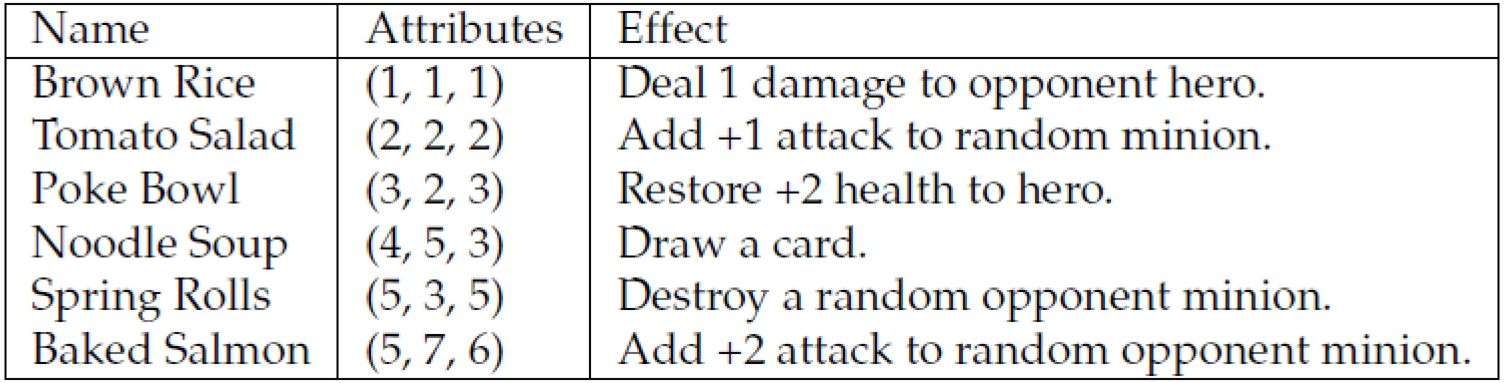
AARHUS UNIVERSITET

Software Engineering and Architecture

Mandatory Reflections

EtaStone = Card Effects

- Card Effects = The fun begins
  - This is the core game mechanics of HearthStone

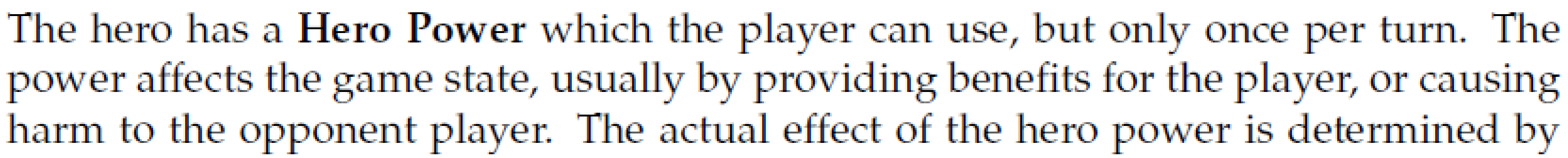


Name	Attributes	Effect
Brown Rice	(1, 1, 1)	Deal 1 damage to opponent hero.
Tomato Salad	(2, 2, 2)	Add +1 attack to random minion.
Poke Bowl	(3, 2, 3)	Restore +2 health to hero.
Noodle Soup	(4, 5, 3)	Draw a card.
Spring Rolls	(5, 3, 5)	Destroy a random opponent minion.
Baked Salmon	(5, 7, 6)	Add +2 attack to random opponent minion.

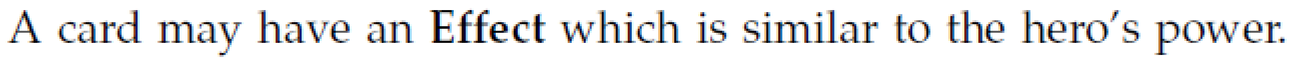
- What is it?
  - *Given a trigger, modify the game's state (trigger = play card)*
- This resembles something we have seen before?

Effects

- Seen before, yes


The hero has a **Hero Power** which the player can use, but only once per turn. The power affects the game state, usually by providing benefits for the player, or causing harm to the opponent player. The actual effect of the hero power is determined by

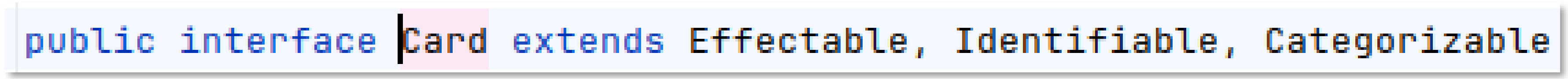
  - *Given a trigger, modify the game's state (trigger = use power)*


A card may have an **Effect** which is similar to the hero's power.

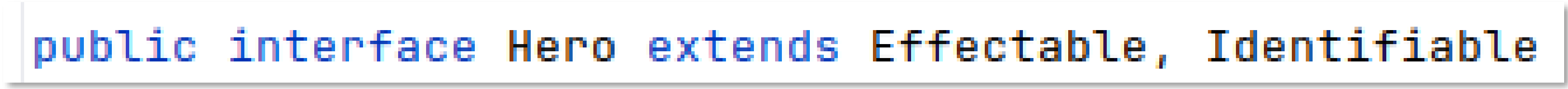
- It is a **role** that a game object can play ☺
  - Role Interface: *Effectable* ('able to trigger an effect')

Effectable

- So

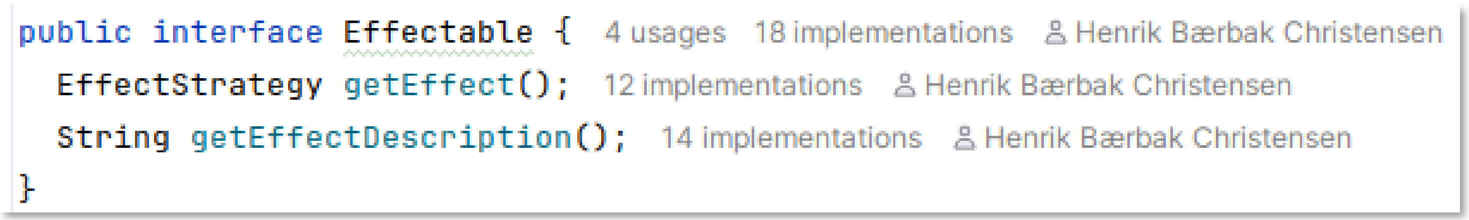


```
public interface Card extends Effectable, Identifiable, Categorizable
```



```
public interface Hero extends Effectable, Identifiable
```

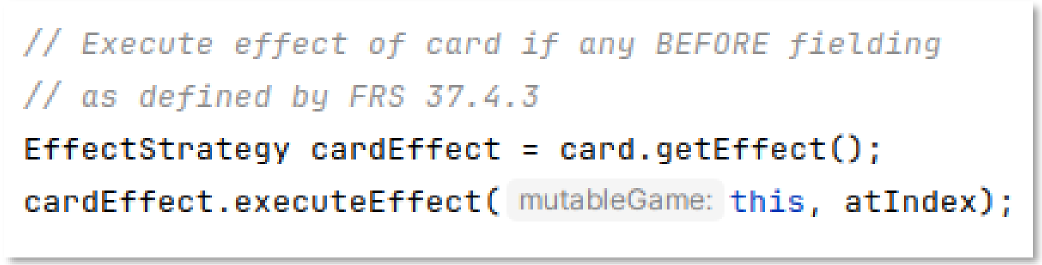
- Which signals that cards and heroes have the following *responsibilities*



```
public interface Effectable {  4 usages  18 implementations  Henrik Bærbak Christensen
  EffectStrategy getEffect();  12 implementations  Henrik Bærbak Christensen
  String getEffectDescription();  14 implementations  Henrik Bærbak Christensen
}
```

Thus triggering an Card Effect

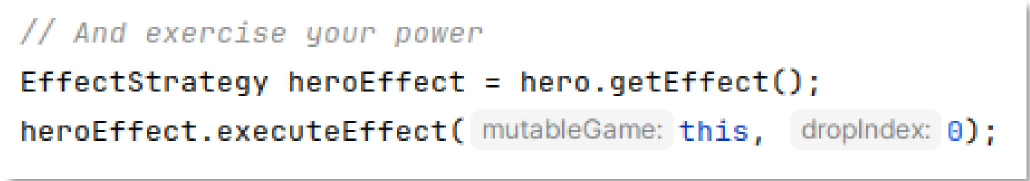
- As Game knowns that a card has the responsibility to provide its effect, it can simply (during playCard()) do
  - "Card, hand me your effect, and then I will execute it."



```
// Execute effect of card if any BEFORE fielding
// as defined by FRS 37.4.3
EffectStrategy cardEffect = card.getEffect();
cardEffect.executeEffect( mutableGame: this, atIndex);
```

Do the same thing, the same way...

- In usePower() method, same thing goes on

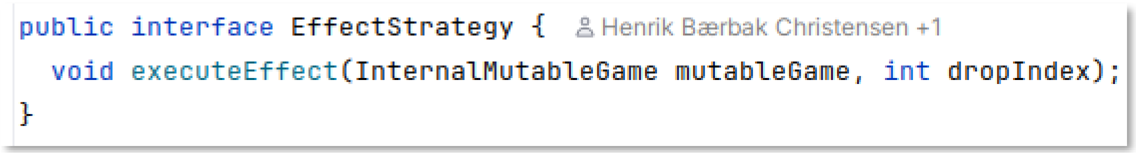


```
// And exercise your power
EffectStrategy heroEffect = hero.getEffect();
heroEffect.executeEffect( mutableGame: this, dropIndex: 0);
```

AARHUS UNIVERSITET

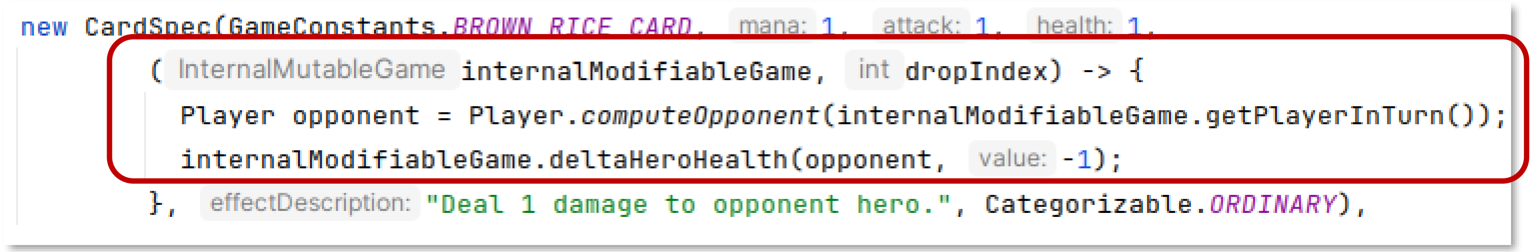
Effects as Lambdas

- My EffectStrategy only has a *single method*...



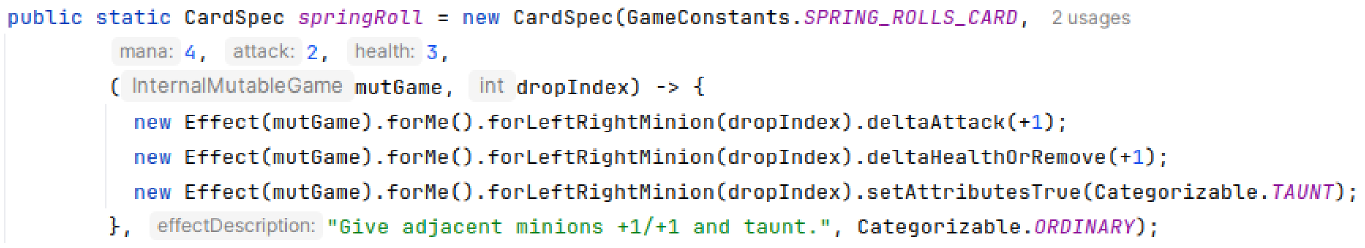
```
public interface EffectStrategy {  Henrik Bærbak Christensen +1
  void executeEffect(InternalMutableGame mutableGame, int dropIndex);
}
```

- ... So I can code them as *lambda functions*



```
new CardSpec(GameConstants.BROWN_RICE_CARD,  mana: 1,  attack: 1,  health: 1,
        ( InternalMutableGame internalModifiableGame,  int dropIndex) -> {
          Player opponent = Player.computeOpponent(internalModifiableGame.getPlayerInTurn());
          internalModifiableGame.deltaHeroHealth(opponent,  value: -1);
        },  effectDescription: "Deal 1 damage to opponent hero.", Categorizable.ORDINARY),
```

- *From my PiStone, using WizardHub*

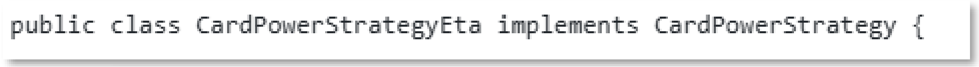


```
public static CardSpec springRoll = new CardSpec(GameConstants.SPRING_ROLLS_CARD,   2 usages
         mana: 4,  attack: 2,  health: 3,
        ( InternalMutableGame mutGame,  int dropIndex) -> {
          new Effect(mutGame).forMe().forLeftRightMinion(dropIndex).deltaAttack(+1);
          new Effect(mutGame).forMe().forLeftRightMinion(dropIndex).deltaHealthOrRemove(+1);
          new Effect(mutGame).forMe().forLeftRightMinion(dropIndex).setAttributesTrue(Categorizable.TAUNT);
        },  effectDescription: "Give adjacent minions +1/+1 and taunt.", Categorizable.ORDINARY);
```

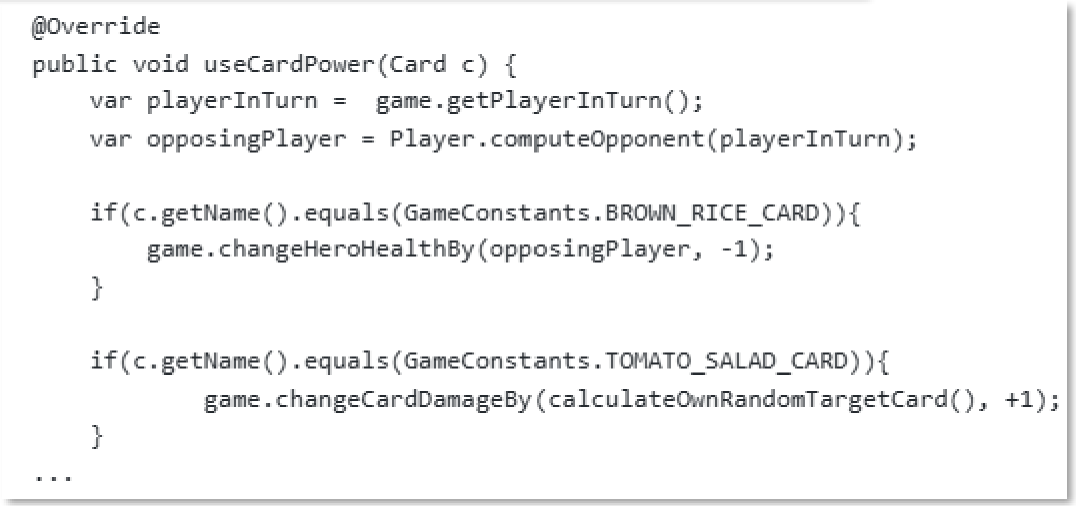
AARHUS UNIVERSITET

Some opt for "flatter design"

- The effect strategy is associated with Game, not Card
  - A good design: *open for extension, closed for modification*
    - It has the 'card type switch' **in the strategy, not in game**



```
public class CardPowerStrategyEta implements CardPowerStrategy {
```



```
@Override
public void useCardPower(Card c) {
    var playerInTurn =  game.getPlayerInTurn();
    var opposingPlayer = Player.computeOpponent(playerInTurn);

    if(c.getName().equals(GameConstants.BROWN_RICE_CARD)){
        game.changeHeroHealthBy(opposingPlayer, -1);
    }

    if(c.getName().equals(GameConstants.TOMATO_SALAD_CARD)){
            game.changeCardDamageBy(calculateOwnRandomTargetCard(), +1);
    }
...
```

- Liability
  - The switch can become pretty long
  - (HearthStone => 1000 cards)

AARHUS UNIVERSITET

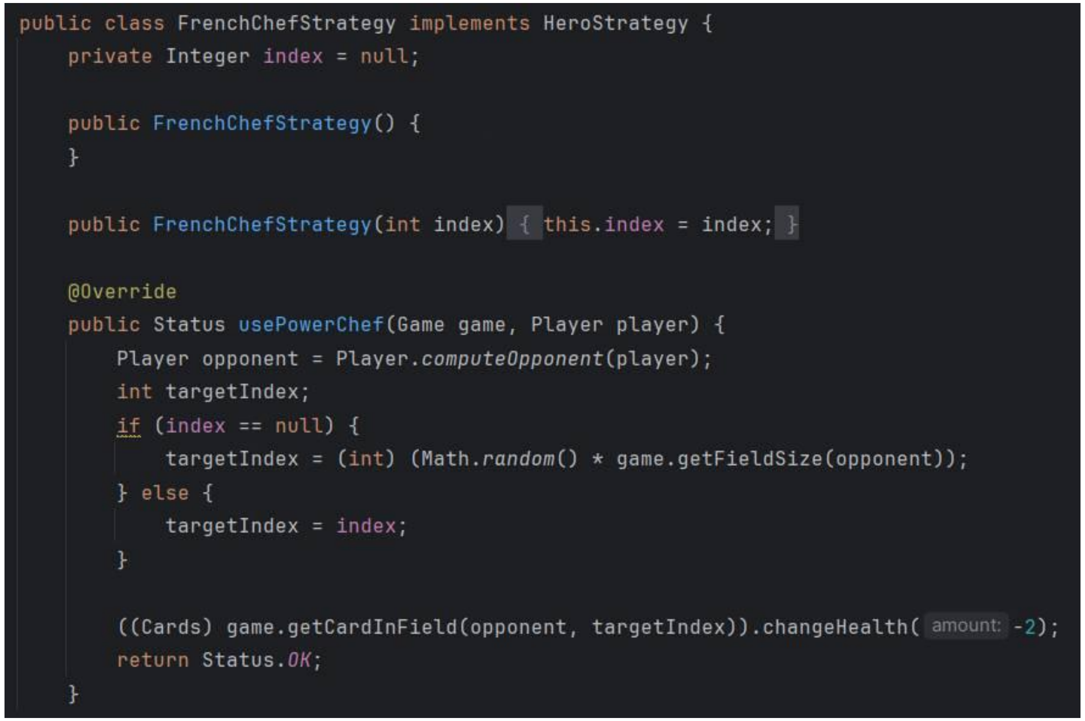
From last week's cancelled lecture...

One major issue that needs to be stated...

No Doubles in Production

- What is the issue with this "test stub"?

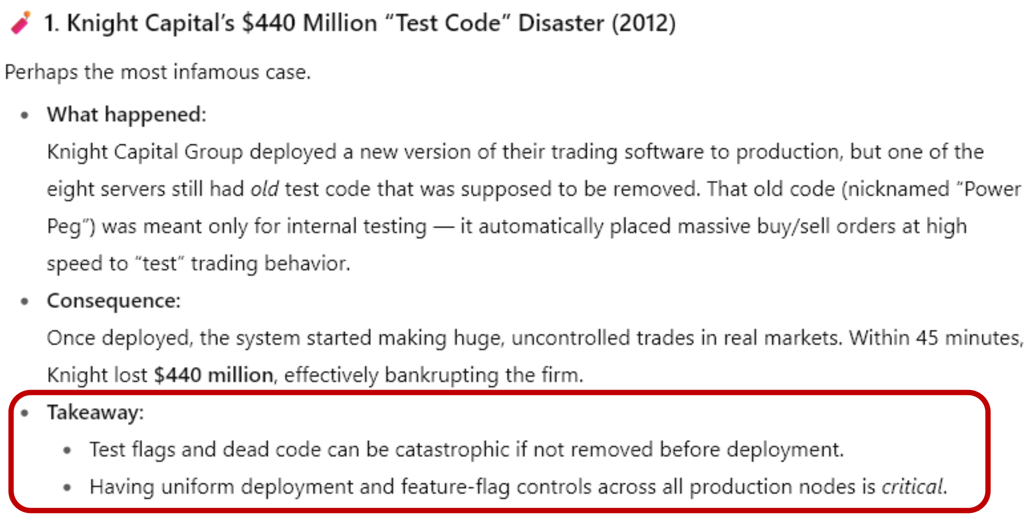
- (Which per definition is not a test stub…)



```
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

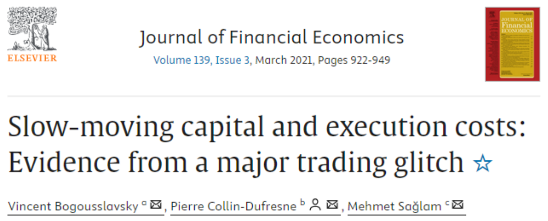
- One such example
  - Thanks to ChatGPT



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

Perhaps the most infamous case.

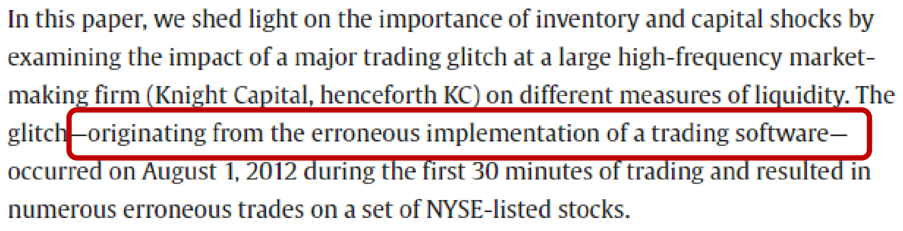
- **What happened:**
  Knight Capital Group deployed a new version of their trading software to production, but one of the eight servers still had *old* test code that was supposed to be removed. That old code (nicknamed "Power Peg") was meant only for internal testing — it automatically placed massive 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]

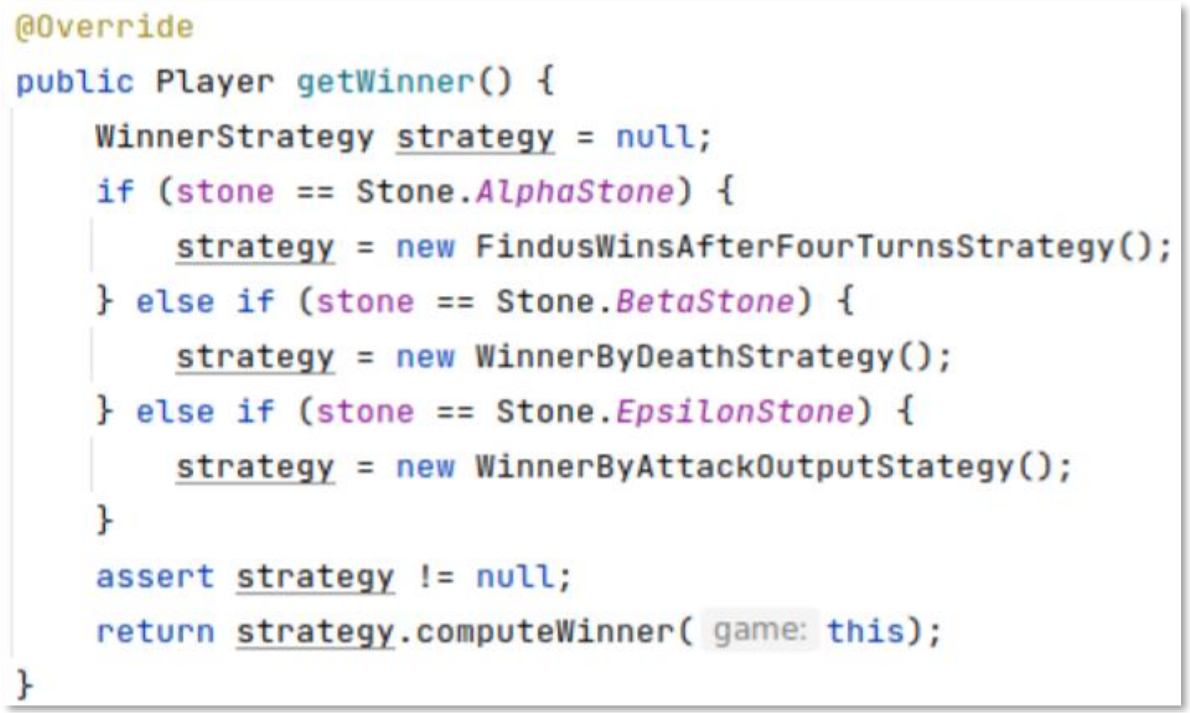


In this paper, we shed light on the importance of inventory and capital shocks by examining the impact of a major trading glitch at a large high-frequency market-making firm (Knight Capital, henceforth KC) on different measures of liquidity. The glitch—originating from the erroneous implementation of 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.

From Earlier Years

From the Trenches

- What is problematic here?
  - Assuming this method is in Game?

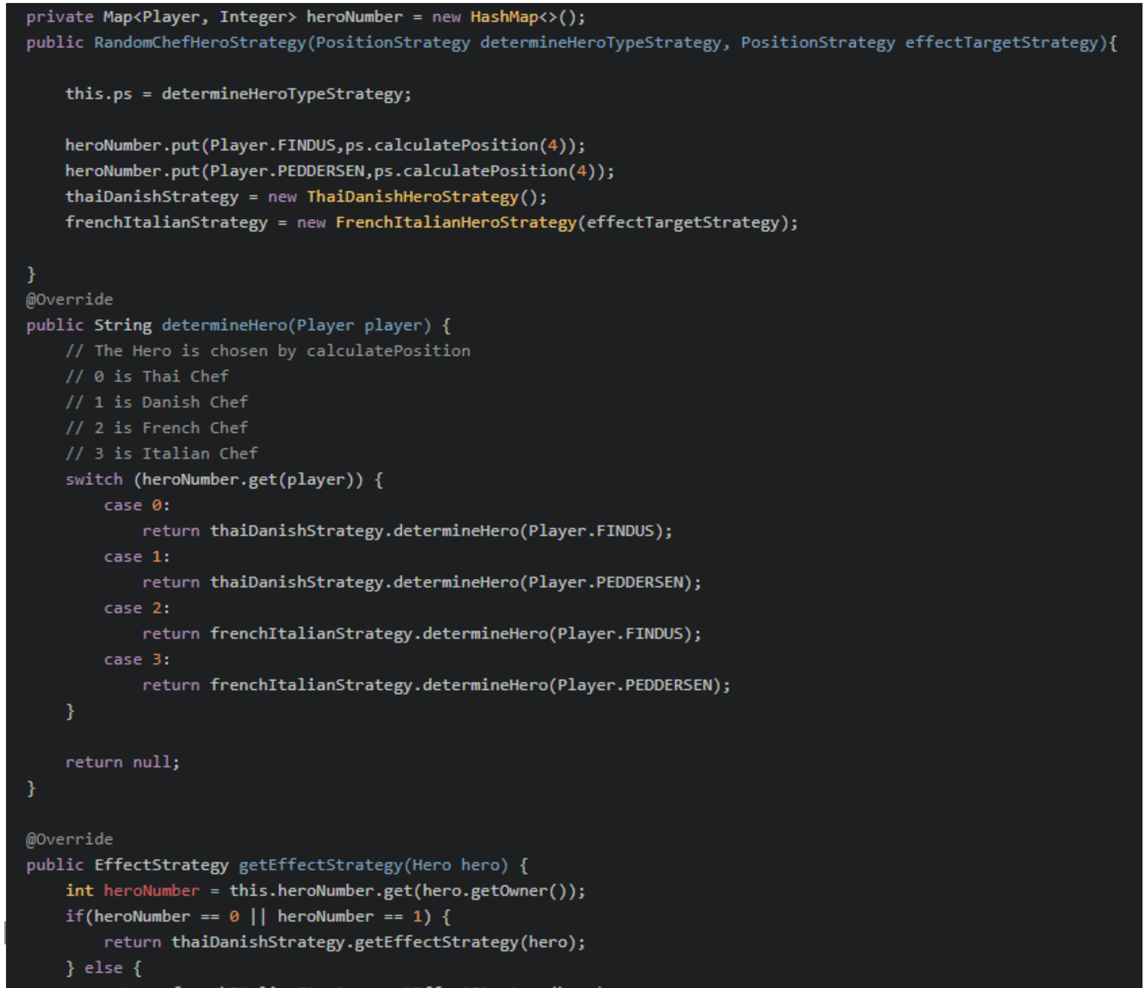


```java
@Override
public Player getWinner() {
    WinnerStrategy strategy = null;
    if (stone == Stone.AlphaStone) {
        strategy = new FindusWinsAfterFourTurnsStrategy();
    } else if (stone == Stone.BetaStone) {
        strategy = new WinnerByDeathStrategy();
    } else if (stone == Stone.EpsilonStone) {
        strategy = new WinnerByAttackOutputStategy();
    }
    assert strategy != null;
    return strategy.computeWinner( game: this);
}
```

Overengineering

- What is happening?
  - SemiStone’s “pick random hero”

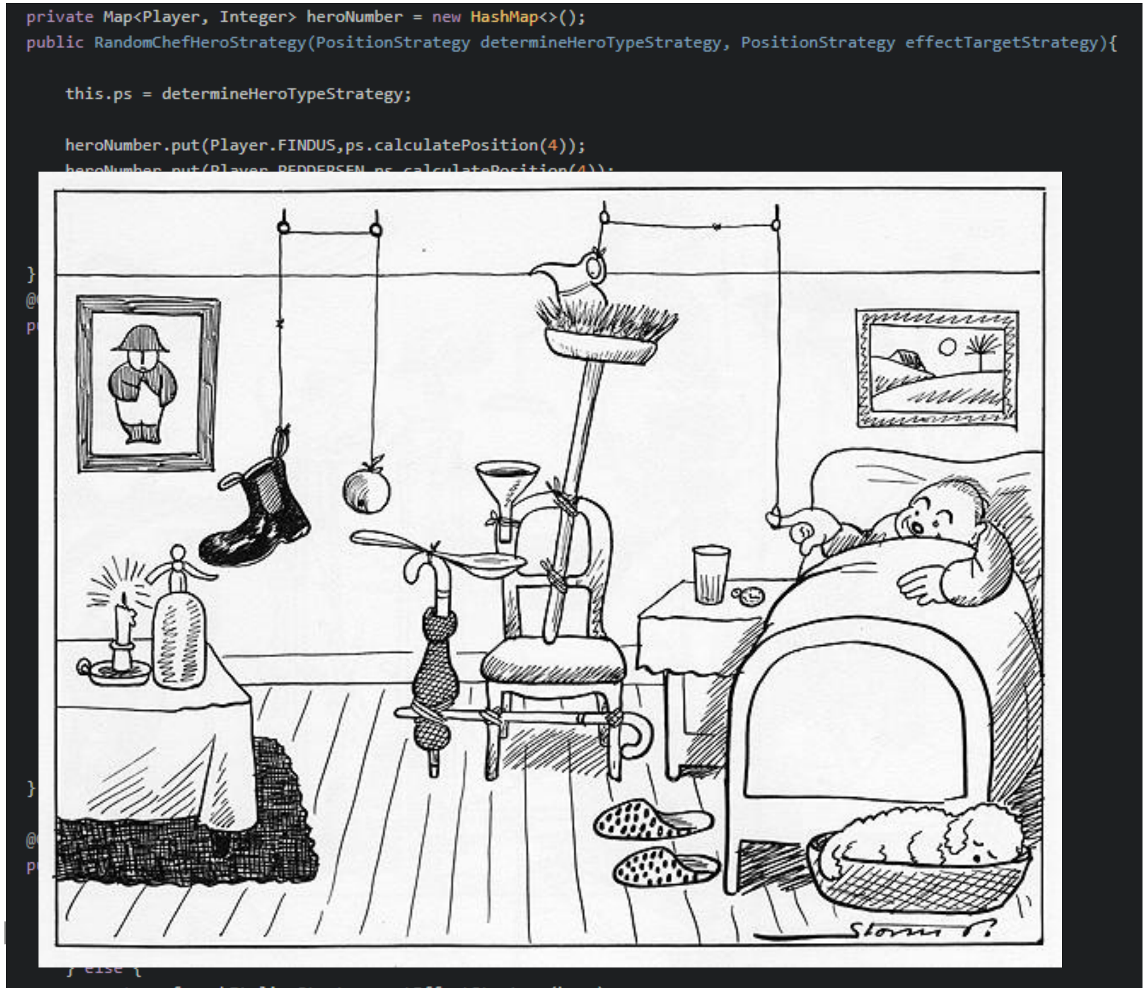
- ???



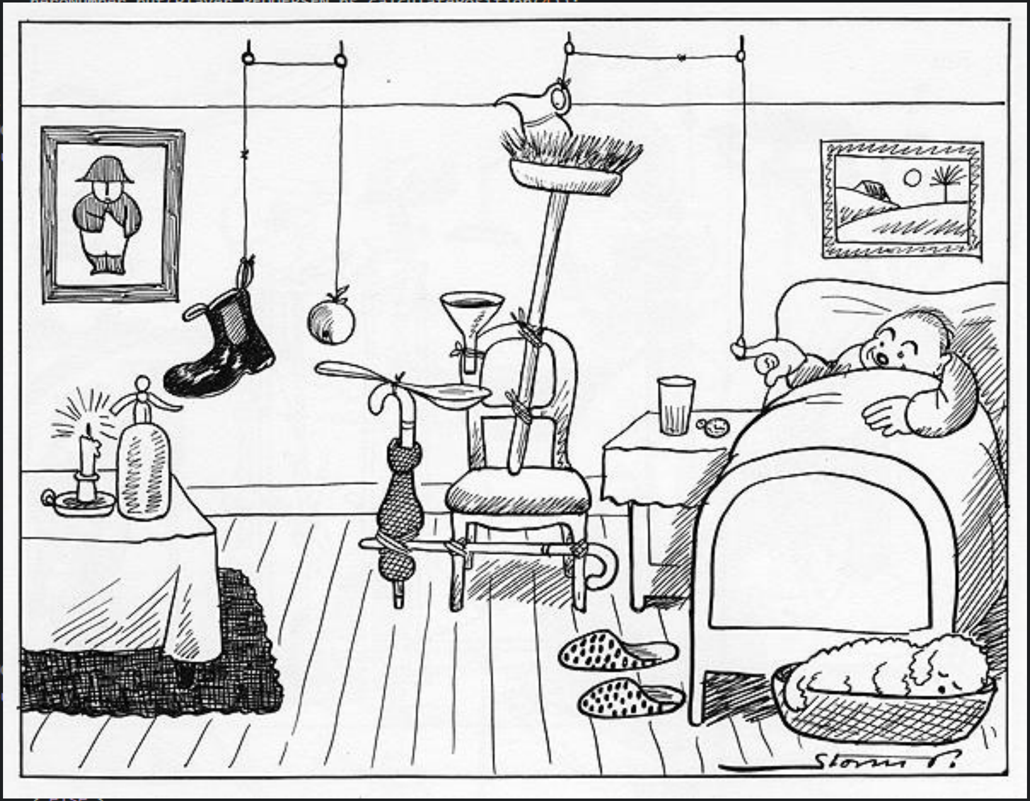
```java
private Map<Player, Integer> heroNumber = new HashMap<>();
public RandomChefHeroStrategy(PositionStrategy determineHeroTypeStrategy, PositionStrategy effectTargetStrategy){

    this.ps = determineHeroTypeStrategy;

    heroNumber.put(Player.FINDUS,ps.calculatePosition(4));
    heroNumber.put(Player.PEDDERSEN,ps.calculatePosition(4));
    thaiDanishStrategy = new ThaiDanishHeroStrategy();
    frenchItalianStrategy = new FrenchItalianHeroStrategy(effectTargetStrategy);

}
@Override
public String determineHero(Player player) {
    // The Hero is chosen by calculatePosition
    // 0 is Thai Chef
    // 1 is Danish Chef
    // 2 is French Chef
    // 3 is Italian Chef
    switch (heroNumber.get(player)) {
        case 0:
            return thaiDanishStrategy.determineHero(Player.FINDUS);
        case 1:
            return thaiDanishStrategy.determineHero(Player.PEDDERSEN);
        case 2:
            return frenchItalianStrategy.determineHero(Player.FINDUS);
        case 3:
            return frenchItalianStrategy.determineHero(Player.PEDDERSEN);
    }

    return null;
}

@Override
public EffectStrategy getEffectStrategy(Hero hero) {
    int heroNumber = this.heroNumber.get(hero.getOwner());
    if(heroNumber == 0 || heroNumber == 1) {
        return thaiDanishStrategy.getEffectStrategy(hero);
    } else {
```

Overengineering

- What is happening?
  - SemiStone’s “pick random hero”
- Overengineer
  - Overly complex solution to simple problem
- Storm P
  - Avoid his creations

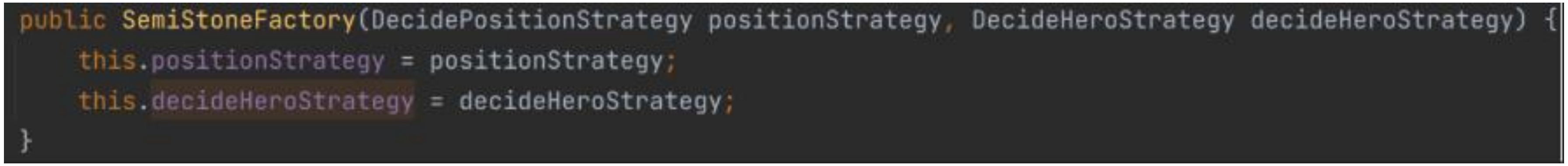


```
private Map<Player, Integer> heroNumber = new HashMap<>();
public RandomChefHeroStrategy(PositionStrategy determineHeroTypeStrategy, PositionStrategy effectTargetStrategy){

    this.ps = determineHeroTypeStrategy;

    heroNumber.put(Player.FINDUS,ps.calculatePosition(4));
```



Factory (?)

- A factory for SemiStone
  - Or – is it???



```
public SemiStoneFactory(DecidePositionStrategy positionStrategy, DecideHeroStrategy decideHeroStrategy) {
    this.positionStrategy = positionStrategy;
    this.decideHeroStrategy = decideHeroStrategy;
}
```

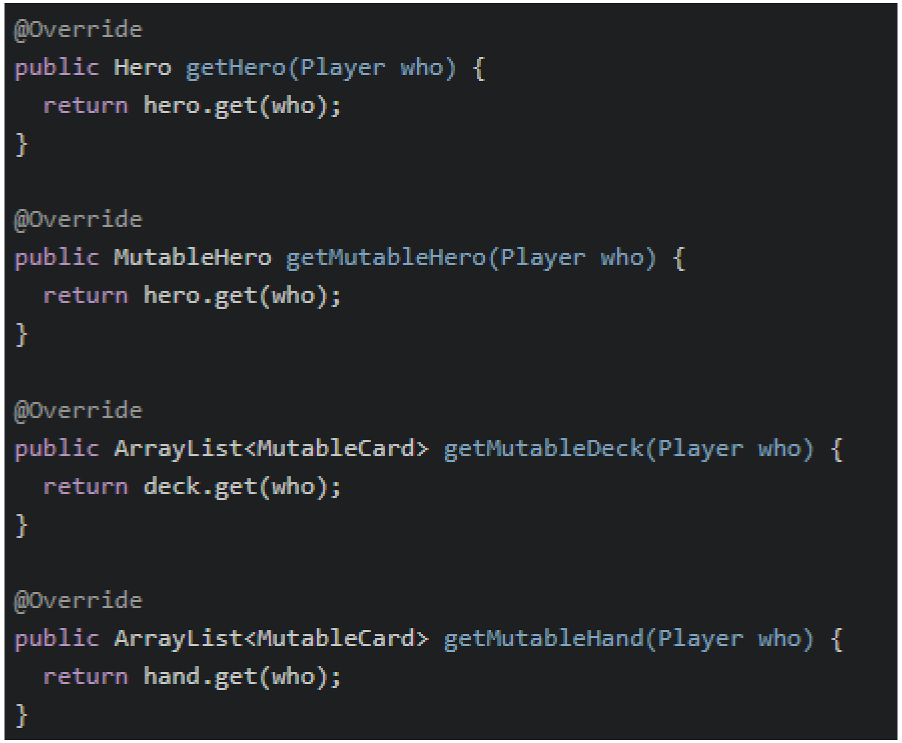
- Find two aspects that are problematic here

AARHUS UNIVERSITET

Cast to Interface

- Cast to an interface is not problematic; cast to class is.
  - MutableHero h = (MutableHero) getHero(who);

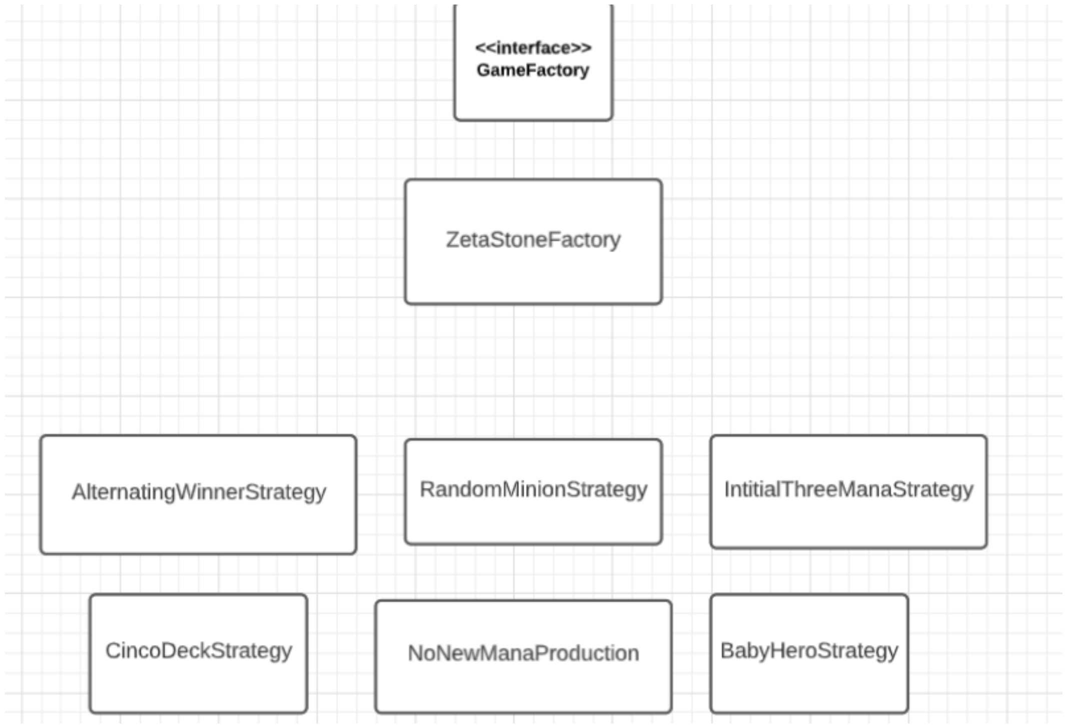
- One example of avoiding it
  - Benefits?
  - Liabilities?



```
@Override
public Hero getHero(Player who) {
  return hero.get(who);
}

@Override
public MutableHero getMutableHero(Player who) {
  return hero.get(who);
}

@Override
public ArrayList<MutableCard> getMutableDeck(Player who) {
  return deck.get(who);
}

@Override
public ArrayList<MutableCard> getMutableHand(Player who) {
  return hand.get(who);
}
```

Simple, but too Simple

- A UML with 1000 lines is worthless, but do not fall in the other pitfall ☺


<<interface>>
GameFactory
ZetaStoneFactory
AlternatingWinnerStrategy
RandomMinionStrategy
IntitialThreeManaStrategy
CincoDeckStrategy
NoNewManaProduction
BabyHeroStrategy