AARHUS UNIVERSITET

Software Engineering and Architecture

From the Trenches

AARHUS UNIVERSITET

Disclaimer

- The following examples are ***your code.***
- I do not know who they belong to! I do not care!
- This is **not to embarrass anyone!**
  - **You do not learn to bicycle without falling off the bike!**
- It is for the purpose of reflection and discussion!
  - You learn a lot from seeing what others do…

AARHUS UNIVERSITET

Test Ordering

- TDD is about taking *small steps*
  - *But the **order** in which you take steps is important!*
- *Example:*
  - A player's hand is a List of cards which has a size

```
public class Game {

  private Map<Player, Integer> playerHandSize = new HashMap<>();

  public Status playCard(Player who, Card card, int atIndex) {
    StandardHotStoneHero hero = (StandardHotStoneHero) getHero(who);
    int heroMana = hero.getMana();

    int manaCost = card.getManaCost();
    hero.setMana(heroMana - manaCost);

    int newHandSize = playerHandSize.get(who) - 1;
    playerHandSize.replace(who,newHandSize);
    return Status.OK;
  }
}
```

Handling size before list is in place is 'lot of work for nothing'

Comparison

- My implementation looks like this

```
@Override   Henrik Bærbak Christensen
public int getDeckSize(Player who) {
  return deckMap.get(who).size();
}
```

- … because I knew that the TDD of the size of hand/deck will automatically be implemented once the hand/deck lists were correctly TDD'ed in place…
- … because 'lookup in Hashmap' is better than switches

AARHUS UNIVERSITET

And this one…

- *Large is **not** beautiful in software development…*

```
private int findusDeckSize = 4;
private int peddersenDeckSize = 4;

@Override
public int getDeckSize(Player who) {
  if (turnNumber >= 2) {
    for (int i = turnNumber; i >= 2; i--) {
      // Determine which player's turn it is
      boolean isFindusTurn = (i%2 == 0);
      if (isFindusTurn && Player.FINDUS.equals(who)) {
        findusDeckSize--; // Decrease Findus's deck size only when it is his turn
      } else if (!isFindusTurn && Player.PEDDERSEN.equals(who)) {
        peddersenDeckSize--; // Decrease Peddersen's deck size only when it is his turn
      }
    }
  }

  // Return the current deck size for the specified player
  if (who.equals(Player.FINDUS)) {
    return findusDeckSize;
  } else {
    return peddersenDeckSize;
  }
}
```

Compare complexity to:

```
@Override   Henrik Bærbak Christensen
public int getDeckSize(Player who) {
  return deckMap.get(who).size();
}
```

AARHUS UNIVERSITET

Use your Datastructures

- What can be constructively criticized here?

```
@Override  2 usages
public int getFieldSize(Player who) {
  //Given player is Findus
  if (who == Player.FINDUS){
    return Findus_Field.size();
    // Given player is Peddersen
  } else {
    return Peddersen_Field.size();
  }

}
```

- *Use a datastructure that can lookup the list*

```
@Override  Henrik Bærbak Christensen
public int getDeckSize(Player who) {
  return deckMap.get(who).size();
}
```

AARHUS UNIVERSITET

TDD of ‘getHand()’

- getHand() returns an Iterable<Card> to allow clients to easily iterate cards ala
  - for (Card c : game.getHand()) { … do something with c; }
- The UI will use that all the time!
- But I guess you already have testet ‘getCardInHand(..)’?
- Then TDD of getHand() is simple
  - Call it and ensure it returns something…

AARHUS UNIVERSITET

My single test case

- Is more or less just a call…
  - TDD is a ‘programming technique’ …
  - … not a ‘comprehensive testing technique’
  - We will return to systematic testing later

```
@Test   Henrik Bærbak Christensen +1
public void shouldSupportIteration() {
  int count = 0;
  for (Card c : game.getHand(Player.PEDDERSEN)) {
    count++;
  }
  assertThat(count, is( value: 3));

  count = 0;
  for (Card c : game.getField(Player.FINDUS)) {
    count++;
  }
  assertThat(count, is( value: 0));
}
```

AARHUS UNIVERSITET

Objects = Unique Identity

- Cards represent physical objects
  - Two Uno cards are --- well – **two Uno cards!**
- **What is wrong here?**

```
public void initializeDecks() {  1 usage
    // creating the cards
    Card card1 = new StandardCard(GameConstants.UNO_CARD, mana: 1, damage: 1, health: 1);
    Card card2 = new StandardCard(GameConstants.DOS_CARD, mana: 2, damage: 2, health: 2);
    Card card3 = new StandardCard(GameConstants.TRES_CARD, mana: 3, damage: 3, health: 3);
    Card card4 = new StandardCard(GameConstants.CUATRO_CARD, mana: 2, damage: 3, health: 1);
    Card card5 = new StandardCard(GameConstants.CINCO_CARD, mana: 3, damage: 5, health: 1);
    Card card6 = new StandardCard(GameConstants.SEIS_CARD, mana: 2, damage: 1, health: 3);
    Card card7 = new StandardCard(GameConstants.SIETE_CARD, mana: 3, damage: 2, health: 4);

    //creating the ArrayList decks, and adding the cards
    ArrayList<Card> findusDeck = new ArrayList<~>();
    findusDeck.add(card1); findusDeck.add(card2); findusDeck.add(card3); findusDeck.add(card4);
    findusDeck.add(card5); findusDeck.add(card6); findusDeck.add(card7);
    ArrayList<Card> peddersenDeck = new ArrayList<~>();
    peddersenDeck.addAll(findusDeck);;
```

To see it

- What happens when Peddersen attacks Findus Tres Card?

- Exercise:
  - If you not understand the problem, you should **stay on this page until you do understand, or you will never become good at programming ☺**

AARHUS UNIVERSITET

Another example

- A similar issue
  - The idea of a CardCollection is fine, but this is **not** the way to do it.

- What is the issue here:

```
public class CardCollection {  4 usages
    public static Card UNO_CARD = new StandardHotStoneCard(GameConstants.UNO_CARD,  mana: 1,  health: 1,  attack: 1);
    public static Card DOS_CARD = new StandardHotStoneCard(GameConstants.DOS_CARD,  mana: 2,  health: 2,  attack: 2);
}
```

AARHUS UNIVERSITET

Over-methodication ☺

- Do not pollute the method space
  - Discussed at the Kata…

```
private void changeHealth(int change) { health += change; }  2 usages

public void loseHealth(int loss) {changeHealth(-loss);  }  1 usage

public void gainHealth(int gain) { changeHealth(gain);}  1 usage

public void setHealth(int value) { health = value; }  1 usage
```

AARHUS UNIVERSITET

Constructive Critique?

```java
@Test
public void shouldNotPlayCardWhenManaIsTooLow() {
  // given a game, initial mana for Findus, the mana cost of a card in hand (3 in this case)
  int initialMana = game.getMana(Player.FINDUS);
  int cardManaValue = game.getCardInHand(Player.FINDUS, 0).getManaCost();
  // when the card is played, and it costs 3
  // then the move is legal, and the mana is updated legally
  assertThat(game.playCard(Player.FINDUS, game.getCardInHand(Player.FINDUS, 0)), is(Status.OK));
  assertThat(game.getMana(Player.FINDUS), is(initialMana - cardManaValue));
  // when another card is played
  // then there is no mana left, so we should get the status 'not enough mana'
  if(cardManaValue == 3) {
    assertThat(game.playCard(Player.FINDUS, game.getCardInHand(Player.FINDUS, 0)), is(Status.NOT_ENOUGH_MANA));
  }
  else {
    // fail assertion
    assertThat(1, is(0));
  }
}
```

AARHUS UNIVERSITET

From My Own TDD

- Note:
  - Concrete values [Testcases are concrete!]
  - Use names for cards to make test evident
  - No ifs! Ever!

- *Evident Tests*

```
@Test  Henrik Bærbak Christensen +2
public void shouldAllowFindusToPlayUno() {
  // Given initialized game
  // When playing card #2 (Uno) from hand to field
  Card unoCard = game.getCardInHand(Player.FINDUS,  indexInHand: 2);
  Status status = game.playCard(Player.FINDUS, unoCard,  atIndex: 0);
  // Then it is allowed
  assertThat(status, is(Status.OK));

  // And Findus' hand is now size 2
  int count = game.getHandSize(Player.FINDUS);
  assertThat(count, is( value: 2));

  // And Uno is present on field at index 0
  Card uno = game.getCardInField(Player.FINDUS,  indexInField: 0);
  assertThat(uno.getName(), is(GameConstants.UNO_CARD));

  // And Uno is sleeping / not active
  assertThat(uno.isActive(), is( value: false));
}
```

AARHUS UNIVERSITET

Meaningful Names

- Use names that say what they are
  - Why is findus hero – not called ‘findusHero’?

```
hero1 = new StandardHero(Player.FINDUS);
hero2 = new StandardHero(Player.PEDDERSEN);

cardHand1 = new ArrayList<>();
cardHand2 = new ArrayList<>();

cardDeck1 = new ArrayList<>();
cardDeck2 = new ArrayList<>();
```