AARHUS UNIVERSITET

Software Engineering and Architecture

Mandatory Reflections

Congrats !

- You have covered a lot of ground!
- There are only *three out of nine* exam topics that we still have to cover!
  – And one of these (frameworks) is actually more or less covered…

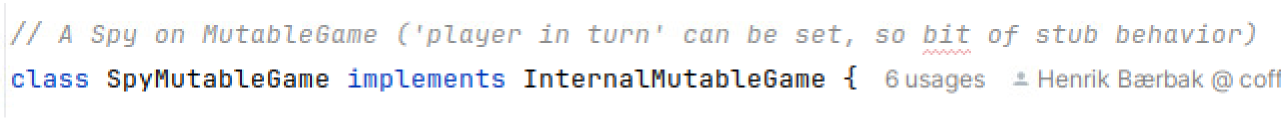
StudyCafe Experience

- Private Interface?
  - Perhaps should be renamed to ‘friend interface’
    - Your friends are allowed to ‘make more private requests’ than a stranger is…

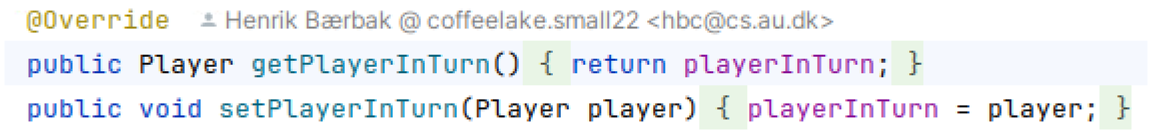
- So, a Card should expose a ‘private interface’ to its good friend, the Game,
  - “Dear Game, you are allowed to change my health”
  - Whereas, a stranger (the UI) is not…
  - Interface MutableCard extends Card { void changeHealth(…); }

Stubs vrs Spies

- Stub/Spy are categories of test doubles
  - I.e. they are *roles* a double may play
  - And it may play **both** and often do!
- Indeed – to make a MutableGameSpy work so it can help us test the card effects, it *needs* to feed 'stub' indirect input.
  - Example: Brown Rice effect likely needs to call 'game.getPlayerInTurn()' to compute the opponent…
- My 'spy':



```
// A Spy on MutableGame ('player in turn' can be set, so bit of stub behavior)
class SpyMutableGame implements InternalMutableGame {  6 usages  Henrik Bærbak @ coff
```



```
@Override   Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
public Player getPlayerInTurn() { return playerInTurn; }
public void setPlayerInTurn(Player player) { playerInTurn = player; }
```

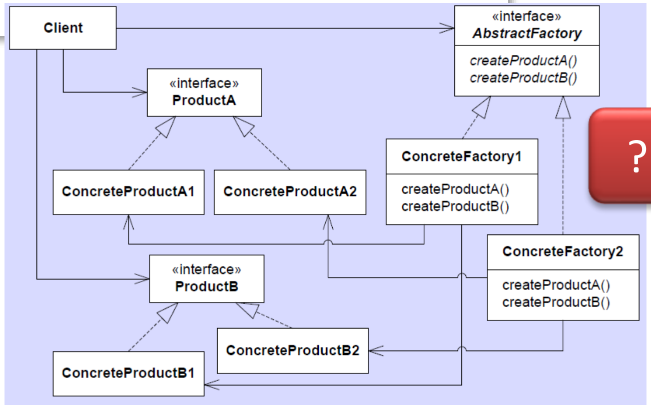
AARHUS UNIVERSITET

Static Fields?

```
// HotstoneFactory.java
public class HotstoneFactory {
    public final ManaStrategy mana;
    public final DeckStrategy deck;
    public final HeroSetupStrategy heroes;
    public final FatigueStrategy fatigue;
    public final WinConditionStrategy win;
    public final HeroPowerStrategy powers; // ny


    public HotstoneFactory(

            ManaStrategy mana,
            DeckStrategy deck,
            HeroSetupStrategy heroes,
            FatigueStrategy fatigue,
            WinConditionStrategy win,
            HeroPowerStrategy powers)
    {
        this.mana = mana;
        this.deck = deck;
        this.heroes = heroes;
        this.fatigue = fatigue;
        this.win = win;
        this.powers = powers;
    }

    public HeroPowerStrategy getHeroPowerStrategy() {
        return powers;
    }
```

```
// BetaFactory.java
public class BetaFactory {
    public static final HotstoneFactory BETA = new HotstoneFactory(
            new GrowingByRound(),
            new DefaultDecks(),
            new BabyHeroesBoth(),
            new BetaFatigue2(),
            new ZeroHealthWins(),
            new NoHeroPowers()
    );
}
```

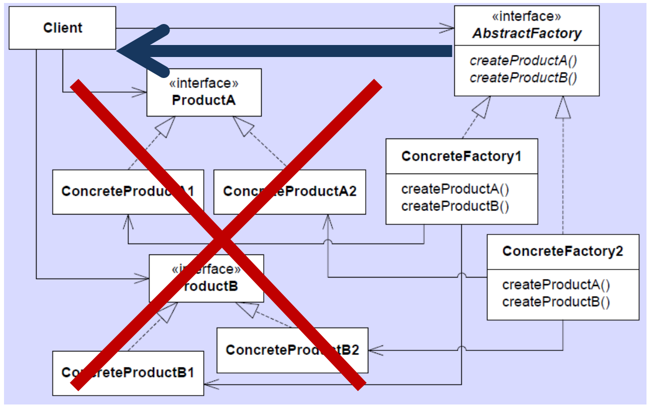
```
// StandardGame.java
  public StandardHotStoneGame(HotstoneFactory factory) { ... }
}
```



?

GameFactory?

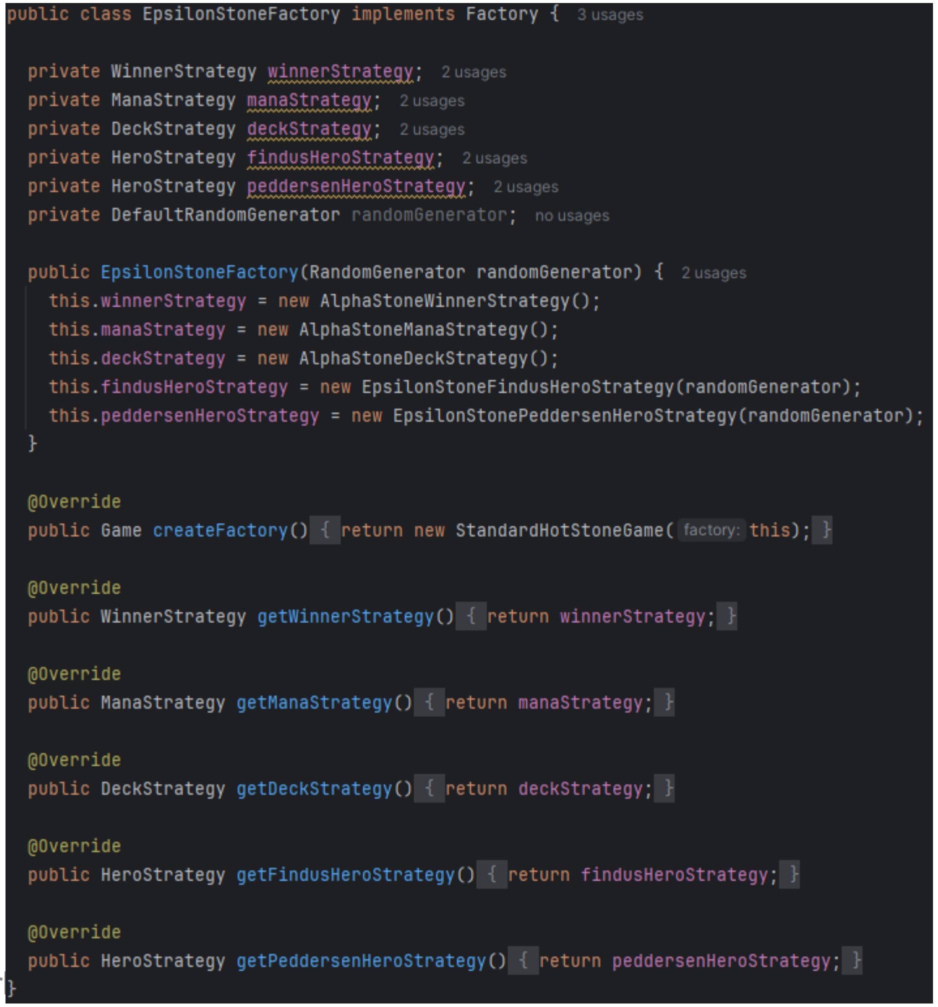
- Factories do *not* create Games !
  - In the UML, it translates to **Factory Create Client!**



```
public class AlphaFactory implements HotStoneFactory {
    @Override
    public StandardHotStoneGame createHotStoneGame() {

        WinCon winConAlpha = new AlphaWin();
        ManaCon manaConAlpha = new AlphaManaCon();
        SOGCon sogConAlpha = new AlphaSOGCon();
        InitializeHeroCon iniHeroConAlpha = new AlphaHeroCon();
        IniCardCon iniCardConAlpha = new AlphaIniCardCon();
        HeroPowerCon heroPowerConAlpha = new AlphaHeroPowerCon();


        return new StandardHotStoneGame(
                Player.FINDUS,
                winConAlpha,
                manaConAlpha,
                sogConAlpha,
                iniHeroConAlpha,
                iniCardConAlpha,
                heroPowerConAlpha
        );
    }
}
```

And a Mix

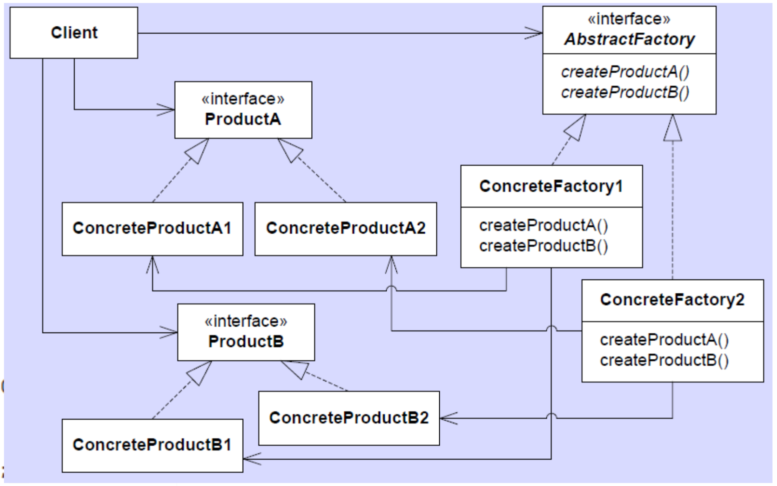
- A bit of both?
- 'createFactory()' return game?
- "This is an apple"

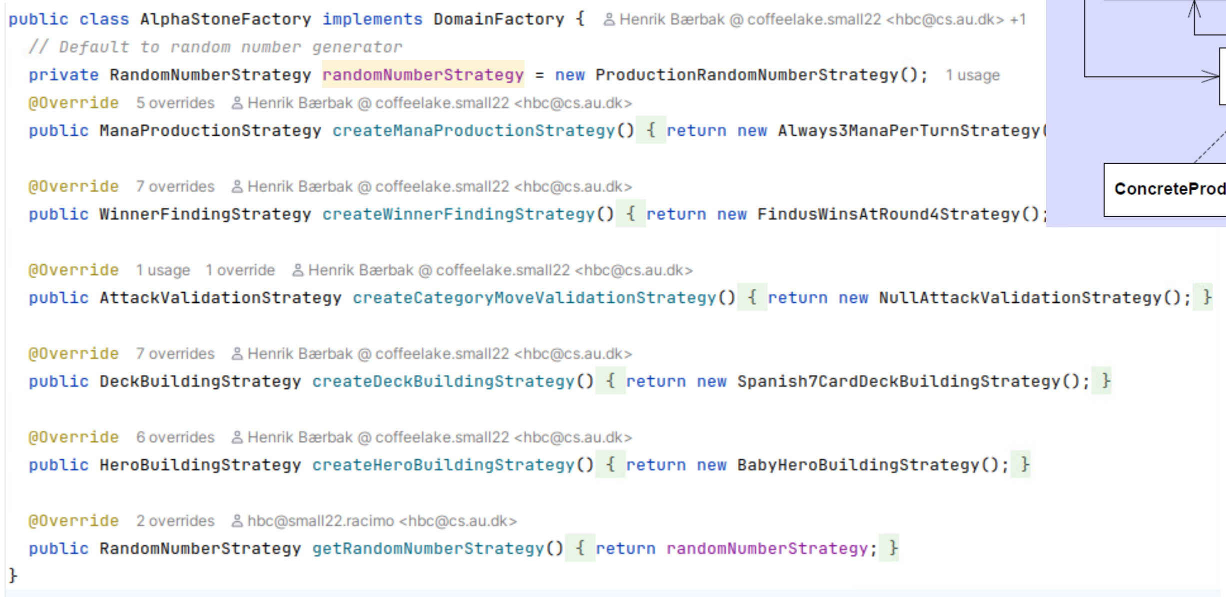


```
public class EpsilonStoneFactory implements Factory {  3 usages

  private WinnerStrategy winnerStrategy;  2 usages
  private ManaStrategy manaStrategy;  2 usages
  private DeckStrategy deckStrategy;  2 usages
  private HeroStrategy findusHeroStrategy;  2 usages
  private HeroStrategy peddersenHeroStrategy;  2 usages
  private DefaultRandomGenerator randomGenerator;  no usages

  public EpsilonStoneFactory(RandomGenerator randomGenerator) {  2 usages
    this.winnerStrategy = new AlphaStoneWinnerStrategy();
    this.manaStrategy = new AlphaStoneManaStrategy();
    this.deckStrategy = new AlphaStoneDeckStrategy();
    this.findusHeroStrategy = new EpsilonStoneFindusHeroStrategy(randomGenerator);
    this.peddersenHeroStrategy = new EpsilonStonePeddersenHeroStrategy(randomGenerator);
  }

  @Override
  public Game createFactory() { return new StandardHotStoneGame( factory: this); }

  @Override
  public WinnerStrategy getWinnerStrategy() { return winnerStrategy; }

  @Override
  public ManaStrategy getManaStrategy() { return manaStrategy; }

  @Override
  public DeckStrategy getDeckStrategy() { return deckStrategy; }

  @Override
  public HeroStrategy getFindusHeroStrategy() { return findusHeroStrategy; }

  @Override
  public HeroStrategy getPeddersenHeroStrategy() { return peddersenHeroStrategy; }
}
```

ProductFactory!

- Factories creates the delegates that client uses

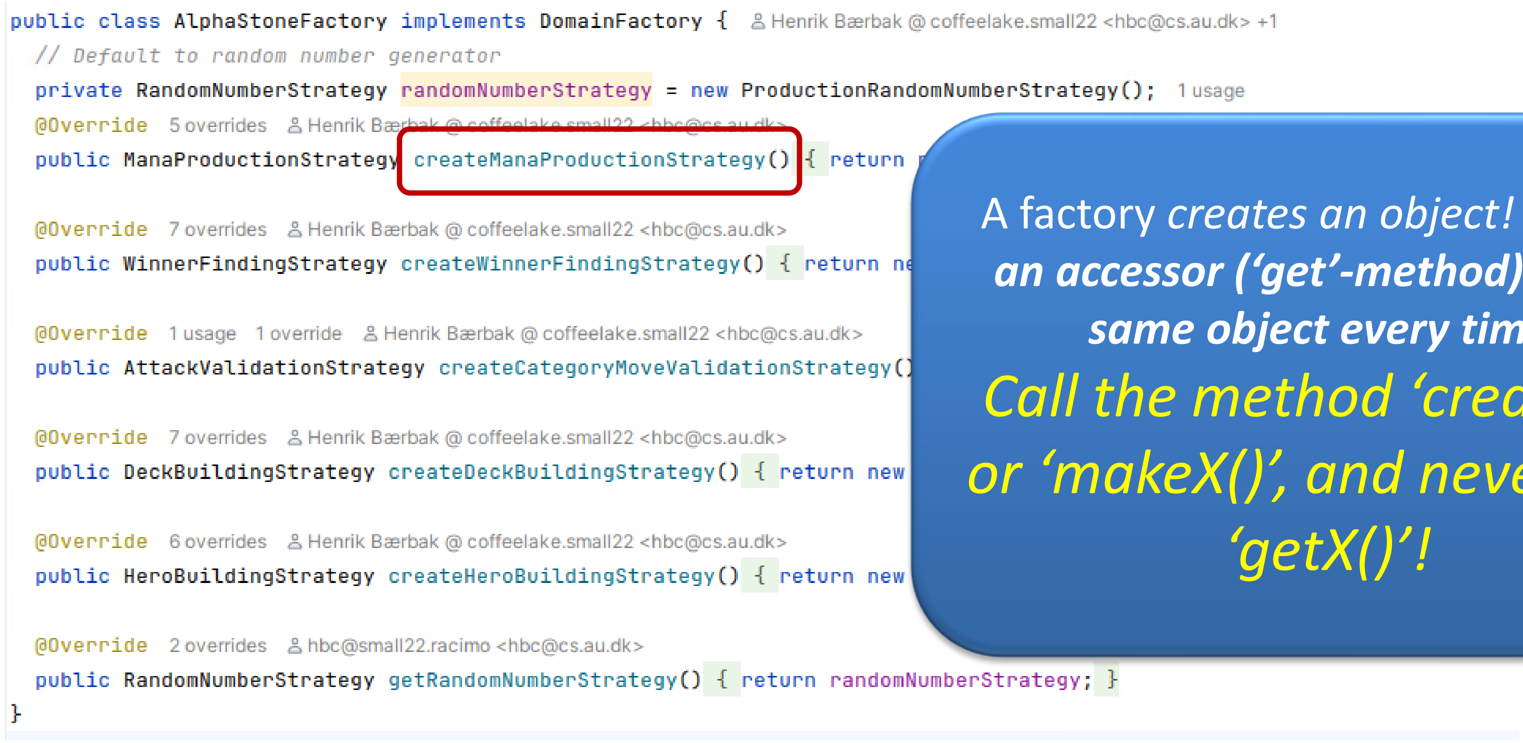




```
public class AlphaStoneFactory implements DomainFactory {   Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk> +1
  // Default to random number generator
  private RandomNumberStrategy randomNumberStrategy = new ProductionRandomNumberStrategy();   1 usage
  @Override   5 overrides   Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
  public ManaProductionStrategy createManaProductionStrategy() { return new Always3ManaPerTurnStrategy(

  @Override   7 overrides   Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
  public WinnerFindingStrategy createWinnerFindingStrategy() { return new FindusWinsAtRound4Strategy();

  @Override   1 usage   1 override   Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
  public AttackValidationStrategy createCategoryMoveValidationStrategy() { return new NullAttackValidationStrategy(); }

  @Override   7 overrides   Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
  public DeckBuildingStrategy createDeckBuildingStrategy() { return new Spanish7CardDeckBuildingStrategy(); }

  @Override   6 overrides   Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
  public HeroBuildingStrategy createHeroBuildingStrategy() { return new BabyHeroBuildingStrategy(); }

  @Override   2 overrides   hbc@small22.racimo <hbc@cs.au.dk>
  public RandomNumberStrategy getRandomNumberStrategy() { return randomNumberStrategy; }
}
```

Factories *Create* stuff

- Important naming of methods



```
public class AlphaStoneFactory implements DomainFactory {  Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk> +1
  // Default to random number generator
  private RandomNumberStrategy randomNumberStrategy = new ProductionRandomNumberStrategy();  1 usage
  @Override  5 overrides  Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
  public ManaProductionStrategy createManaProductionStrategy() { return

  @Override  7 overrides  Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
  public WinnerFindingStrategy createWinnerFindingStrategy() { return ne

  @Override  1 usage  1 override  Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
  public AttackValidationStrategy createCategoryMoveValidationStrategy()

  @Override  7 overrides  Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
  public DeckBuildingStrategy createDeckBuildingStrategy() { return new

  @Override  6 overrides  Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
  public HeroBuildingStrategy createHeroBuildingStrategy() { return new

  @Override  2 overrides  hbc@small22.racimo <hbc@cs.au.dk>
  public RandomNumberStrategy getRandomNumberStrategy() { return randomNumberStrategy; }
}
```

A factory *creates an object!* ***It is not an accessor ('get'-method) on the same object every time!***

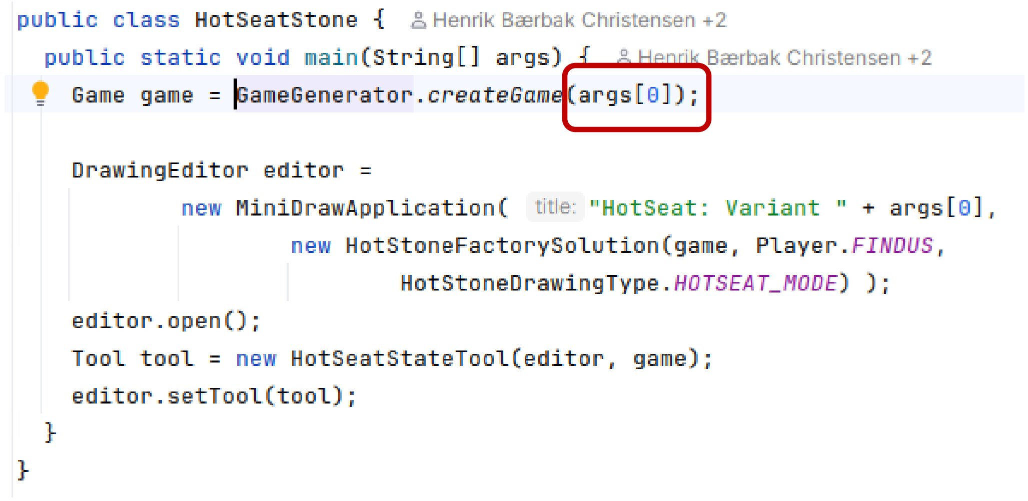
Call the method 'createX()' or 'makeX()', and never ever 'getX()'!

AARHUS UNIVERSITET

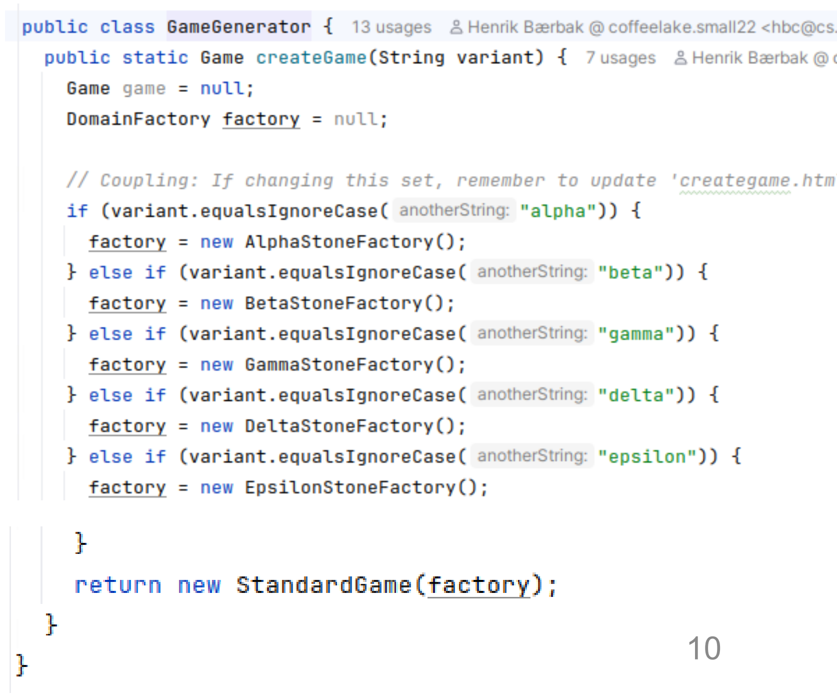
Main Method?

- A ‘switch on a string’ is needed in Main():
  - ‘gradle hotstone –Pvariant=alpha’

Putting that switch in its own abstraction *does* make sense!



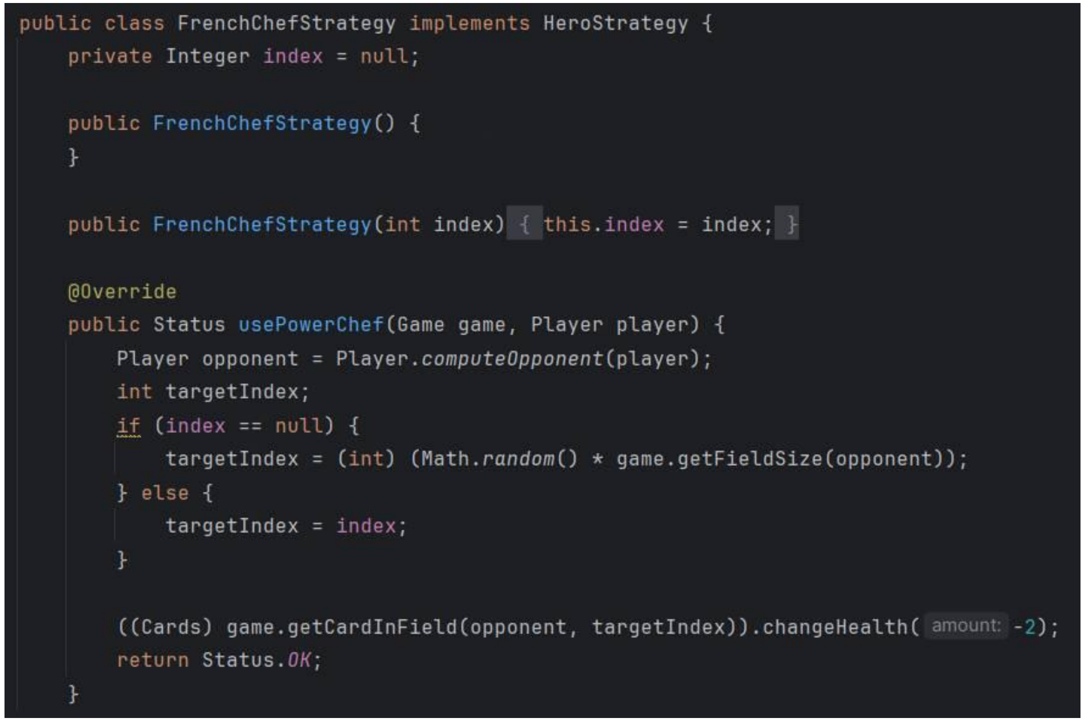
```
public class HotSeatStone {   Henrik Bærbak Christensen +2
  public static void main(String[] args) {   Henrik Bærbak Christensen +2
    Game game = GameGenerator.createGame(args[0]);

    DrawingEditor editor =
            new MiniDrawApplication( title: "HotSeat: Variant " + args[0],
                    new HotStoneFactorySolution(game, Player.FINDUS,
                            HotStoneDrawingType.HOTSEAT_MODE) );
    editor.open();
    Tool tool = new HotSeatStateTool(editor, game);
    editor.setTool(tool);
  }
}
```



```
public class GameGenerator {  13 usages   Henrik Bærbak @ coffeelake.small22 <hbc@cs.
  public static Game createGame(String variant) {  7 usages   Henrik Bærbak @ c
    Game game = null;
    DomainFactory factory = null;

    // Coupling: If changing this set, remember to update 'creategame.htm
    if (variant.equalsIgnoreCase( anotherString: "alpha")) {
      factory = new AlphaStoneFactory();
    } else if (variant.equalsIgnoreCase( anotherString: "beta")) {
      factory = new BetaStoneFactory();
    } else if (variant.equalsIgnoreCase( anotherString: "gamma")) {
      factory = new GammaStoneFactory();
    } else if (variant.equalsIgnoreCase( anotherString: "delta")) {
      factory = new DeltaStoneFactory();
    } else if (variant.equalsIgnoreCase( anotherString: "epsilon")) {
      factory = new EpsilonStoneFactory();

    }
    return new StandardGame(factory);
  }
}
```

No Doubles in Production

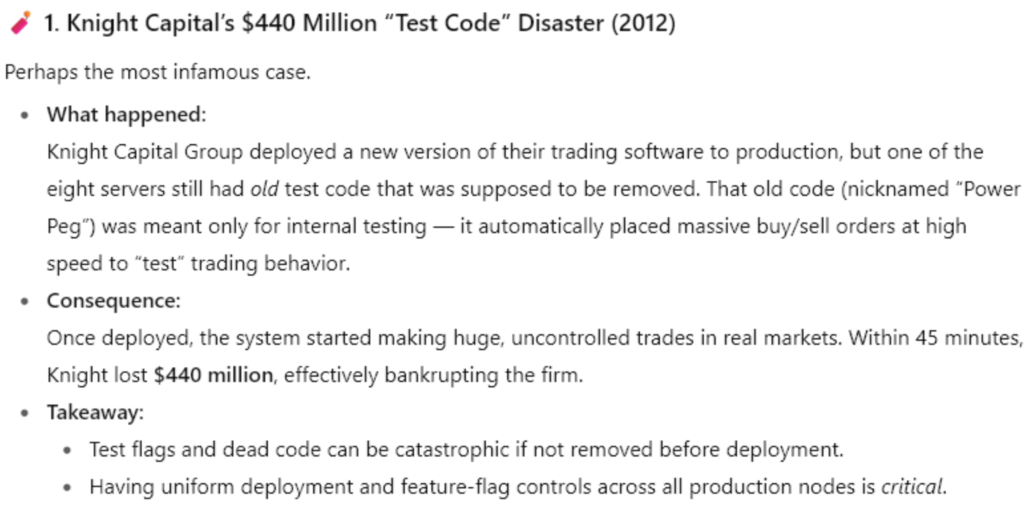
- What is the issue with this test stub?



```java
public class FrenchChefStrategy implements HeroStrategy {
    private Integer index = null;

    public FrenchChefStrategy() {
    }

    public FrenchChefStrategy(int index) { this.index = index; }

    @Override
    public Status usePowerChef(Game game, Player player) {
        Player opponent = Player.computeOpponent(player);
        int targetIndex;
        if (index == null) {
            targetIndex = (int) (Math.random() * game.getFieldSize(opponent));
        } else {
            targetIndex = index;
        }

        ((Cards) game.getCardInField(opponent, targetIndex)).changeHealth( amount: -2);
        return Status.OK;
    }
```

Test Code in Production

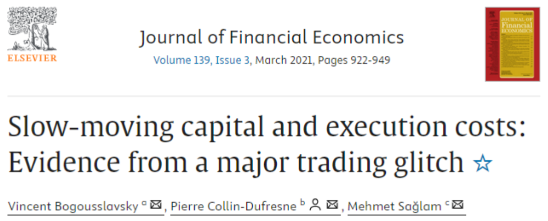
- One such example
  - Thanks to ChatGPT



1. Knight Capital's $440 Million "Test Code" Disaster (2012)

Perhaps the most infamous case.

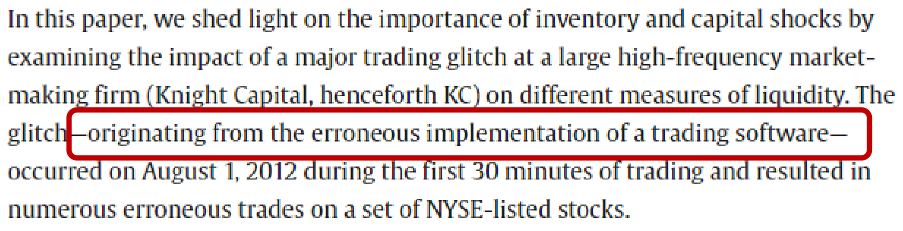
- **What happened:**
  Knight Capital Group deployed a new version of their trading software to production, but one of the eight servers still had *old* test code that was supposed to be removed. That old code (nicknamed "Power Peg") was meant only for internal testing — it automatically placed massive buy/sell orders at high speed to "test" trading behavior.
- **Consequence:**
  Once deployed, the system started making huge, uncontrolled trades in real markets. Within 45 minutes, Knight lost **$440 million**, effectively bankrupting the firm.
- **Takeaway:**
  - Test flags and dead code can be catastrophic if not removed before deployment.
  - Having uniform deployment and feature-flag controls across all production nodes is *critical*.



Journal of Financial Economics
Volume 139, Issue 3, March 2021, Pages 922-949

Slow-moving capital and execution costs: Evidence from a major trading glitch ☆

Vincent Bogousslavsky [a], Pierre Collin-Dufresne [b], Mehmet Sağlam [c]



In this paper, we shed light on the importance of inventory and capital shocks by examining the impact of a major trading glitch at a large high-frequency market-making firm (Knight Capital, henceforth KC) on different measures of liquidity. The glitch—originating from the erroneous implementation of a trading software—occurred on August 1, 2012 during the first 30 minutes of trading and resulted in numerous erroneous trades on a set of NYSE-listed stocks.

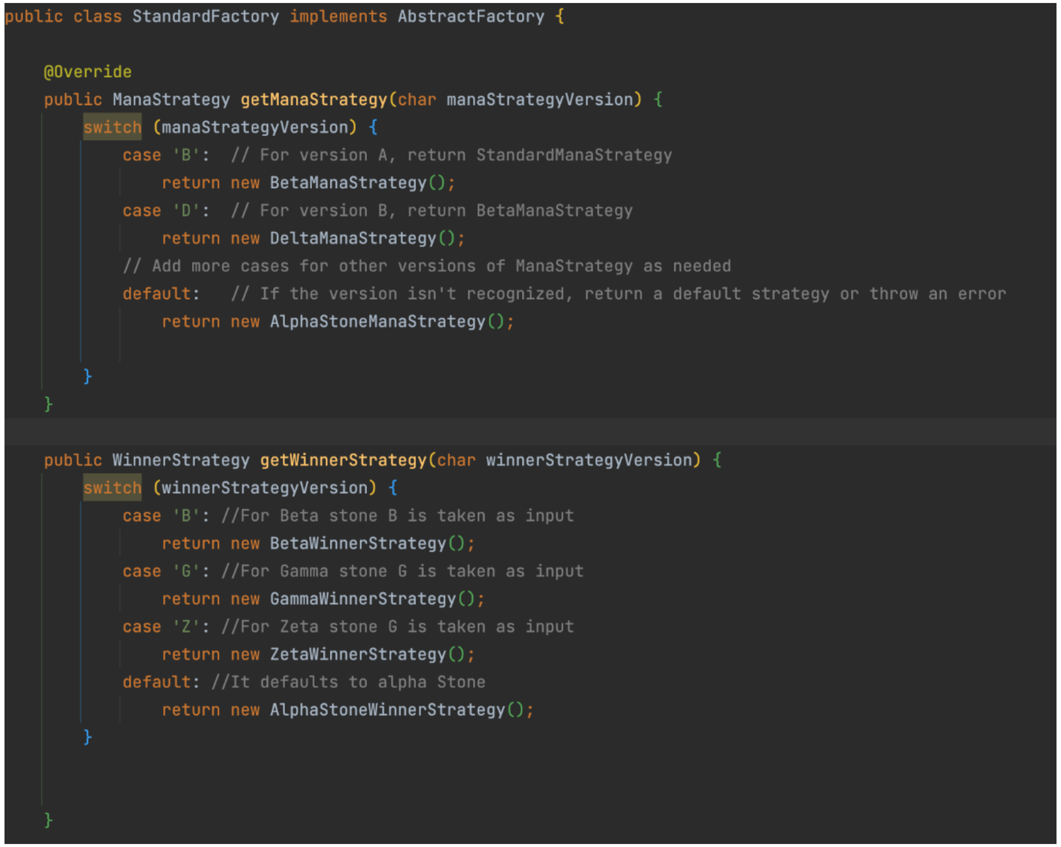
AARHUS UNIVERSITET

From 2024

AARHUS UNIVERSITET

Parametric Factory

- What is the problem here?

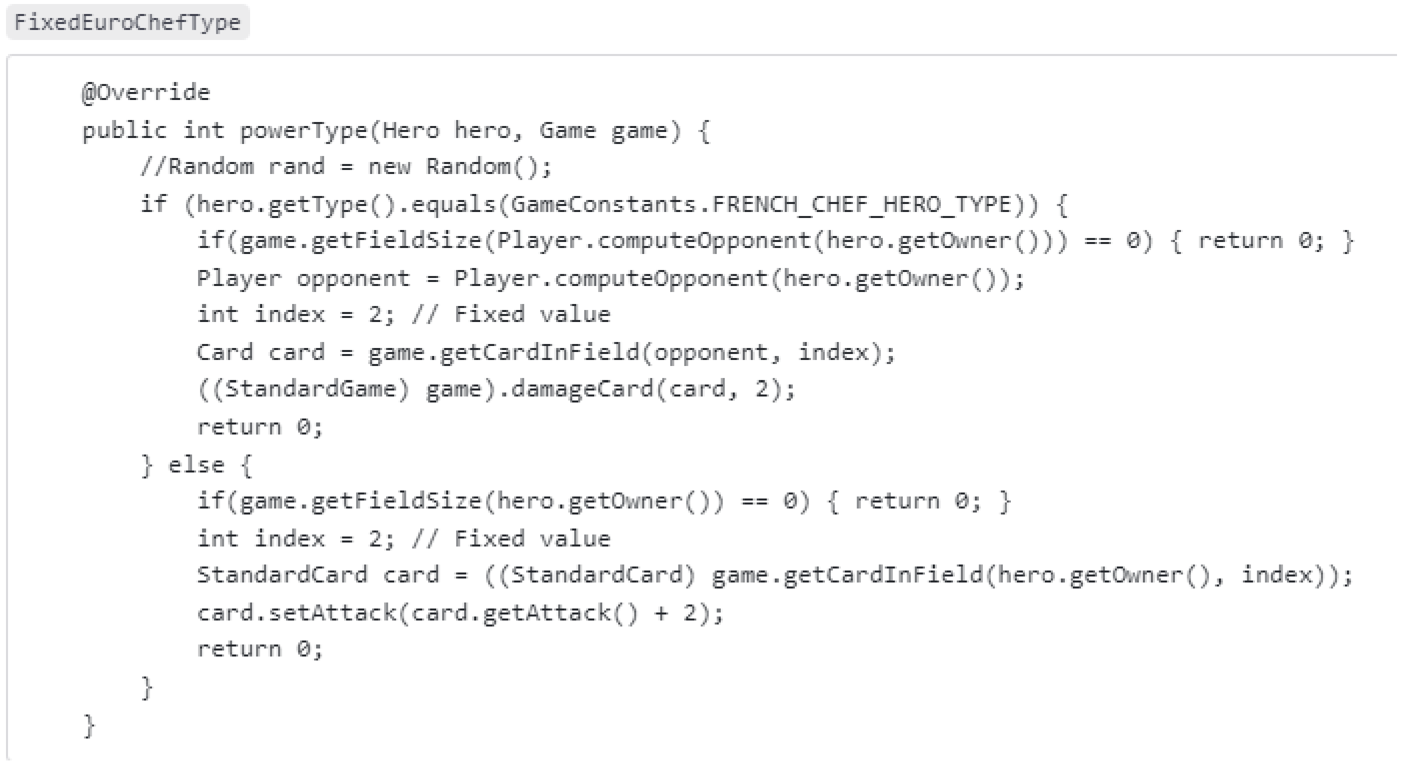


```java
public class StandardFactory implements AbstractFactory {

    @Override
    public ManaStrategy getManaStrategy(char manaStrategyVersion) {
        switch (manaStrategyVersion) {
            case 'B':  // For version A, return StandardManaStrategy
                return new BetaManaStrategy();
            case 'D':  // For version B, return BetaManaStrategy
                return new DeltaManaStrategy();
            // Add more cases for other versions of ManaStrategy as needed
            default:   // If the version isn't recognized, return a default strategy or throw an error
                return new AlphaStoneManaStrategy();

        }
    }

    public WinnerStrategy getWinnerStrategy(char winnerStrategyVersion) {
        switch (winnerStrategyVersion) {
            case 'B': //For Beta stone B is taken as input
                return new BetaWinnerStrategy();
            case 'G': //For Gamma stone G is taken as input
                return new GammaWinnerStrategy();
            case 'Z': //For Zeta stone G is taken as input
                return new ZetaWinnerStrategy();
            default: //It defaults to alpha Stone
                return new AlphaStoneWinnerStrategy();
        }


    }
```

AARHUS UNIVERSITET

Kata : Trap not avoided

- Issue?



```
FixedEuroChefType
```

```
@Override
public int powerType(Hero hero, Game game) {
    //Random rand = new Random();
    if (hero.getType().equals(GameConstants.FRENCH_CHEF_HERO_TYPE)) {
        if(game.getFieldSize(Player.computeOpponent(hero.getOwner())) == 0) { return 0; }
        Player opponent = Player.computeOpponent(hero.getOwner());
        int index = 2; // Fixed value
        Card card = game.getCardInField(opponent, index);
        ((StandardGame) game).damageCard(card, 2);
        return 0;
    } else {
        if(game.getFieldSize(hero.getOwner()) == 0) { return 0; }
        int index = 2; // Fixed value
        StandardCard card = ((StandardCard) game.getCardInField(hero.getOwner(), index));
        card.setAttack(card.getAttack() + 2);
        return 0;
    }
}
```

Pesky Spy Exercise

AARHUS UNIVERSITET

First Year With Strong Spy Focus

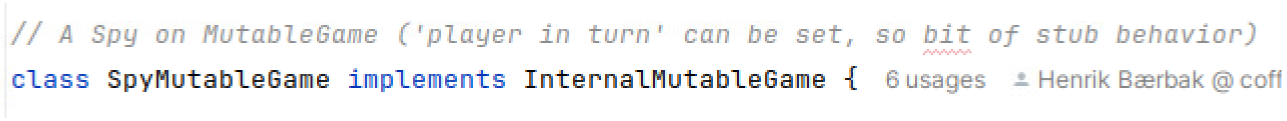
- … so pardon for annoying you with some sharp corners

- Observation
  - A test spy is *not* an object that cannot have *stub* behavior

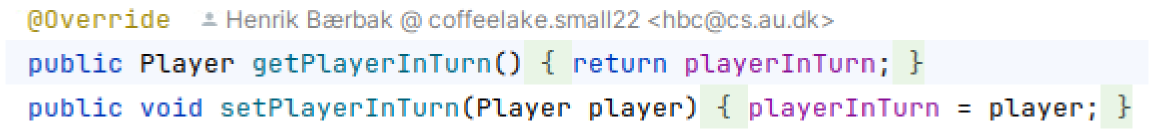
- Indeed – to make a MutableGameSpy work so it can help us test the card effects, it *needs* to feed ‘stub’ indirect input.
  - Example: Brown Rice effect likely needs to call ‘game.getPlayerInTurn()’ to compute the opponent…

Spy With Stub Behavior

- So my spy, for testing eta stone cards *does include stub methods,* like



```
// A Spy on MutableGame ('player in turn' can be set, so bit of stub behavior)
class SpyMutableGame implements InternalMutableGame {  6 usages  Henrik Bærbak @ coff
```



```
@Override  Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
public Player getPlayerInTurn() { return playerInTurn; }
public void setPlayerInTurn(Player player) { playerInTurn = player; }
```

Stubs with Stubs

- I saw quite a few whose card effect strategies retrieve Hero and Card instances
  - Like fetching **the Card, c,** that game must mutate ala
    - mutableGame.reduceHealthOf(**c**, -2);

- Thus the ‘MutableGameSpy’ must thus make ‘stub objects’ for Card to serve the ‘fetching’
  - Card getCardInField(…); must return card objects

- ☹ - lots of coding…

One insight

- One insight to reduce this effort (significantly) is that every hero and every card is *identifiable* via simple data types!

Hero	‘who is it	FINDUS or PEDDERSEN’
Card	‘field index	0..n’

- So instead of
  - reduceHealthOf(**Card c**, int delta)
  - Use
  - reduceHealthOf(**int fieldIndexOfCard**, int delta)
- Similar for hero: use ‘Player’ as type, not ‘Hero’
  - reduceHeroHealth(**Player who**, int delta)

But – it is not uncommon

- I used Mockito for creating test doubles for SparkJava's HTTP library and ended with "*deep doubling*"

```
when(Unirest.get(path)).thenReturn(getRequest);
when(getRequest.asJson()).thenReturn(httpResponse);
when(httpResponse.getStatus()).thenReturn(HttpServletResponse.SC_OK);
when(httpResponse.getBody()).thenReturn(roomPayload);
```

- When GET on url
  - I get a Request **object**
    - That get a httpRequest **object**
      - That should return OK for 'getStatus()'

- **TEDIOS to have deep path of objects to be doubled**

AARHUS UNIVERSITET

Casting Avoided?

- I may have said that introducing, say, MutableCard makes StandardGame free of castings…
  - Like Map<Player, List<**MutableCard**>> fieldMap =…
    - Avoid casting to StandardCard or references to it…
  - Not quite true. Free of casting to **class** but not free of casting to **interface**
    - attackCard(…, **Card** attackingCard, …)
      - Need to be cast ala
      - **MutableCard asMutableCard = (MutableCard) attackingCard**;
- Exercise:
  - Why is cast to interface *much better* than cast to class?