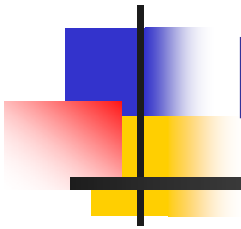
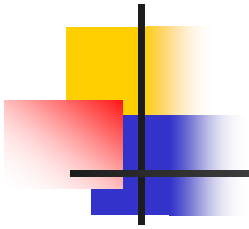


Konsep OOP

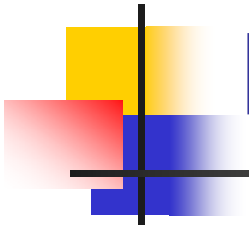
Object Oriented Programming

(Pemrograman Berorientasi Objek)

Object Oriented Programming (OOP)

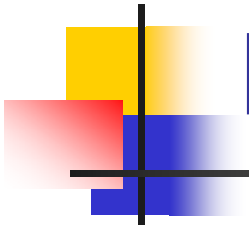


- Merepresentasikan suatu cara pembuatan program yang lebih dekat dengan cara berpikir manusia tentang dunia nyata.
- Object Oriented adalah himpunan tools dan method yang memungkinkan programer membangun program yang :
  - Reliable - reusable
  - User friendly - memenuhi kebutuhan user
  - Maintanable
  - Well documented



Konsep Object Oriented

- Object dan Class
- Inheritance
- Polymorphism

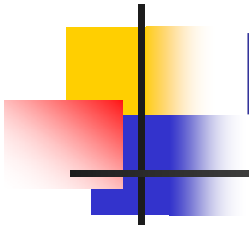


Paradigma OOP

- Contoh kasus:
- Misalkan anda ingin mengirim bunga ke seorang teman bernama Robin yang tinggal di kota lain. Untuk menyelesaikan masalah ini, anda pergi ke toko bunga milik Fred. Anda katakan pada Fred jenis bunga yang dikirim dan memberikan alamat yang dituju. Anda percaya bahwa bunga akan sampai ke teman anda tepat waktu.

Mekanisme Penyelesaian Masalah

1. Pertama kali anda mencari **agent** (Fred) dan melalui agent ini anda mengirimkan **message** (pesan) yang berisi suatu permintaan.
2. Fred memiliki **responsibility** (tanggungjawab) untuk memenuhi permintaan tersebut.
3. Ada beberapa **method** (sebuah algoritma atau kumpulan operasi) yang digunakan Fred untuk melakukan tugas tersebut.
4. Anda tidak perlu tahu method apa yang digunakan Fred. Informasinya tersembunyi (**hidden**)



Konsep OOP

- Sebuah program Object Oriented tersusun atas komunitas dari agen-agen yang saling berinteraksi yang disebut objek.
- Setiap objek memiliki peran yang harus dimainkan.
- Setiap objek menyediakan pelayanan (*service*) atau membentuk suatu aksi yang digunakan oleh anggota lain dalam komunitas

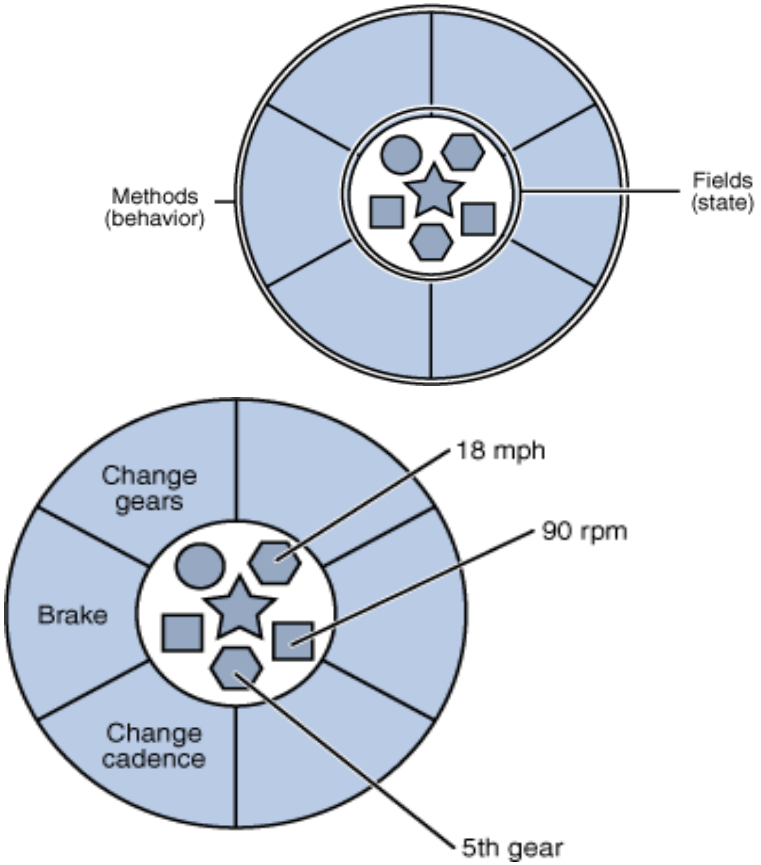
Message and Responsibility

- Anggota komunitas OO saling membuat *request*.
- Aksi dibentuk dalam OOP dengan mengirimkan sebuah pesan (*message*) ke sebuah agen (*object*) yang dipercaya untuk melakukan aksi.
- Pesan (*message*) mengkodekan permintaan untuk sebuah aksi dan digabungkan dengan suatu informasi tambahan (*argumen/parameter*) yang diperlukan untuk memenuhi permintaan tsb.
- Penerima pesan (*receiver*) adalah objek dimana pesan dikirimkan.
- *Receiver* menerima pesan, artinya dia bertanggungjawab (*responsibility*) untuk melakukan aksi tsb.
- Sebagai respon dari pesan, *receiver* akan membentuk beberapa *method* untuk memenuhi permintaan.

Classes dan Instance

- Object adalah instance dari Class
- Fred adalah instance dari class orang
- Fred adalah instance dari class florist
- → Fred adalah object yang merupakan instance dari sebuah class
- Object memiliki **state** dan **behavior**
- **Object Mahasiswa**
  - **State**(NIM, Nama, Kuliah yang diikuti, Umur)
  - **Behavior**(Mengikuti kuliah, mengikuti ujian, mengerjakan tugas, melakukan praktikum)

Encapsulation



- Pengemasan variabel objek oleh method disebut encapsulation
- Keuntungan:
  - Modularity

    Source code masing2 object dapat dikelola secara independent
  - Information hiding

    Sebuah objek memiliki informasi dan method yang dapat diubah tanpa mempengaruhi objek yang lain

Messages

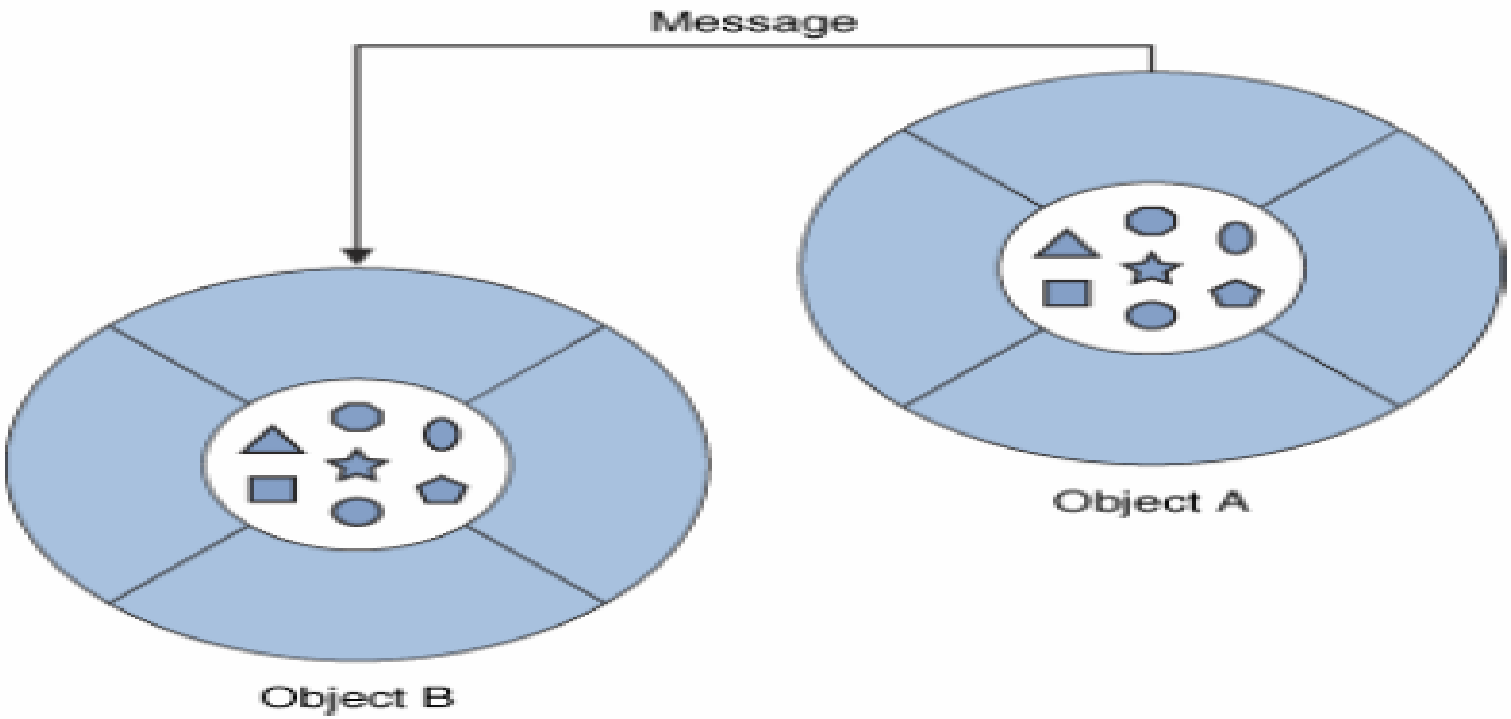
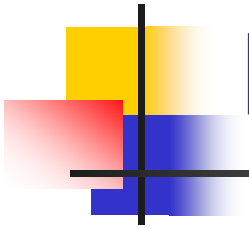
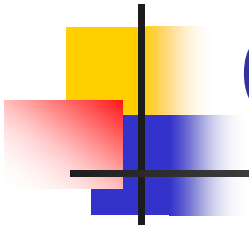


Figure 1.2: A Message



Messages

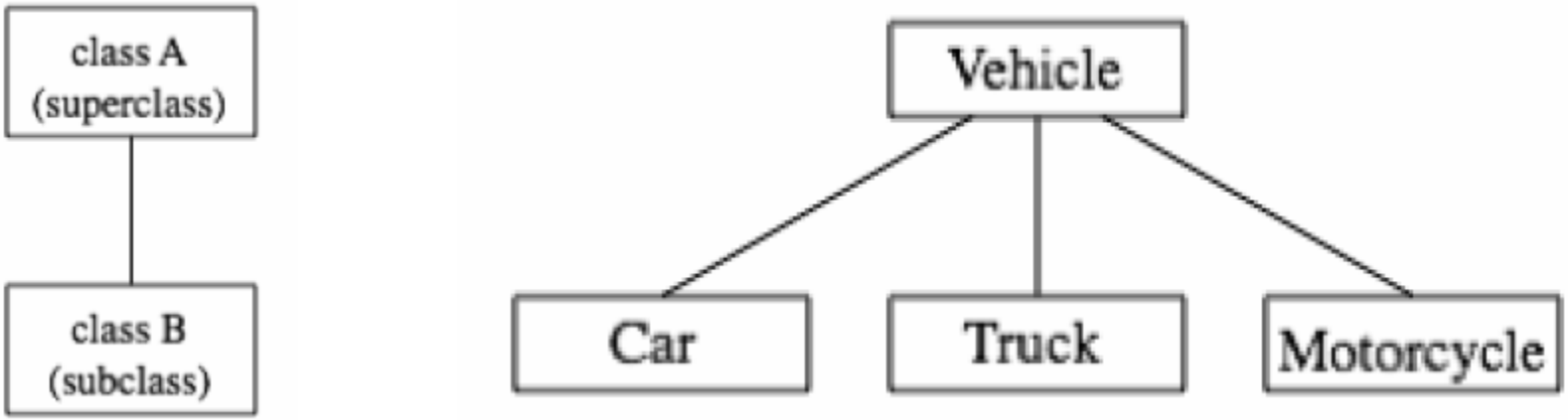
- Objek-objek saling berinteraksi dan berkomunikasi dengan mengirimkan pesan (message).
- Ada tiga bagian dalam pesan
- Contoh : System.out.println{“Hello World’}
  - Object dimana pesan dikirim (System.out)
  - Nama method pembentuk pesan (println)
  - Parameter tambahan (“Hello World”)



Class

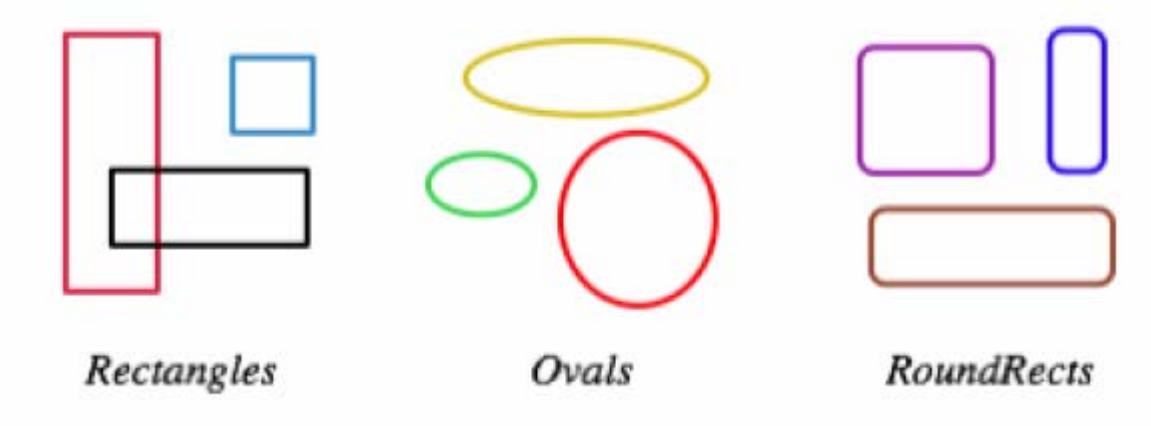
- Class adalah sebuah blueprint yang mendefinisikan variabel dan method untuk semua objek dalam class tersebut.
- Object dari suatu class dikonstruksikan saat program dijalankan (constructor)
- Contoh :
  - Student std
    - Std = new(Student);

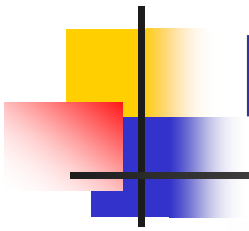
Inheritance



```
class Vehicle {
   int registrationNumber;
   Person owner;  // (Assuming that a Person class has been defined!)
   void transferOwnership(Person newOwner) {
       . . .
   }
   . . .
}
class Car extends Vehicle {
   int numberOfDoors;
   . . .
}
class Truck extends Vehicle {
   int numberOfAxels;
   . . .
}
class Motorcycle extends Vehicle {
   boolean hasSidecar;
   . . .
}
```

Polymorphism





Polymorphism

```
class Shape {

    Color color;    // Color of the shape.  (Recall that class Color
                    // is defined in package java.awt.  Assume
                    // that this class has been imported.)

    void setColor(Color newColor) {
          // Method to change the color of the shape.
       color = newColor; // change value of instance variable
       redraw(); // redraw shape, which will appear in new color
    }

    void redraw() {
          // method for drawing the shape
       ? ? ?  // what commands should go here?
    }

    . . .           // more instance variables and methods

} // end of class Shape
```