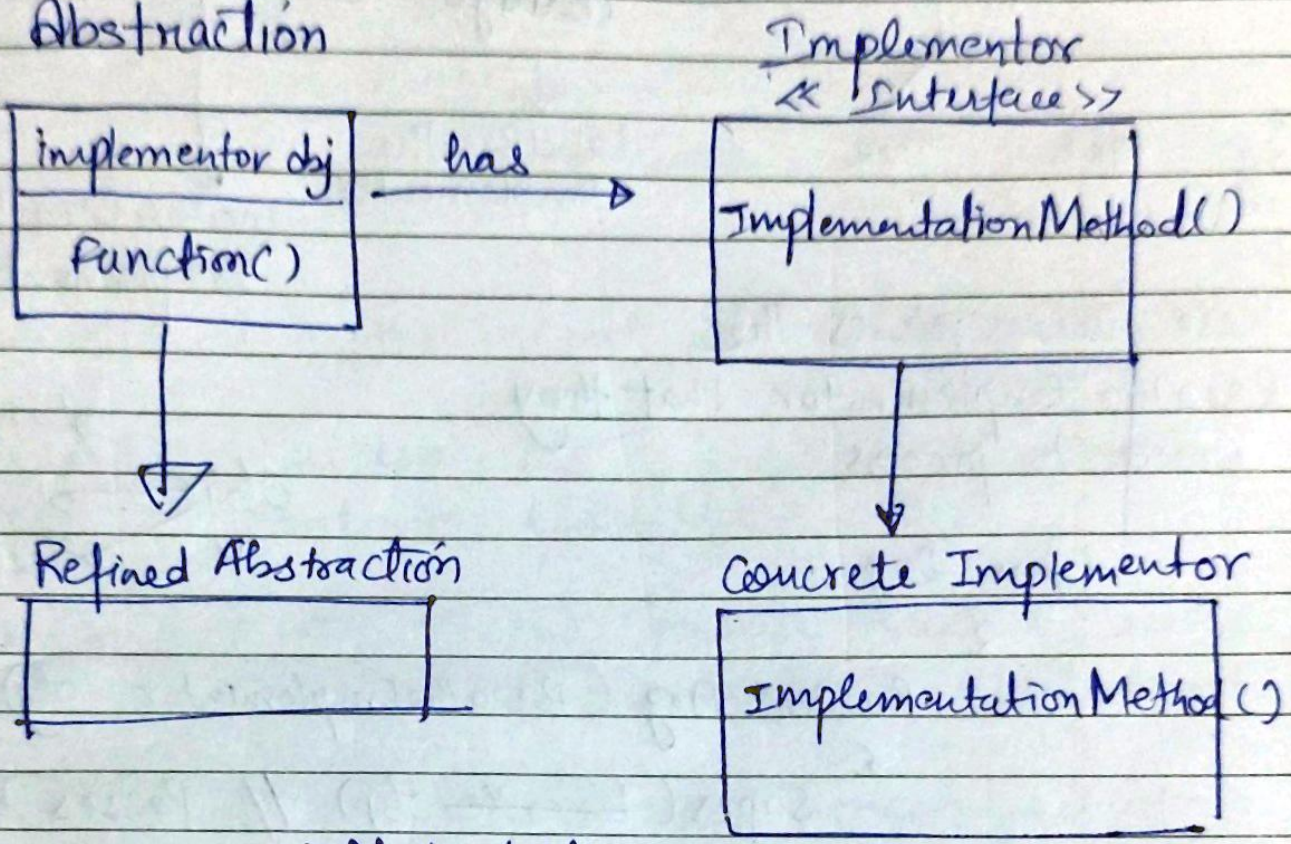
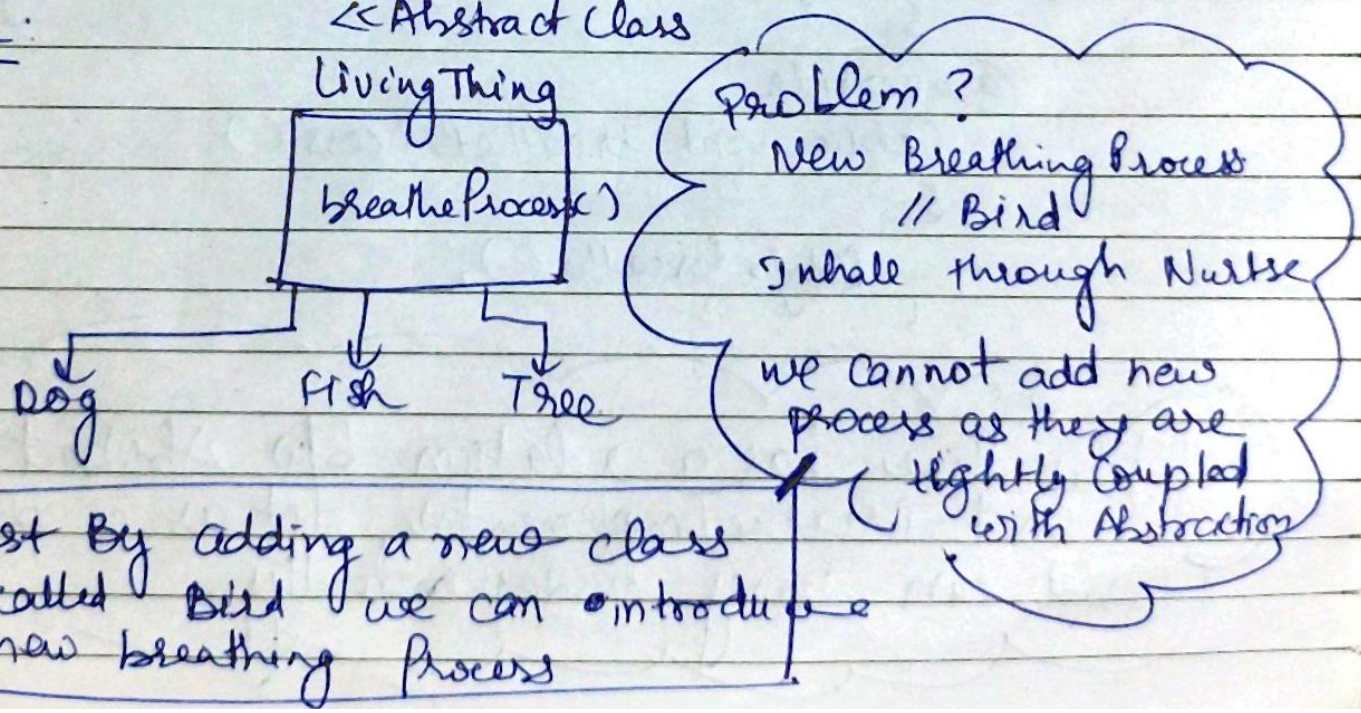
Bridge Design Pattern

Bridge Design pattern decouples an abstraction from its implementation so that two can vary independently

UML

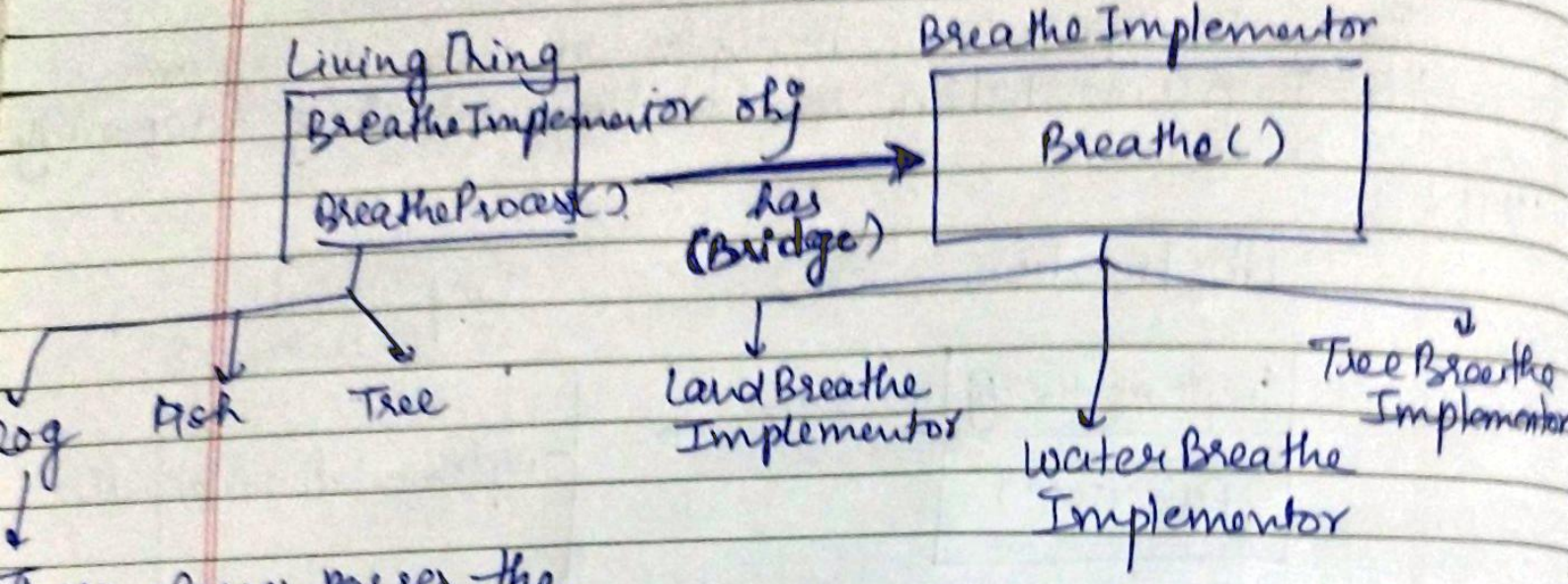


Ex:

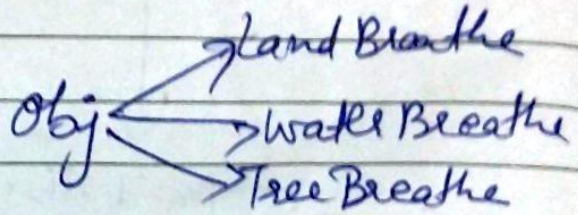


DATE : __/__/__
PAGE : _____

Solution to the Problem



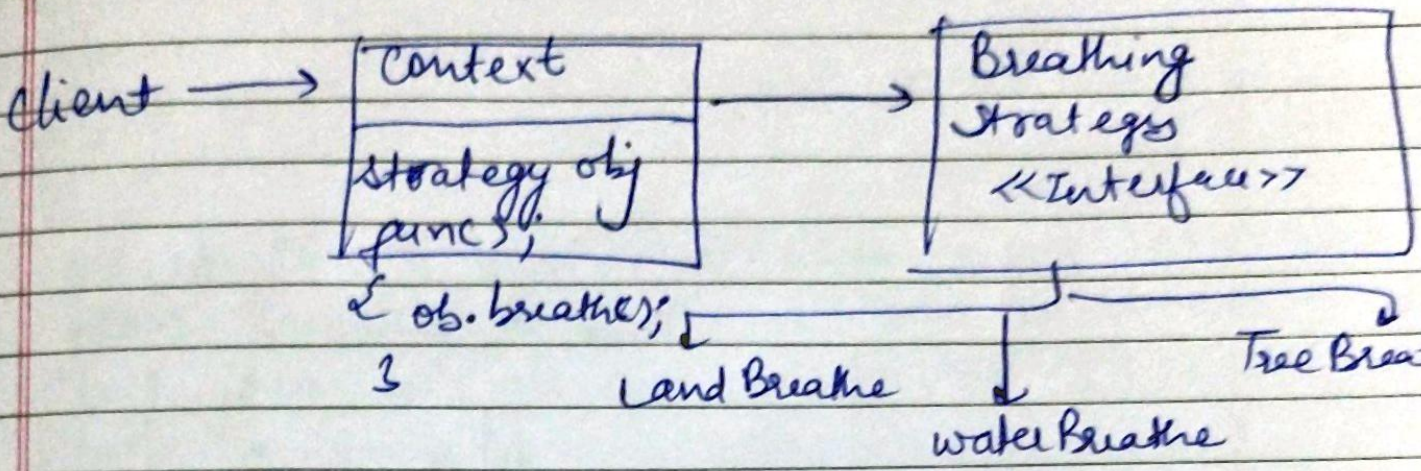
These classes passes the BreatheImplementor that they want to process




```
Class Dog
  {
    Public Dog (BreatheImplementor obj)
     {
       super(~~breath~~ obj) // Passes the breathe Implementor
     }
  @override
    Public void breatheProcess()
    {
      Obj.breathe();
    }
}
```

Now Here has a relation b/w Abstract class and BreatheImplementor act as a bridge and can vary independently.

Strategy Vs Bridge



Difference is in the intent ~~for~~ b/w Context and Abstract

In Strategy pattern, let say client create an object of ~~Context~~ class

```
Context obj = new Context( new landBreathe());
              new context (new waterBreathe());
```

These changes can be done in same object at ~~runtime~~ dynamically at Runtime.

In Bridge Pattern

If let say the Context/Abstract grows then it must seperate the logic and grow independently