AARHUS UNIVERSITET

Software Engineering and Architecture

Some General Observations on the Mandatory

AARHUS UNIVERSITET

From the Trenches…

- Code duplication is…
- … a place where you *clean code*

```java
@Override
public void endTurn() {
  if (counterOfTurn % 2 == 0){
    playerInTurn = Player.PEDDERSEN;
    Heros.get(Player.PEDDERSEN).mana = 3;
    for (Card i : getField(Player.PEDDERSEN)){
      ((StandardCard) i).active = true;
    }
    if (Decks.get(Player.PEDDERSEN).isEmpty()){
      Heros.get(Player.PEDDERSEN).health -= 2;
    } else {
      Card temp = Decks.get(Player.PEDDERSEN).pop();
      Hands.get(Player.PEDDERSEN).add(0, temp);
    }
  } else {
    playerInTurn = Player.FINDUS;
    Heros.get(Player.FINDUS).mana = 3;
    for (Card i : getField(Player.FINDUS)){
      ((StandardCard) i).active = true;
    }
    if (Decks.get(Player.FINDUS).isEmpty()){
      Heros.get(Player.FINDUS).health -= 2;
    } else {
      Card temp = Decks.get(Player.FINDUS).pop();
      Hands.get(Player.FINDUS).add(0, temp);
    }
  }
  counterOfTurn++;
  }
```

AARHUS UNIVERSITET

“Inner” and “Outer”

Encapsulation:
Who can do what?

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 class StandardGame {
  public StandardGame(String variant) {
    if (variant.equals("AlphaStone")) {
      manaProductionStrategy = new Always3ManaPerTurnStrategy();
      winnerStrategy = new FindusWinsAtRound4Strategy();
      ....;
    } else if (variant.equals("BetaStone")) {
      manaProductionStrategy = new OneManaPerRoundStrategy();
      winnerStrategy = new WinnerDefeatsOpponentHeroStrategy();

    } else if (variant.equals("GammaStone")) {
      ...
    }
  }
}
```

- What happens when I want a SigmaStone variant?
  - I have to call *you guys* to resolder the wires inside!

AARHUS UNIVERSITET

From the Trenches

- Or – your own code…

```java
public StandardHotStoneGame(Player starting, String variant) {
  //strategies
  HashMap<String,Object> variantMap = Utility.getGameVariantSettings(variant);
  deckStrat = (DeckStrategy) variantMap.get("deckStrategy");
  heroStrat = (HeroStrategy) variantMap.get("heroStrategy");
  manaStrat = (ManaStrategy) variantMap.get("manaStrategy");
  winStrat = (WinnerStrategy) variantMap.get("winnerStrategy");
```

```java
public static HashMap<String, Object> getGameVariantSettings(String variant) {
  switch (variant) {
    case "Alpha" ->  {
      HashMap<String, Object> AlphaVariant = new HashMap<>();
      AlphaVariant.put("deckStrategy", new AlphaDeckStrategy());
      AlphaVariant.put("heroStrategy", new AlphaHeroStrategy());
      AlphaVariant.put("manaStrategy", new AlphaManaStrategy());
      AlphaVariant.put("winnerStrategy", new AlphaWinnerStrategy());
      return AlphaVariant;
    }
    case "Beta" -> {
      HashMap<String, Object> betaVariant = new HashMap<>();
      betaVariant.put("deckStrategy", new AlphaDeckStrategy());
      betaVariant.put("heroStrategy", new AlphaHeroStrategy());
      betaVariant.put("manaStrategy", new BetaManaStrategy());
      betaVariant.put("winnerStrategy", new BetaWinnerStrategy());
      return betaVariant;
    }
    case "Gamma" -> {
      HashMap<String, Object> gammaVariant = new HashMap<>();
      gammaVariant.put("deckStrategy", new AlphaDeckStrategy());
      gammaVariant.put("heroStrategy", new GammaHeroStrategy());
      gammaVariant.put("manaStrategy", new AlphaManaStrategy());
      gammaVariant.put("winnerStrategy", new AlphaWinnerStrategy());
      return gammaVariant;
    }
    case "Delta" -> {
      HashMap<String, Object> deltaVariant = new HashMap<>();
      deltaVariant.put("deckStrategy", new DeltaDeckStrategy());
      deltaVariant.put("heroStrategy", new AlphaHeroStrategy());
      deltaVariant.put("manaStrategy", new DeltaManaStrategy());
      deltaVariant.put("winnerStrategy", new AlphaWinnerStrategy());
      return deltaVariant;
    }
    default -> throw new RuntimeException("no such game variant");
  }

}
```

From the Trenches

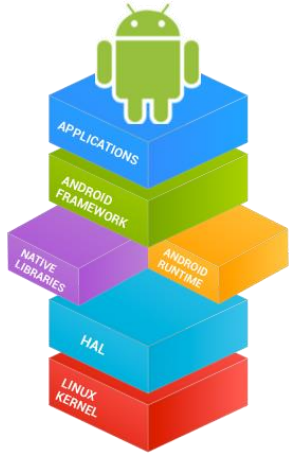
- Or…

```
public StandardHotStoneGame(String version){

  switch (version){
    case "A": // AlphaStone
      break;
    case "B": // BetaStone
      manaStrategy = new betaManaStrategy();
      winnerStrategy = new betaWinnerStrategy();
      break;
    case "Γ": // GammaStone
      heroStrategy = new gammaHeroStrategy();
      heroPowerStrategy = new gammaHeroPowerStrategy();
      break;
    case "Δ": // DeltaStone
      manaStrategy = new deltaManaStrategy();
      deckStrategy = new deltaDeckStrategy();
      break;
  }
```

Frameworks

- What is the process in the mandatory exercises?
  - *To use TDD and compositional design to transform an AlphaStone application into a **HotStone 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!?!*



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 StandardGame(String variant) {
  if (variant.equals("AlphaStone")) {
    manaProductionStrategy = new Always3ManaPerTurnStrategy();
    winnerStrategy = new FindusWinsAtRound4Strategy();
    ....;
  } else if (variant.equals("BetaStone")) {
```

- … is *not suitable for implementing frameworks…*
  - I have to open the TV to solder the wires inside ☹
  - You have to call Google to make your HCI project app ☹

So…

- Keep StandardGame, (StandardHero, StandardCard), … *closed for modification! General enough to handle many variants*
  - They form the framework that is reused *as-is*

- Allow adapting HotStone to a *new variant* by *addition*
  - **I can** code a new DeckBuildingStrategy which allows users to load a deck that they have crafted in a deck editor…
  - **I can** code a PriestHero which can heal minions on field…
  - And provide **my** strategies in the constructor **of your StdGame**
  - 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(WinnerStrategy winnerStrategy, ……..)*
- We will refactor HotStone soon to fix it…
  - Abstract Factory…

You Can Do More Outside

Inner and Outer have different Rules!

Parametric Variant

- Example:
  - GammaStone requirement: Heroes do different things

HotStone Framework Code

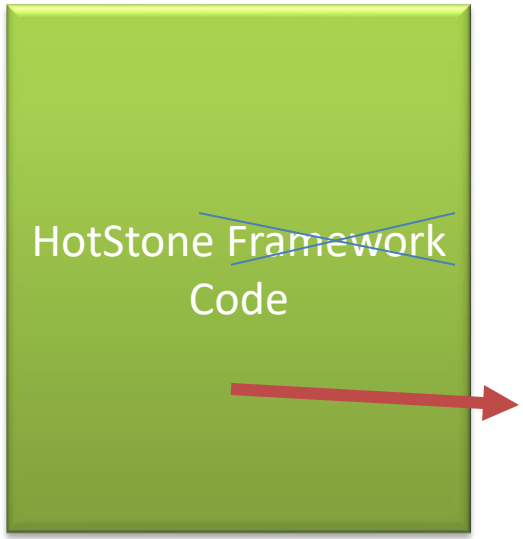
Coded by switching *within StandardGame*

```
@Override
public Status usePower(Player who) {
  // ... validation here...
  String heroType = game.getHero(who).getType();
  if (heroType.equals(GameConstants.THAI_CHEF_HERO_TYPE)) {
    // Chili = damage opponent 2
    // ... Damage opponent hero
  } else if (heroType.equals(GameConstants.DANISH_CHEF_HERO_TYPE)) {
    Card sovs = new StandardCard(who, "Sovs", 1,1,1);
    // ...Field sovs on my battlefield

  }
  return Status.OK;
}
```

Parametric Variant

- Example:
  - GammaStone requirement: Heroes do different things



Liability: HotStone has hard bindings to specific Hero types. *Only Change by Modification!*

```java
@Override
public Status usePower(Player who) {
  // ... validation here...
  String heroType = game.getHero(who).getType();
  if (heroType.equals(GameConstants.THAI_CHEF_HERO_TYPE)) {
    // Chili = damage opponent 2
    // ... Damage opponent hero
  } else if (heroType.equals(GameConstants.DANISH_CHEF_HERO_TYPE)) {
    Card sovs = new StandardCard(who, "Sovs", 1,1,1);
    // ...Field sovs on my battlefield

  }
  return Status.OK;
}
```

The *Worst of All Worlds* design

- Example:
  - GammaStone requirement: Heroes do different things

```java
@Override
public Status usePower(Player who) {
  // ... validation here...
  HeroActionStrategy heroStrategy = null;
  String heroType = game.getHero(who).getType();
  if (heroType.equals(GameConstants.THAI_CHEF_HERO_TYPE)) {
    // Chili = damage opponent 2
    heroStrategy = new HurtOpponentStrategy();
  } else if (heroType.equals(GameConstants.DANISH_CHEF_HERO_TYPE)) {
    // ...Field sovs on my battlefield
    heroStrategy = new FieldSovsStrategy();

  }
  heroStrategy.usePower();
  return Status.OK;
}
```

The *Worst of All Worlds* design

- Example:
  - GammaStone requirement: Heroes do different things

```
@Override
public Status usePower(Player who) {
  // ... validation here...
  HeroActionStrategy heroStrategy = null;
  String heroType = game.getHero(who).getType
  if (heroType.equals(GameConstants.THAI_CHEF
    // Chili = damage opponent 2
    heroStrategy = new HurtOpponentStrategy()
  } else if (heroType.equals(GameConstants.DA
    // ...Field sovs on my battlefield
    heroStrategy = new FieldSovsStrategy();

  }
  heroStrategy.usePower();
  return Status.OK;
}
```

legates

Strategy

REALLY BAD: If you have hero-type switching code in GameImpl, you ***still have a parametric solution with all its liabilities!!!***
Plus all the extra interfaces of the compositional approach.
DO THE SAME THING THE SAME WAY!!!

Compositional Variant

- Example:
  - GammaStone requirement: Heroes do different things

HotStone Framework Code:

GammaStone Delegates

ThaiDanishHeroStrategy

```
@Override
public Status usePower(Player who) {
  // ... validation here...
  heroActionStrategy.applyPower(who, this);
  return Status.OK;
}
```

```
public class ThaiDanishHeroActionStrategy implements HeroActionStrategy {
  @Override
  public void applyPower(Player who, StandardGame game) {
    String heroType = game.getHero(who).getType();
    if (heroType.equals(GameConstants.THAI_CHEF_HERO_TYPE)) {
      // Chili = damage opponent 2
      // ... Damage opponent hero
    } else if (heroType.equals(GameConstants.DANISH_CHEF_HERO_TYPE)) {
      Card sovs = new StandardCard(who, "Sovs", 1,1,1);
      // ...Field sovs on my battlefield

    }
  }
}
```

Compositional Variant

- Example:
  - GammaStone requirement: Heroes do different things

HotStone F

What would Henrik say? Is this Parametric design???

GammaStone Delegates

ThaiDanishHeroStrategy

```
@Override
public Status usePower(Player who) {
  // ... validation here...
  heroActionStrategy.applyPower(who, this);
  return Status.OK;
}
```

```
public class ThaiDanishHeroActionStrategy implements HeroActionStrategy {
  @Override
  public void applyPower(Player who, StandardGame game) {
    String heroType = game.getHero(who).getType();
    if (heroType.equals(GameConstants.THAI_CHEF_HERO_TYPE)) {
      // Chili = damage opponent 2
      // ... Damage opponent hero
    } else if (heroType.equals(GameConstants.DANISH_CHEF_HERO_TYPE)) {
      Card sovs = new StandardCard(who, "Sovs", 1,1,1);
      // ...Field sovs on my battlefield

    }
  }
}
```

Compositional Variant

- Example:
  - GammaStone requirement: Heroes do different things

This is **a much better design!**
Why?
Because a) No hard binding in Game b) GammaStone requirements are *expressed explicitly in a single piece of code that bears the correct name!*

GammaStone Delegates

ThaiDanishHeroStrategy

```
@Override
public Status usePower(Player who) {
  // ... validation here...
  heroActionStrategy.applyPower(who, this);
  return Status.OK;
}
```

```
aiDanishHeroActionStrategy implements HeroActionStrategy {

  pplyPower(Player who, StandardGame game) {
  Type = game.getHero(who).getType();
  if (heroType.equals(GameConstants.THAI_CHEF_HERO_TYPE)) {
    // Chili = damage opponent 2
    // ... Damage opponent hero
  } else if (heroType.equals(GameConstants.DANISH_CHEF_HERO_TYPE)) {
    Card sovs = new StandardCard(who, "Sovs", 1,1,1);
    // ...Field sovs on my battlefield

  }
 }
}
```

Compositional Variant

- And you can avoid the switching completely
  - Put the ‘power strategy’ into the Hero implementation instead; fetch it, and then apply it…
  - Requires a ‘HeroBuildingStrategy’ to create the proper Hero types, then…

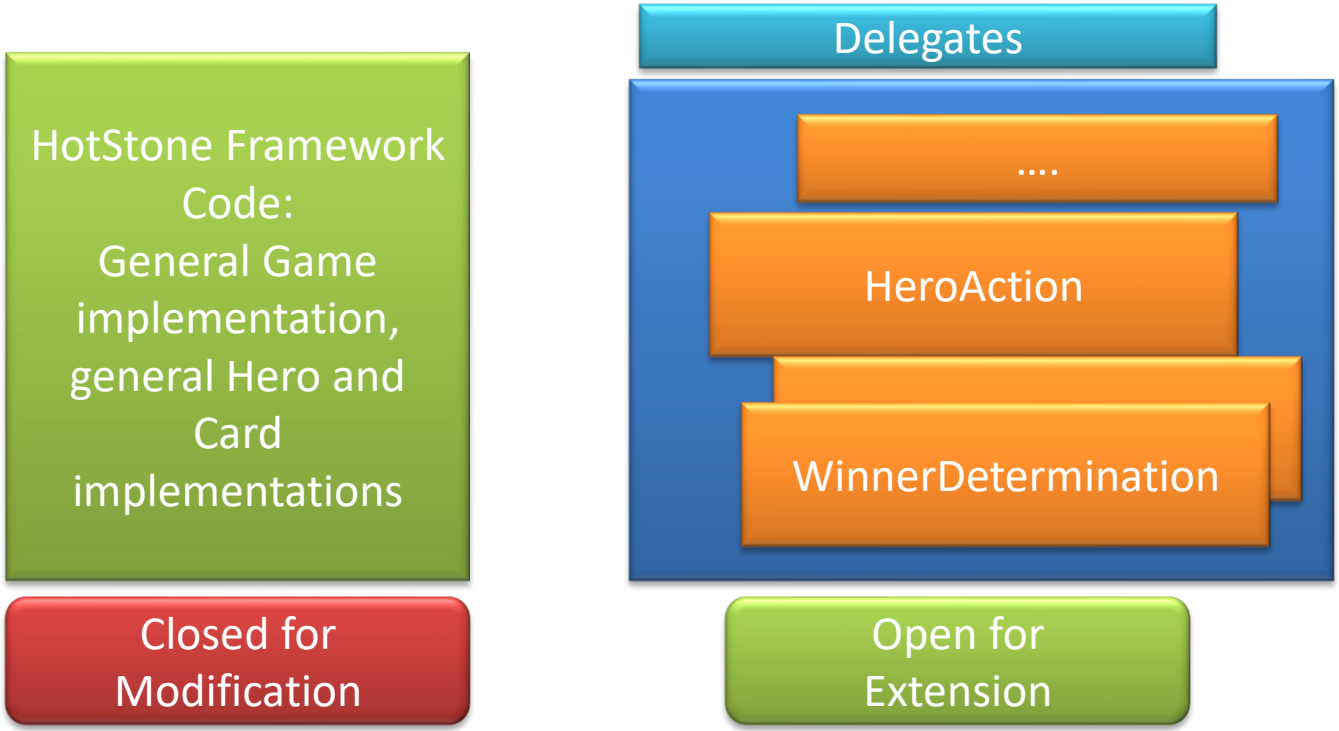
```
@Override
public Status usePower(Player who) {
  // ... validation here...
  // Now - exercise the hero power

  EffectStrategy heroPower = hero.getEffectStrategy();
  heroPower.apply(this);
  return Status.OK;
}
```

- Critique: *Approaching is a way to implement subclassing by hand ☺…*

Inner and Outer

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



AARHUS UNIVERSITET

Clean Code: Prefer Exceptions

Uncle Bob

- *Prefer exceptions over returning status codes*
  - Throw PageDoesNotExistException instead of return E_PAGE_DOES_NOT_EXIST
- So why have you decided on the Status enum???
- Argument
  - Hm hm hm… None!
  - Conclusion: On the ToDo list for next year
- But keep it as the GUI will assume it!