



POLITEKNIK STATISTIKA STIS
For Better Official Statistics

DESIGN PATTERNS

(PART II)

OUTLINE


- Observer
- Decorator
- Factory Method
- Singleton

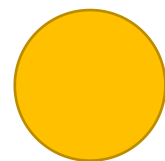


OBSERVER

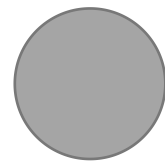
PROBLEM

Bagaimana agar suatu objek dapat mengetahui perubahan yang terjadi pada objek lain?

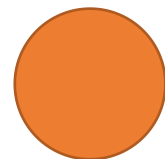
`r = 3;`
↑
`int r = 2;`

Object X



Object A



Object B

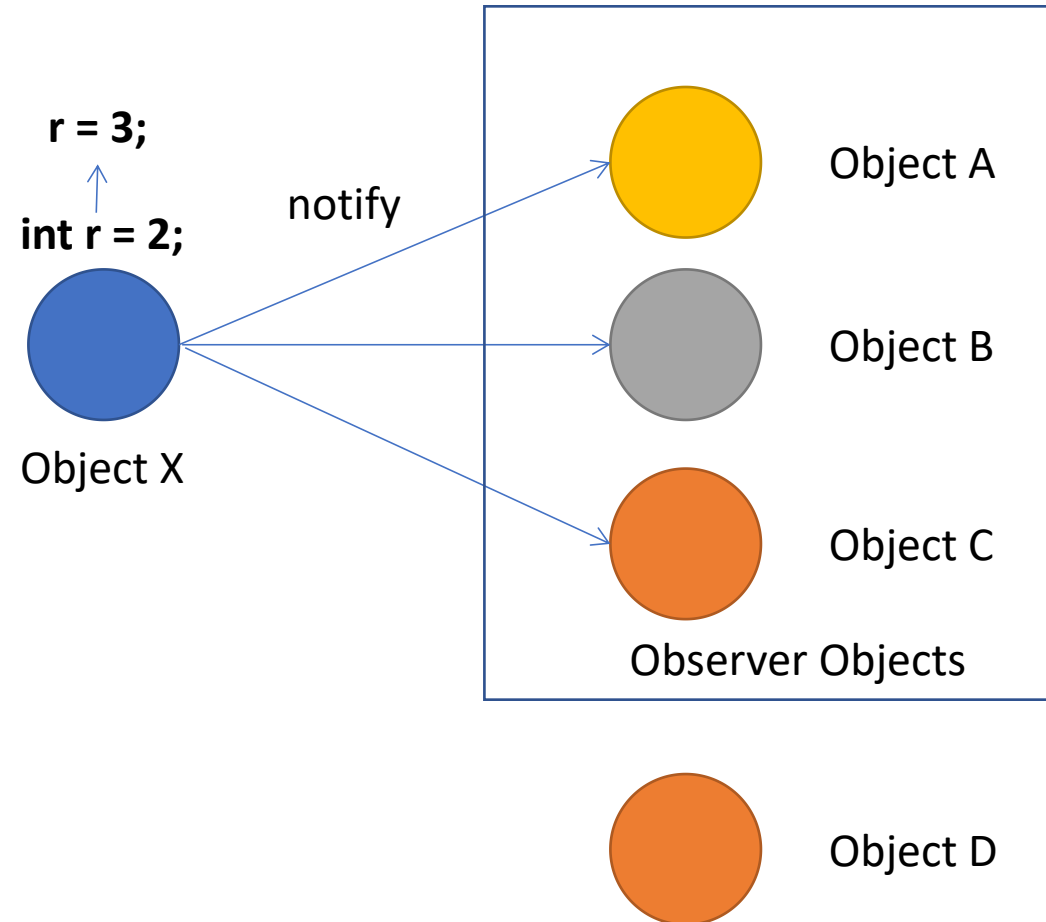


Object C

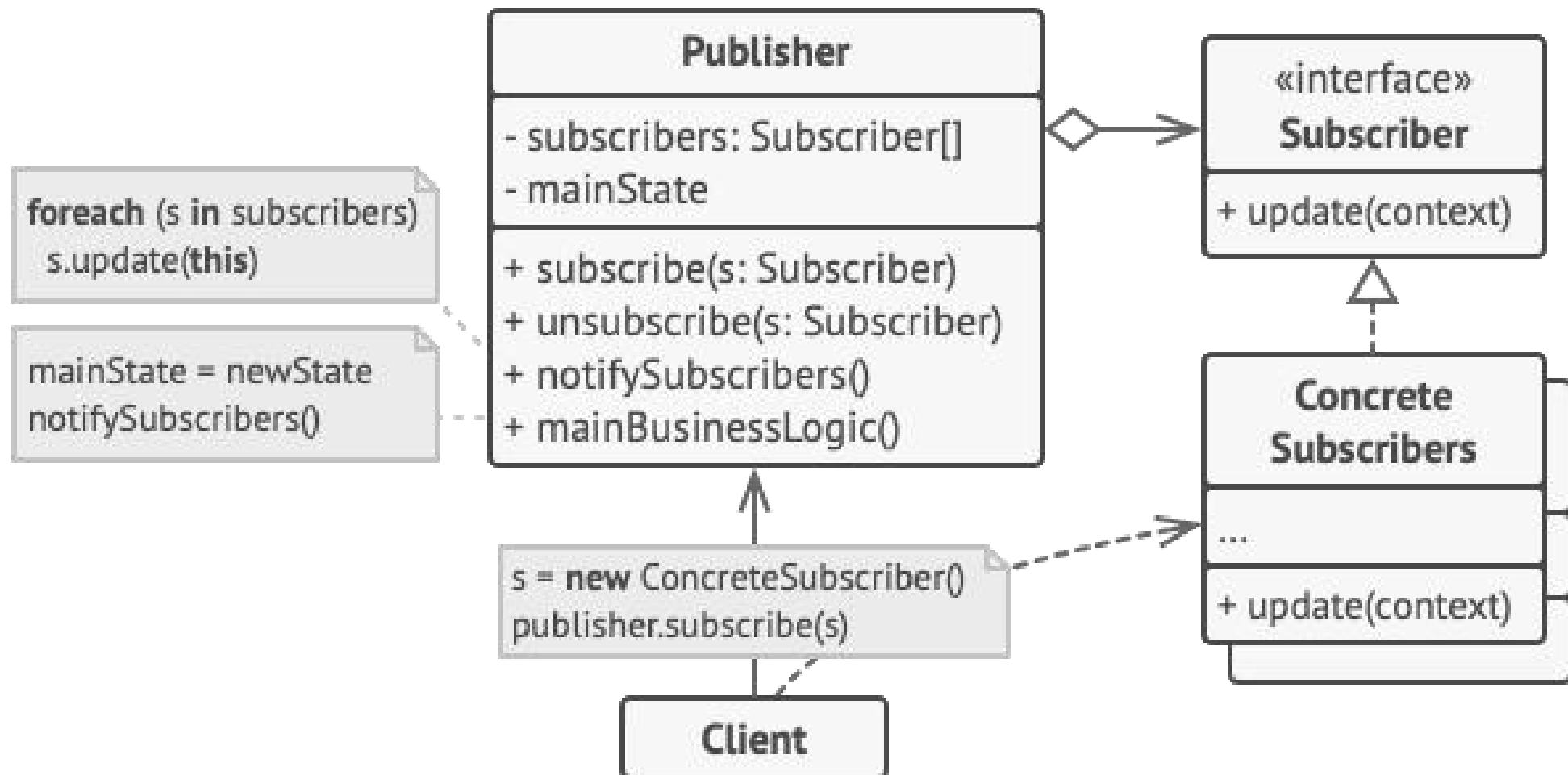
Object X...??

Observer

salah satu *behavioural design pattern* yang menyediakan mekanisme **berlangganan (*subscription*)** bagi **objek-objek** yang ingin mengetahui **perubahan** yang terjadi pada suatu **objek yang diamati**.



STRUKTUR KELAS



KAPAN DIGUNAKAN?

Ketika perubahan *state* (perubahan nilai atribut dll) dari suatu objek memerlukan perubahan pada objek-objek lainnya.

CONTOH



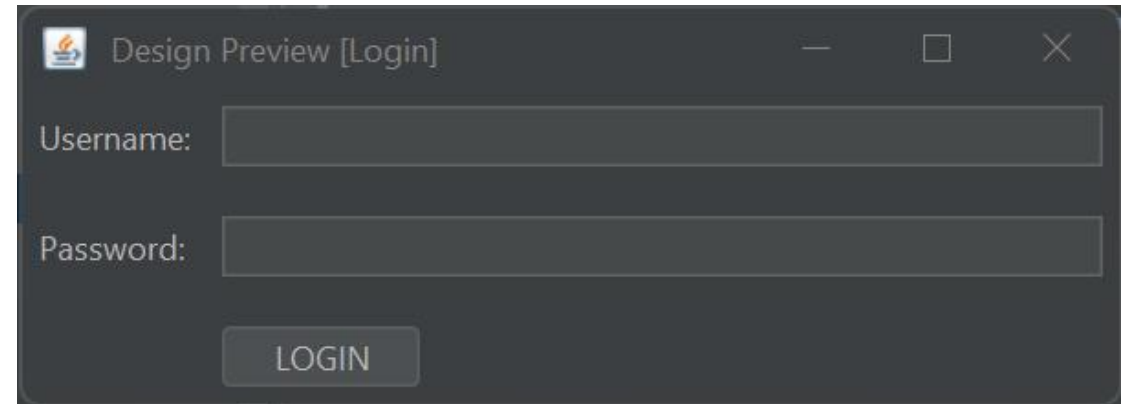
- Aplikasi Weather menampilkan kondisi suhu suatu wilayah.
- Jika suhu mengalami perubahan maka tampilan suhu juga harus berubah.
- Tampilan yang berubah adalah suhu sekarang dan suhu min dan max hari ini.
- Bagaimana design programnya?

KAPAN DIGUNAKAN?

Gunakan *observer pattern* saat beberapa objek pada aplikasi harus mengamati yang lain tetapi hanya untuk waktu yang terbatas atau dalam kasus tertentu.

CONTOH

- Implementasi action listener pada Component Java Swing.

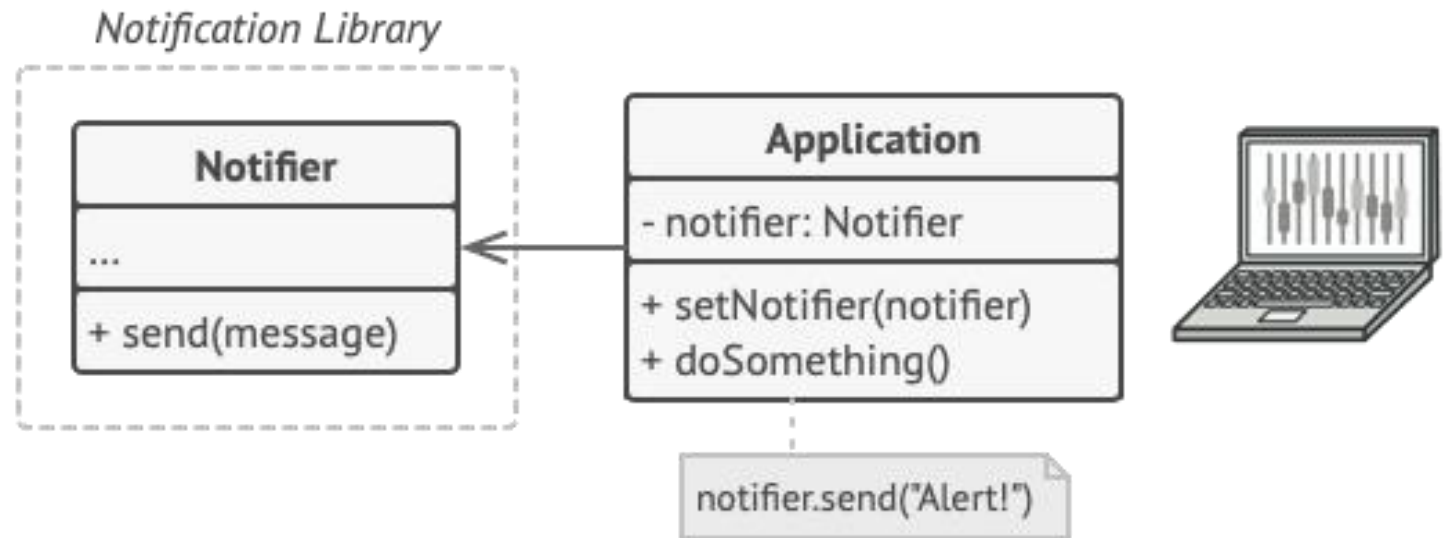




DECORATOR

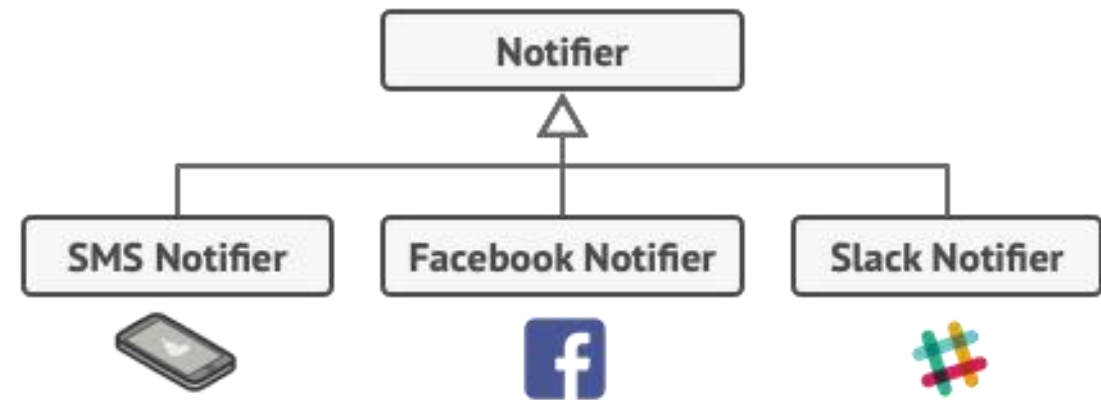
PROBLEM

- Sebuah library notifikasi menyediakan fungsi mengirim notifikasi ke pengguna aplikasi.
- Notifikasi dikirim melalui email.
- Fungsi tersebut diimplementasikan pada method **send(message)** di kelas **Notifier**
- Method ini dieksekusi pada kelas **Application**



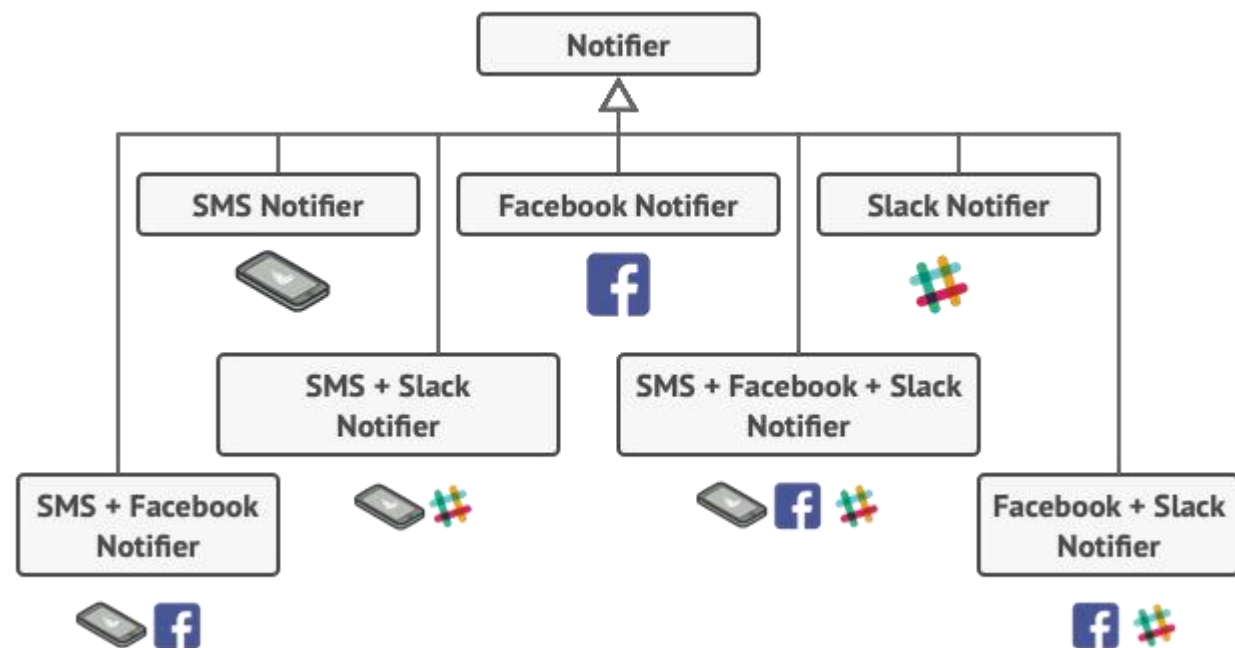
PROBLEM

- Bagaimana jika ada permintaan untuk mengirimkan notifikasi melalui SMS, Facebook, Slack, dll?
- Apa yang kita dilakukan?
- Kita bisa membuat *subclass* **Notifier** dari masing-masing media notifikasi: **SMSNotifier**, **FacebookNotifier**, **SlackNotifier**



PROBLEM

- Bagaimana jika pengguna ingin mendapatkan notifikasi dari 2 media sekaligus?
- Kita bisa membuat subclass kombinasi misal: SMSFacebookNotifier untuk mengirimkan notifikasi melalui SMS dan Facebook sekaligus.
- Bagaimana jika banyak kombinasinya?
- Jumlah subclass akan bertambah banyak.



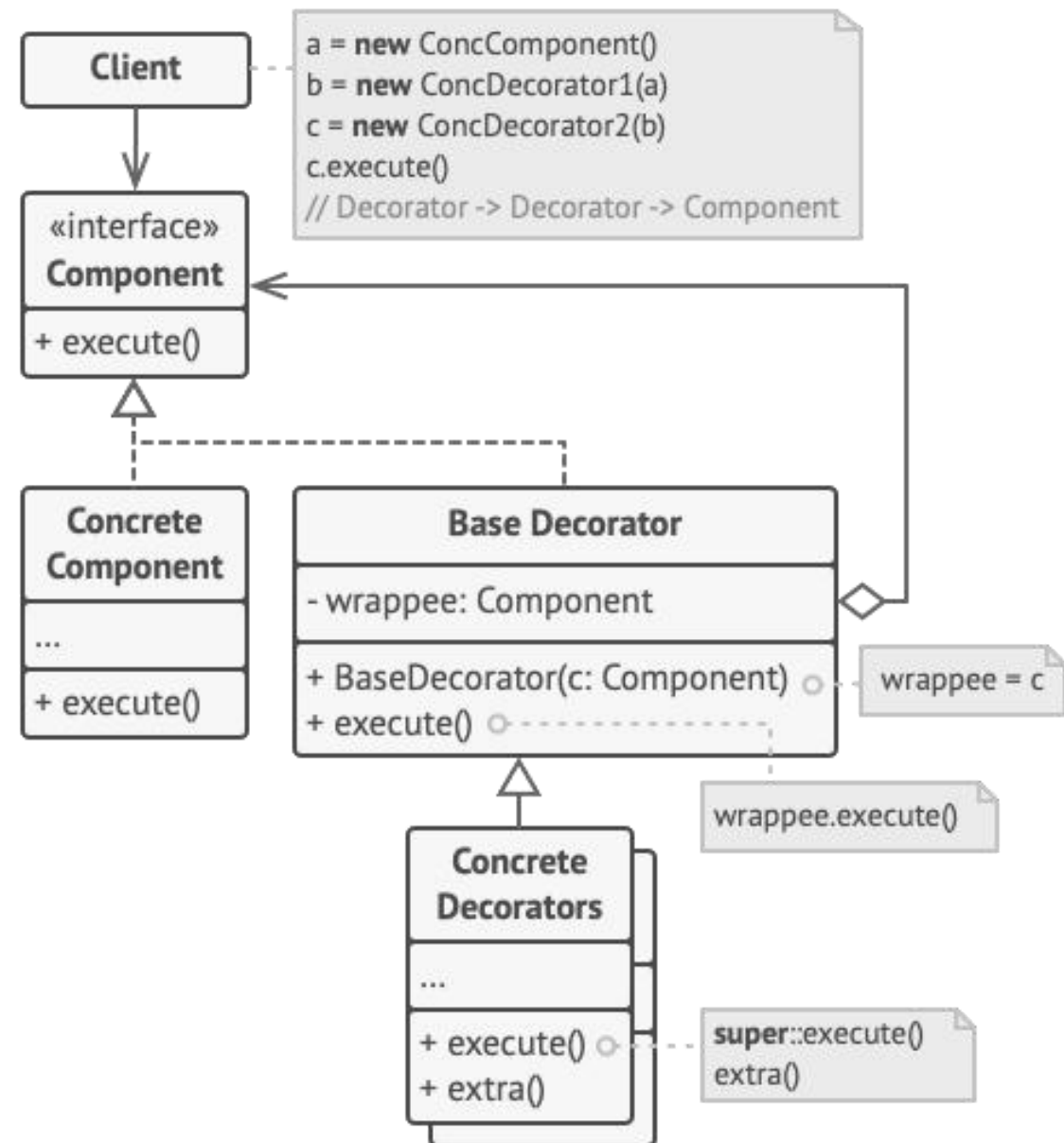
PROBLEM

Bagaimana membuat desain kelas agar dapat mengakomodir penambahan *behavior* baik *behavior* baru ataupun kombinasi *behavior* yang sudah ada?

Decorator

Salah satu *structural design pattern* yang memungkinkan penambahan *behavior* baru ke suatu objek dengan menempatkan objek tersebut ke dalam kelas *wrapper* yang memiliki *behavior* yang ingin ditambahkan.

STRUKTUR KELAS



KAPAN DIGUNAKAN?

- Gunakan *decorator pattern* ketika ingin dapat menambahkan *behavior* tambahan ke suatu objek saat *runtime* tanpa harus mengubah kode program yang menggunakan objek tersebut.
- Gunakan *decorator pattern* ketika tidak mungkin memperluas *behavior* suatu objek menggunakan *inheritance*.

FACTORY METHOD

PROBLEM

```
public interface Shape {  
    public void draw();  
}
```

```
public class Rectangle implements Shape{  
    @Override  
    public void draw() {  
        System.out.println("Draw a rectangle.");  
    }  
}
```

```
Shape shape = new Rectangle();
```

- Pembuatan objek dari *interface* dimaksudkan agar kode program menjadi lebih fleksibel.
- Jika terjadi perubahan implementasi, kita bisa membuat kelas konkrit lainnya. Kemudian mengganti inisiasi objek dengan kelas konkrit tersebut.

PROBLEM

```
Shape shape = null;
if(type.equals("circle")){
    shape = new Circle();
}else if(type.equals("rectangle")){
    shape = new Rectangle();
}else if(type.equals("square")){
    shape = new Square();
}
```

- Bagaimana jika ada penambahan implementasi?
- Ketika ada beberapa kelas konkrit, hal yang sering dilakukan adalah menginisiasi seperti kode disamping.
- Ada beberapa kelas konkrit yang diinisiasi dan kelas mana yang akan diinisiasi tergantung pada kondisi tertentu.

PROBLEM

- Bagaimana jika terdapat penambahan atau pengurangan kelas konkrit? Sementara kode di samping mungkin tidak hanya ada di satu tempat, mungkin bisa di tempat yang lain.
- Jika terdapat penambahan atau pengurangan kelas konkrit maka kita juga perlu mengubah di tempat yang lainnya.

```
Shape shape = null;
if(type.equals("circle")){
    shape = new Circle();
}else if(type.equals("rectangle")){
    shape = new Rectangle();
}else if(type.equals("square")){
    shape = new Square();
}else if(type.equals("triangle")){
    shape = new Triangle();
}else if(type.equals("diamond")){
    shape = new Diamond();
}
```

SOLUSI

```
public class ShapeFactory {  
    public static Shape createShape(String type){  
        Shape shape = null;  
        if(type.equals("circle")){  
            shape = new Circle();  
        }else if(type.equals("rectangle")){  
            shape = new Rectangle();  
        }else if(type.equals("square")){  
            shape = new Square();  
        }else if(type.equals("triangle")){  
            shape = new Triangle();  
        }else if(type.equals("diamond")){  
            shape = new Diamond();  
        }  
        return shape;  
    }  
}
```

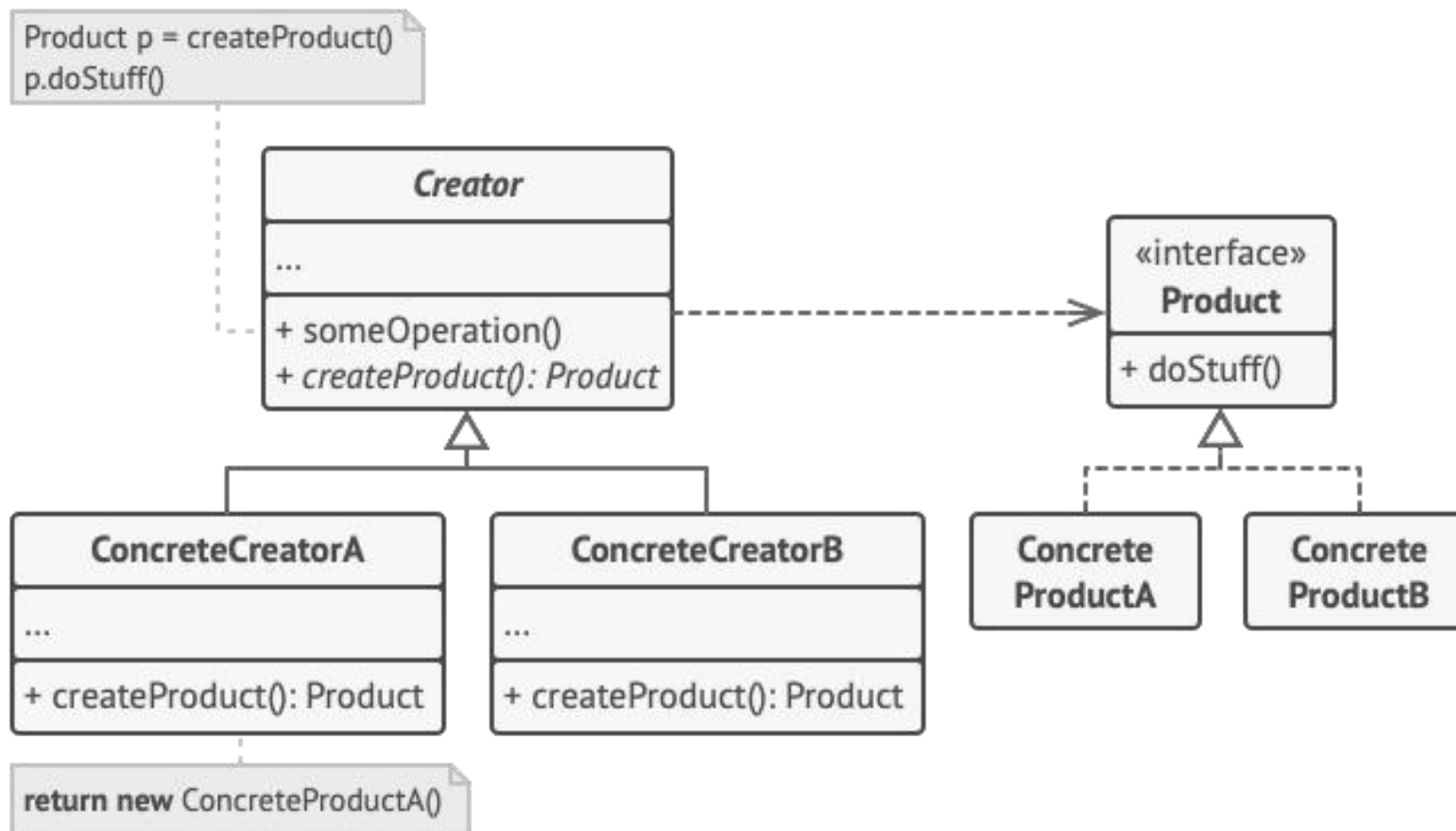
```
Shape shape = ShapeFactory.createShape(type);
```

Simple Factory Method

Factory Method

Salah satu creational design pattern yang menyediakan sebuah interface untuk membuat objek di dalam superclass tetapi membolehkan subclassnya untuk mengubah tipe dari objek yang dibuat.

STRUKTUR KELAS



KAPAN DIGUNAKAN?

- Use the Factory Method when you don't know beforehand the exact types and dependencies of the objects your code should work with.
- Use the Factory Method when you want to provide users of your library or framework with a way to extend its internal components.
- Use the Factory Method when you want to save system resources by reusing existing objects instead of rebuilding them each time.



SINGLETON

PROBLEM

```
public class DatabaseConnection {  
  
    public DatabaseConnection() {  
        System.out.println("Create database connection...");  
    }  
  
    public void execute(String sql){  
        System.out.println("execute sql: "+ sql);  
    }  
  
}
```

- Kelas **DatabaseConnection** digunakan untuk membuat koneksi ke database dan mengeksekusi SQL.

PROBLEM

```
public class OrderRepository {  
    public void save(String orderId){  
        DatabaseConnection connection = new DatabaseConnection();  
        connection.execute("INSERT INTO orders....");  
    }  
}
```

- Kelas **OrderRepository** digunakan untuk menyimpan data order ke database melalui method `save()`.
- Di dalam method ini dibuat objek **DatabaseConnection** selanjutnya menjalankan eksekusi SQL.

PROBLEM

```
public class OrderDetailRepository {  
    public void save(String orderId, String product){  
        DatabaseConnection connection = new DatabaseConnection();  
        connection.execute("INSERT INTO order_details....");  
    }  
}
```

- Kelas **OrderDetailRepository** digunakan untuk menyimpan data order ke database melalui method `save()`.
- Di dalam method ini dibuat objek **DatabaseConnection** selanjutnya menjalankan eksekusi SQL.

PROBLEM

```
public class Application {  
    public static void main(String[] args) {  
        OrderRepository orderRepository = new OrderRepository();  
        orderRepository.save("0001");  
  
        OrderDetailRepository orderDetailRepository = new OrderDetailRepository();  
        orderDetailRepository.save("0001", "Makanan");  
        orderDetailRepository.save("0001", "Minuman");  
        orderDetailRepository.save("0001", "Pakaian");  
    }  
}
```

Apa yang terjadi jika kode program di atas dijalankan?

PROBLEM

- Dalam sekali eksekusi akan ada 4 objek **DatabaseConnection** yang dibuat. Satu objek pada **OrderRepository** dan 3 objek pada **OrderDetailRepository**.
- Bagaimana jika banyak order datang sekaligus dan setiap order terdapat rata-rata 5 order detail?
- Akan ada banyak objek **DatabaseConnection** yang dibuat.
- Otomatis akan banyak koneksi ke database sehingga beban server database menjadi semakin berat.
- Selain itu untuk membuat satu koneksi ke database juga melibatkan banyak resource seperti jaringan sehingga untuk membuat satu koneksi membutuhkan 'cost' yang banyak.

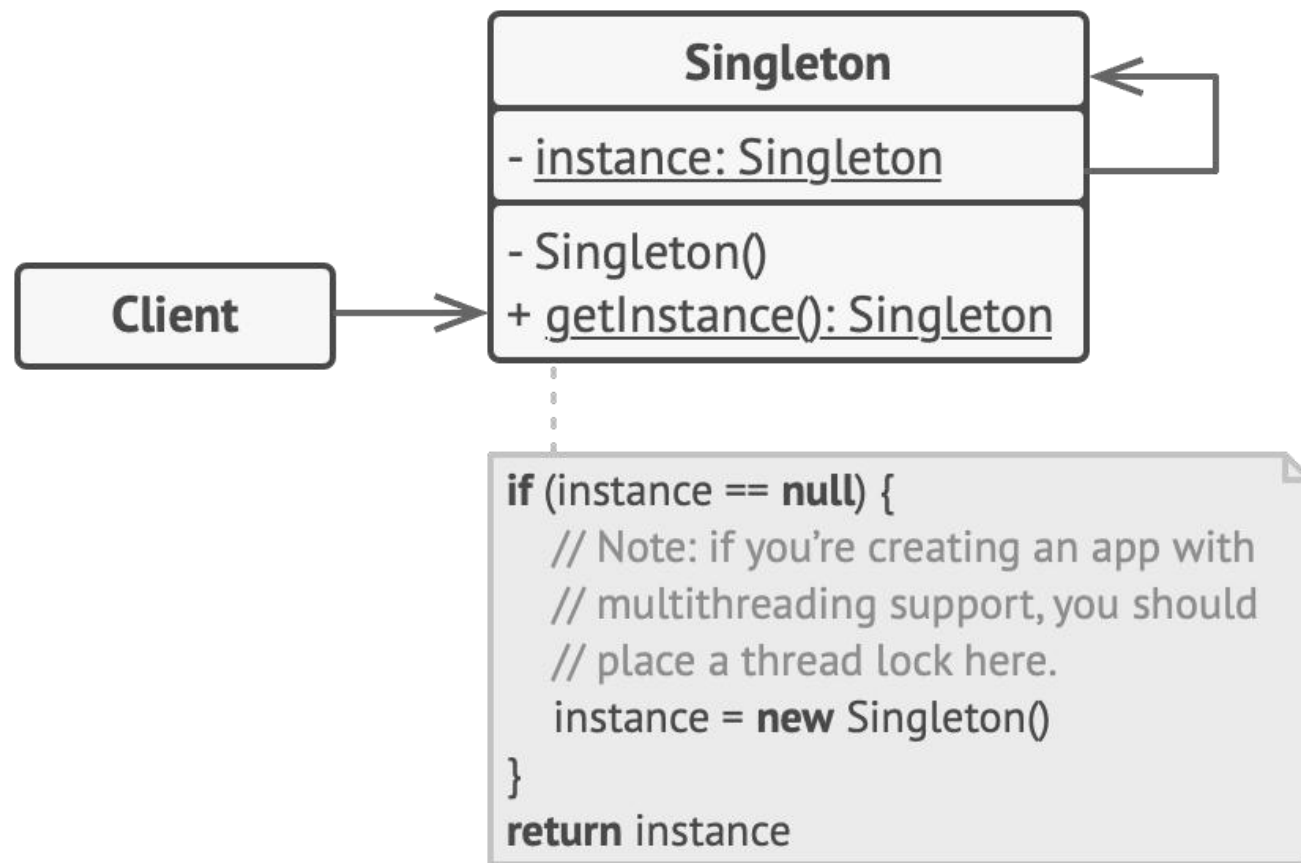
PROBLEM

- Bagaimana solusinya?
- Bisa dibuat satu koneksi ke database untuk satu aplikasi.
- Bagaimana caranya agar satu aplikasi hanya memiliki satu objek koneksi ke database?

Singleton

Creational design pattern yang memastikan hanya ada satu instance dari suatu kelas dan dapat menyediakan akses global ke dalam instance tersebut.

STRUKTUR KELAS



```
public class DatabaseConnection {  
  
    private static DatabaseConnection instance;  
  
    private DatabaseConnection() {  
        System.out.println("Create connection...");  
    }  
  
    public static DatabaseConnection getInstance(){  
        if(instance==null){  
            instance = new DatabaseConnection();  
        }  
        return instance;  
    }  
  
    public void execute(String sql){  
        System.out.println("execute sql: "+ sql);  
    }  
  
}
```

KAPAN DIGUNAKAN?

- Gunakan singleton ketika suatu kelas hanya tersedia satu instance untuk semua client.
- Gunakan singleton ketika memerlukan kontrol ketat terhadap global variable.

REFERENCES





Terima kasih