AARHUS UNIVERSITET

Software Engineering and Architecture

Interface Segregation Principle

Role- and Private Interfaces

Roles…

- **Role** (Da: Rolle / function)
- … will be treated in much more detail soon, but…
- We know it from plays and acting
  - Daniel Craig plays the role of *James Bond*
- And in the more general understanding of a *function*
  - It was the nurse's role to tend to the patient
  - It is the physician's role to plan a treatment for the patient

Definition: **Role (General)**

A function or part performed especially in a particular operation or process.

Role-Person

- Roles are ‘played’/’fulfilled’ by persons
  - The one that ‘does the function / fulfill the responsibilities’
  - James Bond is played by Sean Connery in this movie
  - The SWEA censor today is Clemens Klokmose
  - The physician on duty on the ward today is Jens Madsen
  - The host on the news broadcast today is …
- **A role can be played by many different persons**
  - Of course, if they are trained and skilled to fulfill the role…

AARHUS UNIVERSITET

Role-Person

- **A person can play many different roles**
  - Example: *My roles:*
    - Teacher, researcher, husband, father, taxpayer, colleague, friend, …
- But *they are only accessible for a given audience*
  - I am a father and a teacher. Each with responsibilities…
  - But *not all have access to all my roles, of course*
    - Student: “Could you explain the ‘Role’ concept in programming?”
      - Yes, I will do that. That responsibility belongs to the Teacher role.
    - Student: “Please, Henrik, can you fix my flat bike tire?”
      - **No, I will not! That responsibility belongs to the Father role**
    - Child: “Can you fix my flat bike tire?”
      - Hm hm, ok, but I am going to teach you to do it yourself…

AARHUS UNIVERSITET

Roles define Access

- So – depending on the *role of the requester* some requests are allowed and some are not
  - Only persons in ‘my children role’ interact with my father interface
    - Example: *fix flat tire* *responsibility*
  - Only Danish Tax may interact with my taxpayer interface
    - Example: *provide yearly earning* *responsibility*
  - Only the person in ‘my wife role’ interact with my husband interface
    - No examples ☺

Bottomline

- Role – Person is a *many-to-many* relation
  - One role played by many persons
    - But the person needs to fulfill the requirements/training to play role
  - One Person plays many roles
    - But access to a given role, requires a specific role by the audience

AARHUS UNIVERSITET

Relating to Software

Roles…

- We use *interface* in our OO language to express a *role*
  - *interface Game*
    - Express the responsibilities of our HotStone game
      - Allow clients to inspect game state, play a card, attack a hero, etc.
  - *Interface Card*
    - Express the responsibilities of a HotStone card
      - Store mana cost, health, etc. Allow clients to get those values.
  - *Interface RateStrategy*
    - Express the responsibility to compute parking time based upon a given amount of money

Definition: **Role (Software)**

A set of responsibilities and associated protocols.

AARHUS UNIVERSITET

Role-Object

- Objects implement/"play" these roles
  - Class StandardGame implements Game { … }
  - Game game = new StandardGame(….);
  - Now we know that 'game' object can *act* as defined by the role 'Game'
- *A role can be played by many different objects*
  - 'Game' is currently implemented by about 90 SWEA groups!
    - All different (except if a group cheats ☺)!
  - And AU GitLab probably have > 500 implementations of Game

Role-Object

- Does it make sense to also think:
- *One object plays many different roles, depending upon who wants to access it?*

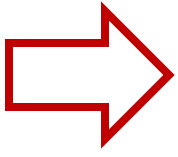
- **Yes** – we have already such a situation in HotStone!

- The Game object is already used in two different contexts

AARHUS UNIVERSITET

Game Roles

- The UI needs to interact with the Game object
  - But *only* by interacting via the Game interface's method
    - Which allows Game to ensure all HotStone rules are obeyed
      - playCard(…) OK
      - modifyHeroHealth(Findus, -2) **Not OK**



Game Roles

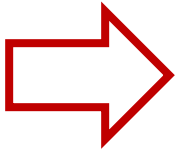
- But we also have *strategies* that need to do *more*
  - The Strategi to handle Hero Power needs more specialized access
    - playCard(…) OK
    - modifyHeroHealth(Findus, -2) **Perfectly OK**

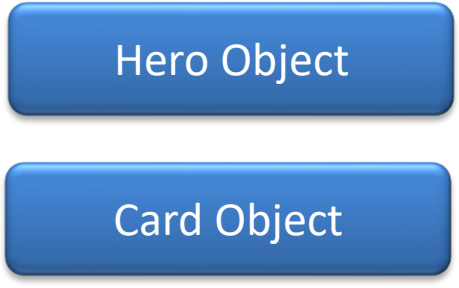
Hero Power. The ThaiChef's power *Chili* will decrease the opponent hero's health by 2. Description: *Deal 2 damage to opponent hero.* The DanishChef's power *Sovs* will field a special minion "Sovs" of value (attack, health) = (1,1). Description: *Summon Sovs card.*

AARHUS UNIVERSITET

And Actually…

- We have the exact same situation with Card and Hero
  - The UI must never change, say, health of Hero/Card
    - As Game object then cannot guaranties rules are obeyed





  - But the Game object of course needs to change all state in all heroes and all cards
    - card.deltaHealth(-3);

ISP

- This is the **Interface Segregation Principle**
- ‘Do not depend on methods that you do not use’

> **Definition: Interface Segregation Principle**
>
> In the field of software engineering, the interface segregation principle (ISP) states that no code should be forced to depend on methods it does not use. ISP splits interfaces that are very large into smaller and more specific ones so that clients will only have to know about the methods that are of interest to them. Such shrunken interfaces are also called role interfaces.

- Or – ‘delimit what methods a given object may access’
  - My children are the only with access to my father interface
  - My SWEA students are the only with access to my teacher intf.

Fine-grained Roles

- The ‘more specific’ role is expressed as a Role Interface

> **Definition: Role Interface**
>
> A *role interface* is defined by looking at a specific interaction between suppliers and consumers. A supplier component will usually implement several role interfaces, one for each of these patterns of interaction.

Martin Fowler

- Ala again
  - I provide a ‘teacher’ interface (one role interface)
  - And a ‘taxpayer’ interface (another role interface)
  - Etc.

AARHUS UNIVERSITET

So – Our Game Can

- Have the UI oriented interface
  - The current Game interface, in the handed-out code
- Have a more specific interface, with more privileges to the strategies…
  - Ala a ‘MutableGame’ interface or another appropriate name
    - Only provided to the strategies
    - Contains special mutators like for example
      - deltaHeroHealth(Player who, int deltaValue);
- Then
  - class StandardGame implements Game, MutableGame {…}
  - (class StandardCard implements Card, MutableGard {…})

AARHUS UNIVERSITET

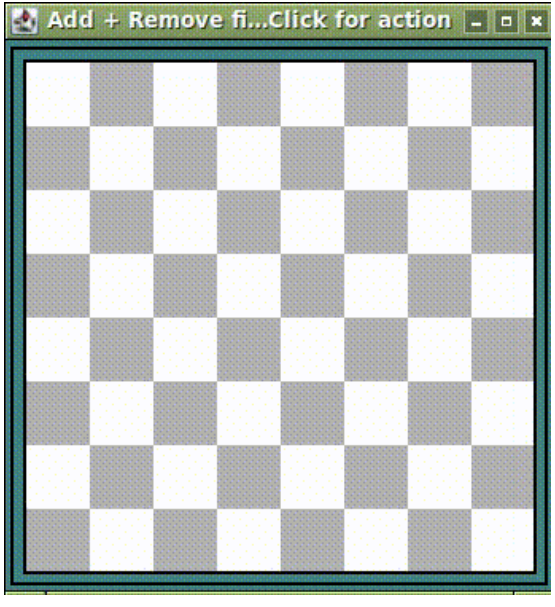
Another Example

- After the Autumn break, we will add a UI to HotStone
- Building a UI for 2D graphics from the bottom up is tedious
- So, we use a library/framework: MiniDraw
  - The central ‘collection’ of graphical elements on-screen is the **Drawing role**

```
public interface Drawing extends FigureCollection, SelectionHandler {
   ...
}
```

Example

- The FigureCollection in MiniDraw only deals with *adding, removing, and iterating the collection of Figures in MiniDraw*




```
public interface FigureCollection extends Iterable<Figure> {
  /**
   * Adds a figure and sets its container to refer to this drawing. If you have
   * several threads that may call add, scope it by the lock/unlock methods.
   * The Drawing role will render figures in the order they are inserted,
   * so if they overlap the LAST added figure will appear on top. Use
   * zOrder method to change ordering.
   *
   * @param figure
   *          the figure to add
   * @return the figure that was inserted.
   */
  Figure add(Figure figure);

  /**
   * Removes a figure. If you have several threads that may call add, scope it
   * by the lock/unlock methods.
   *
   * @param figure
   *          the figure to remove
   * @return the figure removed
   */
  Figure remove(Figure figure);
```

A *specific interaction (add+remove) between the UI and the Drawing, expressed as the Role Interface 'FigureCollection'*

Private Interface

- Role interfaces are often used to enforce more specific *encapsulation* than is possible using *private/public* methods and instance variables…

> Definition: **Private Interface**
>
> Provide a mechanism that allows specific classes to use a non-public subset of a class interface without inadvertently increasing the visibility of any hidden member variables or member functions.

James Newkirk

- Let us make an example, highly inspired by our project…

Example

- We have a system/framework/**Façade** which presents (x,y) points to outside code, but that outside code must *never modify the (x,y) values!*
- *Read-only* **Role interface** is a solution to that.
  - Only accessor methods, no mutator methods…
  - (Analogy:
    - Façade = Game
    - Point = Card)

```
public interface Point {
      int getX();
      int getY();
    }

public interface Facade {
      Point getPoint();
    }
```

Example

- However, internal classes inside the Façade of course needs to mutate the state of these (x,y) points.
- Let us say that one class needs to translate (dx,dy) points
- **Private Interface** is a solution to that

```
public interface TranslatablePoint {
      void translation(int dx, int dy);
    }

public interface PointStrategy() {
      void doSomethingToPoint(TranslatablePoint p);
    }
```

Example

- Now the internal, implementing, class of course implements both

```
public class StandardPoint implements Point, TranslatablePoint {
      [all three methods implemented here]
    }

public class MyFacade implements Facade {
      StandardPoint point;
      Point getPoint() { return point; }
    }
```

- That is, if you use ‘getPoint()’ from the outside you only get access to ‘getX()’ and ‘getY()’

Example

- Now, an internal PointStrategy can translate points like

```
public class Strategy1 implements PointStrategy {
      void doSomethingToPoint(TranslatablePoint p) {
        p.translation(+3, +7);
      }
    }
```

- And can be called internally like

```
strategy.doSomethingToPoint(point);
```

Example

- However, a PointStrategy cannot access (x,y)…

```
public interface TranslatablePoint {
      void translation(int dx, int dy);
    }
```

- However, of course it is often the case, that we need just that.

- *Exercise: How do we solve that?*

AARHUS UNIVERSITET

Solution 1:

- Fine-grained solution: **Missing accessor methods**
  - Just add those methods that are missing

```
public interface TranslatablePoint {
      int getX();
      int getY();
      void translation(int dx, int dy);
    }
```

- Pro
  - Can select just the right set of accessors
    - (here it is both of them, but if read-only had 20, we may just pick the two we need).
- Con
  - Same methods are now present in two interfaces

Ups?

- Uhum – how does that work in Java?

```
public interface Point {
      int getX();
      int getY();
    }
```

```
public interface TranslatablePoint {
      int getX();
      int getY();
      void translation(int dx, int dy);
    }
```

```
public class StandardPoint implements Point, TranslatablePoint {
      [all three methods implemented here]
    }
```

- StandardPoint must now implement ‘getX()’ twice or???

- Exercise: What happens?

AARHUS UNIVERSITET

Solution 2:

- Coarse-grained approach: **Extend existing interface**
  - Just implement both

```
public interface TranslatablePoint extends Point {
      void translation(int dx, int dy);
    }
```

- Pro
  - Less typing
  - ***You can actually Program to an Interface in the façade impl!***

```
public class MyFacade implements Facade {
    TranslatablePoint point;
    Point getPoint() { return point; }
}
```

- Con
  - You get all methods

AARHUS UNIVERSITET

Mandatory Note

- We have *read-only* role interfaces for Card and Hero in HotStone.
  - But Game's implementation and strategies need to manipulate them…
  - *Use private interfaces for that ☺!*
- Strategies needs special mutations of Game
  - *Use private interface(s) for that ☺!*
- ***Now you 'program to an interface'***, and avoid hard coupling to, say, StandardGame etc.

AARHUS UNIVERSITET

Mandatory Note

- And iff you use the ‘extending existing interface’
  - interface MutableGame implements Game { (mutators here) }
- … you can now *remove* most casts and *hard couplings* to your StandardGame/StandardCard etc
  - Map<Player, List<**MutableCard**>> handMap = …
  - Interface HeroPowerStrategy {
  - public void usePower(**MutableGame** game);
  - }
- … and still have the full flexibility of providing any concrete class that implements MutableX in your