AARHUS UNIVERSITET

Software Engineering and Architecture

Some General Observations on the Mandatory

AARHUS UNIVERSITET

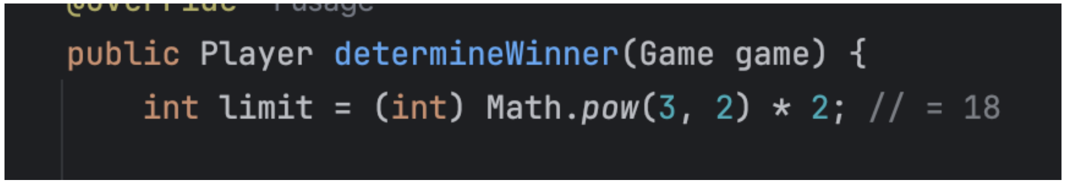
Advice for Epsilon

- EpsilonStone is the ‘introduce test stub to handle indirect input (a random number)’ exercise
- My advice is
  - Study the Weekplan 6 kata on *one way of how **not** to do it!*
  - *A Stub is a replacement for the behavior not under test control*
    - ***And nothing more***
  - That is: Make the ‘indirect input algorithm’ as *small as possible*
    - *Encapsulate the **minimum** behavior*

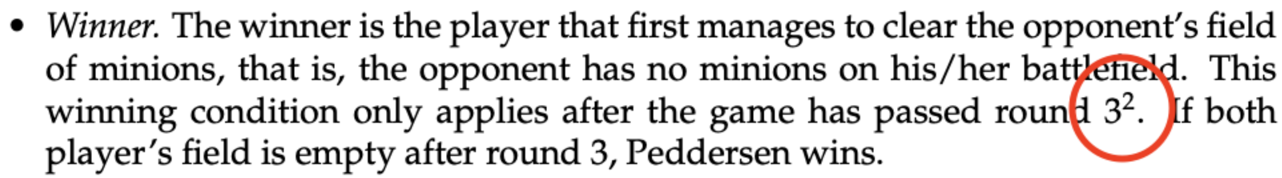
AARHUS UNIVERSITET

From the Trenches

World’s worst footnote ☺



```
public Player determineWinner(Game game) {
    int limit = (int) Math.pow(3, 2) * 2; // = 18
```

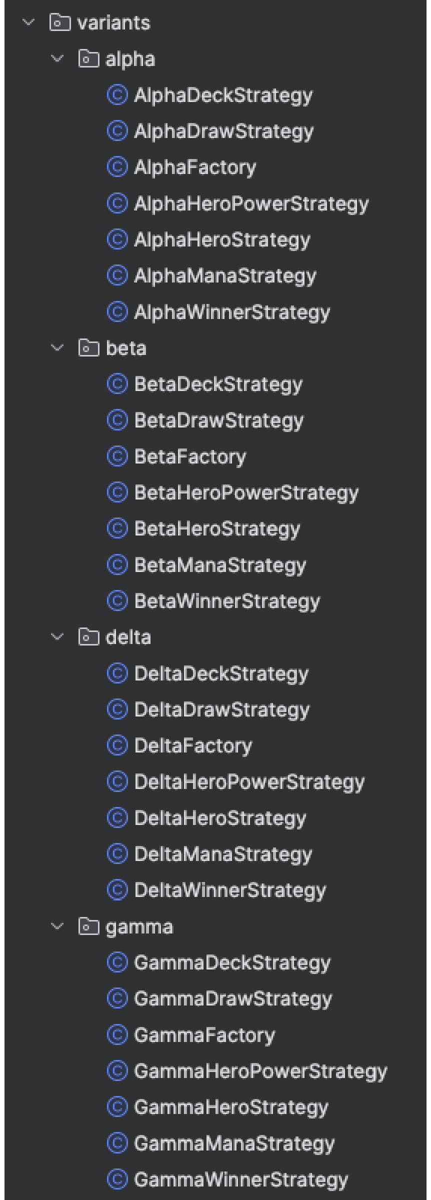


- *Winner.* The winner is the player that first manages to clear the opponent’s field of minions, that is, the opponent has no minions on his/her battlefield. This winning condition only applies after the game has passed round 3[2]. If both player’s field is empty after round 3, Peddersen wins.

AARHUS UNIVERSITET

- Benefits of this solution?
- Critique of this solution?

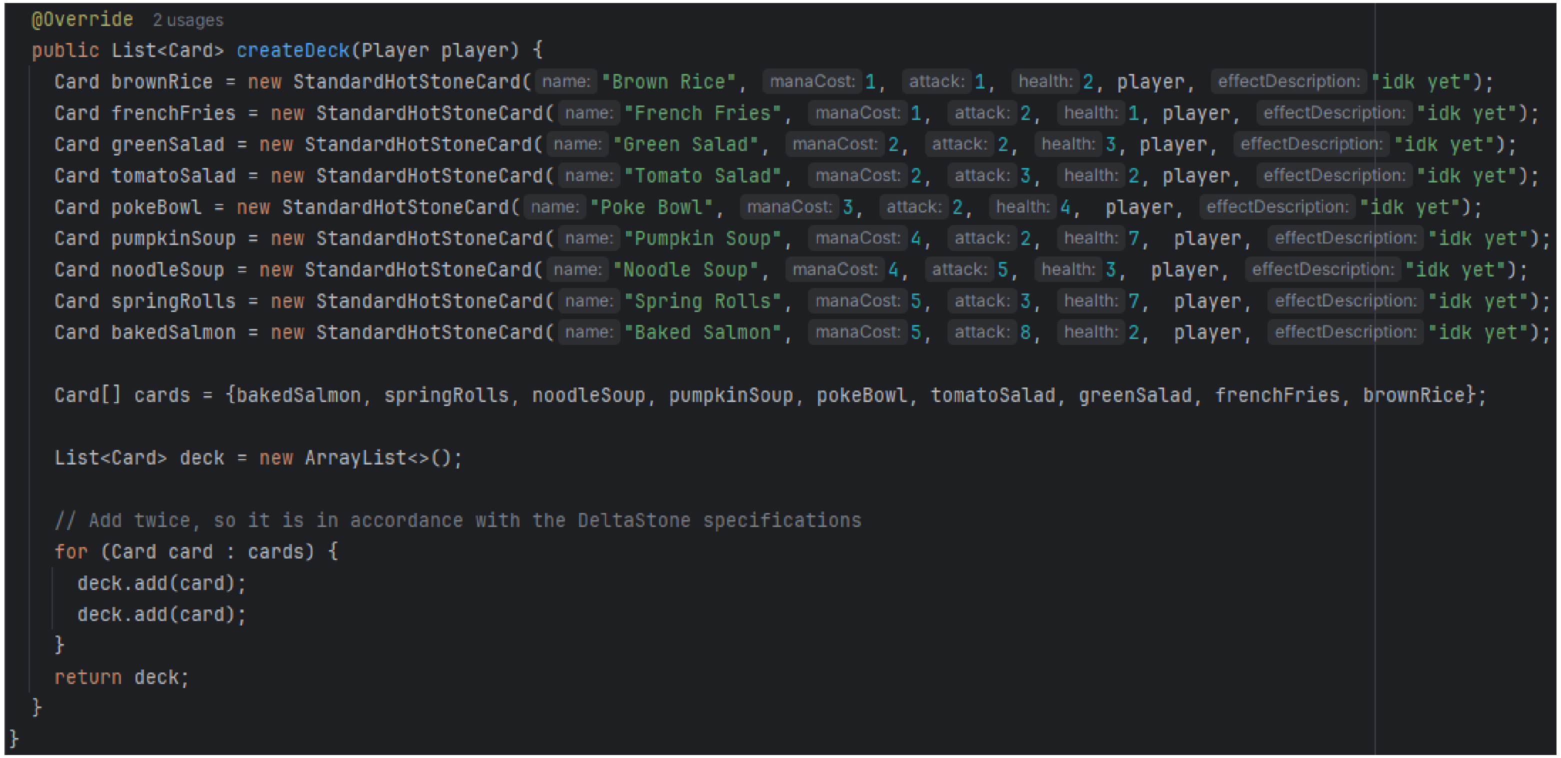
- Morale
  - Naming matters
  - Duplication matters



```
variants
  alpha
    AlphaDeckStrategy
    AlphaDrawStrategy
    AlphaFactory
    AlphaHeroPowerStrategy
    AlphaHeroStrategy
    AlphaManaStrategy
    AlphaWinnerStrategy
  beta
    BetaDeckStrategy
    BetaDrawStrategy
    BetaFactory
    BetaHeroPowerStrategy
    BetaHeroStrategy
    BetaManaStrategy
    BetaWinnerStrategy
  delta
    DeltaDeckStrategy
    DeltaDrawStrategy
    DeltaFactory
    DeltaHeroPowerStrategy
    DeltaHeroStrategy
    DeltaManaStrategy
    DeltaWinnerStrategy
  gamma
    GammaDeckStrategy
    GammaDrawStrategy
    GammaFactory
    GammaHeroPowerStrategy
    GammaHeroStrategy
    GammaManaStrategy
    GammaWinnerStrategy
```

Remember: References

- What’s the issue?



```
  @Override  2 usages
  public List<Card> createDeck(Player player) {
    Card brownRice = new StandardHotStoneCard( name: "Brown Rice",  manaCost: 1,  attack: 1,  health: 2, player,  effectDescription: "idk yet");
    Card frenchFries = new StandardHotStoneCard( name: "French Fries",  manaCost: 1,  attack: 2,  health: 1, player,  effectDescription: "idk yet");
    Card greenSalad = new StandardHotStoneCard( name: "Green Salad",  manaCost: 2,  attack: 2,  health: 3, player,  effectDescription: "idk yet");
    Card tomatoSalad = new StandardHotStoneCard( name: "Tomato Salad",  manaCost: 2,  attack: 3,  health: 2, player,  effectDescription: "idk yet");
    Card pokeBowl = new StandardHotStoneCard( name: "Poke Bowl",  manaCost: 3,  attack: 2,  health: 4,  player,  effectDescription: "idk yet");
    Card pumpkinSoup = new StandardHotStoneCard( name: "Pumpkin Soup",  manaCost: 4,  attack: 2,  health: 7,  player,  effectDescription: "idk yet");
    Card noodleSoup = new StandardHotStoneCard( name: "Noodle Soup",  manaCost: 4,  attack: 5,  health: 3,  player,  effectDescription: "idk yet");
    Card springRolls = new StandardHotStoneCard( name: "Spring Rolls",  manaCost: 5,  attack: 3,  health: 7,  player,  effectDescription: "idk yet");
    Card bakedSalmon = new StandardHotStoneCard( name: "Baked Salmon",  manaCost: 5,  attack: 8,  health: 2,  player,  effectDescription: "idk yet");

    Card[] cards = {bakedSalmon, springRolls, noodleSoup, pumpkinSoup, pokeBowl, tomatoSalad, greenSalad, frenchFries, brownRice};

    List<Card> deck = new ArrayList<>();

    // Add twice, so it is in accordance with the DeltaStone specifications
    for (Card card : cards) {
      deck.add(card);
      deck.add(card);
    }
    return deck;
  }
}
```

Encapsulate What Varies !

- *Make your variability points as precise as possible*
  - What is the issue here?

```
public class FixedMana5 implements ManaStrategy {
    @Override
    public void onTurnBegan(StandardHotStoneGame g) {
        g.getPlayerState(g.getPlayerInTurn()).setMana(5);
        g.getPlayerState(g.getPlayerInTurn()).setCanUsePower(true);
        g.activateField(g.getPlayerInTurn());
    }
}
```

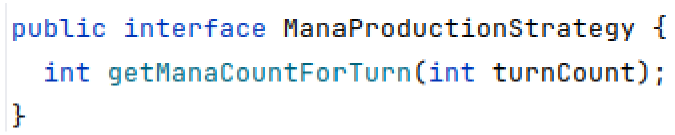
- Alternative way of mutating Game?

```
public class GrowingByRound implements ManaStrategy {
    @Override
    public void
    onTurnBegan(StandardHotStoneGame g) {
        int mana = g.getRoundNumber();
        g.getPlayerState(g.getPlayerInTurn()).setMana(mana);
        g.getPlayerState(g.getPlayerInTurn()).setCanUsePower(true);
        g.activateField(g.getPlayerInTurn());
    }
}
```

AARHUS UNIVERSITET

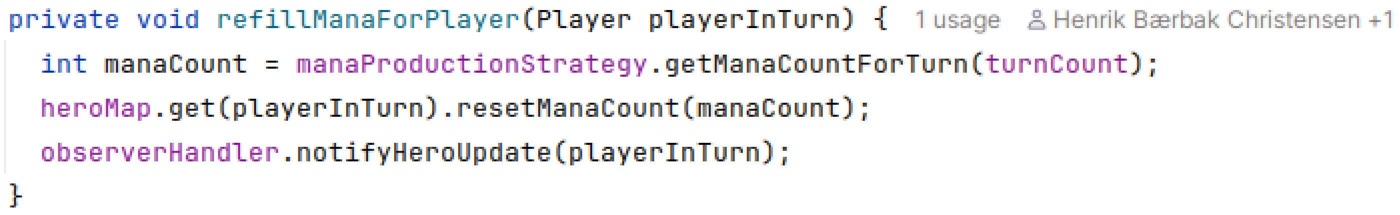
If possible, let Game do it

- If you can compute a *value* in your strategy, then the assignment is in the Game object
  - *Which is much a more cohesive way*
    - *Game changes game state; instead of some strategy*



```
public interface ManaProductionStrategy {
  int getManaCountForTurn(int turnCount);
}
```

In StandardGame:

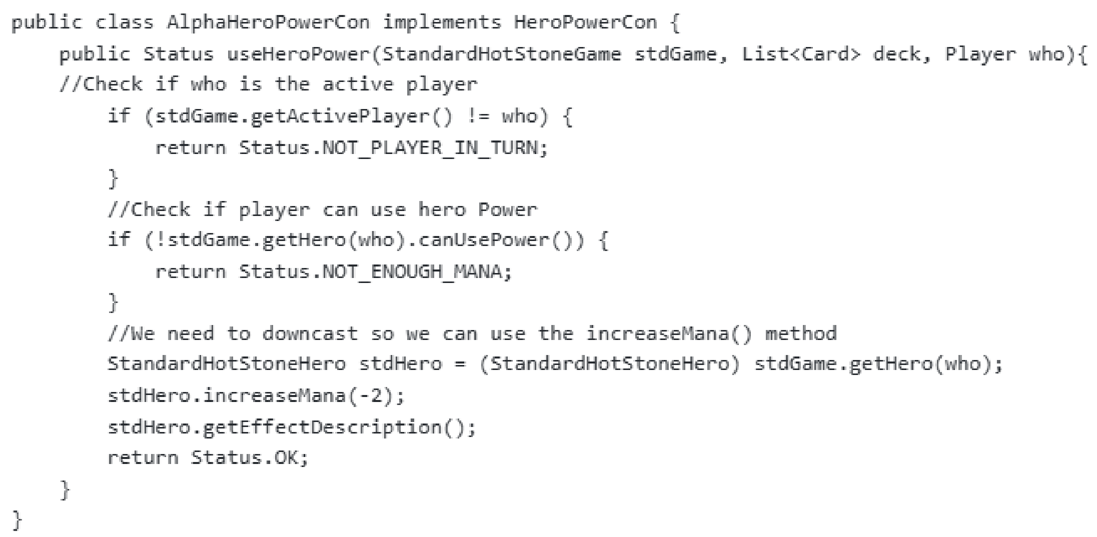


```
private void refillManaForPlayer(Player playerInTurn) {  1 usage  Henrik Bærbak Christensen +1
  int manaCount = manaProductionStrategy.getManaCountForTurn(turnCount);
  heroMap.get(playerInTurn).resetManaCount(manaCount);
  observerHandler.notifyHeroUpdate(playerInTurn);
}
```

Encapsulate What Varies

- Another example

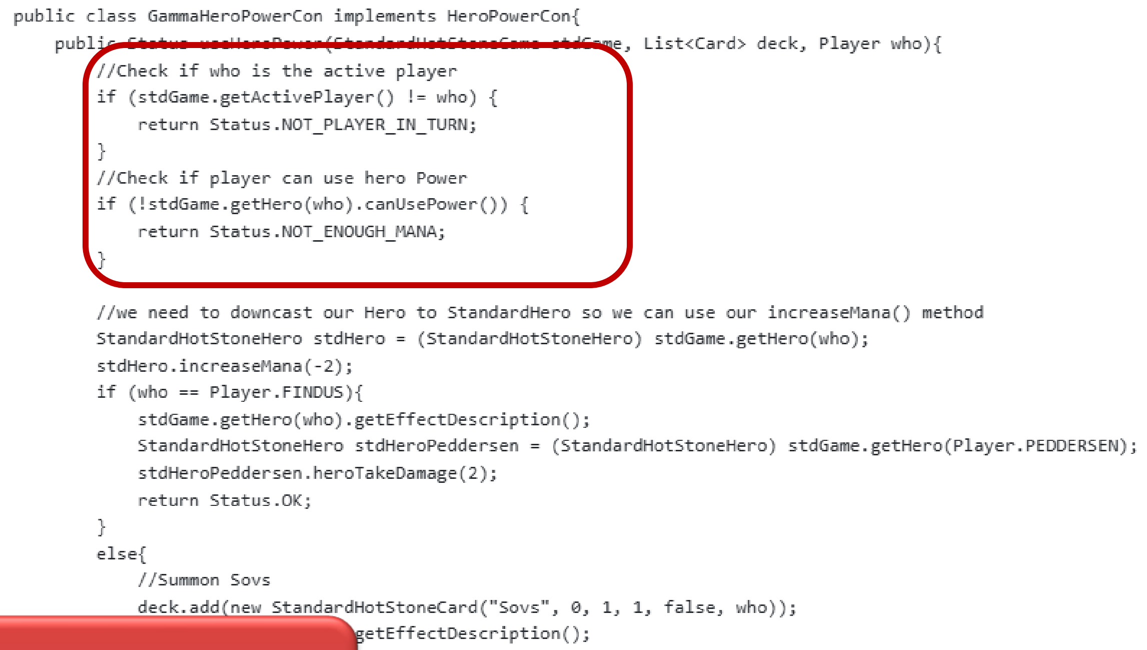
- What will the next strategy look like?



```
public class AlphaHeroPowerCon implements HeroPowerCon {
    public Status useHeroPower(StandardHotStoneGame stdGame, List<Card> deck, Player who){
    //Check if who is the active player
        if (stdGame.getActivePlayer() != who) {
            return Status.NOT_PLAYER_IN_TURN;
        }
        //Check if player can use hero Power
        if (!stdGame.getHero(who).canUsePower()) {
            return Status.NOT_ENOUGH_MANA;
        }
        //We need to downcast so we can use the increaseMana() method
        StandardHotStoneHero stdHero = (StandardHotStoneHero) stdGame.getHero(who);
        stdHero.increaseMana(-2);
        stdHero.getEffectDescription();
        return Status.OK;
    }
}
```

Next Strategy

- Code duplication!

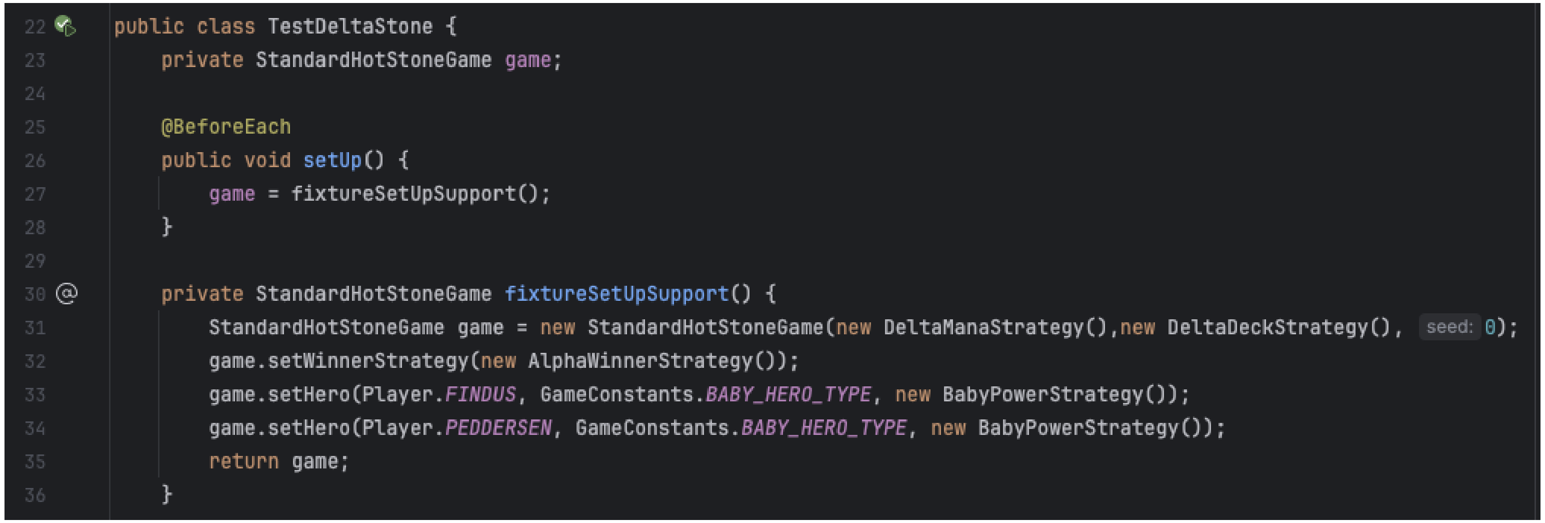


```
public class GammaHeroPowerCon implements HeroPowerCon{
    public Status useHeroPower(StandardHotStoneGame stdGame, List<Card> deck, Player who){
        //Check if who is the active player
        if (stdGame.getActivePlayer() != who) {
            return Status.NOT_PLAYER_IN_TURN;
        }
        //Check if player can use hero Power
        if (!stdGame.getHero(who).canUsePower()) {
            return Status.NOT_ENOUGH_MANA;
        }

        //we need to downcast our Hero to StandardHero so we can use our increaseMana() method
        StandardHotStoneHero stdHero = (StandardHotStoneHero) stdGame.getHero(who);
        stdHero.increaseMana(-2);
        if (who == Player.FINDUS){
            stdGame.getHero(who).getEffectDescription();
            StandardHotStoneHero stdHeroPeddersen = (StandardHotStoneHero) stdGame.getHero(Player.PEDDERSEN);
            stdHeroPeddersen.heroTakeDamage(2);
            return Status.OK;
        }
        else{
            //Summon Sovs
            deck.add(new StandardHotStoneCard("Sovs", 0, 1, 1, false, who));
                                     getEffectDescription();
```

ISO 9126: Stability and Changeability capabilities are suffering ☹

Cohesion? Coupling?

- Try to avoid one variant's requirements affect all others…
- Let Game do what the Game must do…
  - Cohesion again.
- What is the issue (1)? What is the issue (2)?



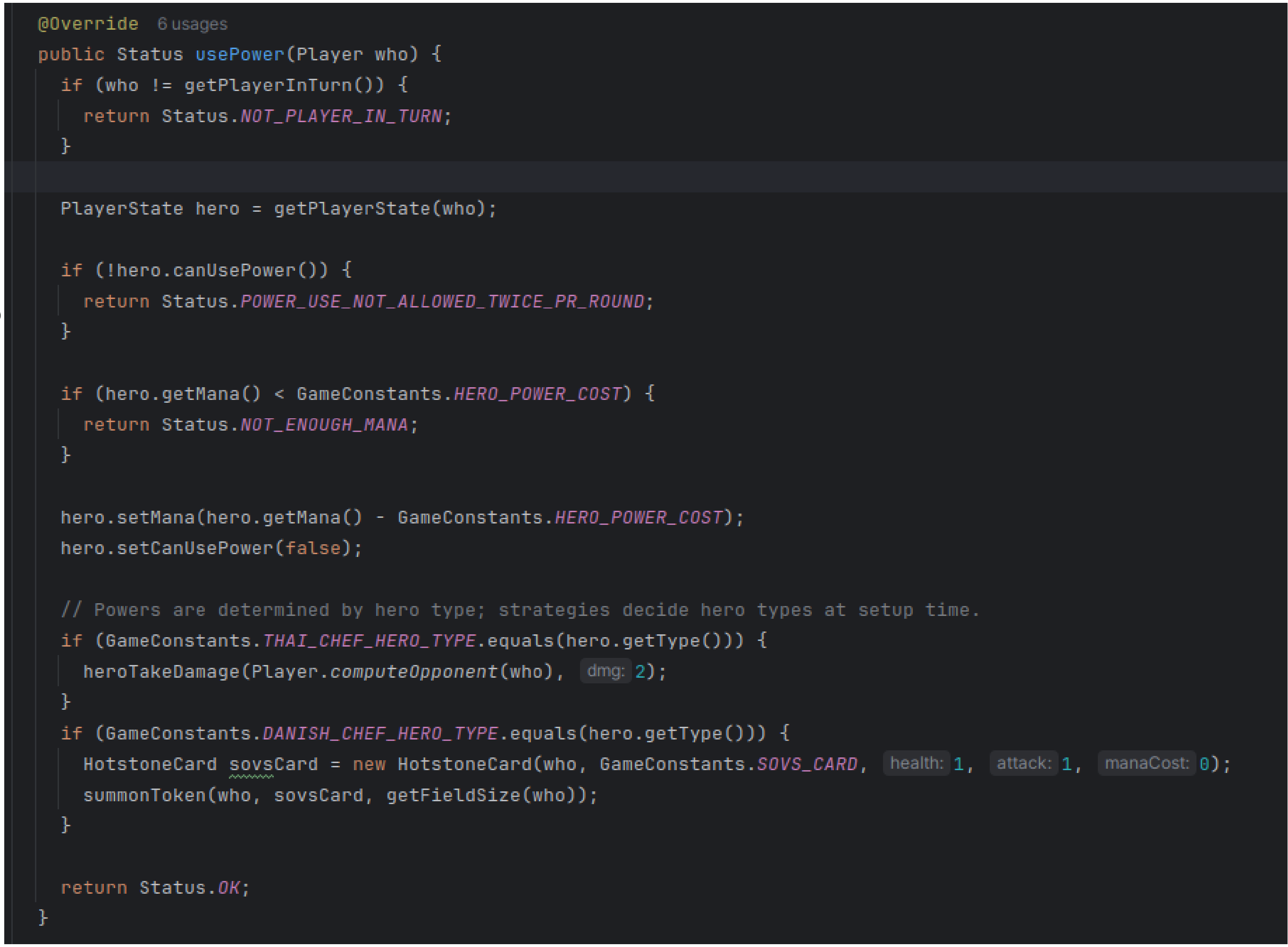
```java
public class TestDeltaStone {
    private StandardHotStoneGame game;

    @BeforeEach
    public void setUp() {
        game = fixtureSetUpSupport();
    }

    private StandardHotStoneGame fixtureSetUpSupport() {
        StandardHotStoneGame game = new StandardHotStoneGame(new DeltaManaStrategy(),new DeltaDeckStrategy(), seed: 0);
        game.setWinnerStrategy(new AlphaWinnerStrategy());
        game.setHero(Player.FINDUS, GameConstants.BABY_HERO_TYPE, new BabyPowerStrategy());
        game.setHero(Player.PEDDERSEN, GameConstants.BABY_HERO_TYPE, new BabyPowerStrategy());
        return game;
    }
```

AARHUS UNIVERSITET

What is the Variability Mgt?

- Src-code copy?
- Parametric?
- Polymorphic?

- Composi-tional?



```
@Override  6 usages
public Status usePower(Player who) {
  if (who != getPlayerInTurn()) {
    return Status.NOT_PLAYER_IN_TURN;
  }

  PlayerState hero = getPlayerState(who);

  if (!hero.canUsePower()) {
    return Status.POWER_USE_NOT_ALLOWED_TWICE_PR_ROUND;
  }

  if (hero.getMana() < GameConstants.HERO_POWER_COST) {
    return Status.NOT_ENOUGH_MANA;
  }

  hero.setMana(hero.getMana() - GameConstants.HERO_POWER_COST);
  hero.setCanUsePower(false);

  // Powers are determined by hero type; strategies decide hero types at setup time.
  if (GameConstants.THAI_CHEF_HERO_TYPE.equals(hero.getType())) {
    heroTakeDamage(Player.computeOpponent(who),  dmg: 2);
  }
  if (GameConstants.DANISH_CHEF_HERO_TYPE.equals(hero.getType())) {
    HotstoneCard sovsCard = new HotstoneCard(who, GameConstants.SOVS_CARD,  health: 1,  attack: 1,  manaCost: 0);
    summonToken(who, sovsCard, getFieldSize(who));
  }

  return Status.OK;
}
```

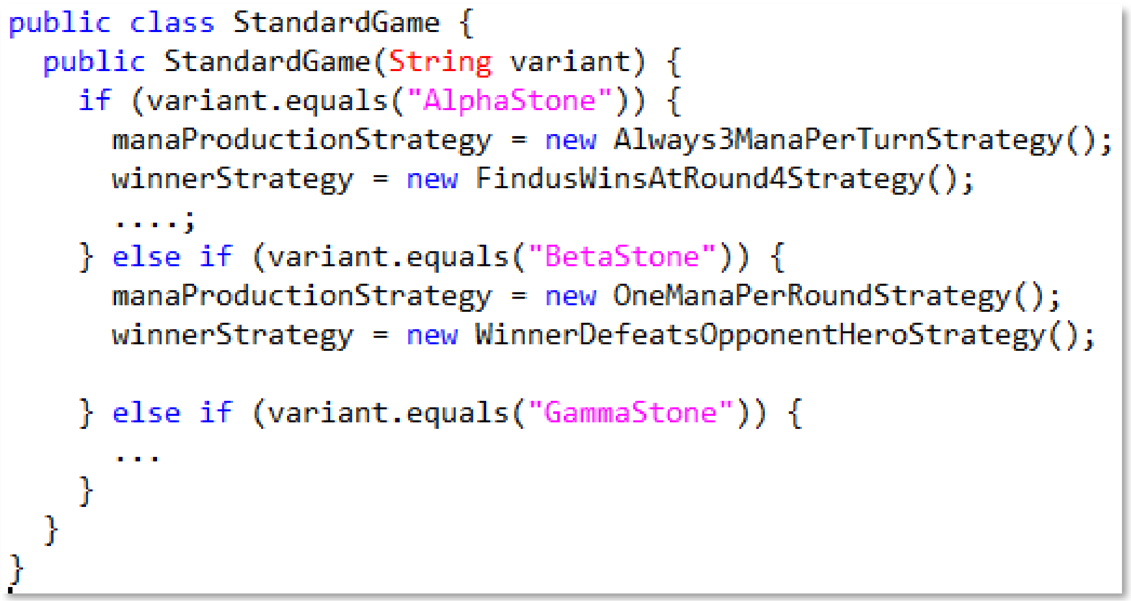
AARHUS UNIVERSITET

“Inner” and “Outer”

Encapsulation:
Who can do what?

AARHUS UNIVERSITET

A Previous Year Example

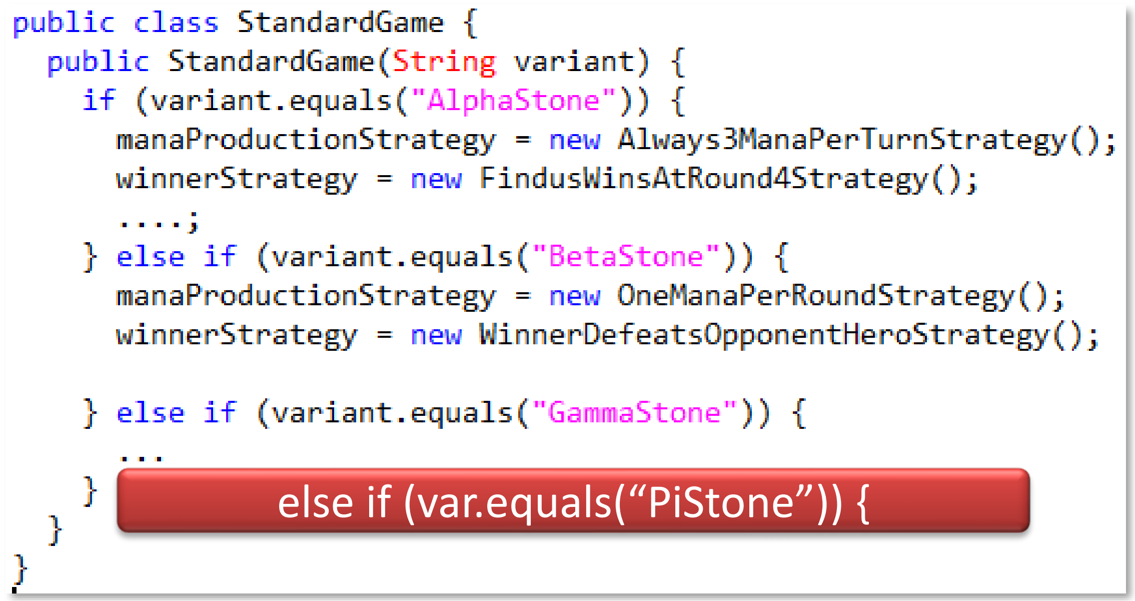


```
public class StandardGame {
  public StandardGame(String variant) {
    if (variant.equals("AlphaStone")) {
      manaProductionStrategy = new Always3ManaPerTurnStrategy();
      winnerStrategy = new FindusWinsAtRound4Strategy();
      ....;
    } else if (variant.equals("BetaStone")) {
      manaProductionStrategy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

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

- What happens when I want a PiStone variant?

AARHUS UNIVERSITET

Last Year Example



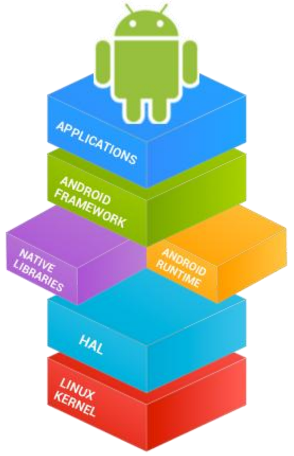
```
public class StandardGame {
  public StandardGame(String variant) {
    if (variant.equals("AlphaStone")) {
      manaProductionStrategy = new Always3ManaPerTurnStrategy();
      winnerStrategy = new FindusWinsAtRound4Strategy();
      ....;
    } else if (variant.equals("BetaStone")) {
      manaProductionStrategy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

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

else if (var.equals(“PiStone”)) {

- **Change by modification ☹**
  - ***Code must be changed every time a new variant is envisioned…***

Frameworks

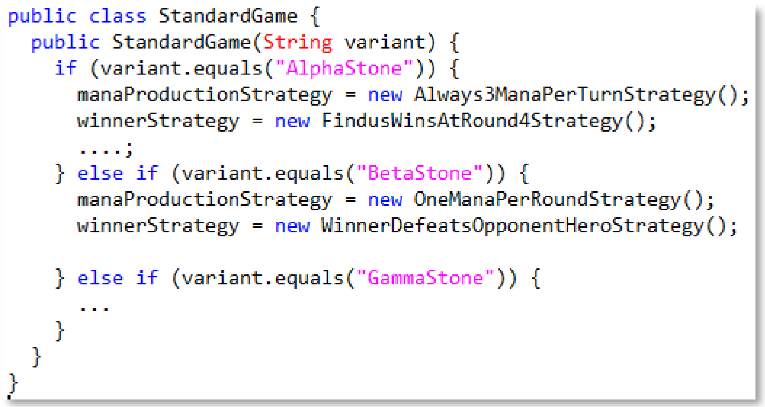
- What is the process in the mandatory exercises?
  - *To use TDD and compositional design to transform an AlphaStone application into a **HotStone framework***
- Frameworks are
  - Reusable software designs and implementations
  - Must be reconfigurable *from the outside*
    - *Just like a TV set or a mobile phone*
- 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

How do I?

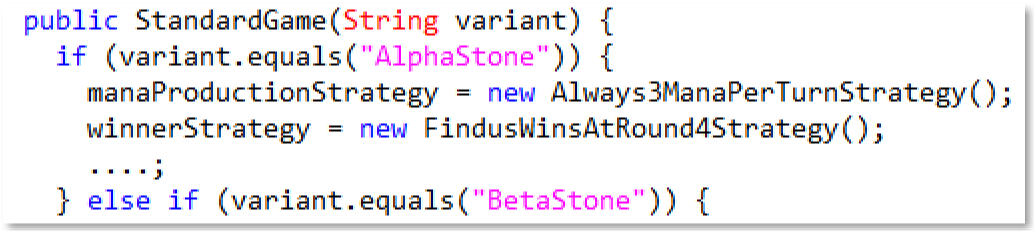
- Switch from channel TV2 to DR on my Samsung TV set?
  - A) Push the Button ‘3’ on the TV’s remote control *interface?*
  - B) Call Samsung to tell them to send a man to re-solder the wire inside the TV set?
- *This code is using method B*



```
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")) {
      ...
    }
  }
}
```

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…*

You Can Do More Outside

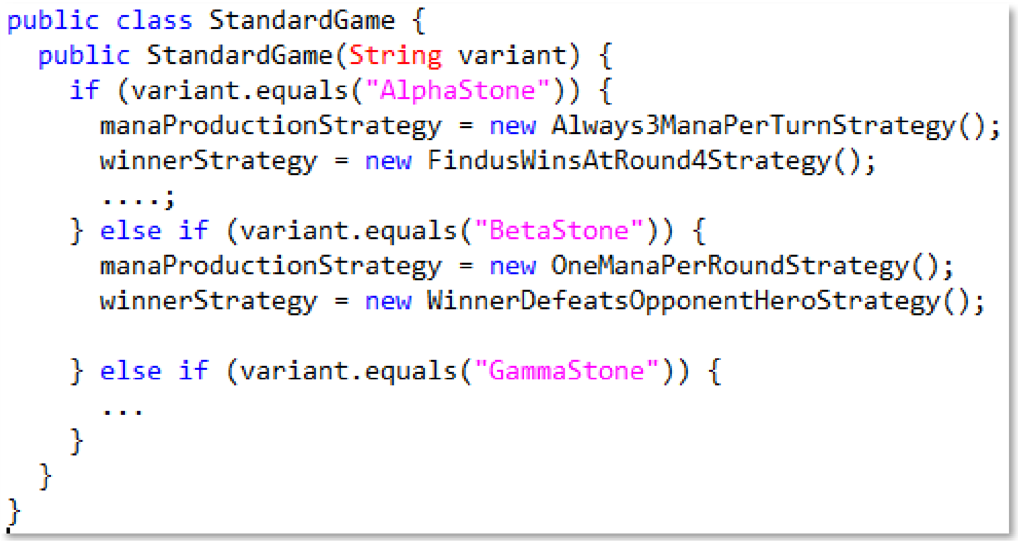
Inner and Outer have different Rules!

Or

Switches are OK in Strategies…

Switching on Variants

- … is parametric design, right?

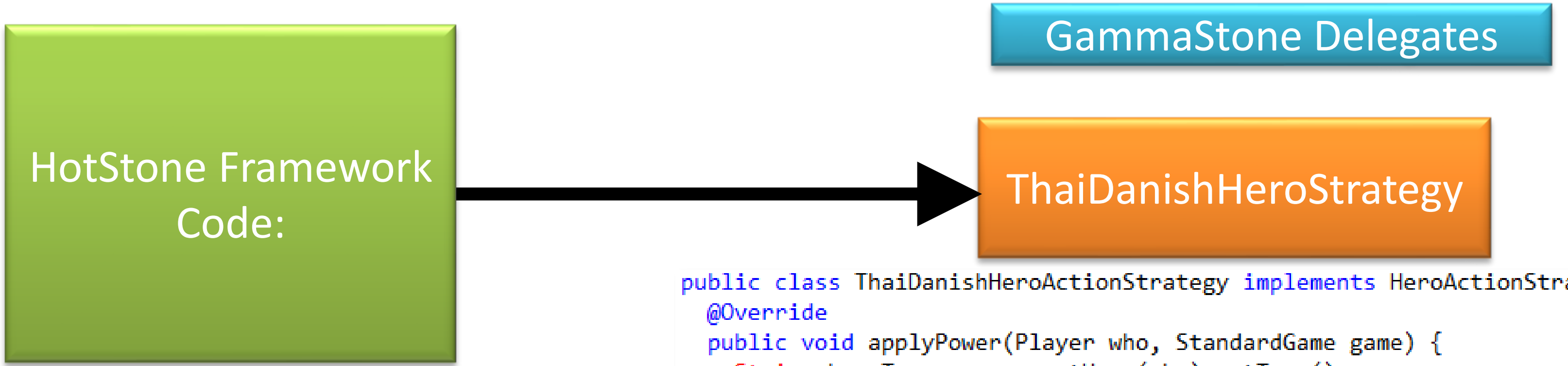


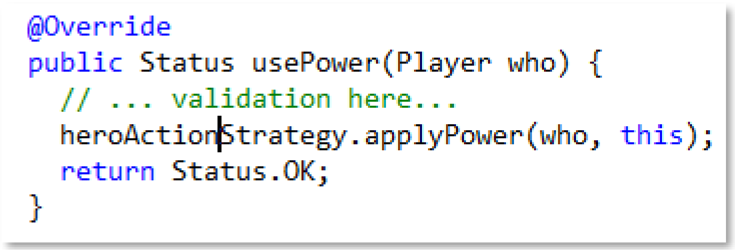
```
public class StandardGame {
  public StandardGame(String variant) {
    if (variant.equals("AlphaStone")) {
      manaProductionStrategy = new Always3ManaPerTurnStrategy();
      winnerStrategy = new FindusWinsAtRound4Strategy();
      ....;
    } else if (variant.equals("BetaStone")) {
      manaProductionStrategy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

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

AARHUS UNIVERSITET

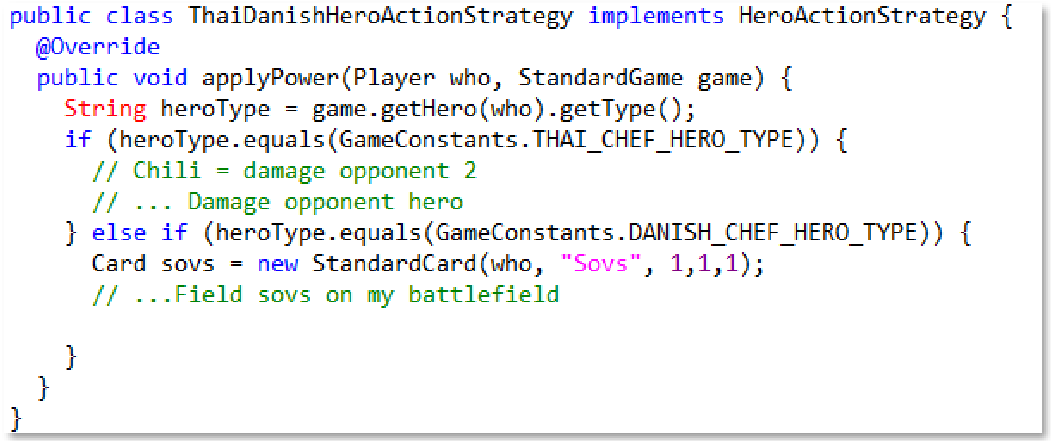
Compositional Variant

- Example:
  - GammaStone requirement: Heroes do different things





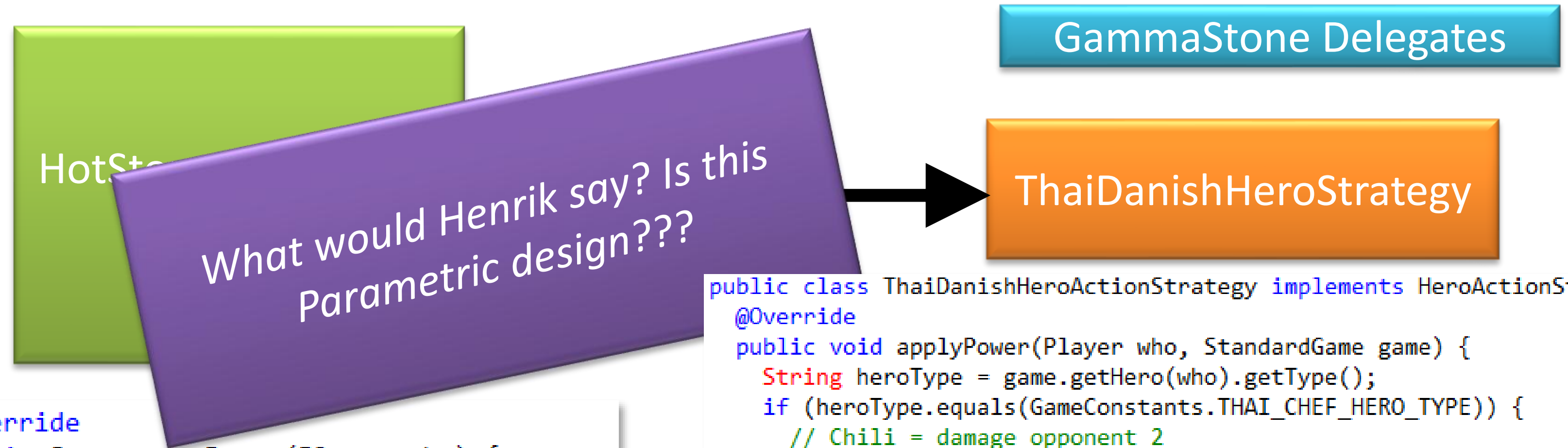
```
@Override
public Status usePower(Player who) {
  // ... validation here...
  heroActionStrategy.applyPower(who, this);
  return Status.OK;
}
```

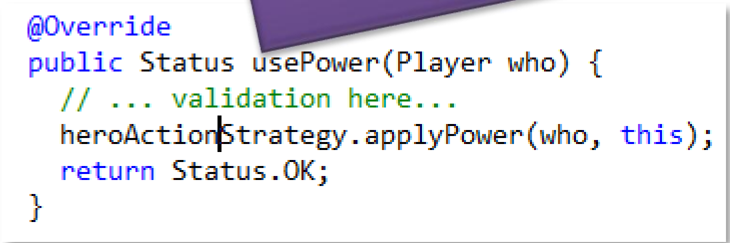


```
public class ThaiDanishHeroActionStrategy implements HeroActionStrategy {
  @Override
  public void applyPower(Player who, StandardGame game) {
    String heroType = game.getHero(who).getType();
    if (heroType.equals(GameConstants.THAI_CHEF_HERO_TYPE)) {
      // Chili = damage opponent 2
      // ... 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

    }
  }
}
```

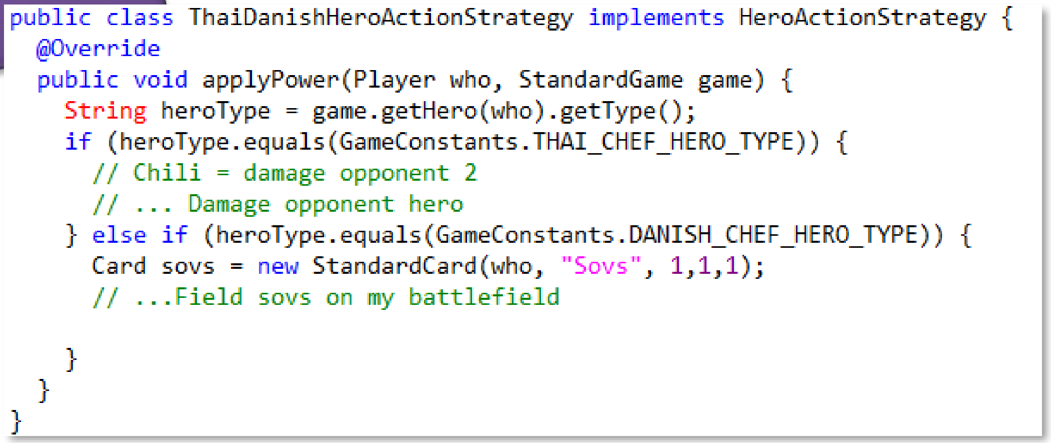
Compositional Variant

- Example:
  - GammaStone requirement: Heroes do different things





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

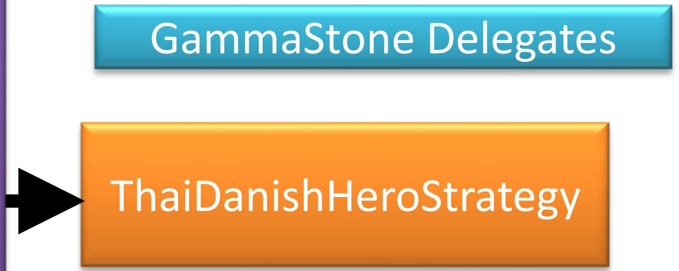


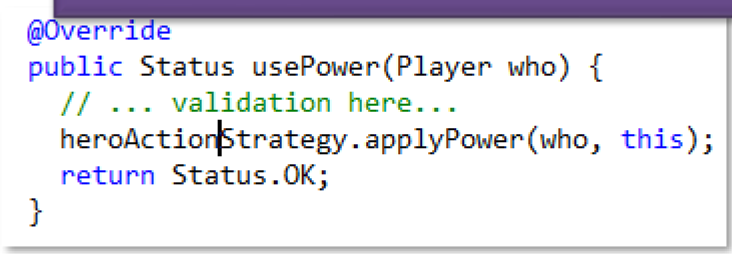
```java
public class ThaiDanishHeroActionStrategy implements HeroActionStrategy {
  @Override
  public void applyPower(Player who, StandardGame game) {
    String heroType = game.getHero(who).getType();
    if (heroType.equals(GameConstants.THAI_CHEF_HERO_TYPE)) {
      // Chili = damage opponent 2
      // ... 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

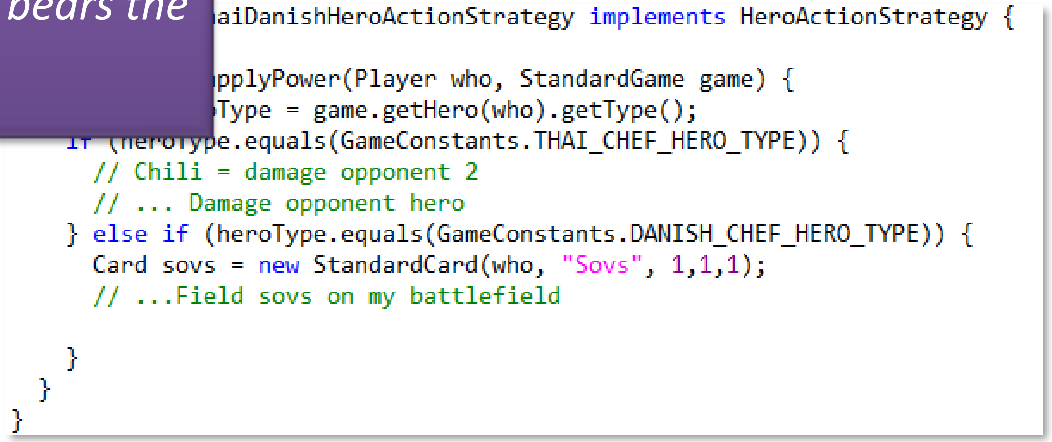
- Example:
  - GammaStone requirement: Heroes do different things

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!*





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

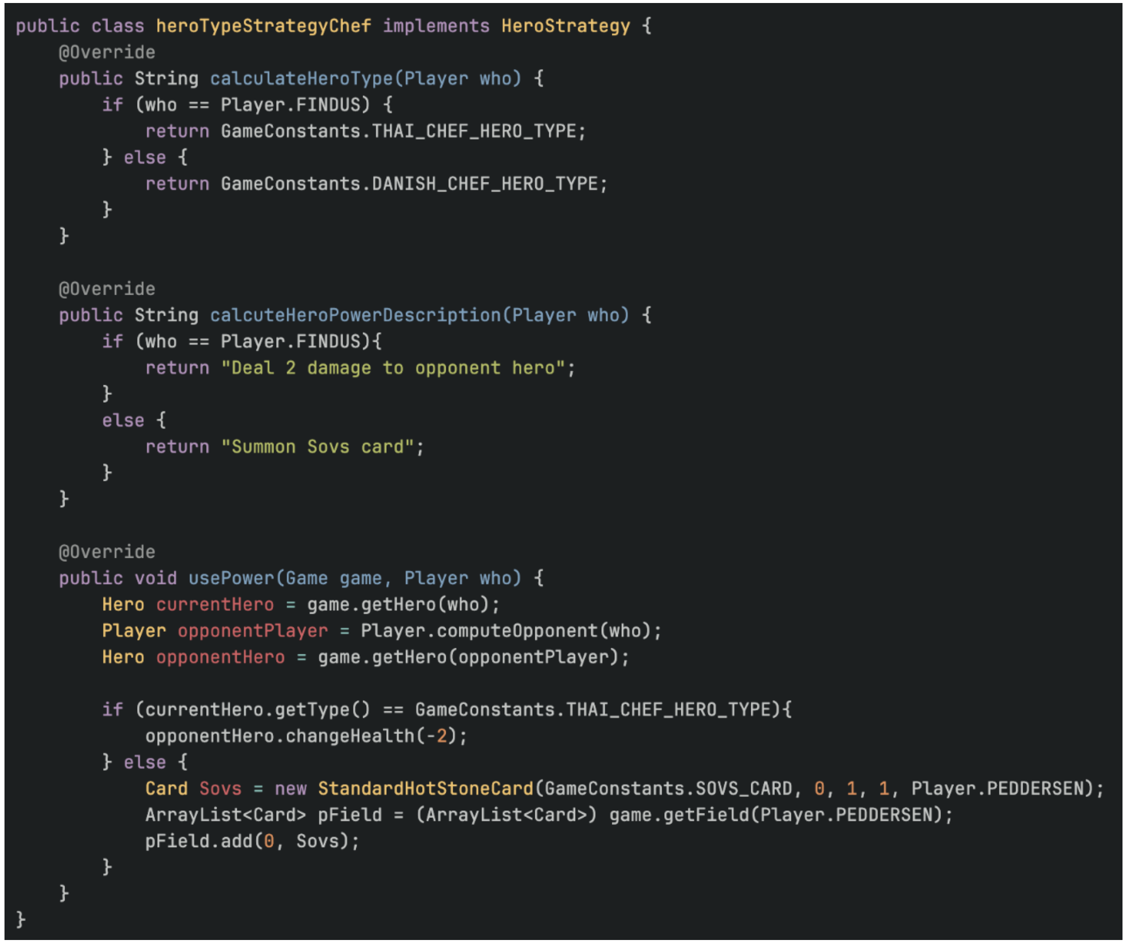


```
aiDanishHeroActionStrategy implements HeroActionStrategy {

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

  }
 }
}
```

Example

- From the trenches

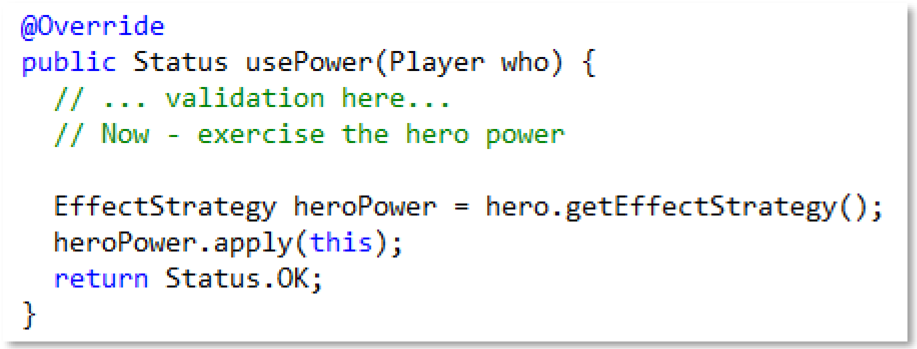
- But other issues
  - Coupling is too hard…



```java
public class heroTypeStrategyChef implements HeroStrategy {
    @Override
    public String calculateHeroType(Player who) {
        if (who == Player.FINDUS) {
            return GameConstants.THAI_CHEF_HERO_TYPE;
        } else {
            return GameConstants.DANISH_CHEF_HERO_TYPE;
        }
    }

    @Override
    public String calcuteHeroPowerDescription(Player who) {
        if (who == Player.FINDUS){
            return "Deal 2 damage to opponent hero";
        }
        else {
            return "Summon Sovs card";
        }
    }

    @Override
    public void usePower(Game game, Player who) {
        Hero currentHero = game.getHero(who);
        Player opponentPlayer = Player.computeOpponent(who);
        Hero opponentHero = game.getHero(opponentPlayer);

        if (currentHero.getType() == GameConstants.THAI_CHEF_HERO_TYPE){
            opponentHero.changeHealth(-2);
        } else {
            Card Sovs = new StandardHotStoneCard(GameConstants.SOVS_CARD, 0, 1, 1, Player.PEDDERSEN);
            ArrayList<Card> pField = (ArrayList<Card>) game.getField(Player.PEDDERSEN);
            pField.add(0, Sovs);
        }
    }
}
```

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…

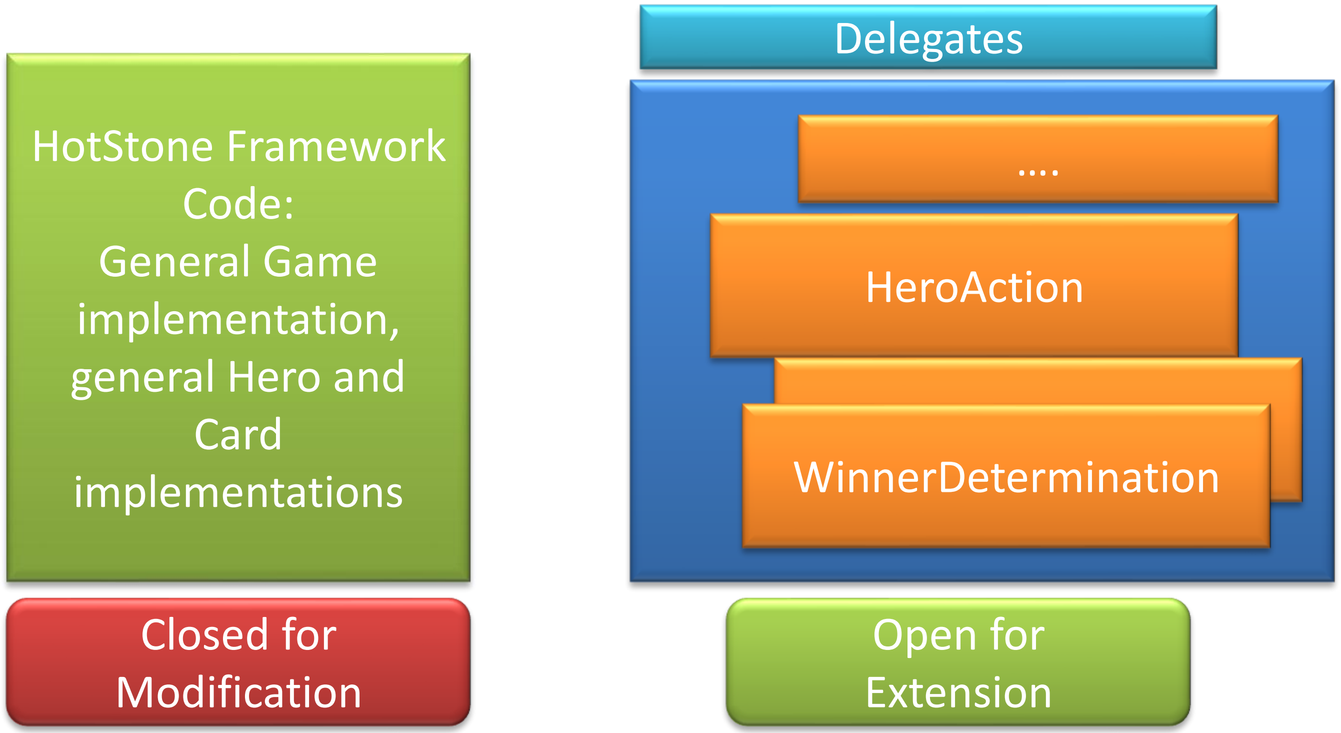


```
@Override
public Status usePower(Player who) {
  // ... validation here...
  // Now - exercise the hero power

  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)!*



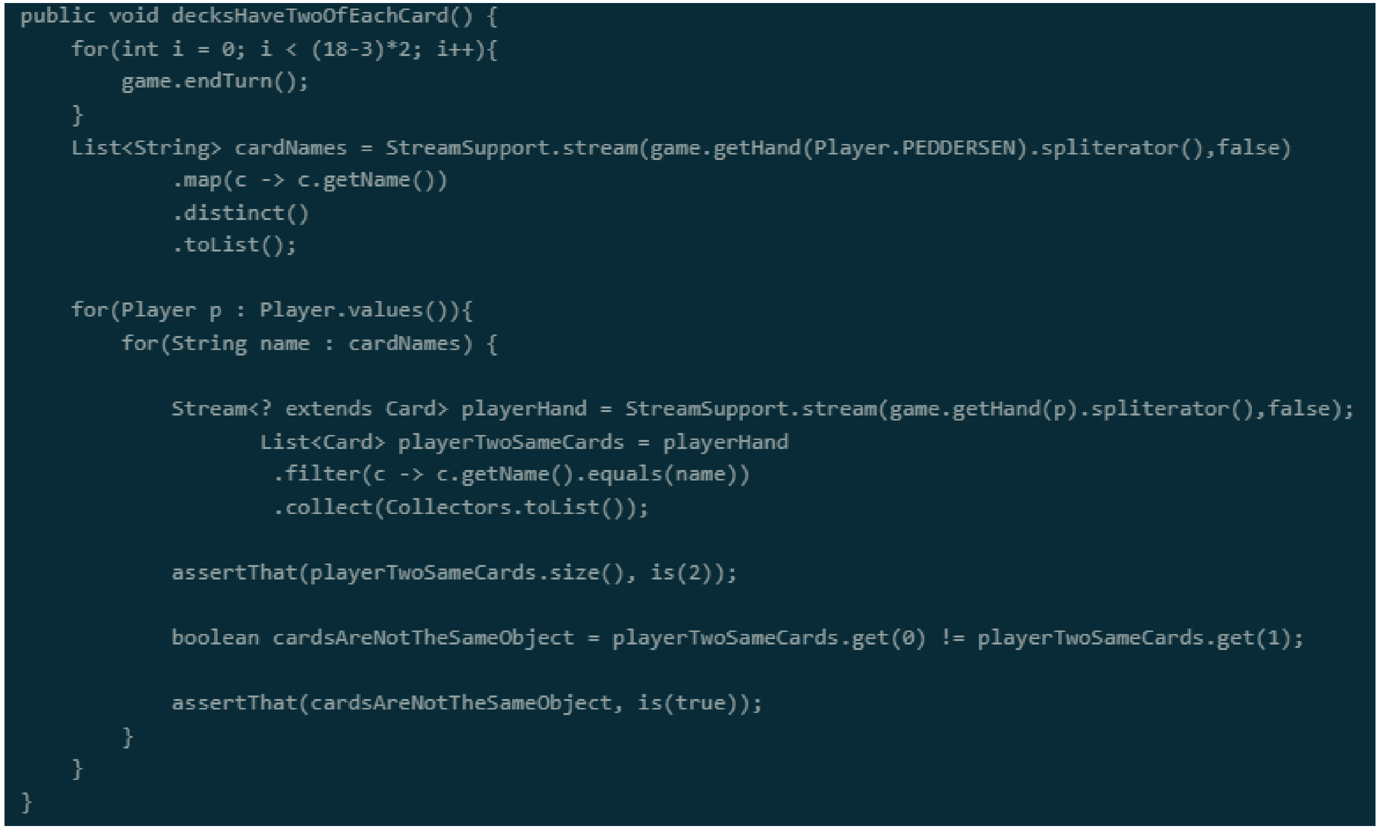
AARHUS UNIVERSITET

From Previous Years

Left for yourself to review…

From the Trenches

- Evident Test?

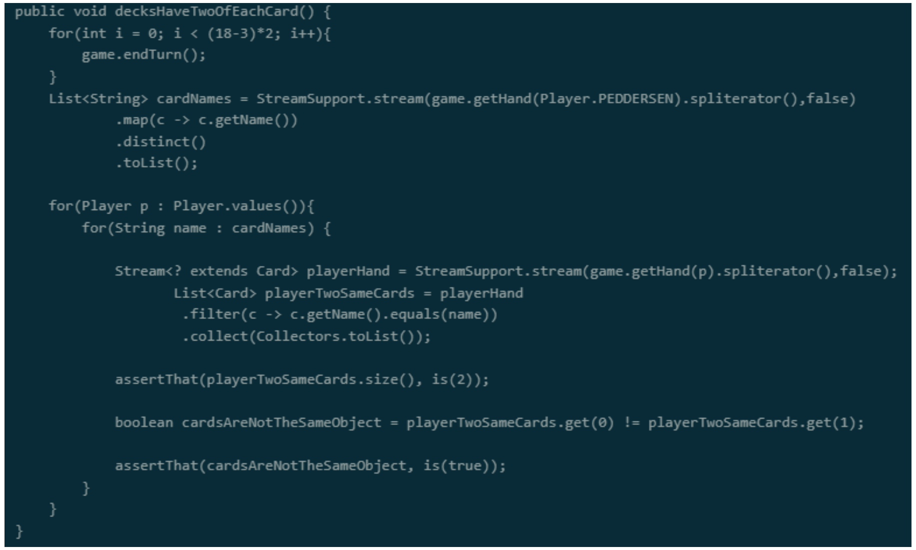


```
public void decksHaveTwoOfEachCard() {
    for(int i = 0; i < (18-3)*2; i++){
        game.endTurn();
    }
    List<String> cardNames = StreamSupport.stream(game.getHand(Player.PEDDERSEN).spliterator(),false)
            .map(c -> c.getName())
            .distinct()
            .toList();

    for(Player p : Player.values()){
        for(String name : cardNames) {

            Stream<? extends Card> playerHand = StreamSupport.stream(game.getHand(p).spliterator(),false);
                   List<Card> playerTwoSameCards = playerHand
                    .filter(c -> c.getName().equals(name))
                    .collect(Collectors.toList());

            assertThat(playerTwoSameCards.size(), is(2));

            boolean cardsAreNotTheSameObject = playerTwoSameCards.get(0) != playerTwoSameCards.get(1);

            assertThat(cardsAreNotTheSameObject, is(true));
        }
    }
}
```

- (No, it is not all bad, this, but…)

Recommendations

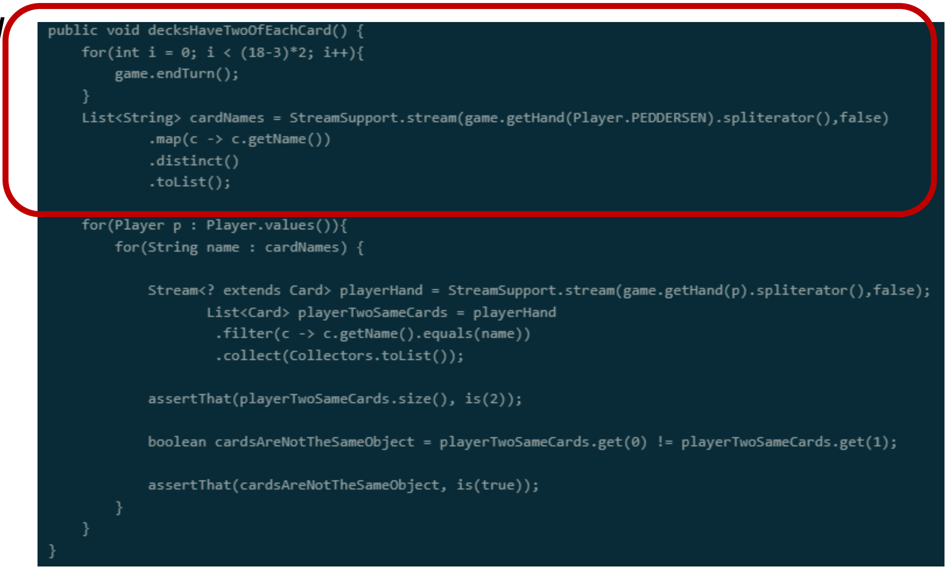
- Use GWT
  - Some parts are a bit obscure here
  - *// **Given** game has drawn all cards to the hand*
  - *// **When** I select all cards of given name for given player*
  - *// **Then** there are only two*
  - *// **Then** the two cards are*
    // unique references



```
public void decksHaveTwoOfEachCard() {
    for(int i = 0; i < (18-3)*2; i++){
        game.endTurn();
    }
    List<String> cardNames = StreamSupport.stream(game.getHand(Player.PEDDERSEN).spliterator(),false)
            .map(c -> c.getName())
            .distinct()
            .toList();

    for(Player p : Player.values()){
        for(String name : cardNames) {

            Stream<? extends Card> playerHand = StreamSupport.stream(game.getHand(p).spliterator(),false);
                   List<Card> playerTwoSameCards = playerHand
                    .filter(c -> c.getName().equals(name))
                    .collect(Collectors.toList());

            assertThat(playerTwoSameCards.size(), is(2));

            boolean cardsAreNotTheSameObject = playerTwoSameCards.get(0) != playerTwoSameCards.get(1);

            assertThat(cardsAreNotTheSameObject, is(true));
        }
    }
}
```

Recommendations

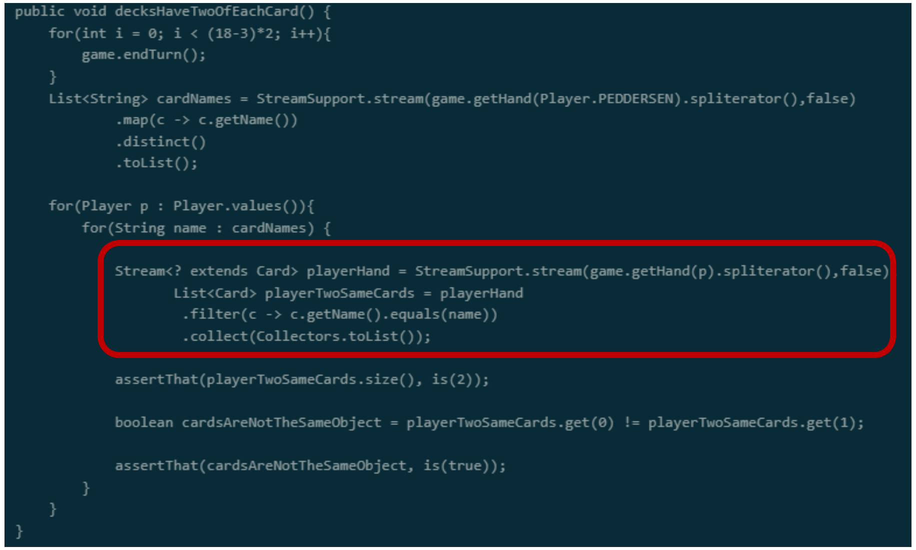
- If possible – use Unit Testing
  - Deck building strategy can just return a list of cards
    - And thus can be unit tested
      - Ala: create deck strategy, get deck, test it
      - *No Game instance involved*



```java
public void decksHaveTwoOfEachCard() {
    for(int i = 0; i < (18-3)*2; i++){
        game.endTurn();
    }
    List<String> cardNames = StreamSupport.stream(game.getHand(Player.PEDDERSEN).spliterator(),false)
            .map(c -> c.getName())
            .distinct()
            .toList();

    for(Player p : Player.values()){
        for(String name : cardNames) {

            Stream<? extends Card> playerHand = StreamSupport.stream(game.getHand(p).spliterator(),false);
                   List<Card> playerTwoSameCards = playerHand
                    .filter(c -> c.getName().equals(name))
                    .collect(Collectors.toList());

            assertThat(playerTwoSameCards.size(), is(2));

            boolean cardsAreNotTheSameObject = playerTwoSameCards.get(0) != playerTwoSameCards.get(1);

            assertThat(cardsAreNotTheSameObject, is(true));
        }
    }
}
```

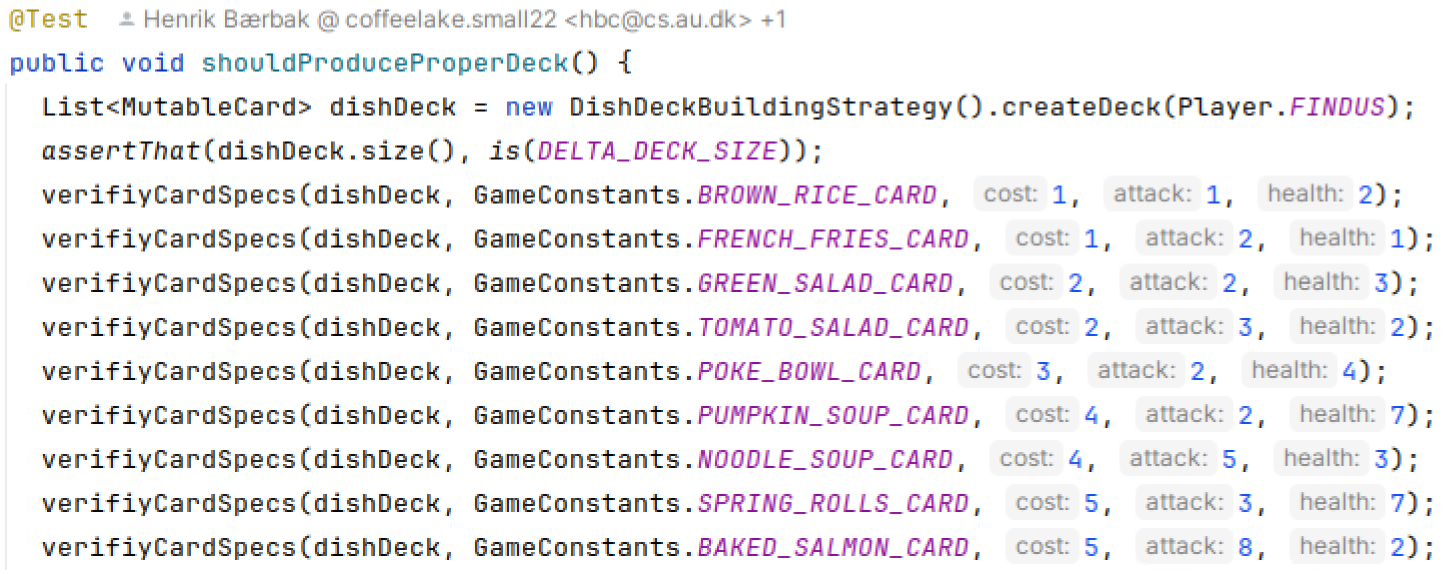
Recommendations

- Make private methods to do ‘One Level of Abstraction
  - private List<Card> getAllCardsOfPlayerWithName(Player who, String name);



```
public void decksHaveTwoOfEachCard() {
    for(int i = 0; i < (18-3)*2; i++){
        game.endTurn();
    }
    List<String> cardNames = StreamSupport.stream(game.getHand(Player.PEDDERSEN).spliterator(),false)
            .map(c -> c.getName())
            .distinct()
            .toList();

    for(Player p : Player.values()){
        for(String name : cardNames) {

            Stream<? extends Card> playerHand = StreamSupport.stream(game.getHand(p).spliterator(),false)
                   List<Card> playerTwoSameCards = playerHand
                    .filter(c -> c.getName().equals(name))
                    .collect(Collectors.toList());

            assertThat(playerTwoSameCards.size(), is(2));

            boolean cardsAreNotTheSameObject = playerTwoSameCards.get(0) != playerTwoSameCards.get(1);

            assertThat(cardsAreNotTheSameObject, is(true));
        }
    }
}
```

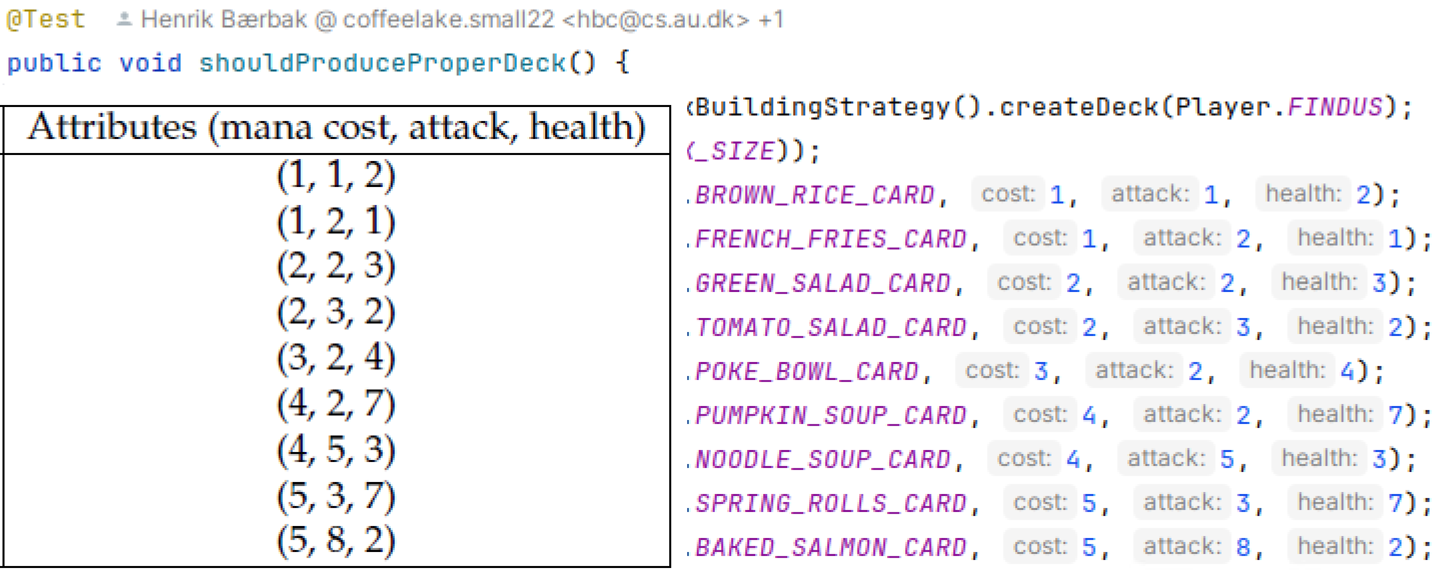
My Tests



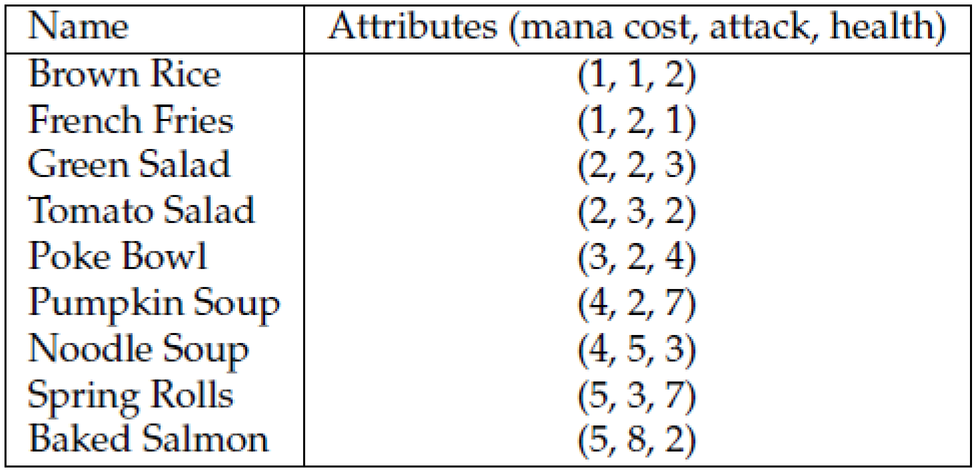
```
@Test   Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk> +1
public void shouldProduceProperDeck() {
  List<MutableCard> dishDeck = new DishDeckBuildingStrategy().createDeck(Player.FINDUS);
  assertThat(dishDeck.size(), is(DELTA_DECK_SIZE));
  verifiyCardSpecs(dishDeck, GameConstants.BROWN_RICE_CARD,  cost: 1,  attack: 1,  health: 2);
  verifiyCardSpecs(dishDeck, GameConstants.FRENCH_FRIES_CARD,  cost: 1,  attack: 2,  health: 1);
  verifiyCardSpecs(dishDeck, GameConstants.GREEN_SALAD_CARD,  cost: 2,  attack: 2,  health: 3);
  verifiyCardSpecs(dishDeck, GameConstants.TOMATO_SALAD_CARD,  cost: 2,  attack: 3,  health: 2);
  verifiyCardSpecs(dishDeck, GameConstants.POKE_BOWL_CARD,  cost: 3,  attack: 2,  health: 4);
  verifiyCardSpecs(dishDeck, GameConstants.PUMPKIN_SOUP_CARD,  cost: 4,  attack: 2,  health: 7);
  verifiyCardSpecs(dishDeck, GameConstants.NOODLE_SOUP_CARD,  cost: 4,  attack: 5,  health: 3);
  verifiyCardSpecs(dishDeck, GameConstants.SPRING_ROLLS_CARD,  cost: 5,  attack: 3,  health: 7);
  verifiyCardSpecs(dishDeck, GameConstants.BAKED_SALMON_CARD,  cost: 5,  attack: 8,  health: 2);
```

- Unit testing + private helper method.
- Note! This is not how the first test case looked like; I have iterated on this test to *clean up* and *make evident* and *do One Level Of Abstraction*

And note the ‘formatting’



```
@Test   Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk> +1
public void shouldProduceProperDeck() {
BuildingStrategy().createDeck(Player.FINDUS);
_SIZE));
.BROWN_RICE_CARD,  cost: 1,  attack: 1,  health: 2);
.FRENCH_FRIES_CARD,  cost: 1,  attack: 2,  health: 1);
.GREEN_SALAD_CARD,  cost: 2,  attack: 2,  health: 3);
.TOMATO_SALAD_CARD,  cost: 2,  attack: 3,  health: 2);
.POKE_BOWL_CARD,  cost: 3,  attack: 2,  health: 4);
.PUMPKIN_SOUP_CARD,  cost: 4,  attack: 2,  health: 7);
.NOODLE_SOUP_CARD,  cost: 4,  attack: 5,  health: 3);
.SPRING_ROLLS_CARD,  cost: 5,  attack: 3,  health: 7);
.BAKED_SALMON_CARD,  cost: 5,  attack: 8,  health: 2);
```

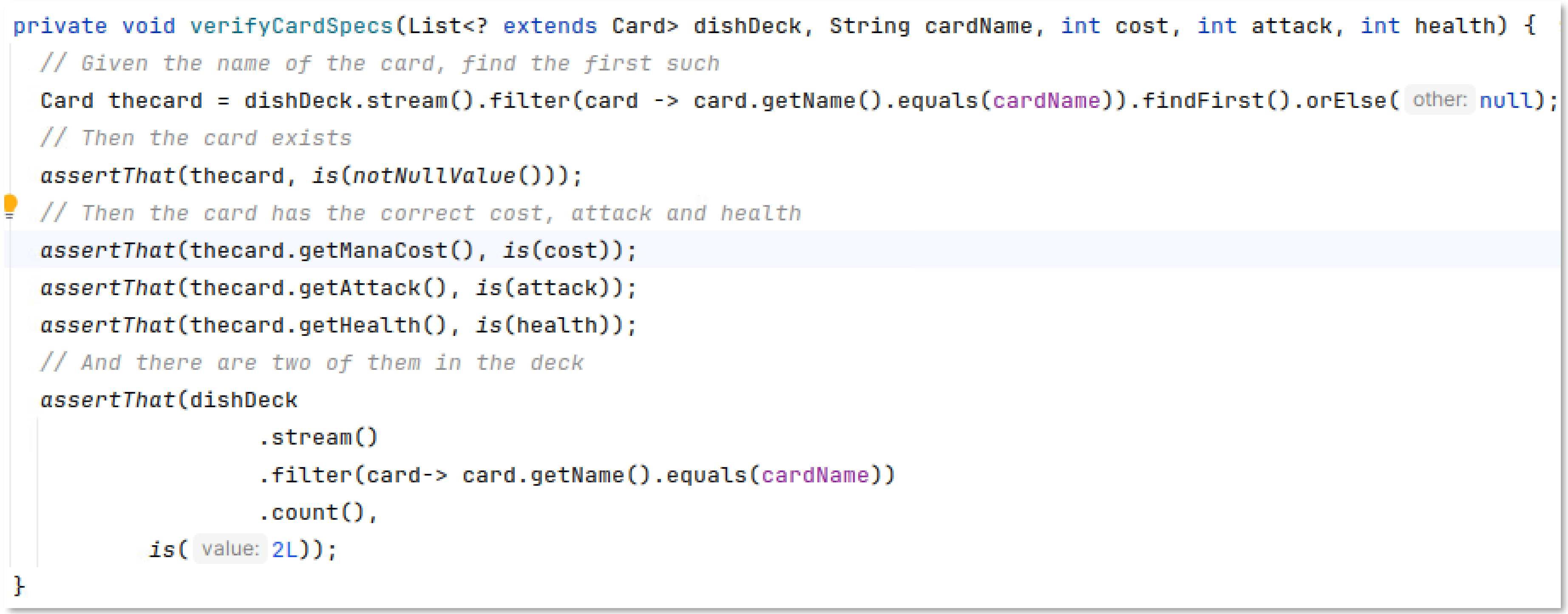


Name	Attributes (mana cost, attack, health)
Brown Rice	(1, 1, 2)
French Fries	(1, 2, 1)
Green Salad	(2, 2, 3)
Tomato Salad	(2, 3, 2)
Poke Bowl	(3, 2, 4)
Pumpkin Soup	(4, 2, 7)
Noodle Soup	(4, 5, 3)
Spring Rolls	(5, 3, 7)
Baked Salmon	(5, 8, 2)

- Note! The visual layout of the test closely matches that of the tabular format, making x-checking it easy
  - *Analyzability*

The Verify method

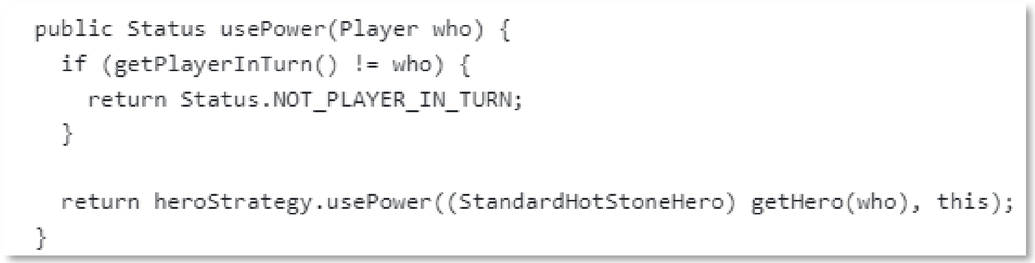
- Looks like



```
private void verifyCardSpecs(List<? extends Card> dishDeck, String cardName, int cost, int attack, int health) {
  // Given the name of the card, find the first such
  Card thecard = dishDeck.stream().filter(card -> card.getName().equals(cardName)).findFirst().orElse( other: null);
  // Then the card exists
  assertThat(thecard, is(notNullValue()));
  // Then the card has the correct cost, attack and health
  assertThat(thecard.getManaCost(), is(cost));
  assertThat(thecard.getAttack(), is(attack));
  assertThat(thecard.getHealth(), is(health));
  // And there are two of them in the deck
  assertThat(dishDeck
                  .stream()
                  .filter(card-> card.getName().equals(cardName))
                  .count(),
          is( value: 2L));
}
```

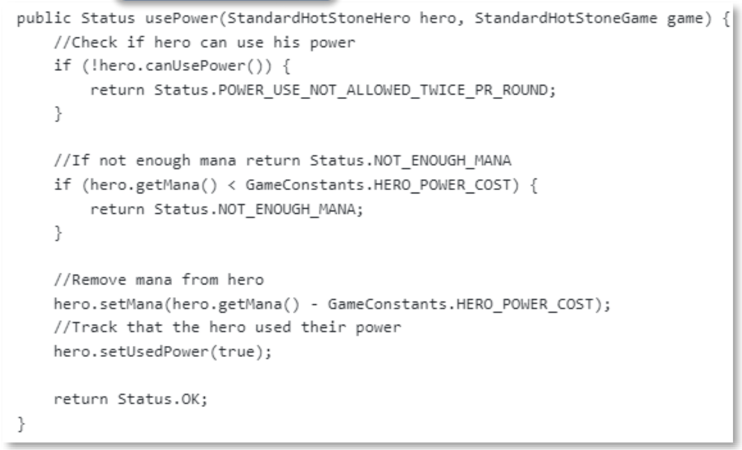
Single Responsibility

- Who is responsible for validating game rules? **Game!**
- What is wrong?



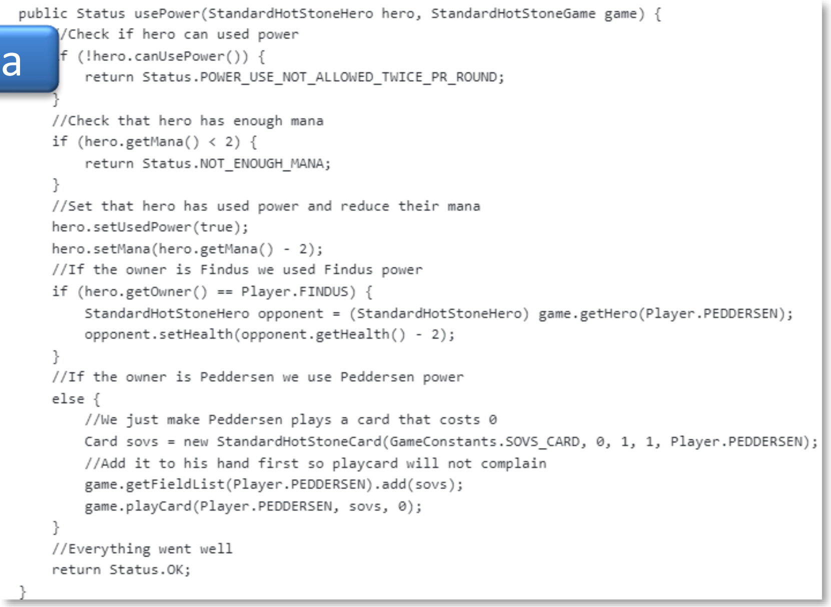
```
public Status usePower(Player who) {
  if (getPlayerInTurn() != who) {
    return Status.NOT_PLAYER_IN_TURN;
  }

  return heroStrategy.usePower((StandardHotStoneHero) getHero(who), this);
}
```

Alpha



```
public Status usePower(StandardHotStoneHero hero, StandardHotStoneGame game) {
    //Check if hero can use his power
    if (!hero.canUsePower()) {
        return Status.POWER_USE_NOT_ALLOWED_TWICE_PR_ROUND;
    }

    //If not enough mana return Status.NOT_ENOUGH_MANA
    if (hero.getMana() < GameConstants.HERO_POWER_COST) {
        return Status.NOT_ENOUGH_MANA;
    }

    //Remove mana from hero
    hero.setMana(hero.getMana() - GameConstants.HERO_POWER_COST);
    //Track that the hero used their power
    hero.setUsedPower(true);

    return Status.OK;
}
```

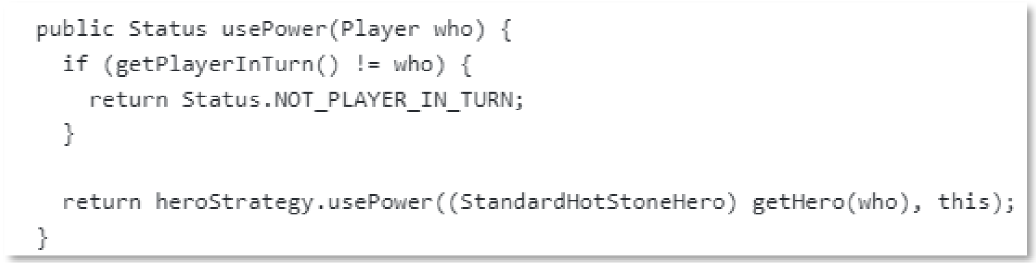
Gamma



```
public Status usePower(StandardHotStoneHero hero, StandardHotStoneGame game) {
    /Check if hero can used power
    f (!hero.canUsePower()) {
        return Status.POWER_USE_NOT_ALLOWED_TWICE_PR_ROUND;
    }
    //Check that hero has enough mana
    if (hero.getMana() < 2) {
        return Status.NOT_ENOUGH_MANA;
    }
    //Set that hero has used power and reduce their mana
    hero.setUsedPower(true);
    hero.setMana(hero.getMana() - 2);
    //If the owner is Findus we used Findus power
    if (hero.getOwner() == Player.FINDUS) {
        StandardHotStoneHero opponent = (StandardHotStoneHero) game.getHero(Player.PEDDERSEN);
        opponent.setHealth(opponent.getHealth() - 2);
    }
    //If the owner is Peddersen we use Peddersen power
    else {
        //We just make Peddersen plays a card that costs 0
        Card sovs = new StandardHotStoneCard(GameConstants.SOVS_CARD, 0, 1, 1, Player.PEDDERSEN);
        //Add it to his hand first so playcard will not complain
        game.getFieldList(Player.PEDDERSEN).add(sovs);
        game.playCard(Player.PEDDERSEN, sovs, 0);
    }
    //Everything went well
    return Status.OK;
}
```

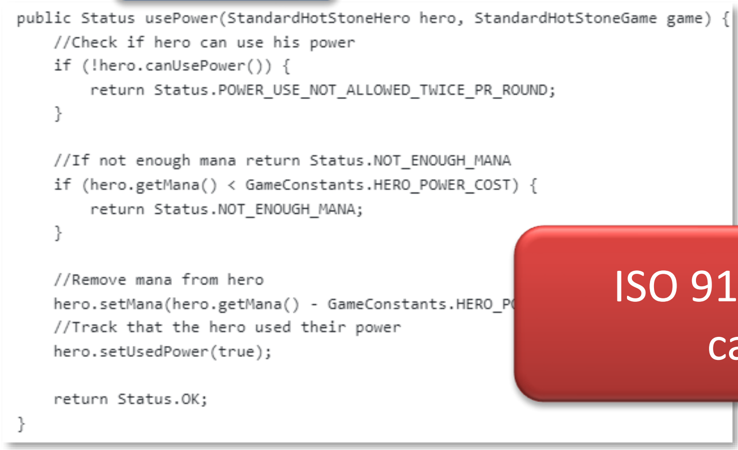
Single Responsibility

- Here: Game does half, Strategy does half of validation ☹
- Result
  - Code duplication
  - Validation in two places



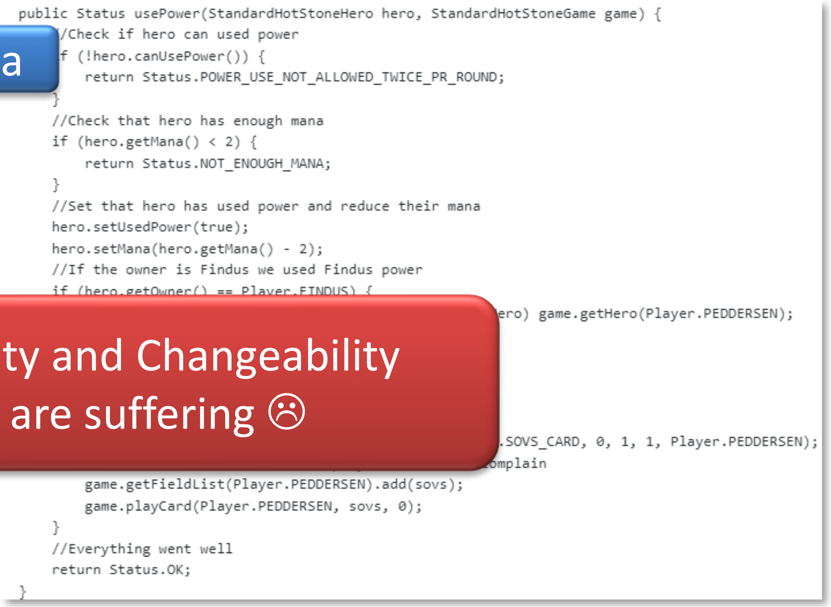
```
public Status usePower(Player who) {
  if (getPlayerInTurn() != who) {
    return Status.NOT_PLAYER_IN_TURN;
  }

  return heroStrategy.usePower((StandardHotStoneHero) getHero(who), this);
}
```

Alpha



```
public Status usePower(StandardHotStoneHero hero, StandardHotStoneGame game) {
    //Check if hero can use his power
    if (!hero.canUsePower()) {
        return Status.POWER_USE_NOT_ALLOWED_TWICE_PR_ROUND;
    }

    //If not enough mana return Status.NOT_ENOUGH_MANA
    if (hero.getMana() < GameConstants.HERO_POWER_COST) {
        return Status.NOT_ENOUGH_MANA;
    }

    //Remove mana from hero
    hero.setMana(hero.getMana() - GameConstants.HERO_P
    //Track that the hero used their power
    hero.setUsedPower(true);

    return Status.OK;
}
```

Gamma

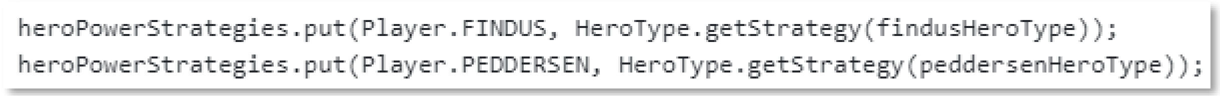


```
public Status usePower(StandardHotStoneHero hero, StandardHotStoneGame game) {
    /Check if hero can used power
    f (!hero.canUsePower()) {
        return Status.POWER_USE_NOT_ALLOWED_TWICE_PR_ROUND;
    }
    //Check that hero has enough mana
    if (hero.getMana() < 2) {
        return Status.NOT_ENOUGH_MANA;
    }
    //Set that hero has used power and reduce their mana
    hero.setUsedPower(true);
    hero.setMana(hero.getMana() - 2);
    //If the owner is Findus we used Findus power
    if (hero.getOwner() == Player.FINDUS) {
                                   ero) game.getHero(Player.PEDDERSEN);



                                   .SOVS_CARD, 0, 1, 1, Player.PEDDERSEN);
                                  omplain
        game.getFieldList(Player.PEDDERSEN).add(sovs);
        game.playCard(Player.PEDDERSEN, sovs, 0);
    }
    //Everything went well
    return Status.OK;
}
```

ISO 9126: Stability and Changeability capabilities are suffering ☹

Another Attempt

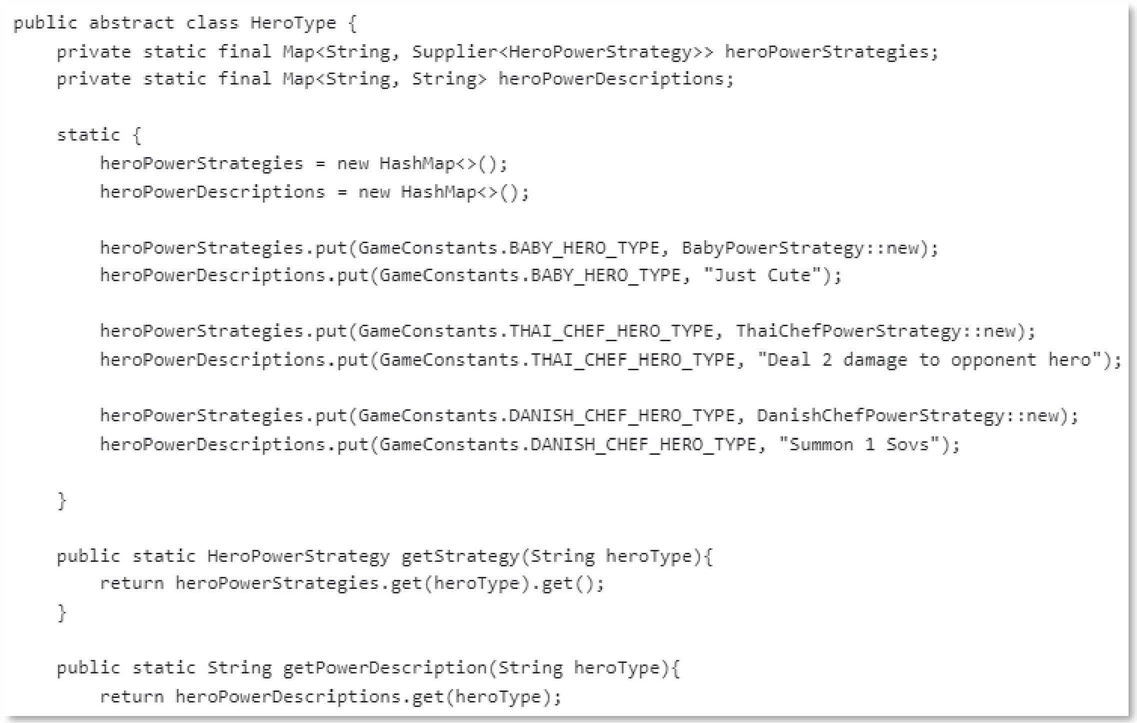
- Game has this ‘configuration’ of heroes’ Power…



```
heroPowerStrategies.put(Player.FINDUS, HeroType.getStrategy(findusHeroType));
heroPowerStrategies.put(Player.PEDDERSEN, HeroType.getStrategy(peddersenHeroType));
```

- Drawing from a single ‘HeroType’

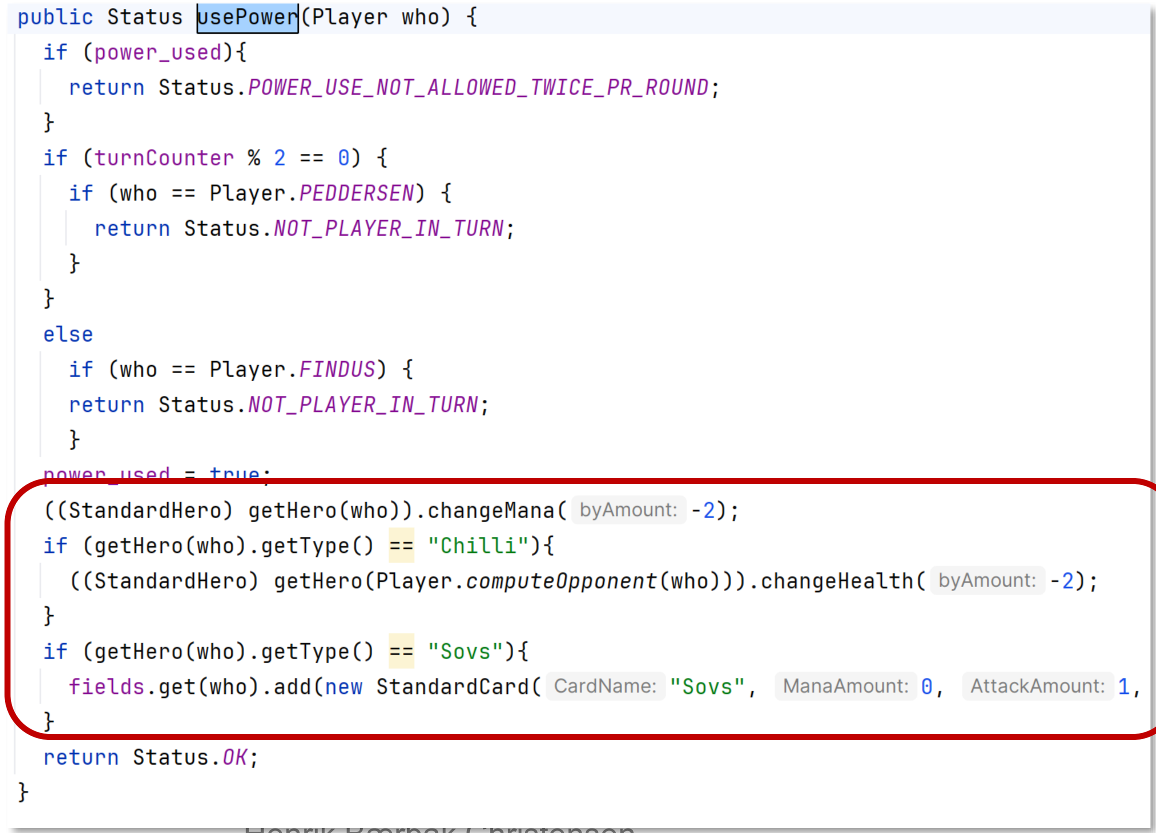
- Pro/Cons?



```
public abstract class HeroType {
    private static final Map<String, Supplier<HeroPowerStrategy>> heroPowerStrategies;
    private static final Map<String, String> heroPowerDescriptions;

    static {
        heroPowerStrategies = new HashMap<>();
        heroPowerDescriptions = new HashMap<>();

        heroPowerStrategies.put(GameConstants.BABY_HERO_TYPE, BabyPowerStrategy::new);
        heroPowerDescriptions.put(GameConstants.BABY_HERO_TYPE, "Just Cute");

        heroPowerStrategies.put(GameConstants.THAI_CHEF_HERO_TYPE, ThaiChefPowerStrategy::new);
        heroPowerDescriptions.put(GameConstants.THAI_CHEF_HERO_TYPE, "Deal 2 damage to opponent hero");

        heroPowerStrategies.put(GameConstants.DANISH_CHEF_HERO_TYPE, DanishChefPowerStrategy::new);
        heroPowerDescriptions.put(GameConstants.DANISH_CHEF_HERO_TYPE, "Summon 1 Sovs");

    }

    public static HeroPowerStrategy getStrategy(String heroType){
        return heroPowerStrategies.get(heroType).get();
    }

    public static String getPowerDescription(String heroType){
        return heroPowerDescriptions.get(heroType);
```

Variability Technique Used?

- What about this one? From the ‘StandardGame’



```
public Status usePower(Player who) {
  if (power_used){
    return Status.POWER_USE_NOT_ALLOWED_TWICE_PR_ROUND;
  }
  if (turnCounter % 2 == 0) {
    if (who == Player.PEDDERSEN) {
      return Status.NOT_PLAYER_IN_TURN;
    }
  }
  else
    if (who == Player.FINDUS) {
    return Status.NOT_PLAYER_IN_TURN;
    }
  power_used = true;
  ((StandardHero) getHero(who)).changeMana( byAmount: -2);
  if (getHero(who).getType() == "Chilli"){
    ((StandardHero) getHero(Player.computeOpponent(who))).changeHealth( byAmount: -2);
  }
  if (getHero(who).getType() == "Sovs"){
    fields.get(who).add(new StandardCard( CardName: "Sovs",  ManaAmount: 0,  AttackAmount: 1,
  }
  return Status.OK;
}
```