T-302-HONN: Lab 3
Design Principles and Patterns

Þórður Friðriksson
thordurf@ru.is

Háskólinn í Reykjavík — Reykjavík University

Inngangur

Takið eftir: Vinsamlegast lesið yfir alla eftirfarandi punkta áður en þið byrjið á verkefninu

1. **Hópar**
   - Þetta verkefni getur verið skilað í 1-2 tveggja manna hópum
   - Þið þurfið eingöngu að skila verkefninu einu sinni per hóp
   - Munið að að merkja verkefnið á forsíðu með nafni og netfangi á öllum hópmeðlimum
2. **Reglur um svindl**
   - Lausnin verður að vera ykkar eigið verk, ef upp kemst um svindl fá allir tengdir aðilar 0 fyrir verkefnið ef þetta er fyrsta brot, ef þetta er endurtekið brot má búast við harðari refsingu. Sjá reglur skólans um verkefnavinnu: ru.is/namid/reglur/reglur-um-verkefnavinnu
3. **Verkefnaskil**
   - Verkefnum skal vera skilað á Canvas bæði sem pdf skrá fyrir öll skrifleg svör og sem zip skrá fyrir kóðann
   - pdf skráin skal heita: `{student1@ru.is}-{student2@ru.is}-lab3.pdf`
   - zip skráin skal heita: `{student1@ru.is}-{student2@ru.is}-lab3.zip`
   - **Ef ekki er fylgt fyrirmælum um verkefnaskil má búast við niðurlækkun á einkun**
4. **Sein skil**
   - Sjá late submission policy

1 Design Principles (40 stig)

Gefið stuttar og hnitmiðaðar lýsingar á design principles þegar það er beðið um þær, útskýrið hvað principle-in ganga út á og hvatan bakvið hvert og eitt

Ath Kóðinn var gerður með **python 3.9**

1.1 Law of Demeter (10 stig)

1. *(Heildar stig: 3)* Útskýrið í stuttu máli *Law of Demeter* og hvatan bakvið það

2. *(Heildar stig: 3)* Hver er helsti gallinn við *Law of Demeter*

3. *(Heildar stig: 4)* Hér má sjá mynd af `ShapeRender` klasa sem hefur fallið `draw_circle` sem býr til `circle` hlut og teiknar circle hlutinn.

```python
class ShapeRenderer:
    def __init__(self, shape_factory: ShapeFactory):
        self.__shape_factory = shape_factory

    def draw_circle(self):
        circle = self.__shape_factory.CreateCircle()
        circle.draw()
```

   (a) *(Heildar stig: 2)* Af hverju er þetta tæknilega séð brot á *Law of Demeter*?

   (b) *(Heildar stig: 2)* Þrátt fyrir að vera tæknilega séð brot á *Law of Demeter*, af hverju er þetta mjög vægt brot og af hverju ætti þetta ekki að vera "lagað" til að uppfylla *Law of Demeter*?

1.2 SOLID (10 stig)

1. Útskýrið öll *SOLID* principle-in í stuttu máli og hvatan bakvið þau

   (a) *(Heildar stig: 2)* ***S**ingle Responsibility Principle*

   (b) *(Heildar stig: 2)* ***O**pen-Closed Principle*

   (c) *(Heildar stig: 2)* ***L**iskov Substitution Principle*

   (d) *(Heildar stig: 2)* ***I**nterface Segregation Principle*

   (e) *(Heildar stig: 2)* ***D**ependency Inversion Principle*

1.3 Berið kennsl á brotin (20 stig)

Í þessum part eigi þið að bera kennsl á þau design principle brot sem eru í hverju og einu dæmi sem og gera tillögur að lagfæringur þegar svo er beðið um

Ath að geta verið eitt eða fleiri design principle brot í hverju dæmi

1. *(Heildar stig: 5)*

   (a) *(Heildar stig: 2.5)* Hvað/hver er/u brotið/in á myndinni fyrir neðan? Útskýrið

```python
class OrderService:
    def __init__(self, order_repository: OrderRepository):
        self.__order_repository = order_repository

    def submit_order(self, order: Order):
        self.__order_repository.save_order(order)
        self.__order_repository.connection.commit()
```

   (b) *(Heildar stig: 2.5)* Hvernig myndu þið laga brotið/in?

2. *(Heildar stig: 5)*

   (a) *(Heildar stig: 2.5)* Hvað/hver er/u brotið/in á myndinni fyrir neðan? Útskýrið

```python
class Bird(ABC):
    @abstractmethod
    def lay_egg(self):
        pass

    @abstractmethod
    def swim(self):
        pass

    @abstractmethod
    def fly(self):
        pass


class Penguin(Bird):
    def lay_egg(self):
        print('laying a penguin egg')

    def swim(self):
        print('swim swim')

    def fly(self):
        raise NotImplementedError('Penguins don\'t fly')
```

   (b) *(Heildar stig: 2.5)* Hvernig getið þið lagað brotið/in en ennþá notað inheritance(semsagt ekki laga með composition)

3. *(Heildar stig: 10 / 20 með bónus)*

   (a) *(Heildar stig: 5)* Hvað/hver er/u brotið/in á myndinni fyrir neðan? Útskýrið

```python
class ShipmentService:
    def ship(self, package: Package):
        if package.shipment_type == ShipmentType.ship:
            ship = Ship()
            ship.reload_fuel_oil()
            ship.ship(package)

        elif package.shipment_type == ShipmentType.plane:
            plane = Plane()
            plane.reload_jet_fuel()
            plane.fly(package)

        elif package.shipment_type == ShipmentType.car:
            car = Car()
            car.reload_gas()
            car.drive(package)

        elif package.shipment_type == ShipmentType.carrier_pigeon:
            pigeon = CarrierPigeon()
            pigeon.eat_seeds()
            pigeon.fly(package)
```

   (b) *(Heildar stig: 5)* Hvernig myndu þið laga brotið/in?

   (c) *(Heildar stig: 10)* **Bónus** Lagið brotið og skilið sem part af zip skránni, kóðann á myndinni má finna í `skeleton_code` sem fylgir með verkefnalýsingunni

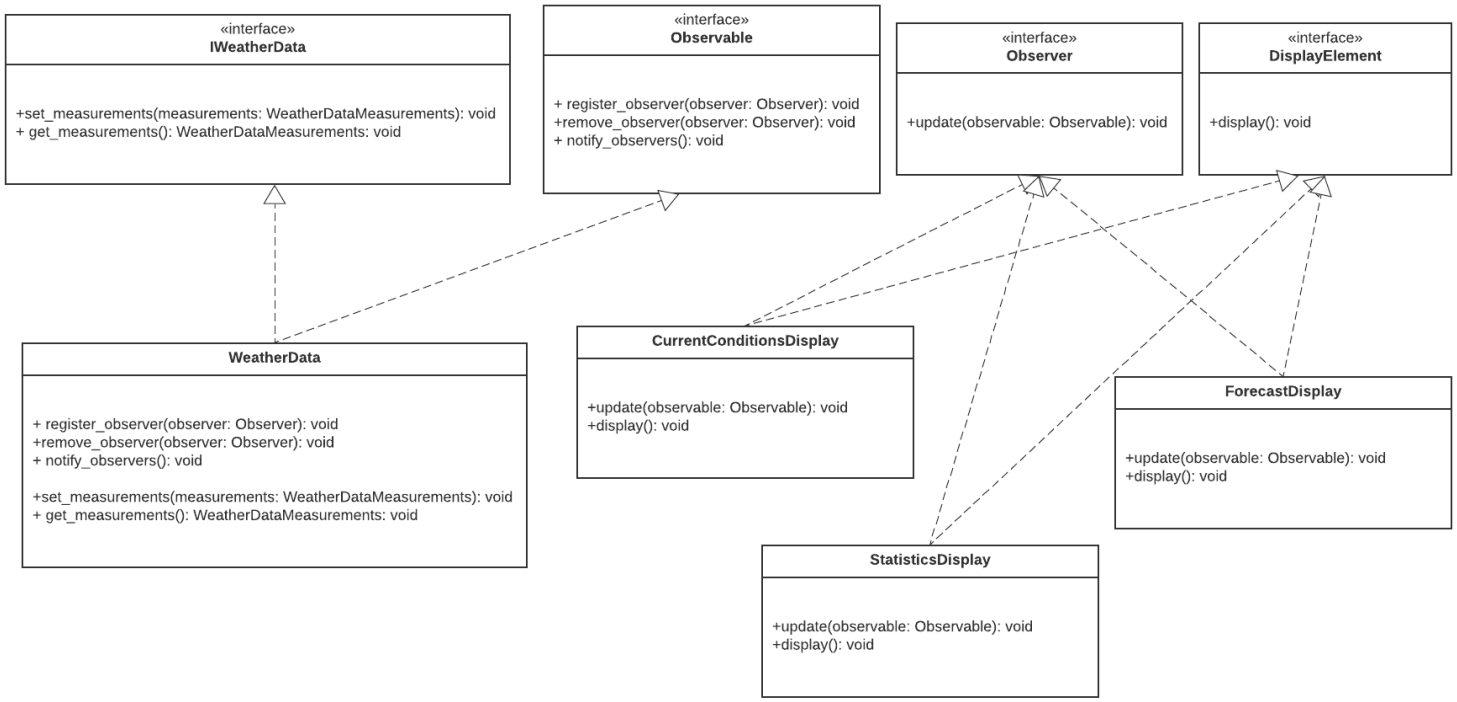
2 Design Patterns (60 stig)

Ath Kóðinn var gerður með **python 3.9**

2.1 The Observer Pattern (35 stig)

2.1.1 (Heildar stig: 30) Weather Station

Í þessu verkefni er markmiðið að útfæra tilbrigiði af WeatherStation application-inu sem við sáum í Observer kaflanum í Head first design patterns bókinni, WeatherStation-ið okkar mun hafa eftirfarandi structure:



Útfærslan mun vera mjög svipuð og sú sem er í bókinni fyrir utan að hvað lítur að nokkrum hlutum:

- Í bókinni er publisher-inn kallaður `Subject` en hér er hann kallaður `Observable`
- Við erum með sér data clasa sem heitir `WeatherDataMeasurement` til að encapsulate-a öll weather gögnin, þessi `WeatherDataMeasurement` klasi samanstendur af pressure, humidity og temperature
- Við erum með `IWeatherData` interface fyrir þær virknir sem líta að WeatherData klasanum sem tengjast ekki beint Observable virkninni þ.e.a.s `set_measurements` og `get_measurements`
- Observer-arnir munu ekki register-a sig sjálfir eins og er gert í bókinni, heldur mun það vera gert af eitthverjum þriðja aðila (`main` fallinu í okkar tilfelli)
- og kannski stærsta breytingin frá útfærslunni í bókinni er að við munum útfæra `pull` observer pattern-ið í staðinn fyrir `push` eins og er sýnt í bókinni. Þetta þýðir að í staðinn fyrir að publisher-inn sendir weather gögnin í `update` föllin á observer-unum (semsagt publisher-inn `push`-ar weather gögnunum) þá sendum við publisher-inn sjálfann inn í `update` fallið á observer-unum og observer-arnir sjá sjálfir um að `pull`-a gögnin frá WeatherData klasanum (sem er publisher-inn), sem þeir munu gera með `get_measurements` fallinu
  .

Ykkur er gefinn skeleton kóði til að hjálpa við útfærsluna sem má finna í `observer_skeleton` möppunni, þar eru öll interface-in skilgreind fyrir ykkur og það eina sem þið þurfið að gera er að útfæra concrete klasana (þá sem útfæra interface-in)

Hér má síðan sjá expected output-ið sem fæst við rétta útfærslu þegar main, sem má einnig finna í skeleton kóðanum, er keyrt.

Command Line

```
$ python main.py
Current conditions: 80.0F degrees and 65.0% humidity
Avg/Max/Min temperature = 80.0/80.0/80.0
Forecast: Improving weather on the way!
---
Current conditions: 80.5F degrees and 65.0% humidity
Avg/Max/Min temperature = 80.2/80.5/80.0
Forecast: Watch out for cooler, rainy weather
---
Current conditions: 78.0F degrees and 90.0% humidity
Avg/Max/Min temperature = 79.5/80.5/78.0
Forecast: Watch out for cooler, rainy weather

------removing forecast display------

Current conditions: 76.0F degrees and 85.0% humidity
Avg/Max/Min temperature = 78.6/80.5/76.0

------removing statistics display------

Current conditions: 81.0F degrees and 84.0% humidity

------removing current conditions display------
```

Eins og er hægt að sjá af output-inu að ofan og í bókinni þá virkar **ForecastDisplay** þannig að þegar pressure hækkar frá seinustu mælingu þá prentast út "Improving weather on the way!", þegar pressure lækkaði frá seinustu mælingu þá prentast út "Watch out for cooler, rainy weather" og þegar pressure er það sama prentast út "More of the same"

Ath. Einhverjir rekið kannski augun í að pull útfærslan á Observer Pattern-inu þarf að brjóta **LSP** þar sem observer-inn þarf að gera ráð fyrir hvernig gögnin sem hann er að pull-a líta út. T.d. með kóða eins og:

```
def update(self, observable: Observable):
    if isinstance(observable, IWeatherData):
        weather_data: IWeatherData = observable
        ...
```

En þetta er mjög algeng leið á hvernig *pull* er útfært og undirstrikar það að principles eru bara viðmið sem má brjóta **þegar á við**.

Ath. Töluleg gildi eiga að prentast út með einum aukastaf

Ath. Skilið Kóðanum í zip kóða skránni undir möppu sem heitir observer

2.1.2 (Heildar stig: 5) Open-Closed

Útskýrið hvernig Observer pattern-ið stuðlar að Open-Closed Principle-inu

2.2 The Decorator Pattern (25 stig)

2.2.1 (Heildar stig: 20) Weather Station continued

Það er komin ný krafa í Weather Station dæmið að ofan, veðurmælingar geta verið með margar litlar sveiflur(i.e fluctuations) og við viljum þar af leiðandi ekki notify-a observer-ana okkar við allar litlar og ómerkilegar breytingar sem eiga sér stað í veður gögnunum.

Við erum þó hrædd við breytingar á fallegu observer virkninni okkar og viljum því bæta þessari virkni við með því að bæta við nýjum kóða í stað þess að gera breytingar á þeim kóða sem til er fyrir (munuð þó þurfa að gera **mjög** litla breytingu á WeatherData klasanum).

Við höfum því ákveðið að gera þessa breytingar með Decorator pattern-inu sem mun bæta við þeirri virkni að observer-ar verða ekki notify-aðir þegar breytingar á öllum veðurgögnum er innan við eina heiltölu af því gildi sem hefur nú þegar verið publish-að til observer-ana. t.d. ef temperature er 100 og það myndi koma breyting sem er 100.9 þá væri sú breyting ekki publish-uð, en ef breyting-in væri 101 eða meira eða 99 eða minna þá væri sú breyting publish-uð.

Í skelton kóðanum sem ykkur er gefið undir `decorator_skelton` má finna 1 skrá sem inniheldur ákveðið interface sem þið ættuð að bæta við WeatherStation kóðann ykkar sem nú þegar er til og mun það interface gefa ykkur ákveðinn byrjunarpunkt

Hér má síðan sjá expected output-ið sem fæst við rétta útfærslu

Command Line

```
$ python main.py
Current conditions: 80.0F degrees and 65.0% humidity
Avg/Max/Min temperature = 80.0/80.0/80.0
Forecast: Improving weather on the way!
---
---
Current conditions: 78.0F degrees and 90.0% humidity
Avg/Max/Min temperature = 79.0/80.0/78.0
Forecast: Watch out for cooler, rainy weather

------removing forecast display------

Current conditions: 76.0F degrees and 85.0% humidity
Avg/Max/Min temperature = 78.0/80.0/76.0

------removing statistics display------

Current conditions: 81.0F degrees and 84.0% humidity

------removing current conditions display------
```

Ath.

- Þið munuð þurfa að gera **litla** breytingu á main file-inu til að fá þetta output
- Skilið Kóðanum í zip kóða skránni undir möppu sem heitir decorator (með öllum gamla observer kóðanum líka til að gera yfirferðina okkar auðveldari)

2.2.2 (Heildar stig: 5) Open-Closed

Útskýrið af hverju stærsti hvatinn á bakvið þessa breytingu er Open-Closed Principle