Chapter 1.

→ Patterns

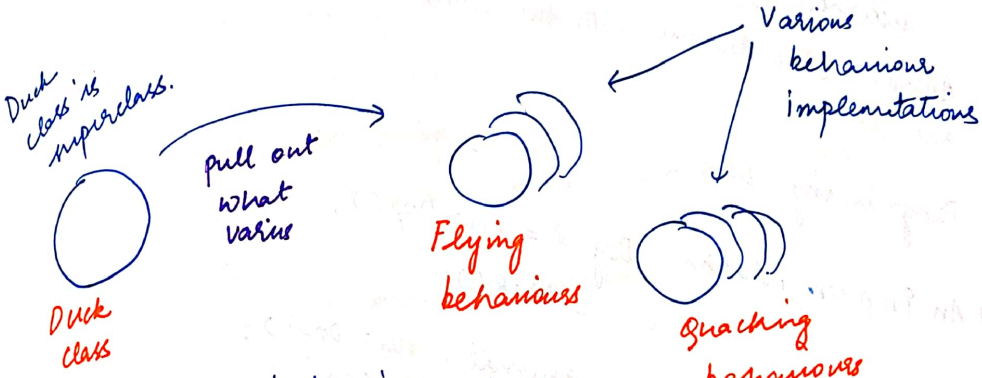
→ A localized update caused a non-local side effect.

→ Interfaces in Java
- It can have variables and methods but they will be abstract.
- blueprint of the class

→ Java interfaces have no implementation code, so no code reuse.

DP: Identify aspects of your application that vary and separate them from what stays the same.

→ let some part of a system vary independently of all other parts.



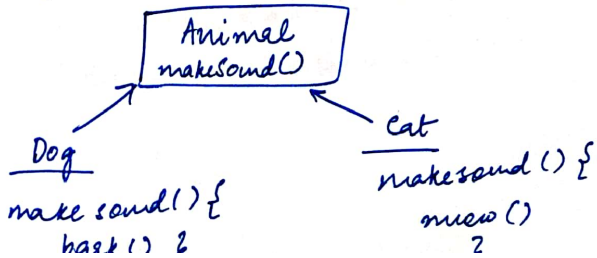
→ we want to change behaviours at runtime

DP: Program to an interface, not an implementation.

→ Program to an interface means program to a supertype. Exploit polymorphism, actual runtime isn't locked.

→ Polymorphism in Java
- Ability of a message to be displayed in more than one form.
- Same person possess different behaviours in different situations.

→



→ The big picture on encapsulated behaviours:

IS-A

HAS-A

IMPLEMENTS

↗ allows to change behaviour at run-time.

DP: Favor composition over inheritance.

→ Instead of inheriting, ducks get behaviour by being 'composed' with right behaviour object.

STRATERGY

- Defines a family of algorithms, encapsulates each one and makes them interchangble.
- Lets algorithm vary independently from clients that use it.

* Abstract class Animal with two concrete implementations Dog and Cat.

Programing to

1) An implementation
```
Dog d = new Dog();
d.bark();
```

2) An interface
```
Animal animal = new Dog();
do animal.makeSound();
```

Best:
```
a = getAnimal();
a.makeSound();
```

* Composition allows us to encapsulate a family of algorithms into thier own set of class, as well as change behaviour at runtime.

* shared vocabulary-patterns are powerful.