Analisa dan Desain Berorientasi Objek

GRASP Pattern

Materi :

- Apa itu Pattern
- GRASP Pattern:
  - GRASP 1: Expert
  - GRASP 2: Creator
  - GRASP 3: High Cohesion
  - GRASP 4: Low Coupling
  - GRASP 5: Controller

Apa itu Pattern

Pattern adalah sebuah solusi yang popular dalam memecahkan permasalahan pada sebuah desain perangkat lunak.

Contoh: Dalam rangka memasak ayam krispi

- Solusi 1: cuci ayam, masukkan ke tepung kering, goreng.
- Solusi 2: cuci ayam, masukkan ke tepung basah, masukkan ke tepung kering, goreng.
- Solusi 3: cuci ayam, masukkan ke tepung basah, goreng.

Dapat kita amati bahwa semua solusi (1-3) tidak ada yang salah. Semuanya bisa menghasilkan ayam krispi.

Namun dari semua solusi, rupanya yang paling populer adalah solusi 2. Oleh karena itu solusi 2 ini disebut pattern dalam hal memasak ayam krispi.

GRASP Pattern

Sebuah pattern yang memungkinkan kita untuk merancang behaviour/method pada class dengan cara yang elegan.

GRASP singkatan dari “General Responsibility Assignment Software Patterns”.

Ada beberapa pattern dalam GRASP:

- Expert, Creator, High Cohesion, Low Coupling, Controller.

GRASP 1

Expert Pattern

Apa itu Expert Pattern

Meletakkan behaviour pada class yang sesuai

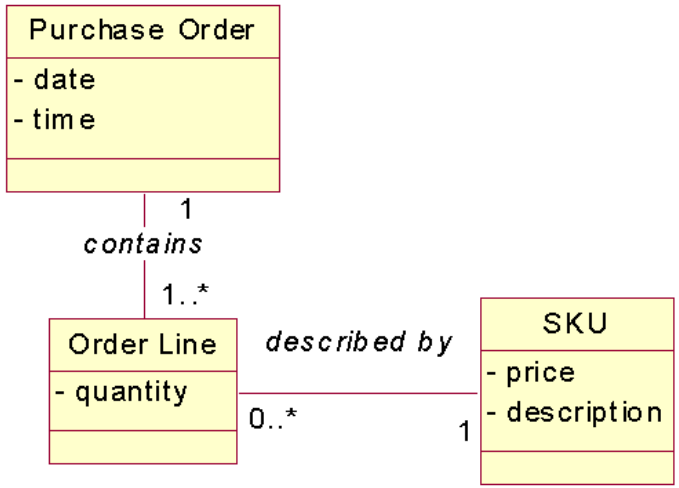
Dengan tujuan agar sistem dapat:

- Dipahami dengan mudah
- Dikembangkan dengan mudah
- Reusable
- Robust

Contoh

Pada contoh kasus sistem pembelian. Terdapat class:

- PurchaseOrder: Setiap ada pembelian, class ini mencatat tanggal dan waktu pembelian. Dan terdapat atribut array of OrderLine.
- SKU: Mendeskripsikan barang yang ada, nama barangnya dan harganya.
- OrderLine: Mendeskripsikan barang apa yang dibeli dan berapa jumlahnya. Dan terdapat atribut SKU.



Contoh (lanjutan)

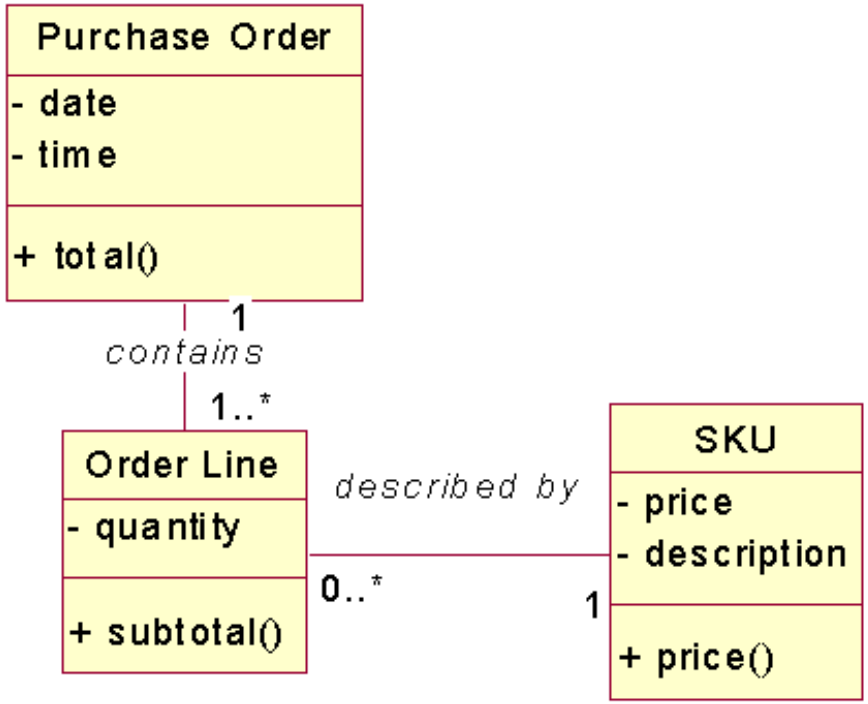
Bagaimana jika sistem harus menampilkan total harga pembelian? Di class apa seharusnya kita letakkan method **total()**?

Menurut Expert Pattern, method yang berurusan dengan total harga pembelian seharusnya diletakkan di class **PurchaseOrder**, karena class tersebutlah yang seharusnya menjadi “expert” dalam hal order pembelian.

Kemudian agar method **total()** dapat menjumlahkan total harga pada pembelian, dibuatlah method **subtotal()** pada class **OrderLine()**

Contoh (lanjutan)

Sehingga conceptual diagramnya menjadi seperti ini:



GRASP 2

Creator Pattern

Apa itu Creator Pattern

Creator pattern adalah penerapan Expert Pattern secara lebih spesifik:

- Class manakah yang bertanggung jawab untuk membuat objek dari sebuah class yang lain?

Jawabannya adalah:

Class A bertanggung jawab untuk membuat objek dari class B jika:

- A mengandung property objek dari class B
- A menggunakan objek B
- A menginisialisasi data yang akan diset ke objek B

Contoh

Pada contoh kasus sistem pembelian.

Ketika ada order pembelian baru, class mana yang bertanggung jawab untuk membuat objek dari class **OrderLine**?

Solusinya adalah, class **PurchaseOrder** lah yang bertanggung jawab untuk membuat objek dari **OrderLine.**

- Karena didalam **PurchaseOrder** terdapat atribut array of **OrderLine**.

GRASP 3

High Cohesion Pattern

Apa itu High Cohesion Pattern

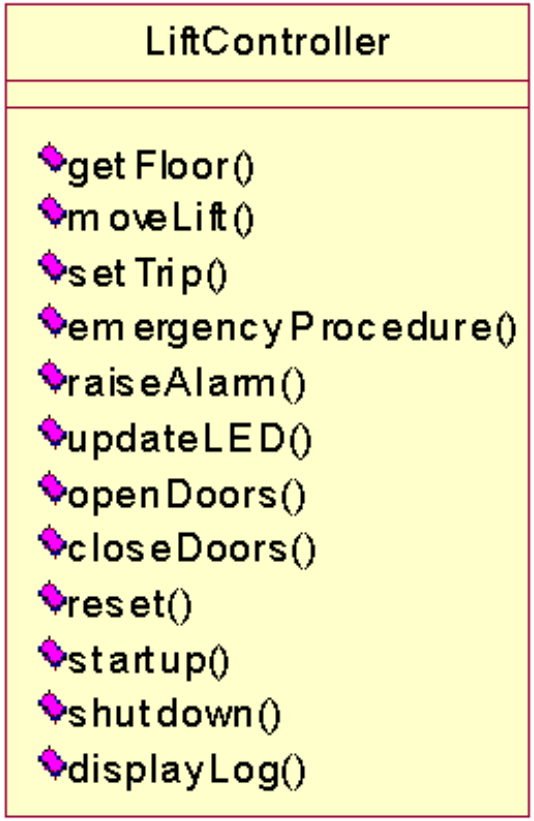
Memastikan bahwa fungsionalitas sebuah class harus fokus di satu hal saja.

Dalam desain yang baik, sebuah class tidak boleh melakukan terlalu banyak hal.

Desain yang baik adalah setiap class hanya memiliki sedikit method saja, yang berkaitan langsung dengan fungsionalitas class itu sendiri.

Contoh

Pada contoh kasus Lift Management System.



Contoh (lanjutan)

Dapat kita amati bahwa class LiftController melakukan banyak tugas: menyalakan alarm, startup/shutdown, menggerakkan lift, update LED, dsb.

Hal tersebut bukan desain yang baik.

- Dengan kata lain: class tersebut **tidak cohesive**

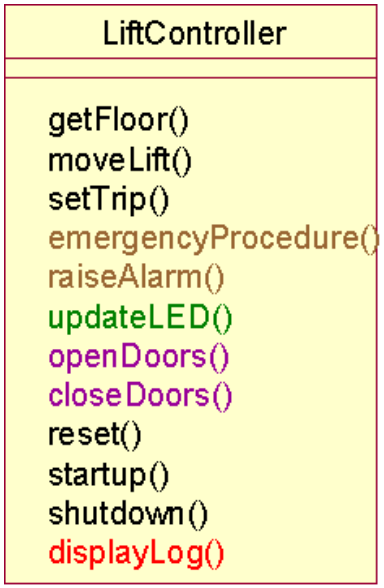
The rule of thumb (seharusnya): Sebuah class haruslah merepresentasikan seperti halnya objek di dunia nyata, atau sebuah “key abstraction”.

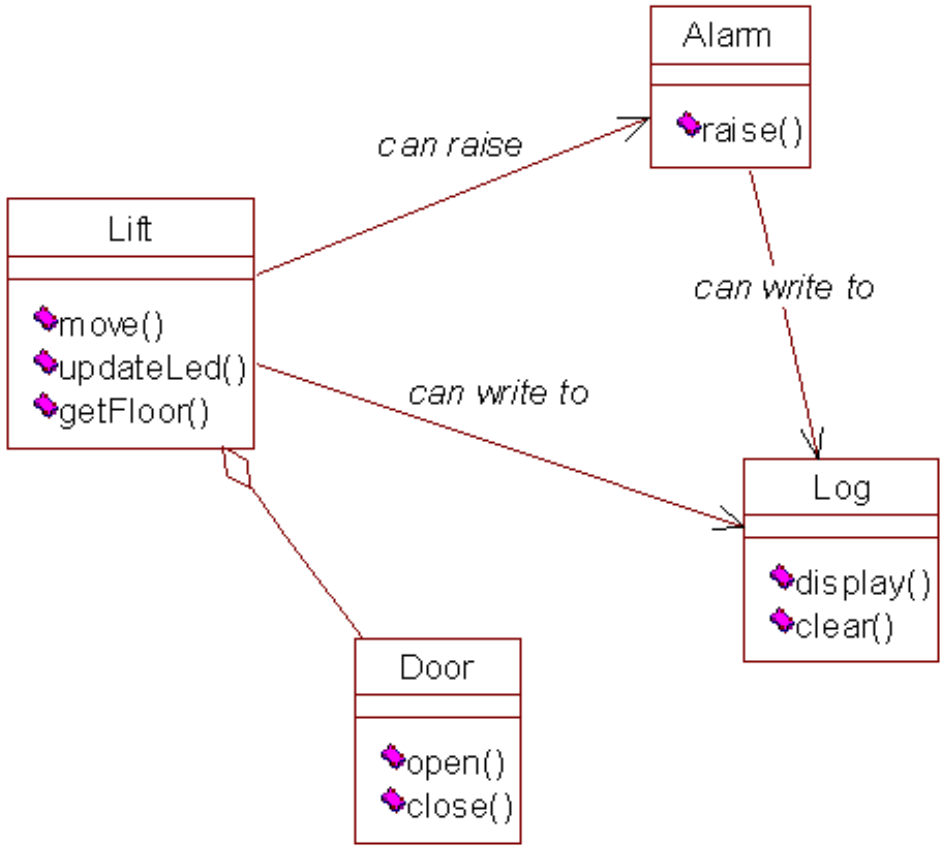
Contoh (lanjutan)

Jika dilihat di class diagram LiftController, class tersebut setidaknya berusaha memodelkan 3 key abstraction: Alarm, Lift, Pintu, dan Log.

Jadi desain yang baik seharusnya memisahkan class LiftController tersebut menjadi beberap class:

Contoh (lanjutan)





GRASP 4

Low Coupling Pattern

Apa itu Low Coupling Pattern

Coupling adalah tingkat ketergantungan sebuah class dengan class yang lain.

High Coupling berakibat pada kode program yang sulit di-maintain.

- Satu perubahan pada sebuah class dapat mempengaruhi fungsionalitas class yang lain.

Desain yang baik adalah yang menganut Low Coupling.

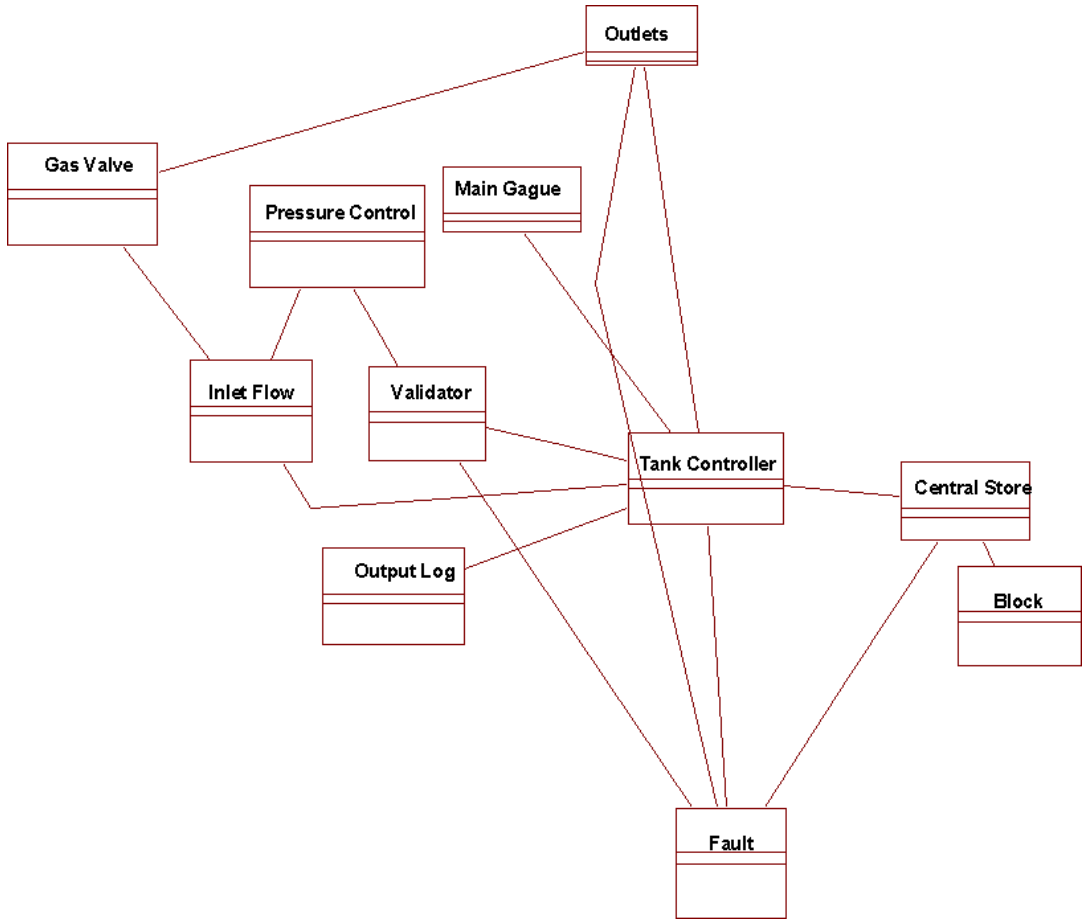
- Satu perubahan pada class tidak akan mempengaruhi fungsionalitas class yang lain.

Contoh 1

Contoh desain High Coupling.

Dapat kita lihat bahwa terlalu banyak relasi asosiasi pada class TankController.

Class tersebut berpotensi tidak cohesive dan melakukan terlalu banyak tugas.



Contoh 2

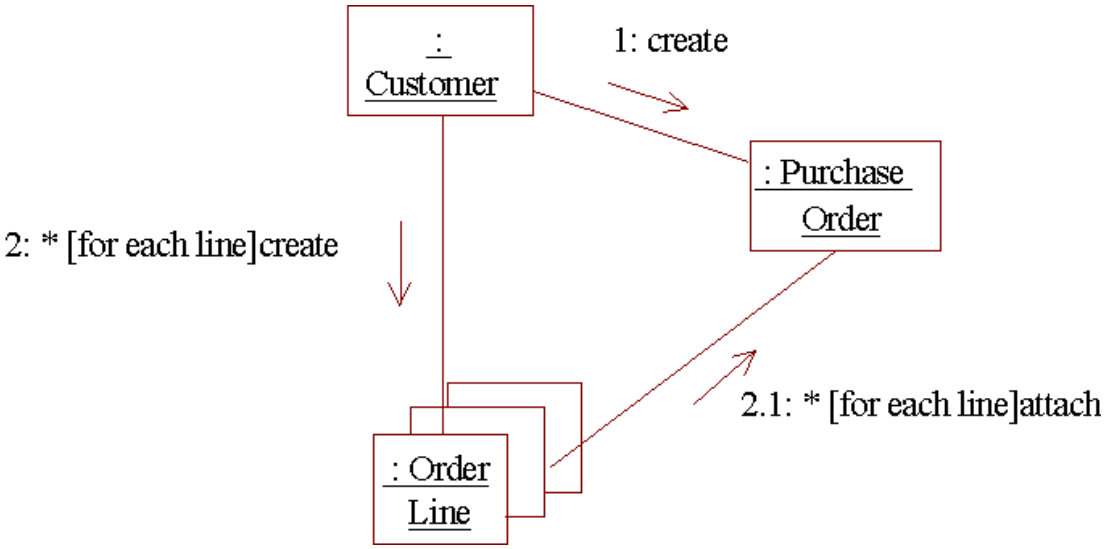
Pada contoh kasus sistem pembelian.

Ketika ada order pembelian, sistem akan meng-create objek Customer, lalu class Customer akan meng-create objek PurchaseOrder.

Yang jadi pertanyaan adalah, class mana yang bertanggung jawab untuk meng-create objek OrderLine?

Contoh 2 (lanjutan)

Solusi 1: Class Customer meng-create PurchaseOrder, lalu tiap barang yang dibeli, Customer meng-create OrderLine

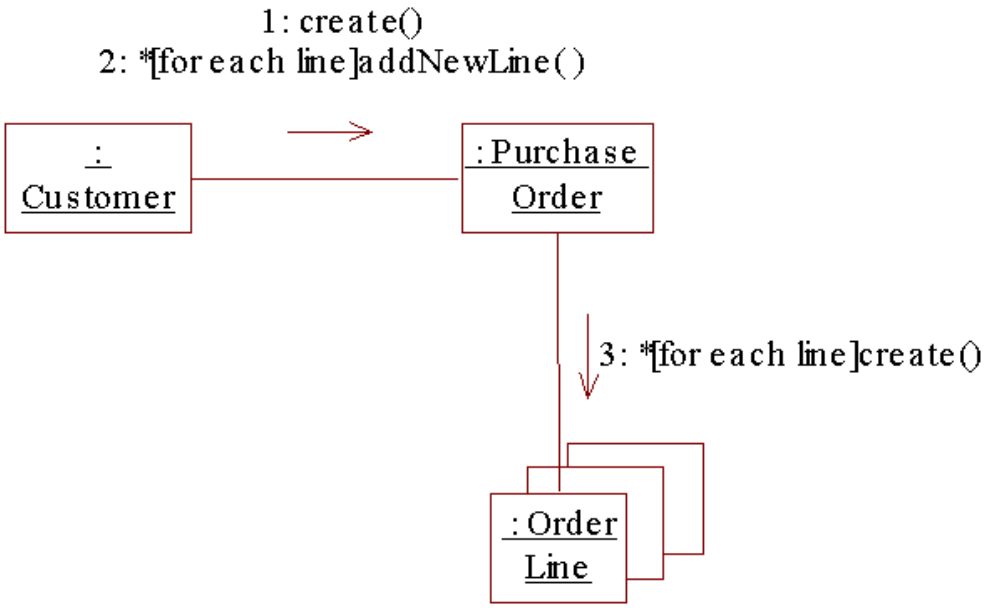


Permasalahannya adalah, ketiga class tersebut jadi saling bergantung satu sama lain. Jika ada perubahan pada OrderLine, class PurchaseOrder dan Customer akan ikut terpengaruh.

- Hal ini termasuk dalam kategori **High Coupling**.

Contoh 2 (lanjutan)

Solusi 2: Class PurchaseOrder yang bertugas meng-create OrderLine



Dengan demikian, jika ada perubahan pada class OrderLine, hanya akan mempengaruhi class PurchaseOrder.

- Dibanding dengan solusi sebelumnya, solusi ini lebih **Low Coupling**

The Law of Demeter

Atau disebut juga: *Don't Talk to Strangers*

Adalah metode yang efektif untuk menghindari high coupling.

Metode ini menyarankan agar sebuah objek hanya boleh memanggil method milik:

- Dirinya sendiri (method didalam class itu sendiri)
- Parameter yang dimasukkan ke method tersebut
- Object yang dibuat didalam class itu sendiri

Pertimbangan Dalam Coupling

Beberapa hal yang perlu dipertimbangkan:

- Jangan pernah membuat atribut/property class public
- Hanya sediakan method getter-setter jika diperlukan saja
- Buat se-sedikit mungkin method yang dapat diakses dari luar class
- Minimalisir data yang dilempar/diteruskan sebagai parameter ke sebuah method
- Jangan terlalu memaksakan Low Coupling yang terlalu ekstrim hingga tidak terjadi interaksi class sama sekali (ingat pattern High Cohesion dan Expert)

GRASP 5

Controller Pattern

Contoh Kasus

Misalkan dalam sebuah sistem Taruhan Pacuan Kuda, atau disebut *BookMaking System*:

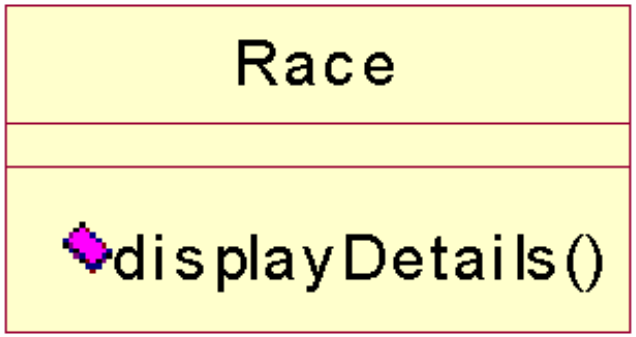
Terdapat class **Race** yang menghandle detail pacuan kuda yang sedang berlangsung.

Jika diharapkan sistem dapat menampilkan ke layar detail pacuan kuda tersebut ke pengguna, siapakah yang bertugas untuk menampilkannya?

Menurut Expert Pattern

Menurut Expert Pattern, karena class Race menghandle detail dari pacuan kuda, maka class tersebut juga bertugas untuk menampilkan detail tersebut ke layar.

Sehingga dibuatlah method **displayDetails()** didalam class tersebut.



Padahal Salah!

Sepertinya solusi tersebut ok.

Tapi ternyata hal itu melanggar Expert Pattern!

Why?

Class Race memang menghandle detail dari pacuan kuda yang sedang berlangsung.

Tapi class Race tidak seharusnya menghandle apapun yang berkaitan dengan Graphical User Interface.

- Dalam hal ini untuk menampilkan detail pacuan kuda tersebut.

Bagaimana Seharusnya?

Jangan mencampuradukkan fungsionalitas class yang berkaitan dengan bisnis proses dengan akses ke GUI (atau database, atau objek fisik lain).

Sebisa mungkin pisahkan class yang berkaitan dengan bisnis proses, dan class yang berkaitan dengan GUI (begitu juga class yang berkaitan dengan database, dsb).

Solusi: Controller Pattern

Kita buat class baru yang menjadi perantara antara pengguna dan class yang berkaitan dengan bisnis proses.

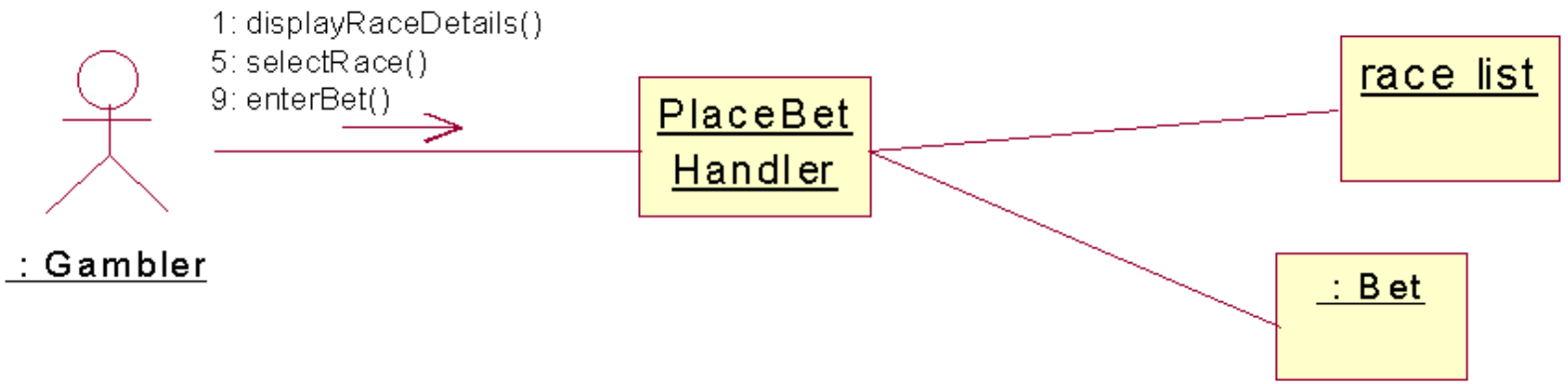
Class ini disebut dengan Controller, dan biasanya diberi nama **<UseCaseName>Handler**.

Dalam kasus ini, dibuatlah class **PlaceBetHandler**.

Class ini bertugas untuk membaca input dari pengguna, dan memutuskan class mana yang dipanggil untuk menangani input tersebut.

Dan class ini juga yang bertugas untuk menampilkan detail pacuan kuda ke layar.

Solusi: Controller Pattern (lanjutan)



Dengan demikian, jika suatu saat dibutuhkan untuk mengganti user interface (misal dari web ke mobile) maka hanya perlu mengubah class Controller-nya.

Thanks!

Any questions?