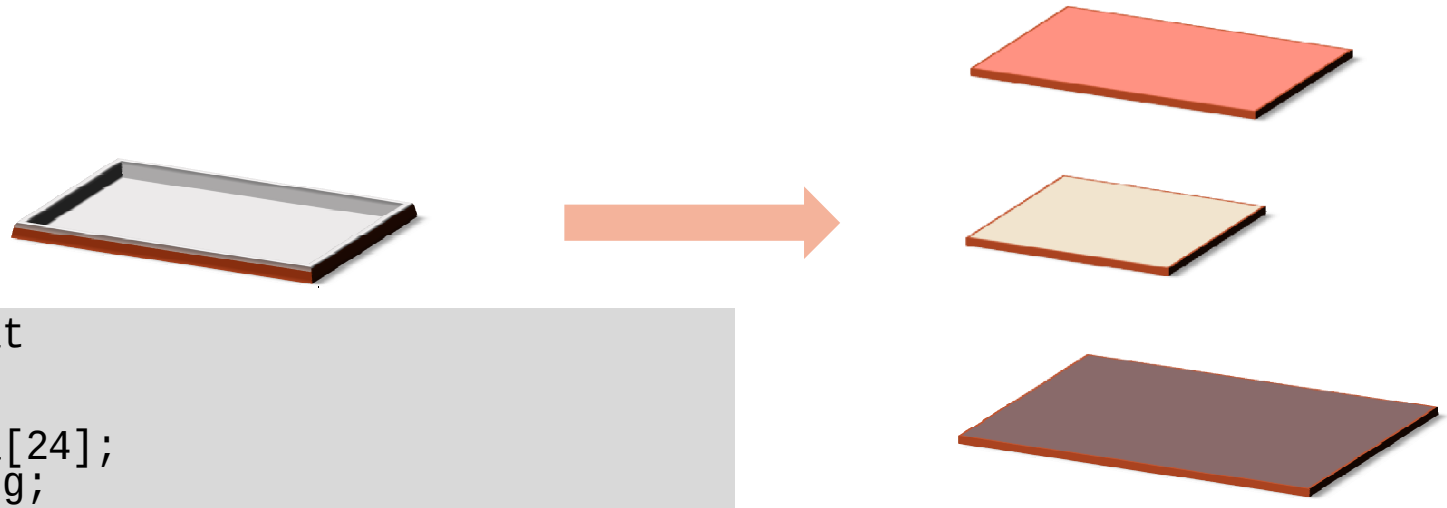


satu template

banyak instance
(wujud nyata)

Object

Sebuah *instance* (perwujudan nyata) dari suatu kelas tertentu



```
class SegiEmpat
{
    private:
    char warna[24];
    int panjang;
    int lebar;

    public:
    void inisialisasi(char *w, int p, int l)
    {
    strcpy(warna, w);
    panjang=p;
    lebar=l;
    }

void setWarna()
    {
     cout<<"Warna  :"<<warna<<endl;
     cout<<"Panjang :"<<panjang<<endl;
     cout<<"lebar  :"<<lebar<<endl;
    }
};
```

```
SegiEmpat sPink,sPutih,sOrange;
sPink.inisialisasi("Pink",20,10);
sPink.setWarna();
sPutih.inisialisasi("Putih",15,10);
sPutih.setWarna();
sOrange.inisialisasi("Orange",30,15);
sOrange.setWarna();
```

Definisi Kelas dan Object

```
class SegiEmpat
{
    private:
    char warna[24];
    int panjang;
    int lebar;

    public:
    void inisialisasi(char *w, int p, int l)
    {
    strcpy(warna, w);
    panjang=p;
    lebar=l;
    }

void setWarna()
    {
     cout<<"Warna     :"<<warna<<endl;
     cout<<"Panjang   :"<<panjang<<endl;
     cout<<"lebar     :"<<lebar<<endl;
    }
};
```

field, menunjukkan atribut/property

constructor, untuk menciptakan object (instance) baru dengan property tertentu

methods atau member functions, mendeskripsikan behaviour atau aktivitas yang bisa dijalankan

```
sPink.inisialisasi("Pink",20,10);
```

memanggil constructor untuk membentuk object baru

Enkapsulasi

- Enkapsulasi: lokalisasi fitur-fitur sebuah object (fields dan methods) dalam definisi object tersebut
- Enkapsulasi menyembunyikan property dan behaviour object dari pihak luar (object yang lain) → object lain melihat object ini sebagai “black box” saja
- Enkapsulasi memisahkan antara bagian publik (yang bisa dilihat oleh pihak luar (object lain) dan bagian privat (internal object itu sendiri) dengan tegas → fitur ini memberi keleluasaan / independensi untuk bekerja dengan aspek internal tanpa harus bergantung pada aspek publik / eksternal → berguna untuk menangani berbagai persoalan interoperabilitas

Enkapsulasi

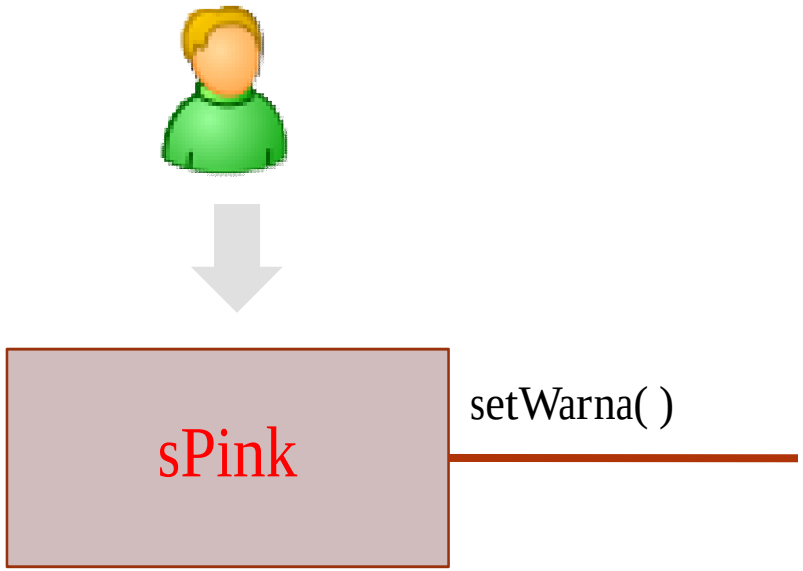
```
class SegiEmpat
{
    private:
    char warna[24];
    int panjang;
    int lebar;

    public:
    void inisialisasi(char *w, int p, int l)
    {
    strcpy(warna, w);
    panjang=p;
    lebar=l;
    }

void setWarna()
    {
     cout<<"Warna    :"<<warna<<endl;
     cout<<"Panjang :"<<panjang<<endl;
     cout<<"lebar    :"<<lebar<<endl;
    }
};
```

```
sPink.inisialisasi("Pink",20,10);
```

Pihak luar (object lain) melihat sPink sebagai sebuah “kotak hitam” yang tidak terlihat isinya, kecuali fields dan methods yang memang dideklarasikan secara publik

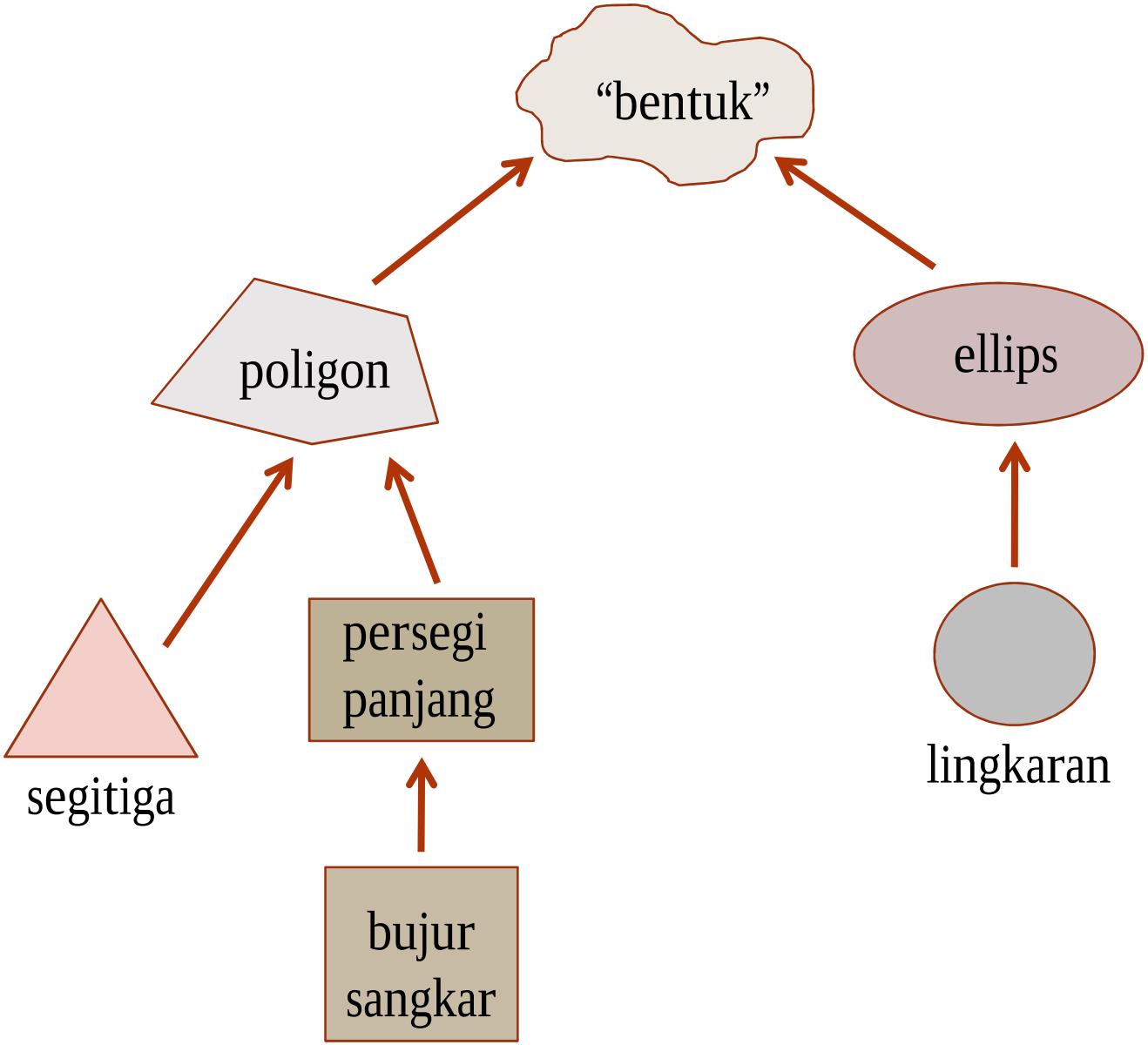


Object lain tidak bisa melihat apa yang dilakukan sPink secara internal

Inheritance (Pewarisan Sifat)

- Dalam dunia OO, beberapa object memiliki sifat/ciri yang mirip dengan yang lain → sebuah kelas object dapat didefinisikan dari kelas yang lain
  - Kemiripan ini membentuk hubungan sifat yang bersifat hirarkis
  - "pesawat penyergap adalah sebuah pesawat terbang yang dilengkapi dengan persenjataan dan dapat terbang melebihi kecepatan suara"
    - "pesawat penyergap" memiliki ciri yang sama dengan "pesawat terbang" → "pesawat penyergap" mewarisi sifat "pesawat terbang"
    - "pesawat penyergap" memiliki ciri yang lebih khusus dibandingkan "pesawat terbang" → "pesawat penyergap" adalah subclass dari "pesawat terbang"
  - Dalam OOP, hubungan inheritance (pewarisan sifat) ini diimplementasikan melalui definisi kelas: sebuah kelas dapat didefinisikan dari kelas yang lain

Inheritance

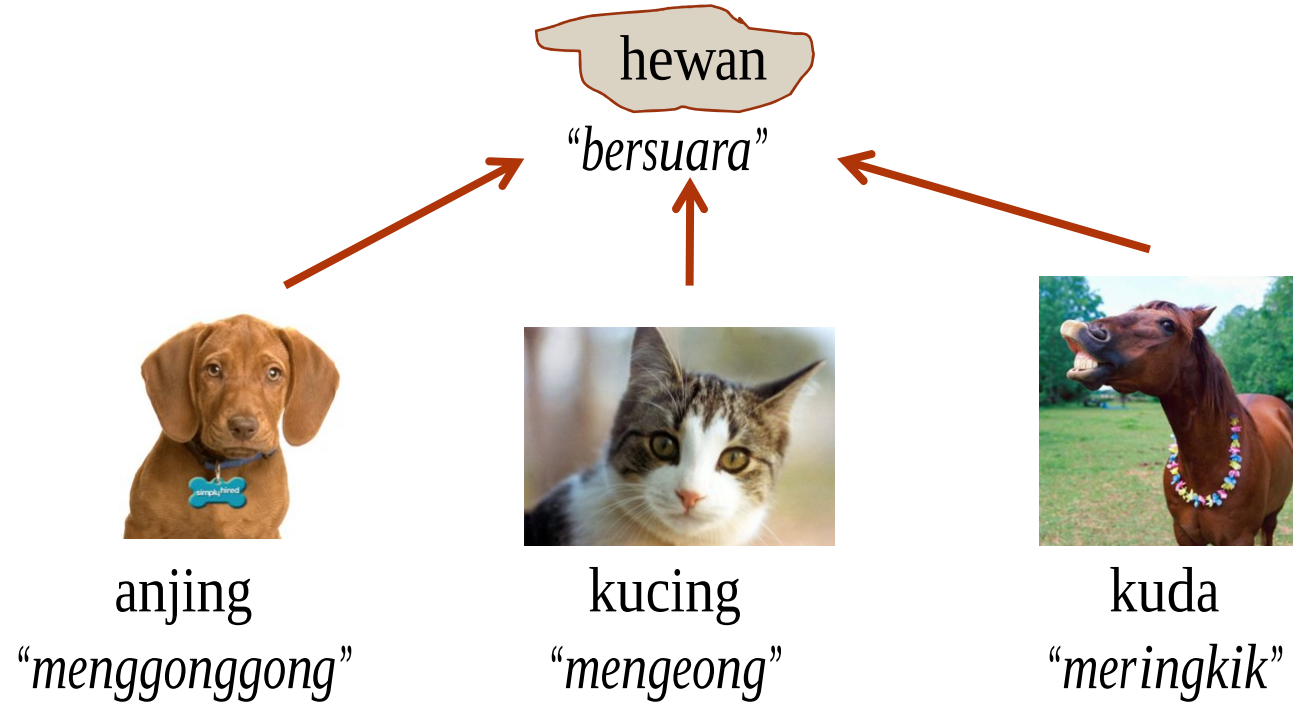


Inheritance

- Dalam pohon hirarki inheritance, sebuah subclass mewujudkan abstraksi yang lebih spesifik dari superclassnya: subclass = superclass + (fitur-fitur baru)
  - Subclass dapat menambahkan fields dan methods baru
  - Subclass dapat mengambil alih *(override)* method milik superclass dan mengubah implementasi method tersebut
- Manfaat inheritance:
  - Pemanfaatan kembali *(reuse)* kelas
  - Menyediakan model yang bersifat generik (abstract class); implementasi spesifiknya bisa dilakukan kemudian (mungkin oleh pemrogram yang berbeda)

Polymorphism

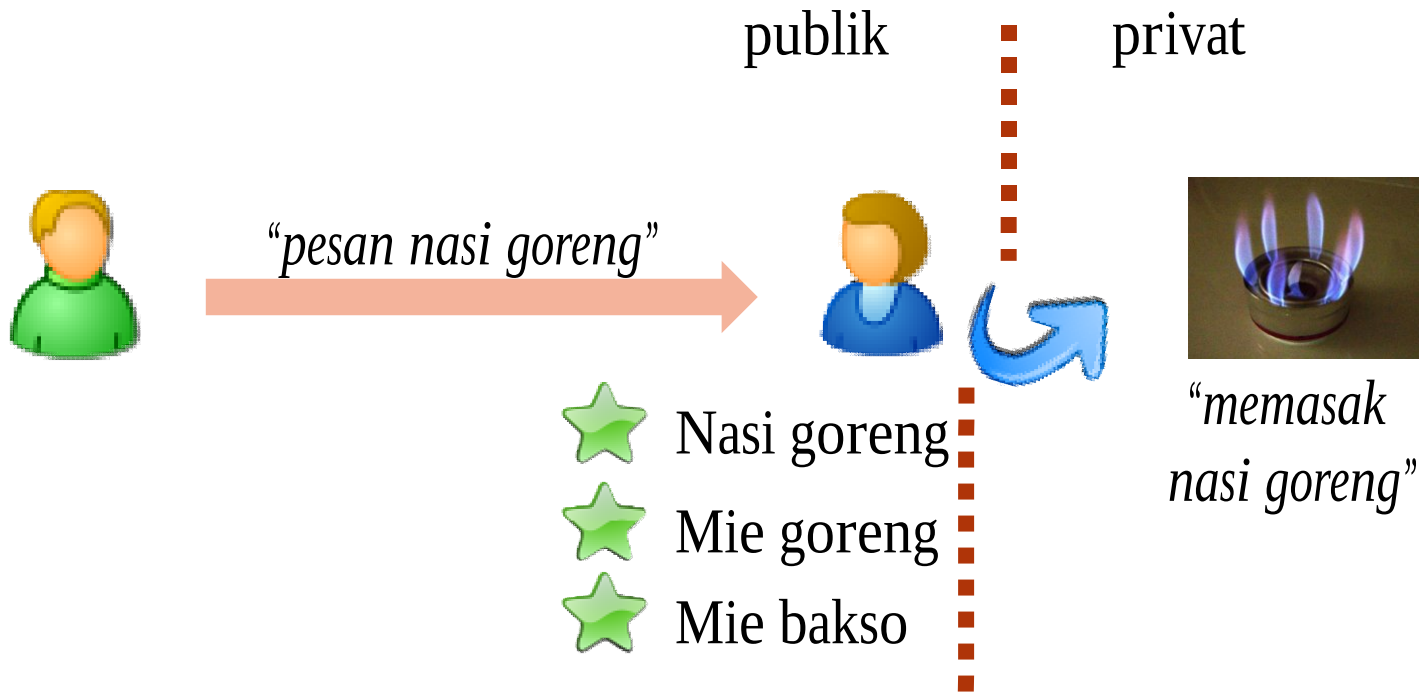
- Poly: banyak; morphism: bentuk → sebuah fitur (generik) bisa diimplementasikan dengan berbagai cara
- Polymorphism diimplementasikan dengan mekanisme inheritance dan overriding



OO dan Interoperabilitas

- Konsep abstraksi dan enkapsulasi membuat “batas” yang tegas antara bagian yang bersifat publik dan bagian yang bersifat privat
  - Abstraksi menjelaskan hal-hal yang terlihat dari luar (bersifat publik)
  - Enkapsulasi menyembunyikan detil-detil yang tidak perlu terlihat dari luar (bersifat privat)
- Independensi antara aspek publik dan privat dieksploitasi untuk menangani isu interoperabilitas: implementasi layanan dapat dilakukan tanpa harus tergantung pada siapa yang mengaksesnya

Pemisahan Publik – Privat



Pemilik warung (server) dapat memasak nasi goreng (implementasi layanan) dengan berbagai cara tanpa harus tergantung pada siapa yang memesannya (client).