AARHUS UNIVERSITET

Software Engineering and Architecture

(Many) Mandatory Reflections

And SE Hints ☺

AARHUS UNIVERSITET

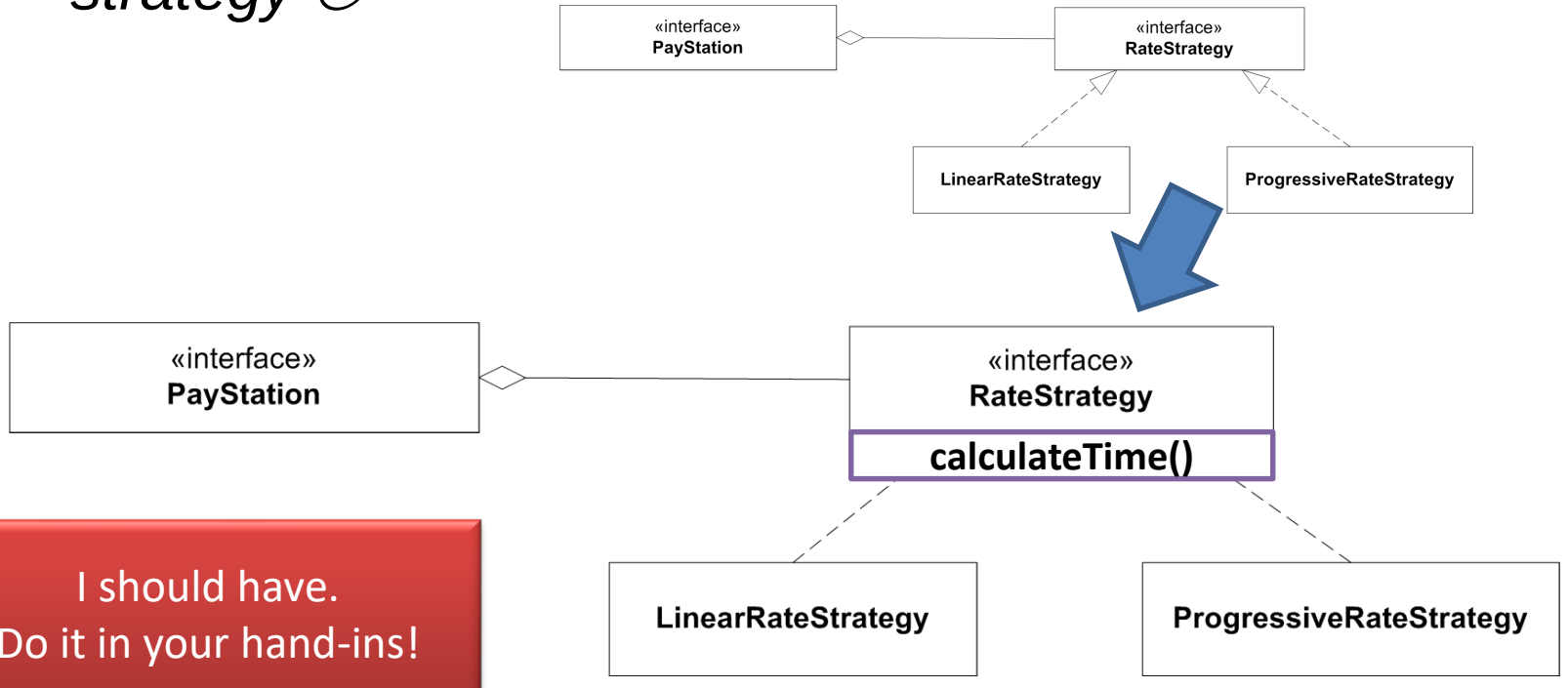
Correction to the book!

I would have made it differently today…

AARHUS UNIVERSITET

Include *central* methods

- Present book: *I do not show the central method in the strategy* ☹



I should have.
Do it in your hand-ins!

AARHUS UNIVERSITET

Inner and Outer

Encapsulation

AARHUS UNIVERSITET

How do I?

- Switch from channel TV2 to DR on my Samsung TV set?
- A) Push the Button ‘3’ on the TV’s remote control *interface?*
- B) Call Samsung to tell them to send a man to re-solder the wire inside the TV set?
- *Some of you accidentally use method B*

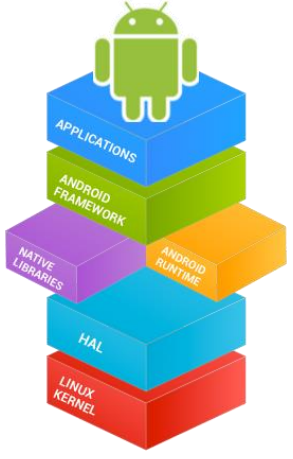
Example

```
public GameImpl(String variant) {
  if (variant.equals("AlphaCiv"){
      ageStrategy = new Decrement100YearsStrategy();
      winnerStrategy = new RedWinsAt3000BCStrategy();
      actionStrategy = new NoActionStrategy();
      ...
  } else if (variant.equals("BetaCiv")){
      ageStrategy = new DeacceleratingTimeScaleStrategy();
      winnerStrategy = new AllCityConquerStrategy();
      actionStrategy = new NoActionStrategy();
  } else if (...)
  {
      ...
  }
}
```

- What happens when I want a WorldWar II HotCiv?
  - I have to call you guys to resolder the wires inside!

Frameworks

- What is the process in the mandatory exercises?
  - *To use TDD and compositional design to transform an AlphaCiv application into a **HotCiv framework***
- Frameworks are
  - Reusable software designs and implementations
  - Must be reconfigurable *from the outside*
    - *Just like a TV set*
- Example
  - Android Google's smartphone OS
  - *You do not call Google to make them rewrite their constructor in order to introduce the App for your HCI course, do you!?!*



AARHUS UNIVERSITET

Open/Closed

- **Open for Extension** (I can adapt the framework)
- **Closed for Modification** (But I cannot rewrite the code)
- *Change by addition, not by modification*
- So
  - public GameImpl(String whichVariant) {
    - If (whichVariant.equals(”Alpha”)) {
      - ageStrategy = new AlphaAgeStrategy()
- Is **wrong!**
  - I have to open the TV to solder the wires inside ☹
  - You have to call Google to make your HCI project app ☹

So…

- Keep GameImpl, UnitImpl, CityImpl, … *closed for modification!*

- Allow adapting HotCiv to a WWII scenario by *addition*
  - **I can** code a WorldLayoutStrategy that produce a Europe map
  - Etc.
  - And provide **my** strategies in the constructor **of your GameImpl**
  - And it will *do the right thing…*

Uncle Bob???

- What about Uncle Bob?
  - *Though shall not have more than two parameters as arguments*
- *Disobey him for now…*
  - *GameImpl(AgeStrategy ageStrategy, ……..)*
- We will refactor HotCiv soon enough to fix it…
  - Abstract Factory…

You Can Do More Outside

Inner and Outer have different Rules!

AARHUS UNIVERSITET

Parametric Variant

- Example:
  - GammaCiv: Settlers make cities, archers fortify

HotCiv Framework Code:
GameImpl, CityImpl, UnitImpl, …

```
@Override
public void performAction(GameImpl gameImpl, UnitImpl unit) {
  if ( unit.getTypeString().equals( GameConstants.ARCHER ) ) {
    // Do Archer stuff - toogle moveability - set defensive strength
  }  else if ( unit.getTypeString().equals( GameConstants.SETTLER ) ) {
    // Do Setterl stuff - tell Game to make city, and remove settler
  }
}
```

AARHUS UNIVERSITET

Parametric Variant

- Example:
  - GammaCiv: Settlers make cities, archers fortify

HotCiv Framework Code:
GameImpl, CityImpl, UnitImpl, …

Bad: HotCiv has hard bindings to specific unit types. *Only Change by Modification!*

```
@Override
public void performAction(GameImpl gameImpl, UnitImpl unit) {
  if ( unit.getTypeString().equals( GameConstants.ARCHER ) ) {
    // Do Archer stuff - toogle moveability - set defensive strength
  }  else if ( unit.getTypeString().equals( GameConstants.SETTLER ) ) {
    // Do Setterl stuff - tell Game to make city, and remove settler
  }
}
```

AARHUS UNIVERSITET

Compositional Variant

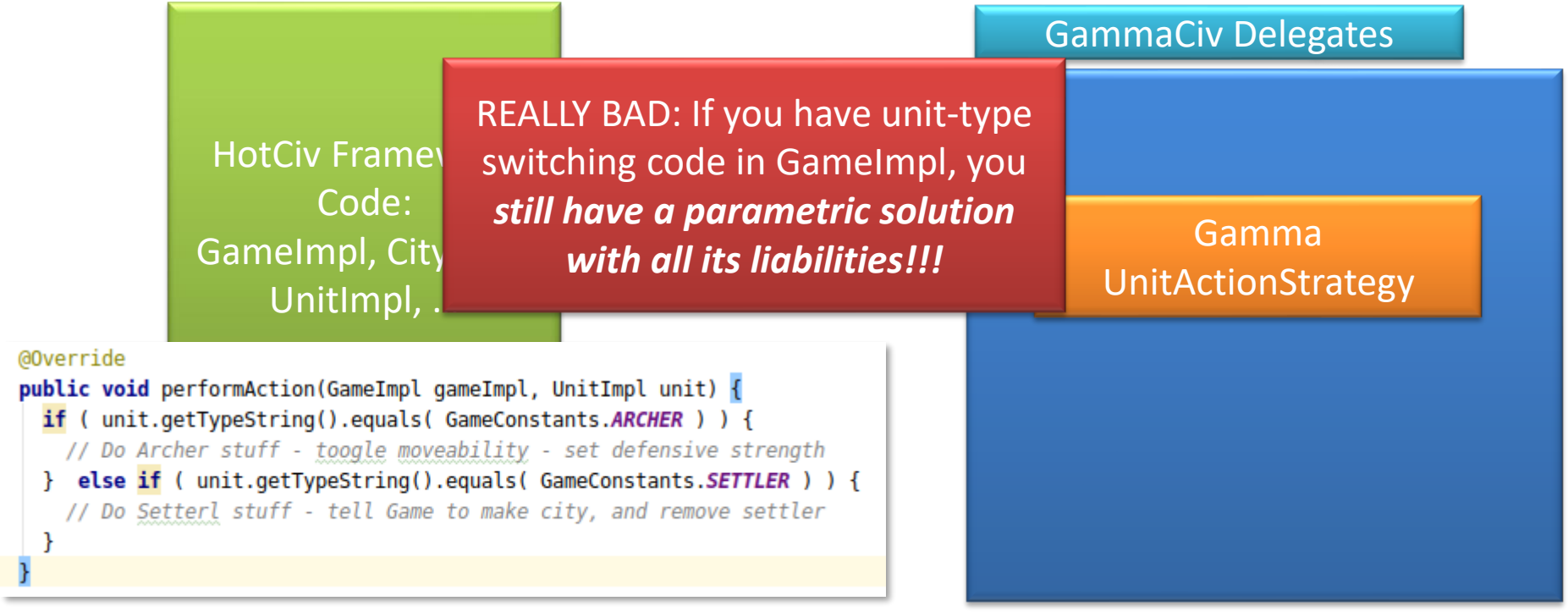
- Example:
  - GammaCiv: Settlers make cities, archers fortify

HotCiv Framework Code:
GameImpl, CityImpl, UnitImpl, …

```
@Override
public void performAction(GameImpl gameImpl, UnitImpl unit) {
  if ( unit.getTypeString().equals( GameConstants.ARCHER ) ) {
    // Do Archer stuff - toogle moveability - set defensive strength
  }  else if ( unit.getTypeString().equals( GameConstants.SETTLER ) ) {
    // Do Setterl stuff - tell Game to make city, and remove settler
  }
}
```

Compositional Variant

- Example:
  - GammaCiv: Settlers make cities, archers fortify



AARHUS UNIVERSITET

Compositional Variant

- Example:
  - GammaCiv: Settlers make cities, archers fortify

HotCiv Framework Code:
GameImpl, CityImpl, UnitImpl, …

GammaCiv Delegates

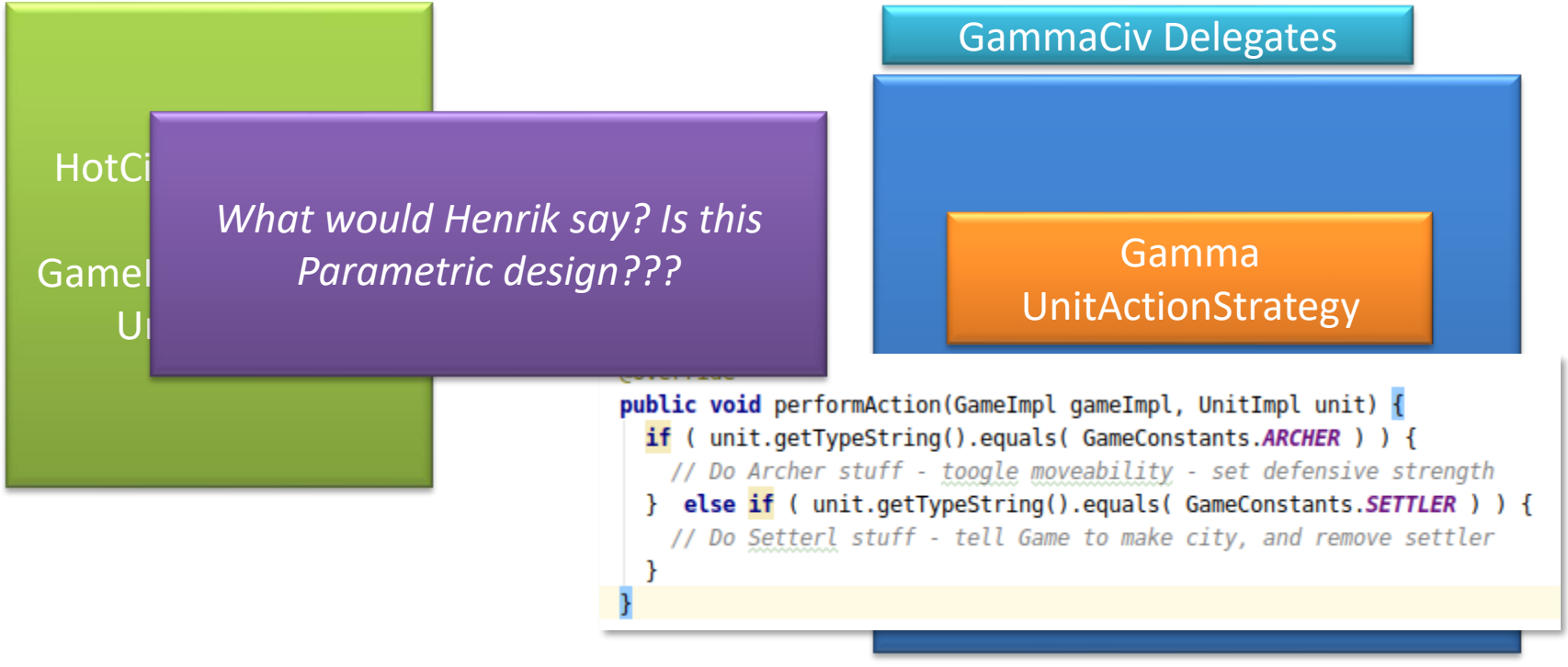
Gamma UnitActionStrategy

```
@Override
public void performAction(GameImpl gameImpl, UnitImpl unit) {
  if ( unit.getTypeString().equals( GameConstants.ARCHER ) ) {
    // Do Archer stuff - toogle moveability - set defensive strength
  }  else if ( unit.getTypeString().equals( GameConstants.SETTLER ) ) {
    // Do Setterl stuff - tell Game to make city, and remove settler
  }
}
```

AARHUS UNIVERSITET

Compositional Variant

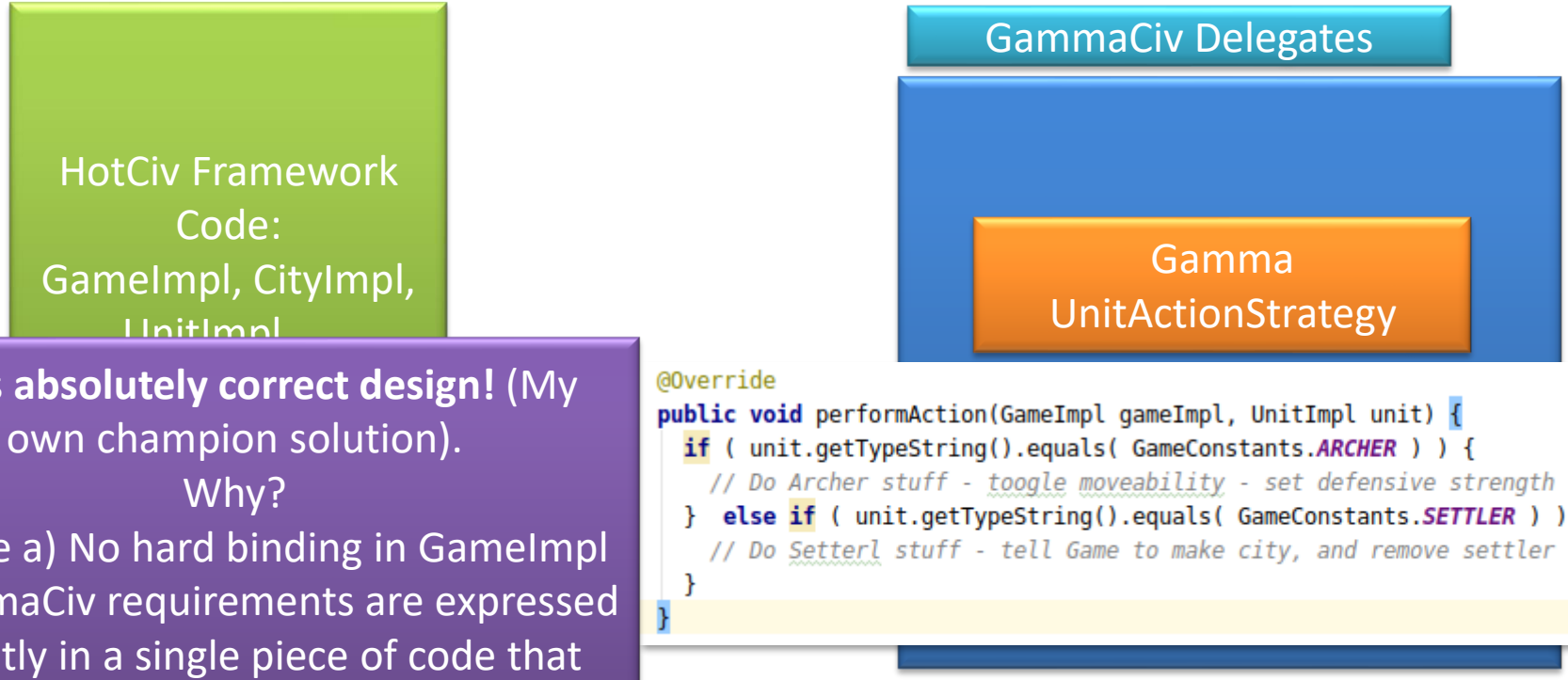
- Example:
  - GammaCiv: Settlers make cities, archers fortify



AARHUS UNIVERSITET

Compositional Variant

- Example:
  - GammaCiv: Settlers make cities, archers fortify

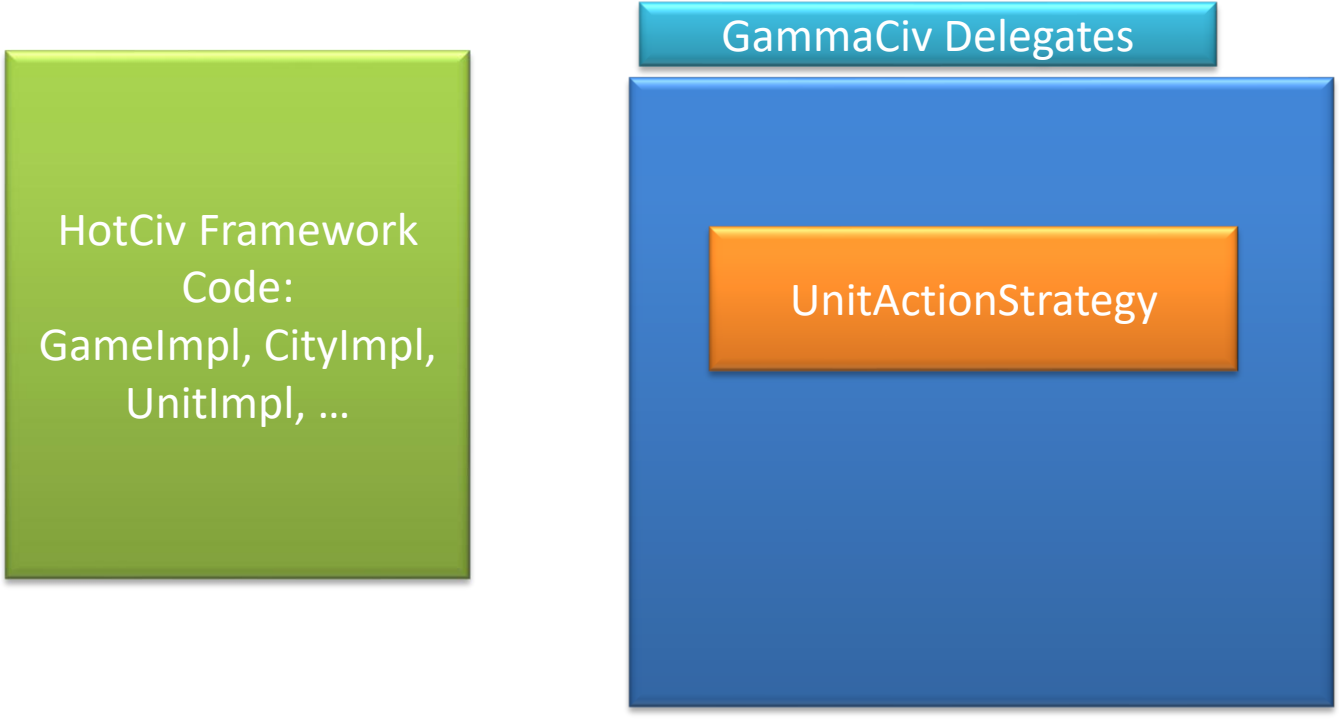


This is **absolutely correct design!** (My own champion solution).
Why?
Because a) No hard binding in GameImpl
b) GammaCiv requirements are expressed explicitly in a single piece of code that bears the correct name!

AARHUS UNIVERSITET

Inner and Outer

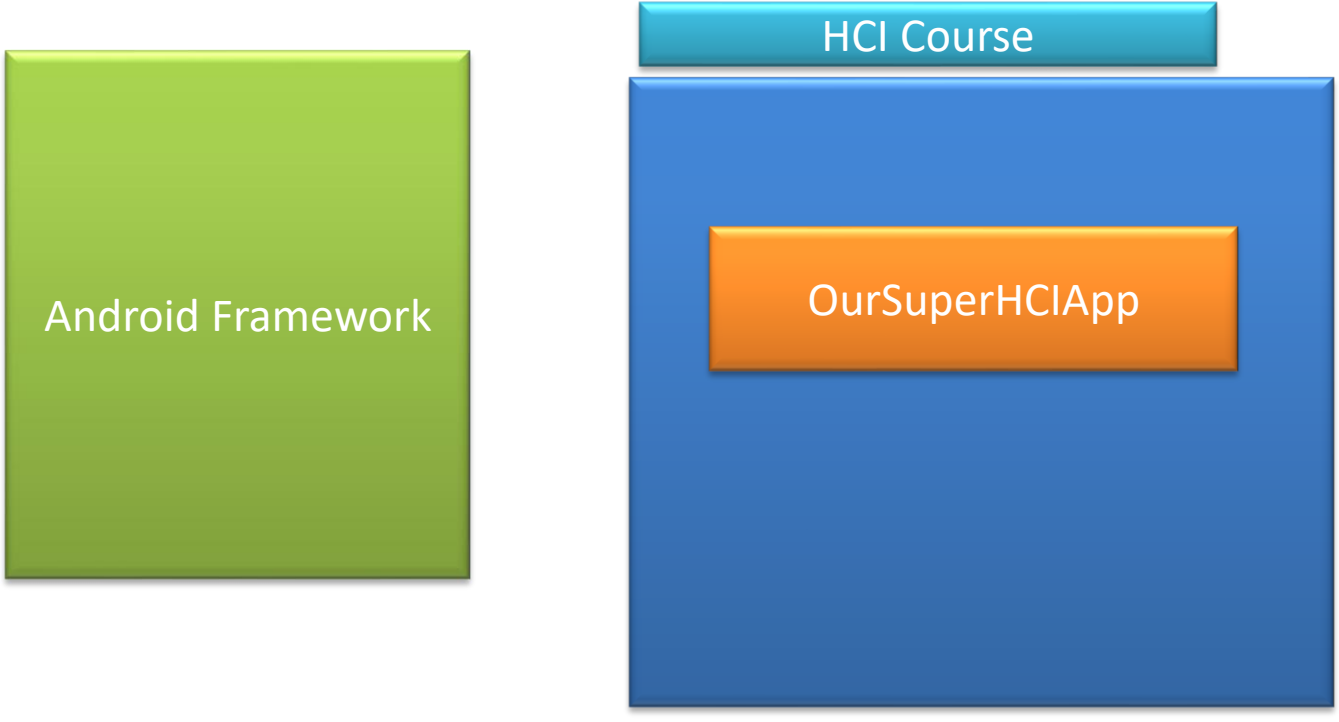
- *Keep inner code (framework code) clean of variability switching code; have it in the outer code (delegates)!*



AARHUS UNIVERSITET

Inner and Outer

- *Keep inner code (framework code) clean of variability switching code; have it in the outer code (delegates)!*



AARHUS UNIVERSITET

Inner Outer Collaboration

Delegates have to tell something

AARHUS UNIVERSITET

But what then…

- GammaCivUnitStrategy has to
  - create a City???
  - destroy a unit???
- **Collaborate via well defined, *descriptively named,* protocol…**
- *Do not reach into actual datastructures!*

```
public class GammaCivActionStrategy implements ActionStrategy {
    public void performUnitAction(Game g, Position p) {
        UnitImpl u = (UnitImpl) g.getUnitAt(p);
        if ( u.getTypeString().equals(GameConstants.ARCHER) ) {
            u.setMoveable(false);
            u.setDefensiveStrength( GammaGameConstants.ARCHER_DEF_STRENGTH );
        } else if ( u.getTypeString().equals(GameConstants.SETTLER) ) {
            GameImpl gameImpl = (GameImpl) g;
            g.createCityAt(p, u.getOwner() );
            g.removeUnitAt(p);
        }
    }
```

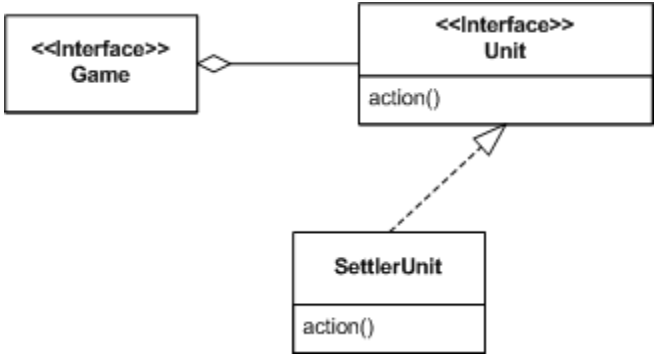
AARHUS UNIVERSITET

The Polymorphic GammaCiv?

AARHUS UNIVERSITET

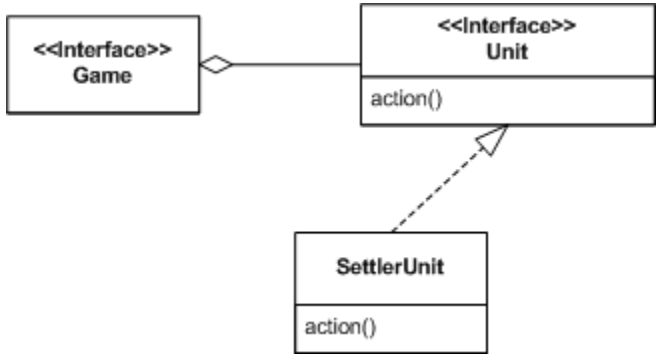
GammaCiv and 36.11

- Polymorphic proposal
  - Add ‘action(…)’ to Unit interface
  - Subtypes implement it
    - SettlerUnit.action(…) does
      - Create city at ‘my’ position
      - Kill myself (?)



GammaCiv and 36.11

- Polymorphic proposal



- *Cohesion*
  - *High: what a settler /archer does; but unit now ”knows a lot”*
- *Coupling*
  - *City : Unit now coupled to City (”must say new StdCity()”);*
  - *World: Either direct access or ask Game to put city*
  - *Destroying: Cannot be done, must ask Game*
  - *Making settlers: abstract factory to make subtype outside HotCiv framework (which contain a switch)– OR a switch on unit type (no support for adding types later)*

And CityImpl Code?

```
class SettlerUnit extends Unit
{
    public void action(GameImpl game, Position p) {
      game.createCityAt(p);
      game.removeUnitAt(p); // "kill me, please :)"
    }
}

How is a unit made in a city

if ( typeWeProduceNow == GameConstants.SETTLER )
{
  newUnit = new SettlerUnit(myPosition);
} else ( typeWeProduceNow == GameConstants.ARCHER ) {
  newUnit = new ArcherUnit(myPosition);
} ...

OR

newUnit = factory.createUnit( typeWeProduceNow, myPosition );
 and
class AlphaCivUnitFactory {
   Unit createUnit( String typeToProduce, Position p ) {
       if ( typeToProduce == GameConstants.SETTLER)
       {
           return new SettlerUnit();
       }
```

Own Experience with polymorphic code:

If (unit.class == SettlerUnit.class) {
SettlerUnit su = (SettlerUnit) unit;
su.doSettleryThingy();
}

Will spread into your code…

AARHUS UNIVERSITET

And CityImpl Code?

- In the *compositional variant,* making units is easy:
  - Unit u = new UnitImpl(whichTypeToMake);

Reuse Strategies

- Note that adding, say, a Chariot unit type can be made purely by *addition* in compositional case...
  - Compositional: add a ”ThetaActionStrategy” which
    - If (type.equals(”Chariot”)) { ...}
    - else {
    - [D].performUnitActionAt(p);
    - }
  - Where [D] is either
    - super in case new strategy inherits from the old one
    - delegate in case we do composition
      - Which is actually the **decorator** pattern
  - Polymorphic: Add ChariotUnit **+ modify city produce code**

AARHUS UNIVERSITET

Generalization or Specialization?

Generalization is not a bad thing…

AARHUS UNIVERSITET

When Need Arise…

- Agile development tells us
  - *You ain't gonna need it*
  - *Simplicity*
    - That is: "Do not generalize (= complicate) design if there is not use of it"
- But do it *when need arise…*
  - Triangulation: Add tests that force generalization into existence.
- Example:
  - Archers need to fortify in GammaCiv
  - Question: Is this a *general feature* or highly GammaCiv *specific?*

Generalize

- Fortify = unit is not moveable + defense strength change

- Making units *non-moveable* sounds general to me…
  - Add accessor to interface ‘isMoveable()’
  - Add mutator to implementing class ‘setMoveable(boolean m)’

- Extra clause in the *bail out fast* of ‘moveUnit(f,t)’
  - boolean isMoveable = getUnit(f).isMoveable();
  - if (! isMoveable) return false;

AARHUS UNIVERSITET

Multi Role'd Abstractions

Big Ball of Mud Strategy

Blob Strategy

Multi-dimensional Variance

AARHUS UNIVERSITET

Consider…

- *One Strategy to Rule Them All!*

```
public interface CivVariantStrategy {
    // Units
    void performUitAction(Position p, GameImpl game);

    // Winner
    Player getWinner(GameImpl game);

    // Age
    int getStartingAge();
    int nextAge(int currentAge);

    // World
    Map<Position, Tile> defineWorld();
    Map<Position, UnitImpl> defineUnits();
    Map<Position, CityImpl> defineCities();
}
```

What is the problem?

```
public class BetaVariantStrategy extends AlphaVariantStrategy {
    @Override
    public Player getWinner(GameImpl game) {
      ....
```

AARHUS UNIVERSITET

The Problem

- ZetaCiv
  - *I want to WinnerStrategy to be a combination of those of BetaCiv and EpsilonCiv*
  - *ZetaCivVariantStrategy extends BetaCivVariant* ***and EpsilonCiv***
- Actually, it is the main discussion in the State Chapter in FRS and the State Pattern lecture ☺

Storing State

Who knows how old the game is?

AARHUS UNIVERSITET

State in Strategies?

- Game age state can be stored in
  - Game:
    - Keep track of how many rounds have been played
  - AgeStrategy:
    - Ok, some suggested keeping state (age) only here…

- Strategy pattern is about *algorithms*
  - Algorithms compute stuff, they do (generally) not *know* stuff…
- I may change the algorithm
  - Change the strategy at runtime

Cohesion again! Keep the state in the object where is belongs

AARHUS UNIVERSITET

Unit and Integration Testing

AARHUS UNIVERSITET

Unit Testing Preferred

```
@Test public void shouldAdhereToBetaCivAging() {
    int roundsPlayed = 0;
    AgeStrategy ageStrategy = new BetaCivAgeStrategy();
    assertThat(ageStrategy.computeAge(roundsPlayed), is(-4000));
    // Around birth of Christ
    roundsPlayed = 39;
    assertThat(ageStrategy.computeAge(roundsPlayed), is(-100));
    roundsPlayed = 40;
    assertThat(ageStrategy.computeAge(roundsPlayed), is(-1));
    roundsPlayed = 41;
    assertThat(ageStrategy.computeAge(roundsPlayed), is(+1));
```

ActionStrategy is *difficult* to test with your present knowledge, so use **integration testing** for that.
(Test spies will help us out later…)

AARHUS UNIVERSITET

Bindings in Game

Keeping Game encapsulated

Why not program to interface internally?

AARHUS UNIVERSITET

Read-Only Interfaces

- *The central statement is:*

- **Only the Game instance is allowed to modify internal game state…**

- **How do we ensure that there are no other mutator methods visible to, say, the GUI programmers than those in Game interface?**

AARHUS UNIVERSITET

Design Issues

- Read-only interfaces for City, Tile, Unit – *Why?*
  - Otherwise **two** ways of changing state in a game
    - Through Game which **will enforce consistency**
    - Through City which will NOT enforce consistency
  - If there are two ways **Bjarne will use the wrong one!**
    - ‘getCityAt(p).setTreasuryTo(10000);’

Morale: Only Game has mutator methods!

- But how to change state?
  - Only CityImpl has mutators; GameImpl only use CityImpl?
  - ModifiableCity interface; GameImpl only use that
  - GameImpl takes on all responsibility?

AARHUS UNIVERSITET

Sketches

- Knowing the Implementation type in Game

```
public class GameImpl implements Game
{
    CityImpl redCity, blueCity;

    private UnitImpl produceUnitInCity( CityImpl city ) {
        UnitImpl unitProduced = null;
        String unitType = city.getProduction();
        int costOfUnit = getCostOfUnitType( unitType );
        if ( city.totalProduction > costOfUnit ) {
            unitProduced = new UnitImpl( unitType );
            city.decreaseTreasuryBy( costOfUnit );
        }
        return unitProduced;
    }
```

- Liabilities
  - Fails if we must allow external clients to configure our Game with new City implementations ☹
    - Dependency injection and stuff...
  - We have *high coupling* between Game and CityImpl

AARHUS UNIVERSITET

Sketches

- Using an ”modifiable interface” that extends read-only intf.

```
public class GameImpl implements Game
{
ModifiableCity redCity, blueCity;

    private UnitImpl produceUnitInCity( CityImpl city ) {
        UnitImpl unitProduced = null;
        String unitType = city.getProduction();
        int costOfUnit = getCostOfUnitType( unitType );
        if ( city.totalProduction > costOfUnit ) {
            unitProduced = new UnitImpl( unitType );
            city.decreaseTreasuryBy( costOfUnit );                ) );
        }
        return unitProduced;
    }
interface ModifiableCity extends City
{
    public int totalProduction();
    public void decreaseProductionBy( int amount );
}
```

- Now the Game can get any class that implements *ModifiableCity* injected and handle it...

AARHUS UNIVERSITET

My opinion

- It is fully OK for GameImpl to
  - Operate through CityImpl, UnitImpl, etc.
  - Do a cast ala (CityImpl) this.getCityAt(p)

  - *Why:*
    - *Because we are developing the internal production code*
    - *These are meant to be long lived (make'em rather dumb PODOs!)*

- The GUI guys only use Game interface
- The Domain guys (= us) use the internals

AARHUS UNIVERSITET

My opinion

- ”More beautiful” to use a ‘ModifiableCity’ interface?

- *Simplicity: maximize work not done*

  - *We do not need it at the moment*
  - *It introduces more complexity*

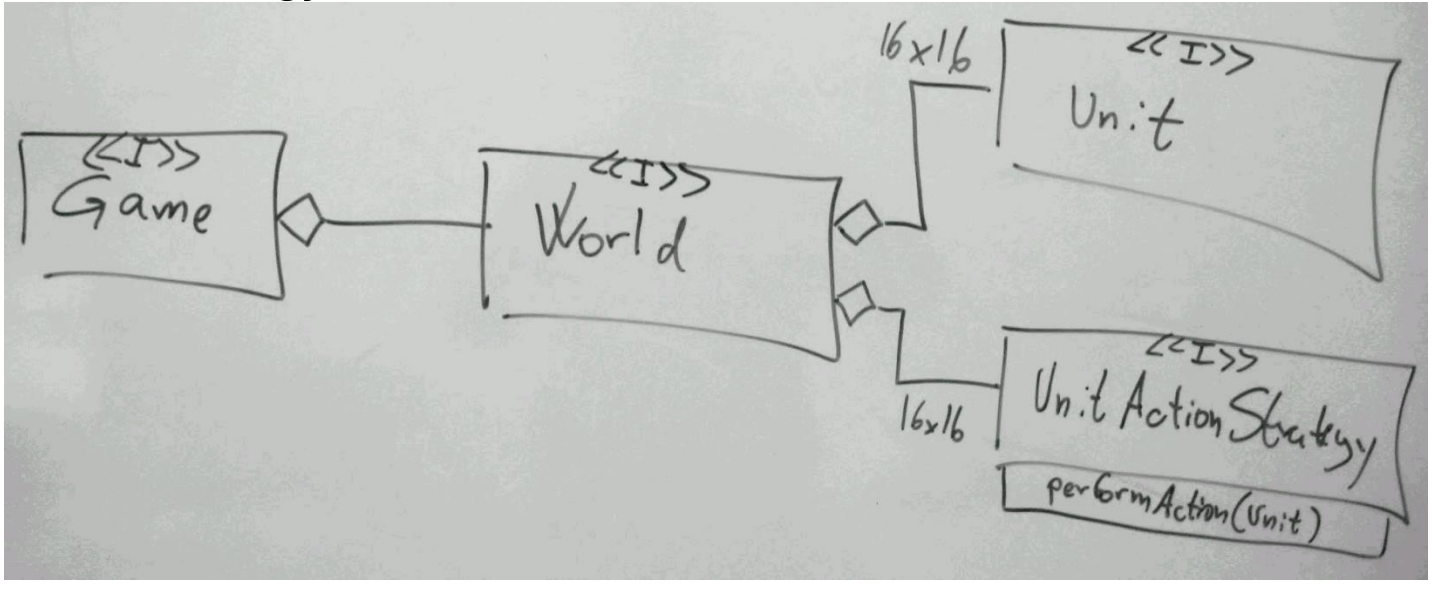
  - *If need arise, easy to refactor*
    - *CityImpl -> ModifiableCity*

Delegation that does not help

On the contrary…

Inheritance by Hand

- Maintain 1-1 mapping between a Unit and a ActionStrategy
  - So at (6,5) there must be a Settler; and at (6,5) therefore a SettlerActionStrategy



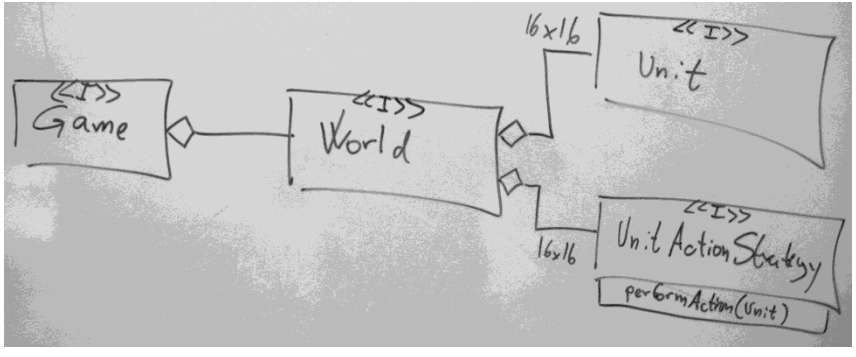
AARHUS UNIVERSITET

Inheritance by Hand

- Then to perform an action

```
game:: performActionAt(Position p) {
  Unit unit = world.getUnitAt(p)
  UnitActionStrategy strategy = world.getActionStrategyAt(p);
  strategy.performActionFor(unit);
}
```

- What is this?
  - It is **not** strategy!



It is an object where we maintain a relation to *one* of its methods **by hand**. It would be easier to just have the method in the Unit, right, and then use subclassing to determine its actual implementation. So this is 'inheritance' but hand coded! Fragile, tedious, low maintainability, an all the way inproper solution…

AARHUS UNIVERSITET

Inheritance by Hand

- Another example:
  - An AlphaGame inside a BetaGame
    - endOfTurn() { alphaCiv.endOfTurn(); }
    - moveUnit(f,t) { return alphaCiv.moveUnit(f,t); }
    - getAge() { [beta civ code here] }
- *This is not Strategy! It is recoding inheritance by hand!!!*
  - *You just hardcode 'super' to be 'alphaCiv'*