The “Poker Game” Kata

Sébastien Mosser

18.10.2017

pictures: sxc.hu

Simple Poll APP 1:24 PM

This poll is anonymous. The identity of all responses will be hidden.

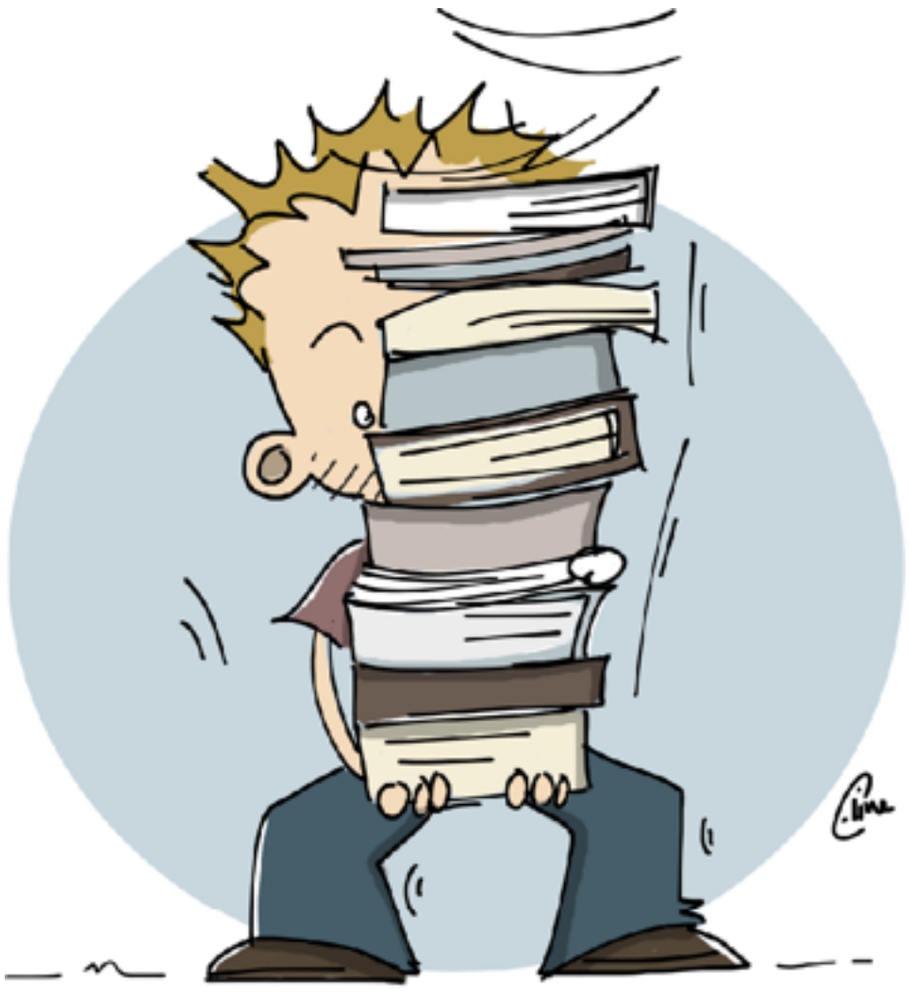
Une séance sur PokerGame

1 Oui, il faut préparer cette séance 33

2 on s'en schtroumpfe 2

1 2 Delete Poll

Simple Poll Edit Settings



Disclaimer

This lecture is mainly about **object-orientation**

Feature-slicing is **experimented** with Takenoko

Disclaimer #2

This lecture is a **walk-through**,

you might encounter **new concepts**.

It complements other courses

Problem Description

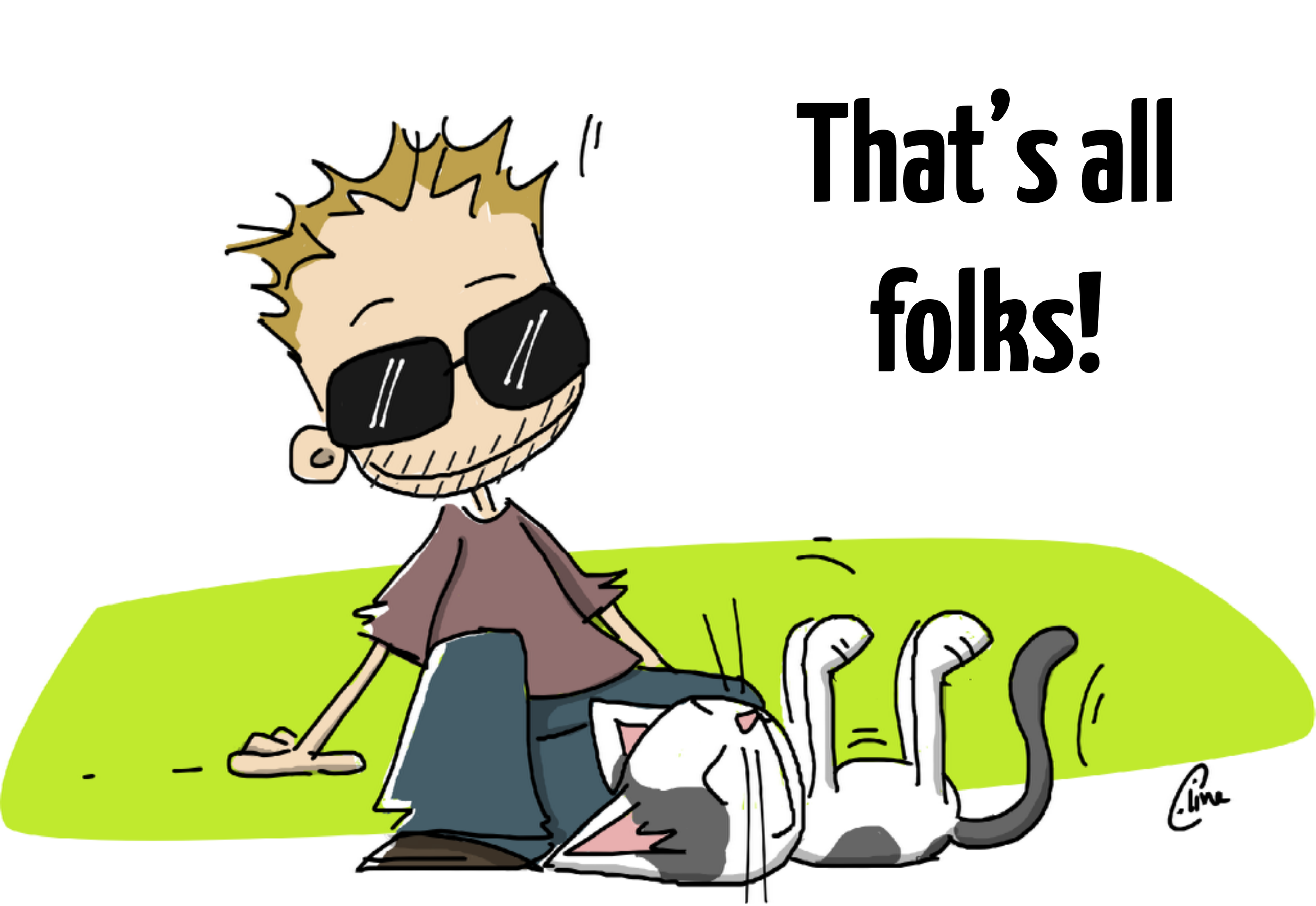
> “Your job is to compare several pairs of poker hands and to indicate which, if either, has a higher rank.

http://codingdojo.org/kata/PokerHands/

Minimal & Viable Product

- Read simple cards from the CLI (e.g., “3 5 7 2”)
- Elect the winner using the “Highest Card” rule
- **That’s all folks**

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Black:");
    String[] line1 = scanner.nextLine().split(" ");
    System.out.println("White:");
    String[] line2 = scanner.nextLine().split(" ");
    Arrays.sort(line1);
    Arrays.sort(line2);
    if(line1[line1.length-1].compareTo(line1[line2.length-1]) > 0 ) {
        System.out.println("Black wins with: " + line1[line1.length-1]);
    } else {
        System.out.println("White wins with " + line2[line2.length-1]);
    }
}
```


That's all folks!

Problems?

Readability?

Compliance with the specs?

Maintainability?

Testability?

Extension to fulfil the specs?

Technical Debt!

```
else if (h1.getType().equals("double paire") && h2.getType().equals("double paire")) {
    if (h1.typeHand.get(0).getValue().ordinal() > h2.typeHand.get(0).getValue().ordinal())
        this.handNumber = 1;
    else if (h1.typeHand.get(0).getValue().ordinal() == h2.typeHand.get(0).getValue().ordinal()) {
        if (h1.typeHand.get(1).getValue().ordinal() > h2.typeHand.get(1).getValue().ordinal()) {
            this.handNumber = 1;
        } else if (h1.typeHand.get(1).getValue().ordinal() == h2.typeHand.get(1).getValue().ordinal()) {
            if (h1.typeHand.get(2).getValue().ordinal() > h2.typeHand.get(2).getValue().ordinal()) {
                this.handNumber = 1;
            } else if (h1.typeHand.get(2).getValue().ordinal() == h2.typeHand.get(2).getValue().ordinal()) {
                this.handNumber = 0;
            } else this.handNumber = 2;
        } else {
            this.handNumber = 2;
        }
    } else this.handNumber = 2;
}
```

Real code, from real project!

TARGET

Software engineering rule of three

Software engineering rule of three

Readability

Software engineering rule of three

Readability

Readability

Software engineering rule of three

Readability

Readability
Readability

Focusing on readability

```
public static void main(String[] args) {
    HandReader reader = new HandReader(System.in);
    Hand black = reader.obtainForPlayer("Black");
    Hand white = reader.obtainForPlayer(“White");

    Referee referee = new Referee();
    GameResult result = referee.decide(black, white);

    System.out.println(result);
}
```

WAT? No getters? no setters?

Are we coding in Java?

It does not even compile!

“One of the **great leaps in OO** is to be able to answer the question “**how does this work?**” with “**I don’t care**”.

- Alan Knight

```
GameResult result = referee.decide(black, white);
System.out.println(result);
```

How does the referee decides?

```
GameResult result = referee.decide(black, white);
System.out.println(result);
```

How does the referee decides?

I don't care

```
GameResult result = referee.decide(black, white);
System.out.println(result);
```

How does the referee decides?

I don't care

How does the game result is printed?

```
GameResult result = referee.decide(black, white);
System.out.println(result);
```

How does the referee decides?

I don't care

How does the game result is printed?

I don't care

```
GameResult result = referee.decide(black, white);
System.out.println(result);
```

How does the referee decides?

I don't care

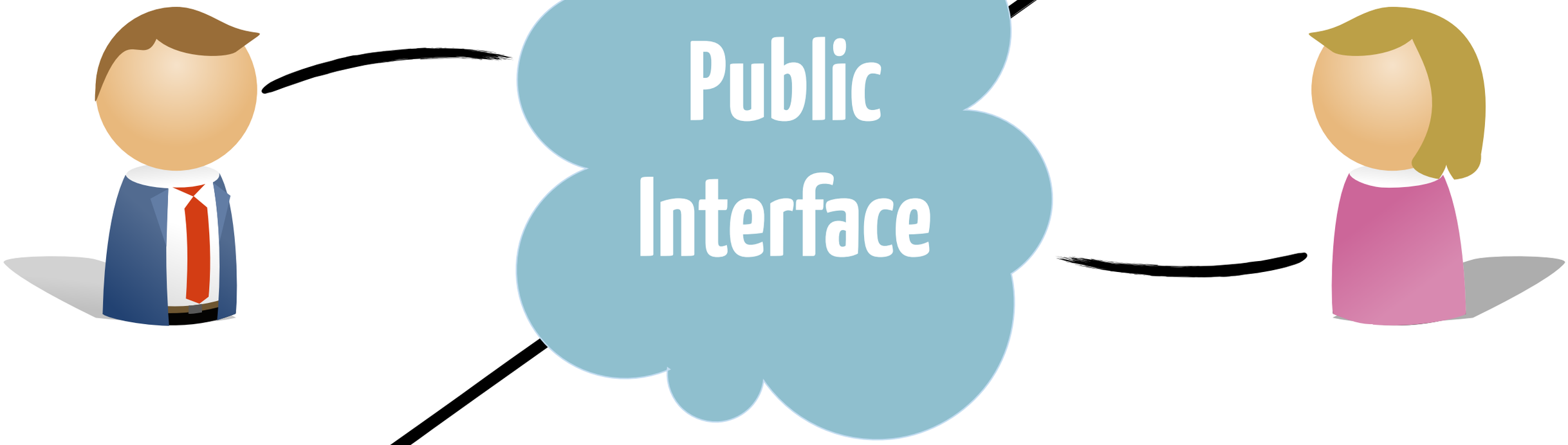
How does the game result is printed?

I don't care

You already got it, isn't it?

```
Hand black = reader.obtainForPlayer("Black");
```

As Bob, class consumer



As Alice, class designer

```
System.out.println("Black:");
String[] line1 = scanner.nextLine().split(" ");
```

SOLIDity applied to OO code (& Jedis)

- S : Single Responsibility
- O: Open / closed principle

- L: Liskov-compliant (substitution)
- I: Interface Segregation
- D: Dependency inversion

Maintenance: Open/Closed principle

Closed for **Modification**

Open for **Extension**

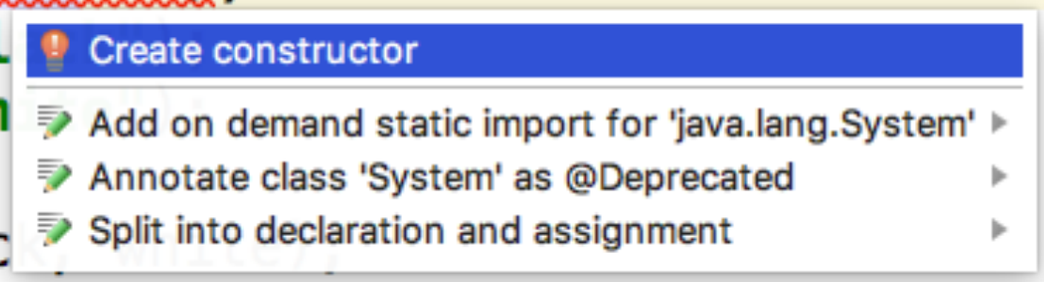
Leveraging an IDE

```
public static void main(String[] args) {
    HandReader reader = new HandReader(System.in);
    Hand b                                    Player("Black");
    Hand w                                    Player("White");
    Refere                                    ();
    GameRe                                    ecide(black, white);
    System
}
```

Create class 'HandReader'
Create enum 'HandReader'
Create inner class 'HandReader'
Create interface 'HandReader'
Add Maven Dependency...
Split into declaration and assignment

```
public class HandReader {}
```

```
public static void main(String[] args) {
    HandReader reader = new HandReader(System.in);
    Hand black = reader.obtainForPlayer("Bl
    Hand white = reader.obtainForPlayer("Wh
    Referee referee = new Referee();
    GameResult result = referee.decide(blac
    System.out.println(result);
}
```




```
public class HandReader {

    public HandReader(InputStream in) {}

}
```

```java
public static void main(String[] args) {
    HandReader reader = new HandReader(System.in);
    Hand black = reader.obtainForPlayer("Black");
    Hand white = reader.obtainForPl
    Referee referee = new Referee()
    GameResult result = referee.decide(black, white);
    System.out.println(result);
}
```

Create method 'obtainForPlayer'

Split into declaration and assignment

```java
public class HandReader {

    public HandReader(InputStream in) {}

    public Hand obtainForPlayer(String playerName) {
        throw new UnsupportedOperationException();
    }
}
```

```
azrael:3A-PokerGame mosser$ mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building PS5 :: PokerGame 1.0
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ poker-game ---
[INFO] Deleting /Users/mosser/work/polytech/3A-PokerGame/target
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ poker-game ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ poker-game ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 5 source files to /Users/mosser/work/polytech/3A-PokerGame/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ poker-game ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ poker-game ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ poker-game ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ poker-game ---
[INFO] Building jar: /Users/mosser/work/polytech/3A-PokerGame/target/poker-game.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.819 s
[INFO] Finished at: 2017-10-17T14:53:59+02:00
[INFO] Final Memory: 18M/304M
[INFO] ------------------------------------------------------------------------
```

Now it compiles.

So what?

Feature orientation!

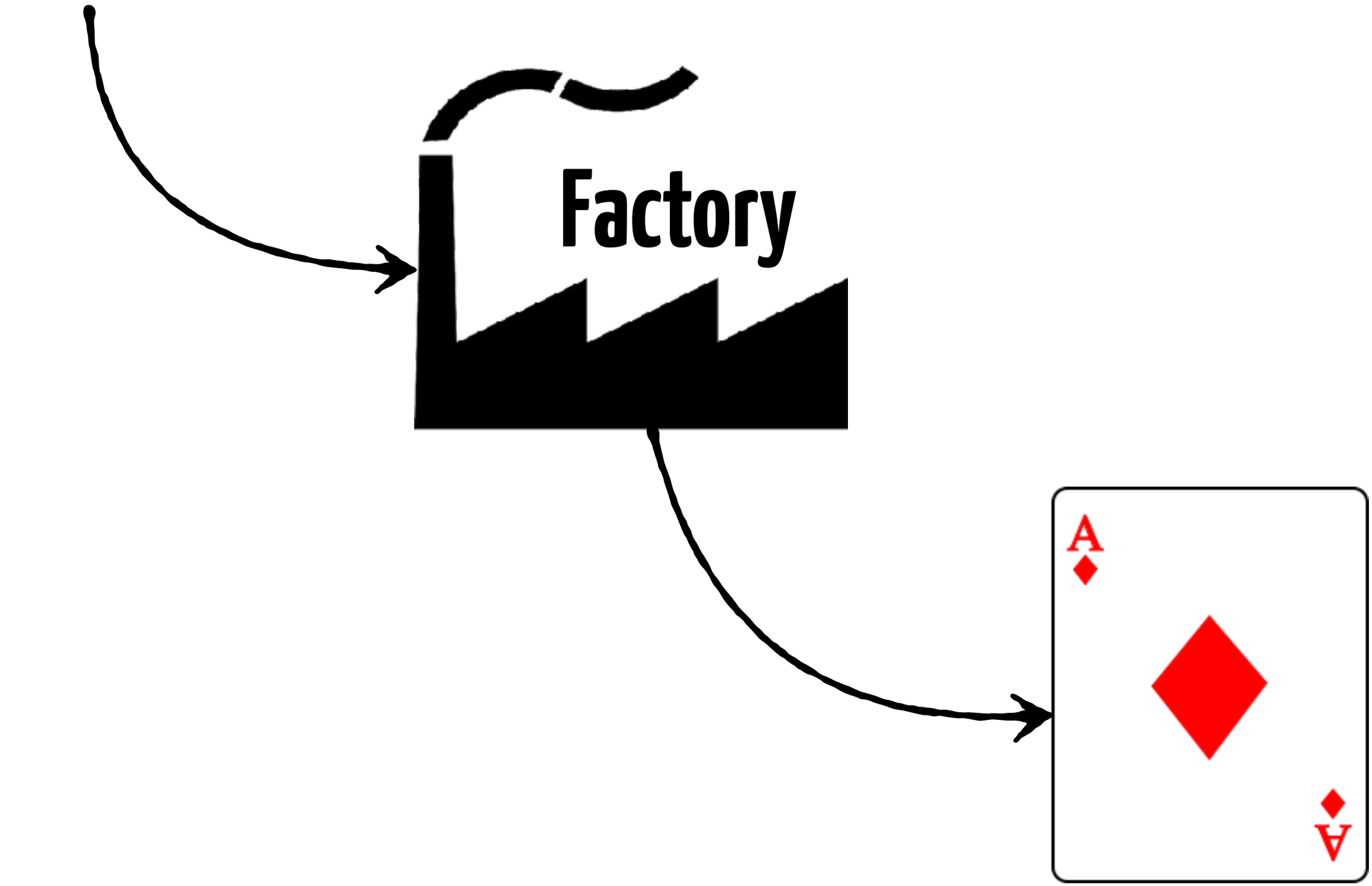
here feature are “technical”

F_1: Reading a Poker Hand

From “AD” to “Ace of Diamonds”

Abstraction: “Factory”

“AD”



```java
public class HandReader {

    private InputStream stream;
    private HandFactory factory;

    public HandReader(InputStream in) {
        this.stream = in;
        this.factory = new HandFactory();
    }

    public Hand obtainForPlayer(String playerName) {
        Scanner scanner = new Scanner(stream);
        System.out.println("Player " + playerName + ": ");
        String data = scanner.nextLine();
        Set<Card> cards = factory.transform(data);
        return new Hand(playerName, cards);
    }
}
```

```java
public enum CardValue {

    TWO(  '2', 2),  THREE('3', 3), FOUR( '4', 4),
    FIVE( '5', 5),  SIX(  '6', 6), SEVEN('7', 7),
    EIGHT('8', 8),  NINE( '9', 9), TEN(  'T', 10),
    JACK( 'J', 11), QUEEN('Q', 12), KING('K', 13);

    private final char symbol;
    private final int value;
    public int getValue() { return value; }

    CardValue(char symbol, int value) {
        this.symbol = symbol;
        this.value = value;
    }


    public static CardValue read(char c) {
        for(CardValue cv: CardValue.values()) {
            if(cv.symbol == c)
                return cv;
        }
        throw new IllegalArgumentException("Unknown card value: "+c);
    }

}
```

Order Relation mapped to Integer's one

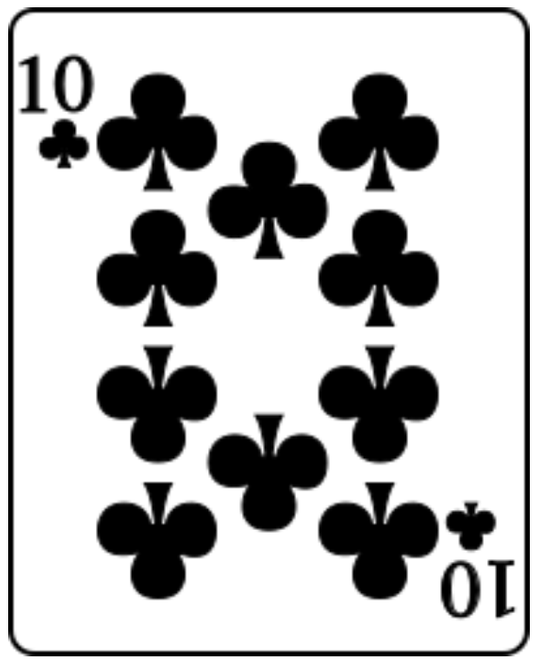
Char -> CardValue translation

```java
public enum Suit {

    CLUBS('C', "Clubs"),  DIAMONDS('D', "Diamonds"),
    HEARTS('H', "Hearts"), SPADES(  'S', "Spades");

    private final char symbol;
    private final String name;
    public String getName() { return name; }

    Suit(char c, String n) {
        this.name = n;
        this.symbol = c;
    }

    public static Suit read(char c) {
        return
          Arrays.stream(Suit.values())
            .filter((Suit s) -> s.symbol == c)
            .findFirst()
            .orElseThrow(() ->
              new IllegalArgumentException("Unknown suit symbol: " + c));
    }
}
```

Char -> Suit translation, using λ

```java
public Card build(String scalar) {
    if(scalar.length() != 2)
        throw new IllegalArgumentException("The data must be exactly two characters");
    CardValue value = CardValue.read(scalar.charAt(0));
    Suit suit =  Suit.read(scalar.charAt(1));
    return new Card(value, suit);
}
```

```java
public class Card {

    private CardValue value;
    private Suit suit;


    public Card(CardValue value, Suit suit) {
        this.value = value;
        this.suit = suit;
    }

    @Override
    public boolean equals(Object o) { … }

    @Override
    public int hashCode() { … }
}
```



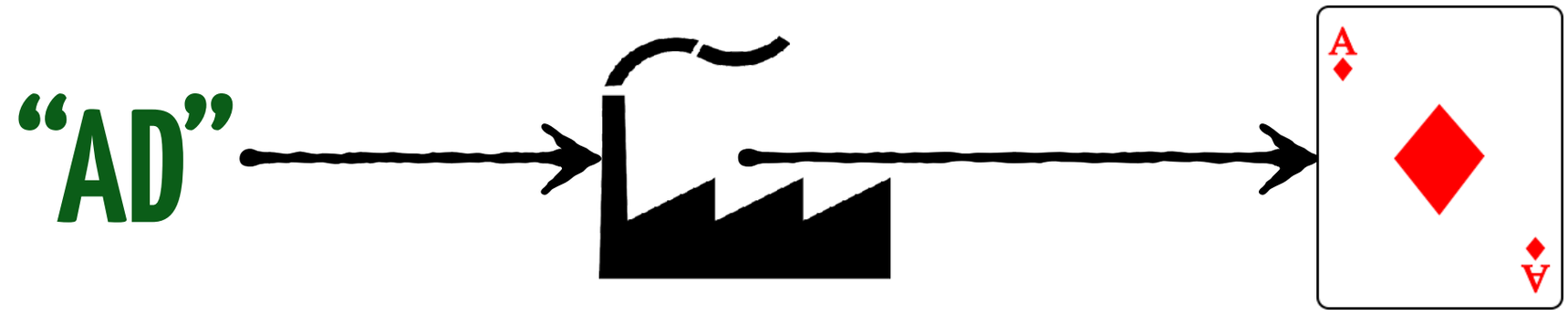
Wait!

Where are the tests?

```java
@Test public void buildClassicalCards() {
    Card queenOfHearts = factory.build("QH");
    assertEquals(new Card(QUEEN, HEARTS), queenOfHearts);

    Card tenOfDiamonds = factory.build("TD");
    assertEquals(new Card(TEN, DIAMONDS), tenOfDiamonds);

    Card twoOfSpades = factory.build("2S");
    assertEquals(new Card(TWO, SPADES), twoOfSpades);

    Card eightOfClubs = factory.build("8C");
    assertEquals(new Card(EIGHT, CLUBS), eightOfClubs);
}
```

Any Issues?


```
@Test public void buildAceOfDiamonds() {
    Card aceOfDiamonds = factory.build("AD");
}
```




```
@Test public void buildAceOfDiamonds() {
    Card aceOfDiamonds = factory.build("AD");
}
```

```
java.lang.IllegalArgumentException: Unknown card value: A

    at poker.cards.CardValue.read(CardValue.java:26)
    at poker.io.HandFactory.build(HandFactory.java:16)
```

```
public enum CardValue {

    TWO(  '2', 2),  THREE('3', 3), FOUR( '4', 4),
    FIVE( '5', 5),  SIX(  '6', 6), SEVEN('7', 7),
    EIGHT('8', 8),  NINE( '9', 9), TEN(  'T', 10),
    JACK( 'J', 11), QUEEN('Q', 12), KING('K', 13);
}
```

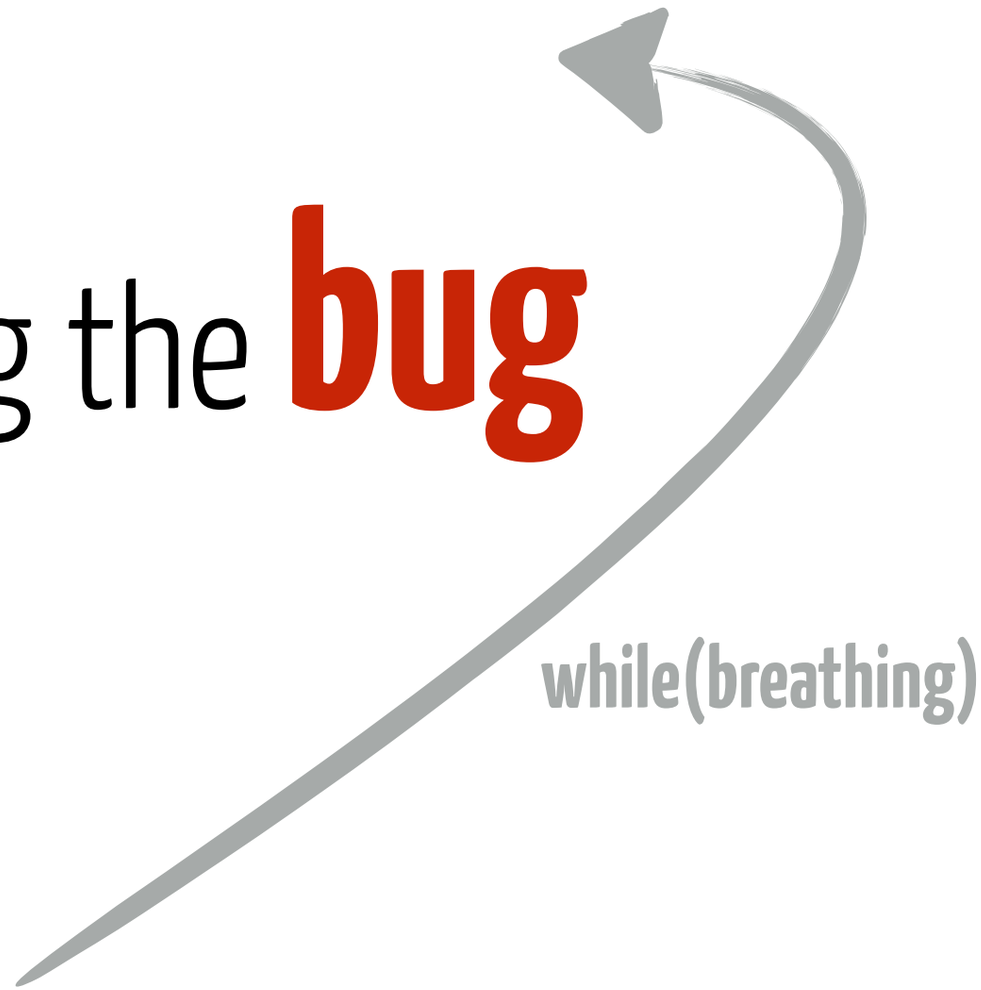
```java
@Test public void buildAllCards() {
    Set<Card> deck = new HashSet<>();
    for(CardValue val: CardValue.values()) {
        for(Suit suit: Suit.values()) {
            Card c = factory.build(""+ val.getSymbol() + suit.getSymbol());
            deck.add(c);
        }
    }
    assertEquals(52, deck.size());
}
```

Bug

=> **red Test** reproducing the **bug**

=> Code **fix**

=> **Test** is now **green**



```java
@Test(expected = IllegalArgumentException.class)
public void detectBadCardValueSymbol() { factory.build("UD"); }

@Test(expected = IllegalArgumentException.class)
public void detectBadSuitSymbol() { factory.build("2X"); }

@Test(expected = IllegalArgumentException.class)
public void detectBadSuitAndValueSymbols() { factory.build("UX"); }
```

```
@Test(expected = IllegalArgumentException.class)
public void detectBadCardValueSymbol() { factory.build("UD"); }

@Test(expected = IllegalArgumentException.class)
public void detectBadSuitSymbol() { factory.build("2X"); }

@Test(expected = IllegalArgumentException.class)
public void detectBadSuitAndValueSymbols() { factory.build("UX"); }
```

We cannot test which exception was thrown

```java
@Test(expected = IllegalArgumentException.class)
public void detectBadCardValueSymbol() { factory.build("UD"); }

@Test(expected = IllegalArgumentException.class)
public void detectBadSuitSymbol() { factory.build("2X"); }

@Test(expected = IllegalArgumentException.class)
public void detectBadSuitAndValueSymbols() { factory.build("UX"); }
```

We cannot test which exception was thrown

Do we (really) care?

```java
public Set<Card> transform(String data) {
    Set<Card> result = new HashSet<>();
    String[] vector = data.split(" ");
    for(String scalar: vector)
        result.add(build(scalar));
    return result;
}
```

```java
@Test public void buildHand() {
    Set<Card> expected = new HashSet<>(
            Arrays.asList(
                    new Card(QUEEN, DIAMONDS),
                    new Card(TEN, SPADES),
                    new Card(TWO, CLUBS),
                    new Card(KING, DIAMONDS),
                    new Card(THREE, CLUBS)
    ));
    assertEquals(expected, factory.transform("QD TS 2C KD 3C"));
    assertEquals(expected, factory.transform("TS QD 3C KD 2C"));
}
```

are we testing it RIGHT?

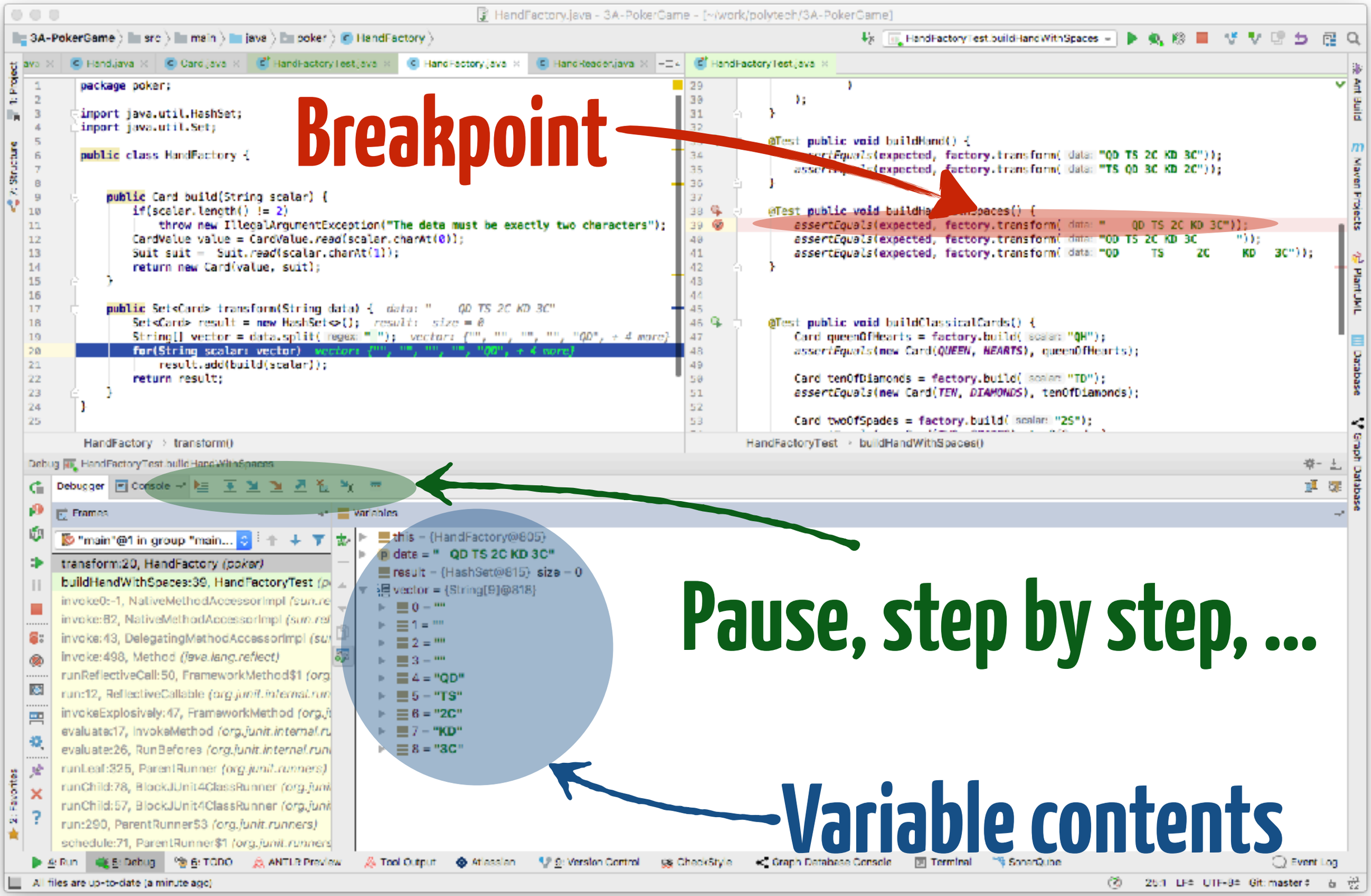
```
@Before public void initExpected() {
    expected = new HashSet<>(
            Arrays.asList(
                    new Card(QUEEN, DIAMONDS),
                    new Card(TEN, SPADES),
                    new Card(TWO, CLUBS),
                    new Card(KING, DIAMONDS),
                    new Card(THREE, CLUBS)
            )
    );
}

@Test public void buildHand() {
    assertEquals(expected, factory.transform("QD TS 2C KD 3C"));
    assertEquals(expected, factory.transform("TS QD 3C KD 2C"));
}

@Test public void buildHandWithSpaces() {
    assertEquals(expected, factory.transform("    QD TS 2C KD 3C"));
    assertEquals(expected, factory.transform("QD TS 2C KD 3C      "));
    assertEquals(expected, factory.transform("QD     TS     2C     KD   3C"));
}
```

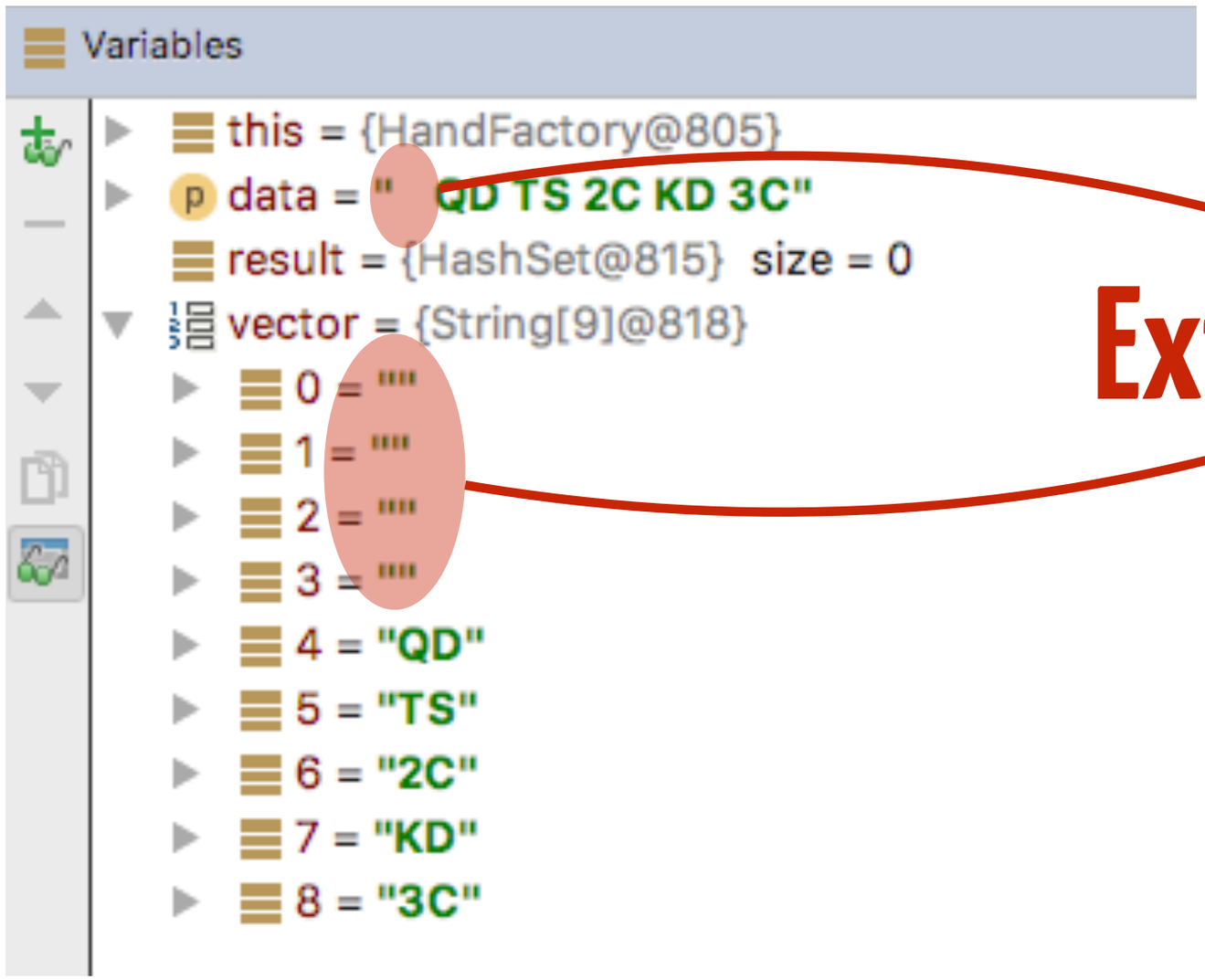
are we testing it RIGHT?

```
@Before public void initExpected() {
    expected = new HashSet<>(
            Arrays.asList(
                    new Card(QUEEN, DIA
                    new Card(TEN, S
                    new Card(TH
                    new Car
                    new
            )
    );
}

@Test
    ass
    asse

@Test publi void buildHandWithSpaces() {
    assertEquals(expected, factory.transform("    QD TS 2C KD 3C"));
    assertEquals(expected, factory.transform("QD TS 2C KD 3C      "));
    assertEquals(expected, factory.transform("QD     TS     2C     KD   3C"));
}
```

```
java.lang.IllegalArgumentException: The data must be exactly two characters
at poker.HandFactory.build(HandFactory.java:11)
at poker.HandFactory.transform(HandFactory.java:21)
at poker.HandFactoryTest.buildHandWithSpaces(HandFactoryTest.java:40)
```

The "Debug" mode (aka entering the matrix)





Extra white spaces !

Remove it!

```
public Set<Card> transform(String data) {
    Set<Card> result = new HashSet<>();
    String cleaned = data.replaceAll("\\s+"," ").trim();
    String[] vector = cleaned.split(" ");
    for(String scalar: vector)
        result.add(build(scalar));
    return result;
}
```

Aprendiz de brujo

Use tools you understand!

(println are fine for noobs, for now)

and try new!

```java
@Test public void buildBigHand() {
    assertEquals(6, factory.transform("JC QD TS 2C KD 3C").size());
}

@Test public void buildSmallHand() {
    assertEquals(4, factory.transform("TS 2C KD 3C").size());
}
```

6 Cards in a Hand?

4 Cards in a Hand?

Who is responsible?

F_2: Validating Hands

Only 5 cards, no duplicates

“Only 5 cards” == Hand intrinsic definition

```
public class Hand {

    private String playerName;
    private Set<Card> cards;

    public Hand(String playerName, Set<Card> cards) {
        this.playerName = playerName;
        if (cards.size() != 5)
            throw new IllegalArgumentException("A hand contains 5 cards!");
        this.cards = cards;
    }

}
```

```
@Test(expected = IllegalArgumentException.class)
public void checkHandWith4Cards() {
    contents.remove(new Card(QUEEN, DIAMONDS));
    Hand myHand = new Hand("Seb", contents);
}

@Test(expected = IllegalArgumentException.class)
public void checkHandWith6Cards() {
    contents.add(new Card(THREE, DIAMONDS));
    Hand myHand = new Hand("Seb", contents);
}
```

```
@Test(expected = IllegalArgumentException.class)
public void checkHandWith4Cards() {
    contents.remove(new Card(QUEEN, DIAMONDS));
    Hand myHand = new Hand("Seb", contents);
}

@Test(expected = IllegalArgumentException.class)
public void checkHandWith6Cards() {
    contents.add(new Card(THREE, DIAMONDS));
    Hand myHand = new Hand("Seb", contents);
}

@Test public void checkHandWith5Cards() {
    Hand myHand = new Hand("Seb", contents);
    assertEquals(true, true);
}
```

“No duplicates” == Referee’s knowledge

$$|cards_b| + |cards_w| = |cards_b \cup cards_w|$$

```
public boolean check(Hand black, Hand white) {
    Set<Card> all = black.getCards();
    all.addAll(white.getCards());
    return (all.size() == 10);
}
```

* can also be done using intersection

```
public class RefereeTest {

    private Referee referee;
    private Set<Card> blackContents;
    private Set<Card> whiteContents;

    @Before public void initReferee() { this.referee = new Referee(); }

    @Before public void initBlack() {
        blackContents = new HashSet<>(Arrays.asList(
                new Card(QUEEN, DIAMONDS), new Card(TEN, SPADES),
                new Card(TWO, CLUBS), new Card(KING, DIAMONDS),
                new Card(THREE, CLUBS)
        ));
    }

    @Before public void initWhite() {
        whiteContents = new HashSet<>(Arrays.asList(
                new Card(KING, SPADES),  new Card(TEN, DIAMONDS),
                new Card(THREE, DIAMONDS), new Card(KING, CLUBS),
                new Card(JACK, HEARTS)
        ));
    }

    @Test public void checkCorrectHands() {
        Hand black = new Hand("Black", blackContents);
        Hand white = new Hand(“White", whiteContents);
        assertTrue(referee.check(black, white));
    }
}
```

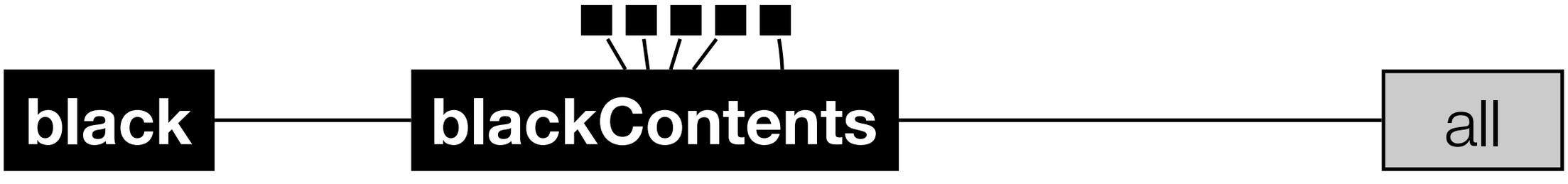
```
public class RefereeTest {

    private Referee referee;
    private Set<Card> blackContents;
    private Set<Card> whiteContents;

    @Before public void init                                  = new Referee(); }

    @Before public void init
        blackContents = new
                new Card(QUE                                  EN, SPADES),
                new Card(TWO                                  DIAMONDS),
                new Card(THR
        ));
    }

    @Before public void init
        whiteContents = new
                new Card(KIN                                  DIAMONDS),
                new Card(THR                                  NG, CLUBS),
                new Card(JAC
        ));
    }

    @Test public void checkCorrectHands() {
        Hand black = new Hand("Black", blackContents);
        Hand white = new Hand(“White", whiteContents);
        assertTrue(referee.check(black, white));
    }
}
```

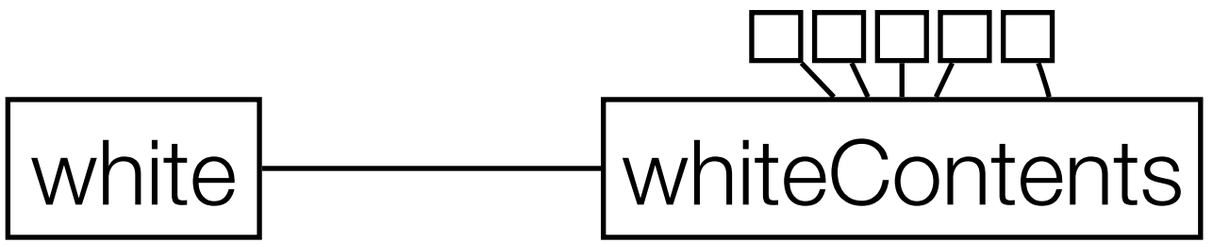
Wrong,

Wrong,

Wrong.

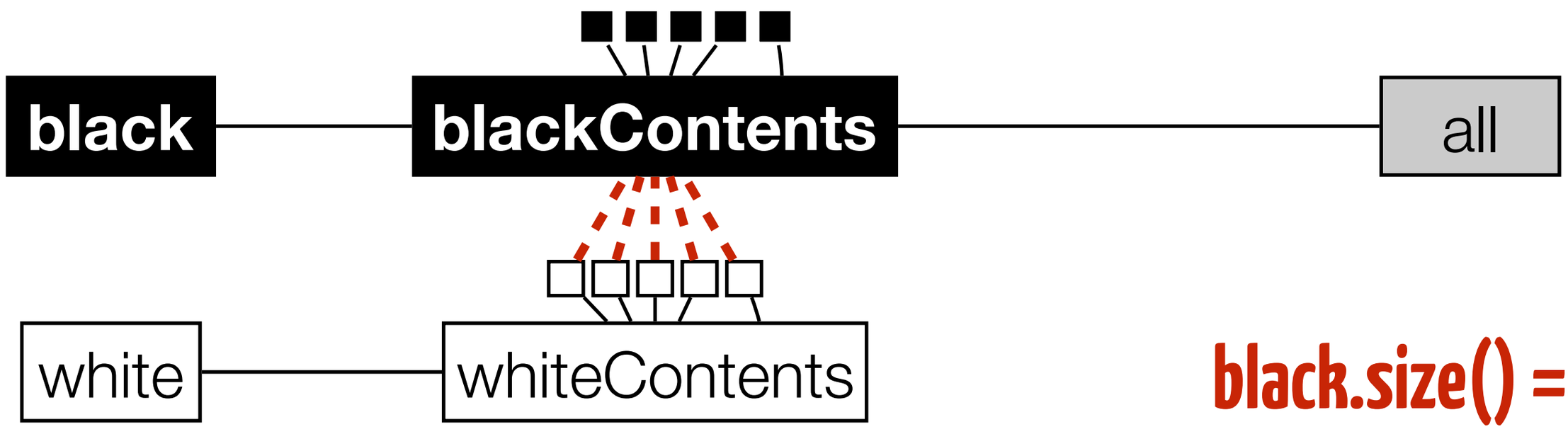
```java
@Test public void checkCorrectHands() {
    Hand black = new Hand("Black", blackContents);
    Hand white = new Hand("White",whiteContents);
    assertTrue(referee.check(black, white));
    assertEquals(5, black.getCards().size());
    assertEquals(5, white.getCards().size());
}
```






```java
public boolean check(Hand black, Hand white) {
    Set<Card> all = black.getCards();
    all.addAll(white.getCards());
    return (all.size() == 10);
}
```

```java
@Test public void checkCorrectHands() {
    Hand black = new Hand("Black", blackContents);
    Hand white = new Hand("White",whiteContents);
    assertTrue(referee.check(black, white));
    assertEquals(5, black.getCards().size());
    assertEquals(5, white.getCards().size());
}
```



black.size() == 10

```java
public boolean check(Hand black, Hand white) {
    Set<Card> all = black.getCards();
    all.addAll(white.getCards());
    return (all.size() == 10);
}
```

```java
public boolean check(Hand left, Hand right) {
    Set<Card> all = new HashSet<>(left.getCards());
    all.addAll(right.getCards());
    return (all.size() == 10);
}
```

Copy ≠ Reference !

```java
@Test public void checkCorrectHands() {
    Hand black = new Hand("Black", blackContents);
    Hand white = new Hand("White",whiteContents);
    assertTrue(referee.check(black, white));
    assertEquals(5, black.getCards().size());
    assertEquals(5, white.getCards().size());
}

@Test public void checkDuplicatedCardsInHands() {
    blackContents.remove(new Card(QUEEN, DIAMONDS));
    blackContents.add(new Card(JACK, HEARTS));
    Hand black = new Hand("Black", blackContents);
    Hand white = new Hand("White",whiteContents);
    assertFalse(referee.check(black, white));
}
```

F_3: Scoring Hands

How to compare two hands?

How to score “Hands”?

$$|\text{Hands}| = \quad ?$$

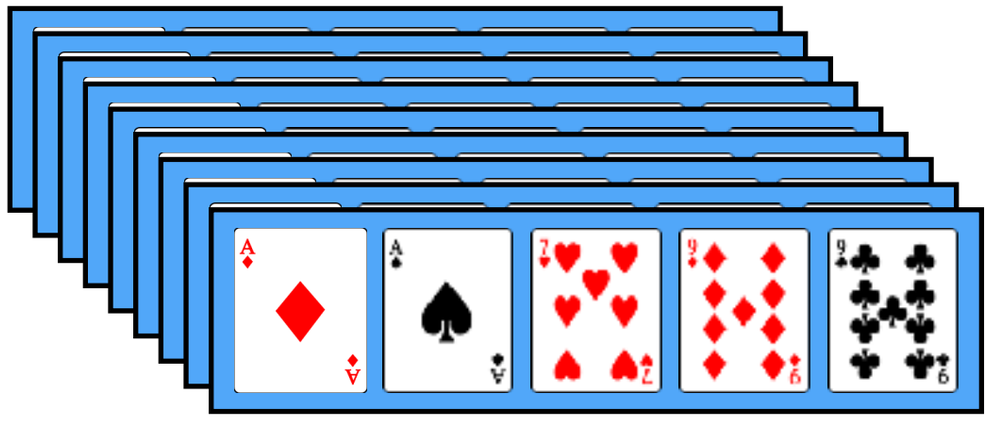
Hint: Reducing to a “simpler” problem,
e.g., an already solved one.

How to score “Hands”?

$$|\text{Hands}| = \binom{5}{52} = 2\,598\,960$$

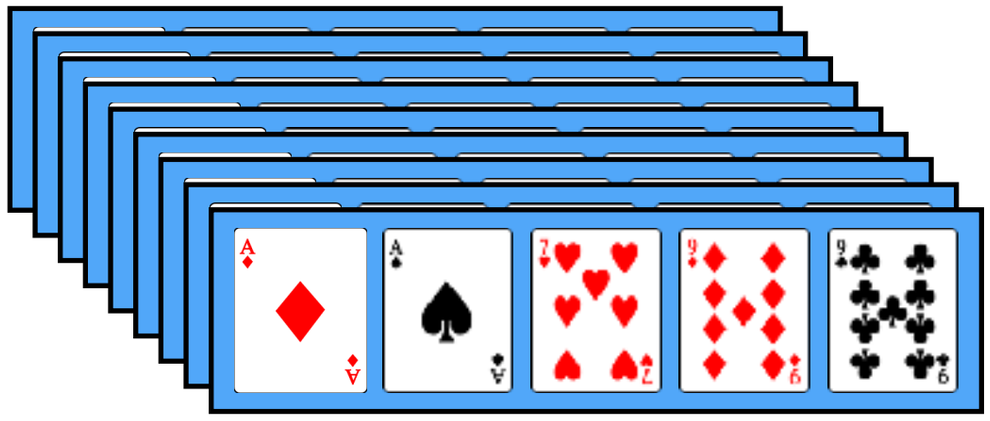
Hint: Reducing to a “simpler” problem, e.g., an already solved one.

Hands



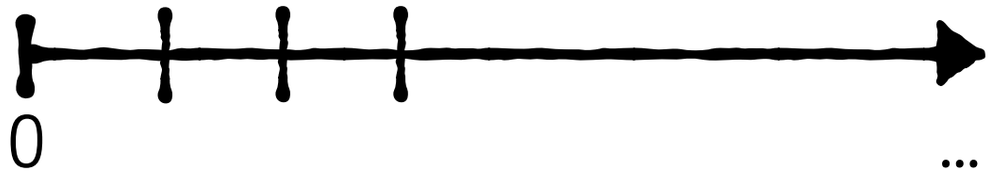
order relation: ?

Hands



order relation: ?

$\mathbb{N}$

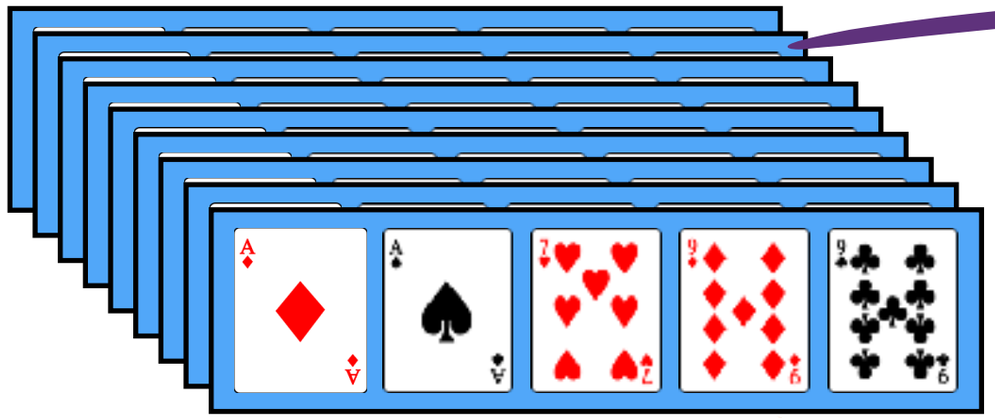


order relation: <

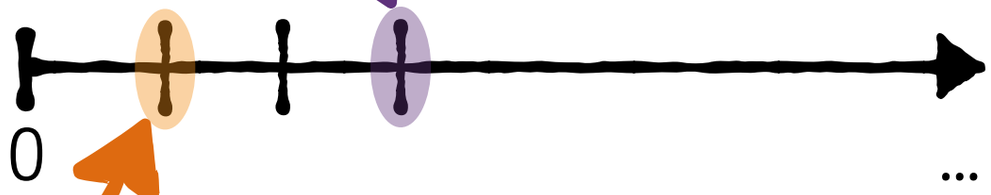
Hands

$\mathbb{N}$

h_2



h_1



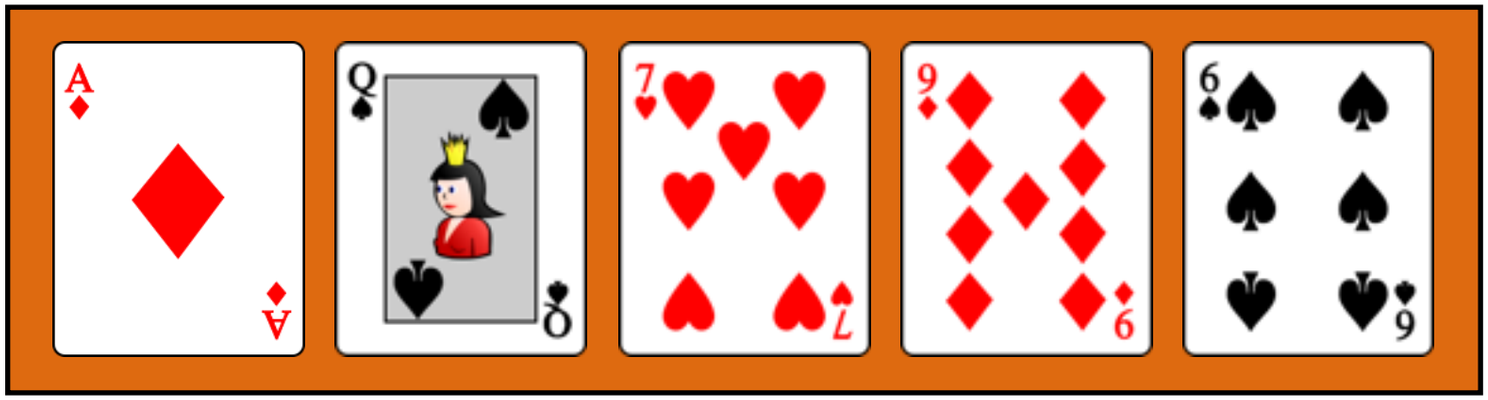
$score(h_2)$

$score(h_1)$

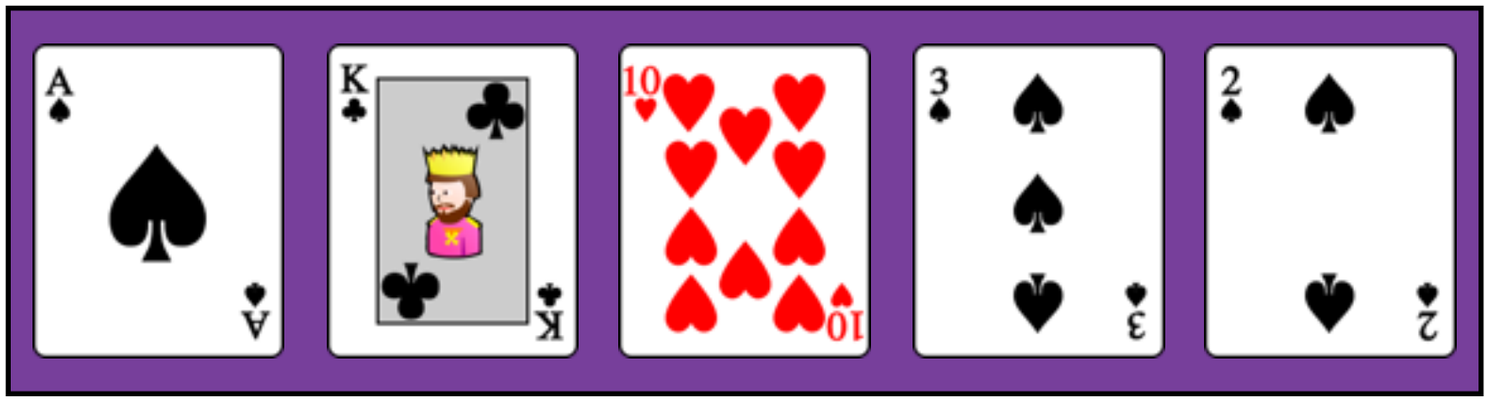
order relation: ?

order relation: <

h_1

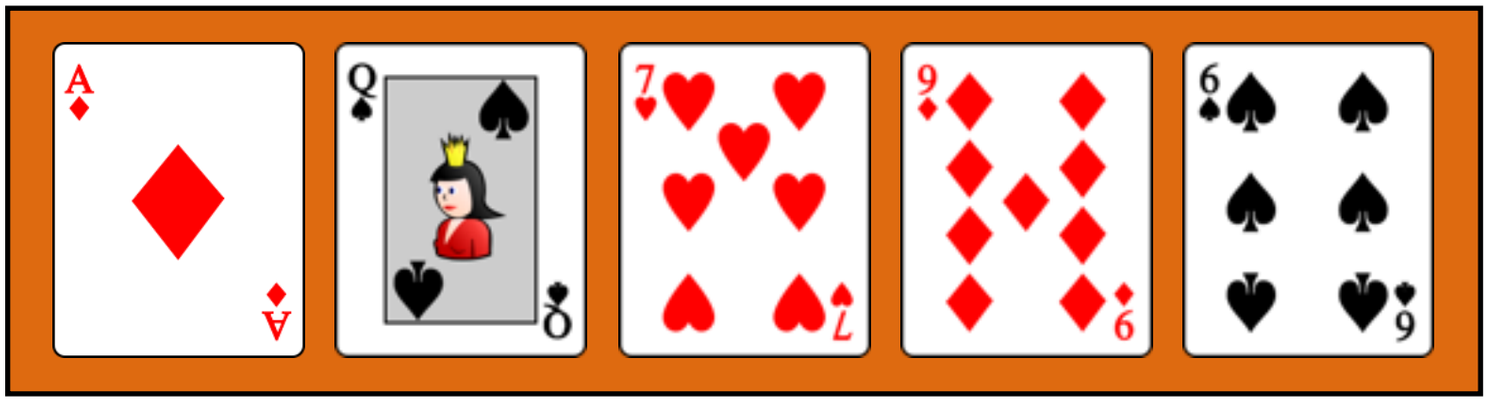


h_2



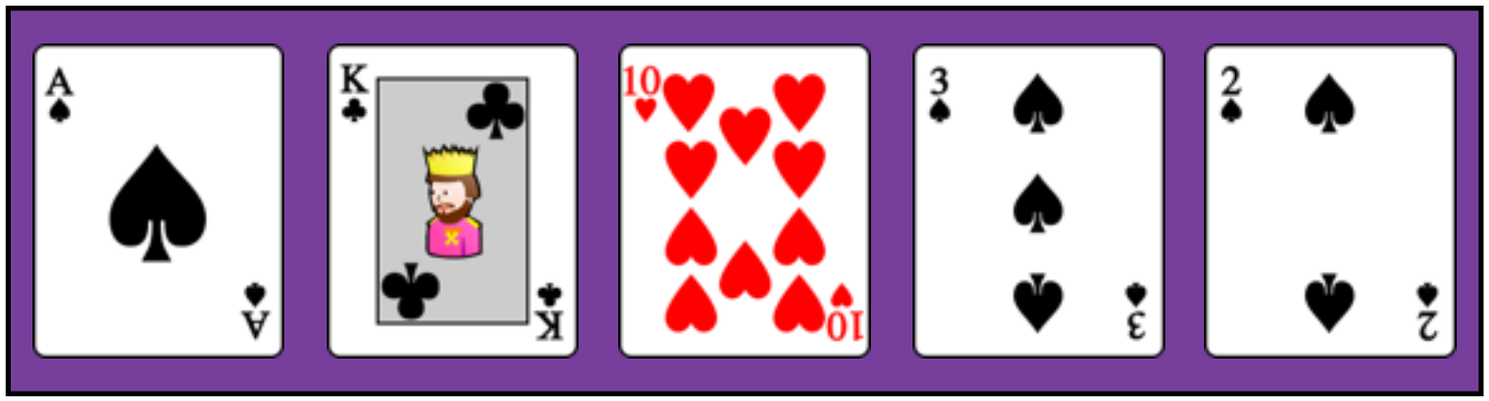
$h_2 > h_1$

h_1



14 12 7 9 6

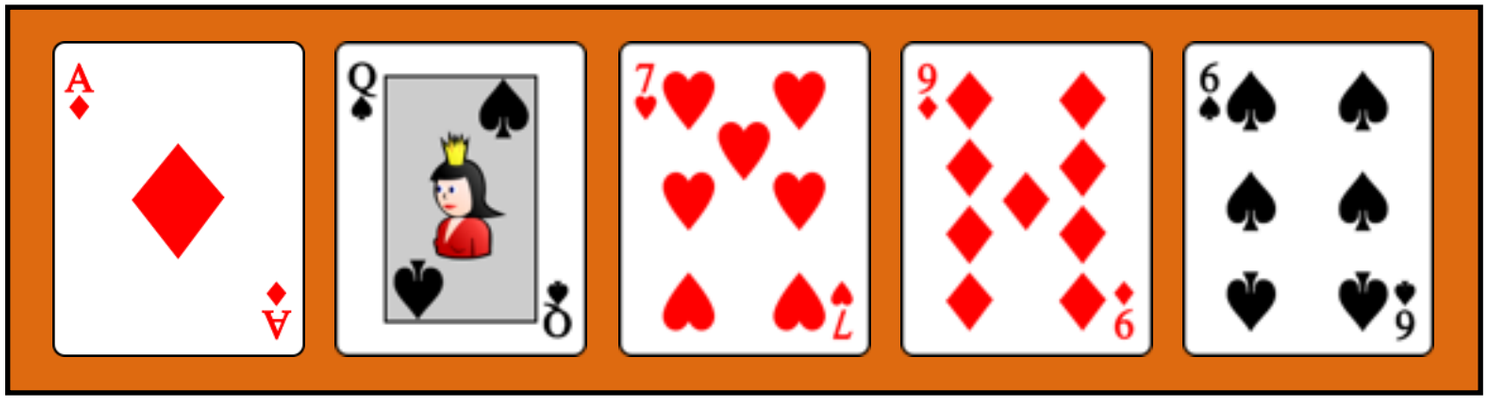
h_2



14 13 10 3 2

$h_2 > h_1$

h_1



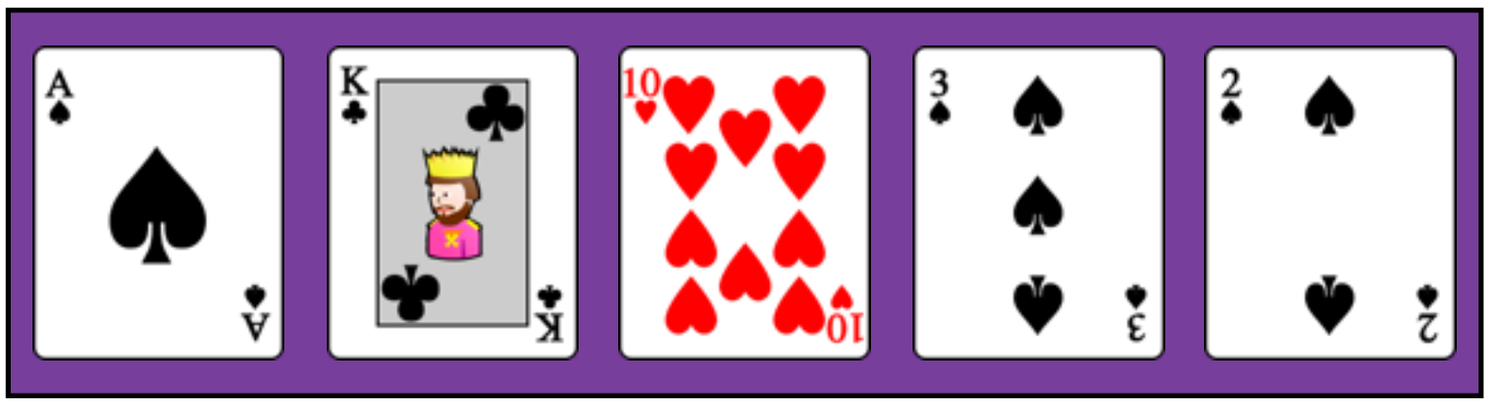
14 12 7 9 6

$score(h_1)$

$=14+12+7+9+6$

$= 42$

h_2



14 13 10 3 2

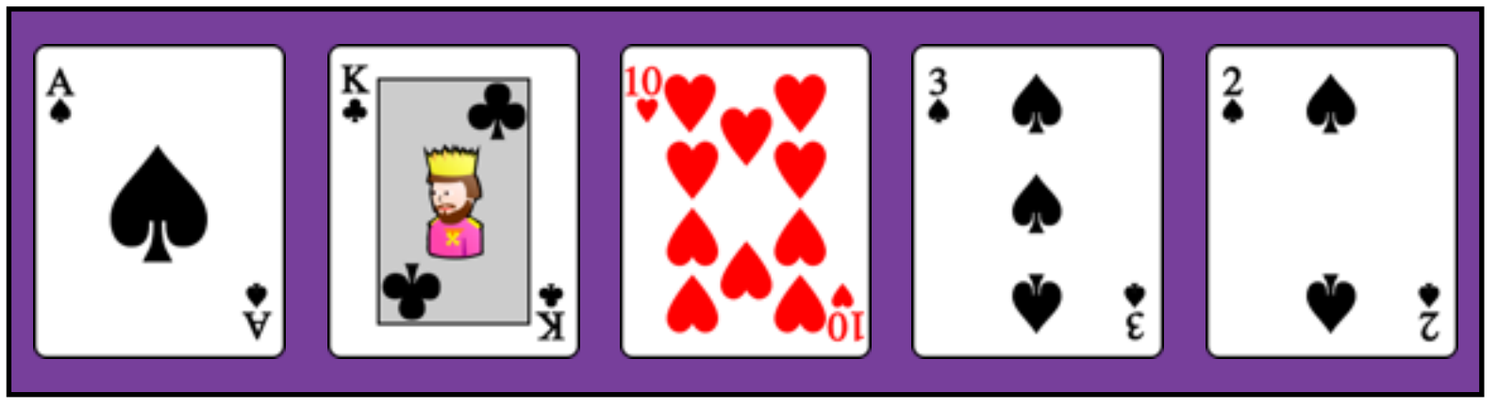
$score(h_2)$

$=14+13+10+3+2$

$= 42$

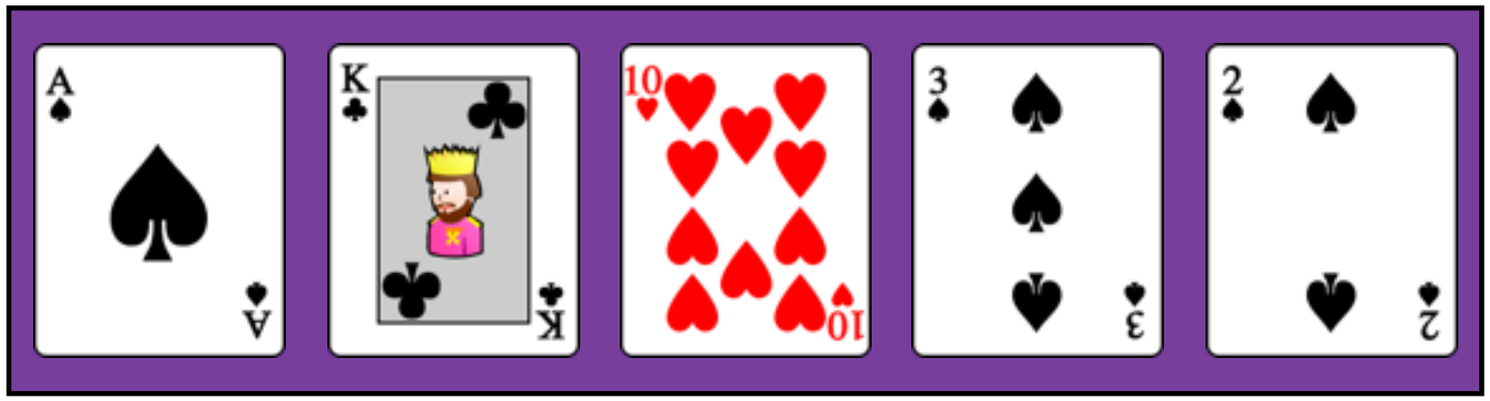
$h_2 > h_1$ $score(h_1) = score(h_2)$

h_2



$h_2 > h_1$

h_2



14 13 10 3 2

score'(h_2) =

2

+ 3 * 10

+ 10 * 100

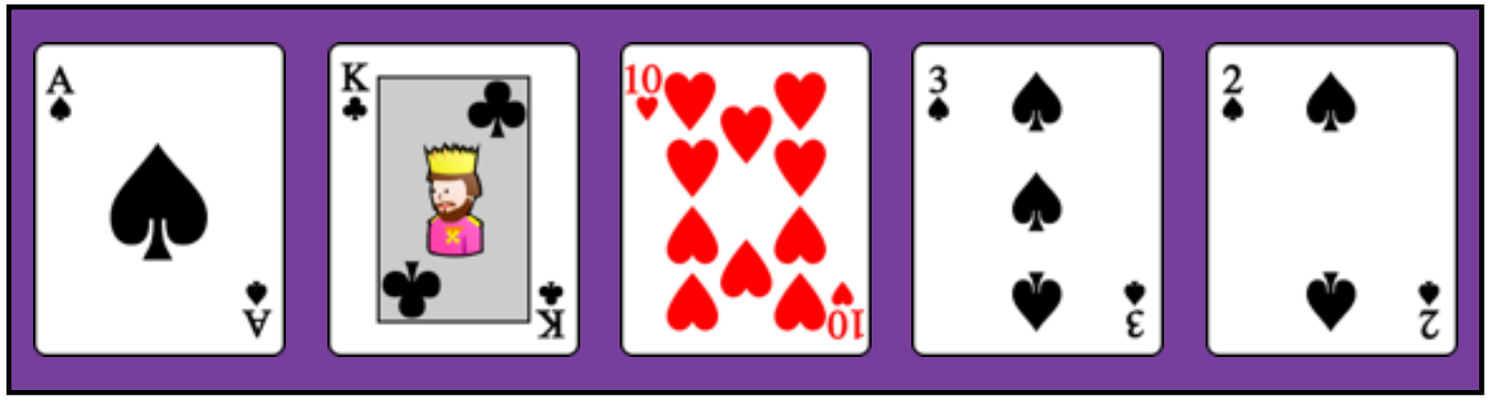
+ 13 * 1000

+ 14 * 10000

= 154,032

h_2 > h_1

h_2



14 13 10 3 2

$score'(h_1) = 152{,}976$

$h_2 > h_1$

$score'(h_2) > score'(h_1)$

$score'(h_2) =$

2

$+ 3 * 10$

$+ 10 * 100$

$+ 13 * 1000$

$+ 14 * 10000$

$= 154{,}032$

Sorting a Hand

Card

```
public class Card implements Comparable<Card> {
…
    @Override public int compareTo(Card that) {
        Integer thisValue = this.value.getValue();
        Integer thatValue = that.value.getValue();
        return thisValue.compareTo(thatValue);
    }
}
```

Hand

```
public List<Card> getOrderedCards() {
    Card[] raw = cards.toArray(new Card[cards.size()]);
    Arrays.sort(raw);
    return Arrays.asList(raw);
}
```

HandTest

```
@Test public void checkCardsSorting() {
    Hand myHand = new Hand("Seb", contents);
    List<Card> sorted = myHand.getOrderedCards();
    assertEquals(new Card(KING, DIAMONDS), sorted.get(4));
    assertEquals(new Card(QUEEN, DIAMONDS), sorted.get(3));
    assertEquals(new Card(TEN, SPADES), sorted.get(2));
    assertEquals(new Card(THREE, CLUBS), sorted.get(1));
    assertEquals(new Card(TWO, CLUBS), sorted.get(0));
}
```

Scoring a Hand

```
public int score() {
    List<Card> sorted = getOrderedCards();
    return  sorted.get(0).getValue().getValue() +
            sorted.get(1).getValue().getValue() * 10 +
            sorted.get(2).getValue().getValue() * 100 +
            sorted.get(3).getValue().getValue() * 1000 +
            sorted.get(4).getValue().getValue() * 10000;
}
```

```
@Test public void checkScore() {
    Hand myHand = new Hand("Seb", contents);
    assertEquals(143032, myHand.score());
}
```

Problem: Bijective hash

Scoring ~ Hashing a Hand

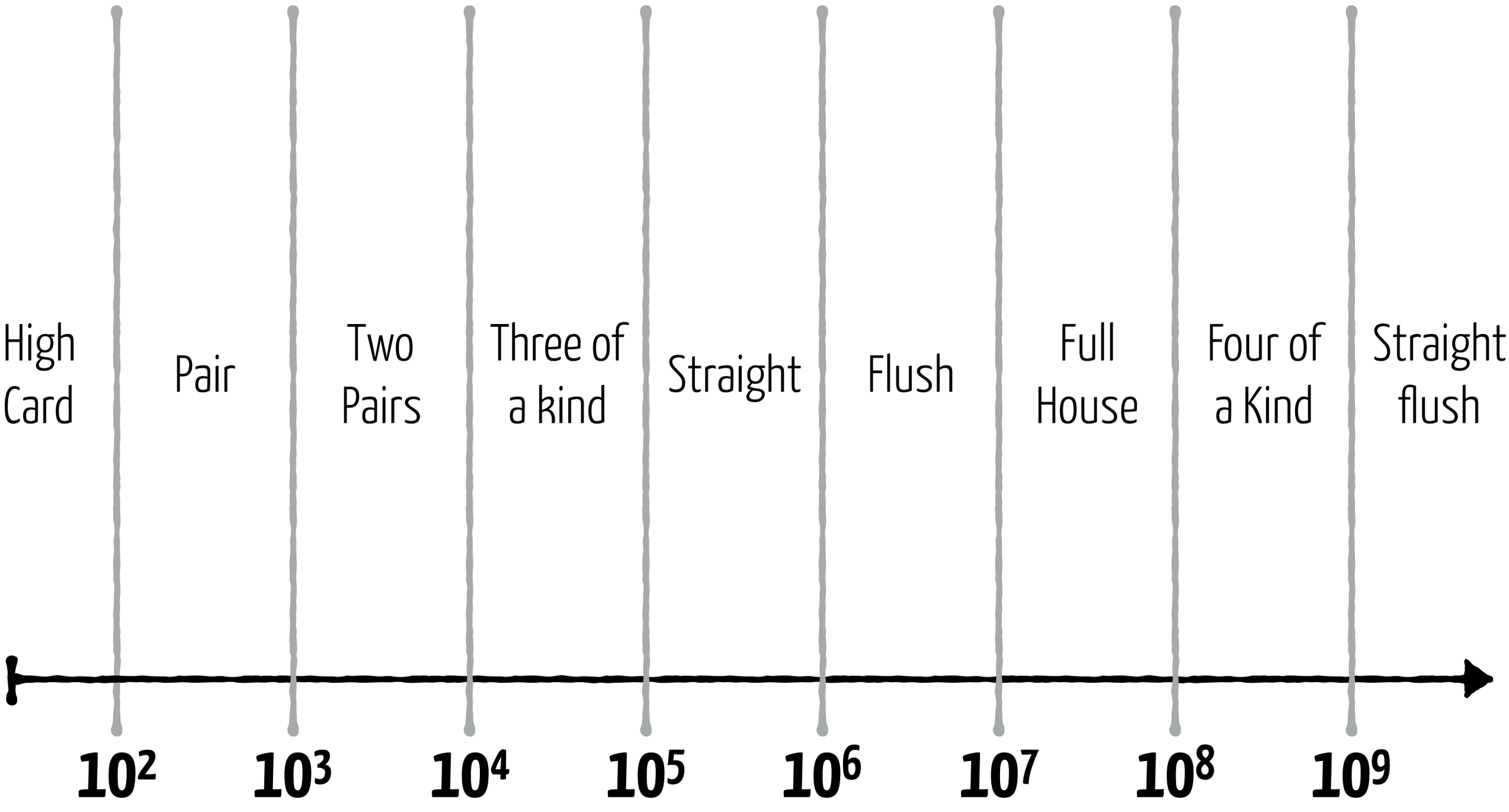
```
@Override
public int hashCode() {
    int result = value != null ? value.hashCode() : 0;
    result = 31 * result + (suit != null ? suit.hashCode() : 0);
    return result;
}
```

Intensively used by Hash* in Java

F_4: Detecting Combinations

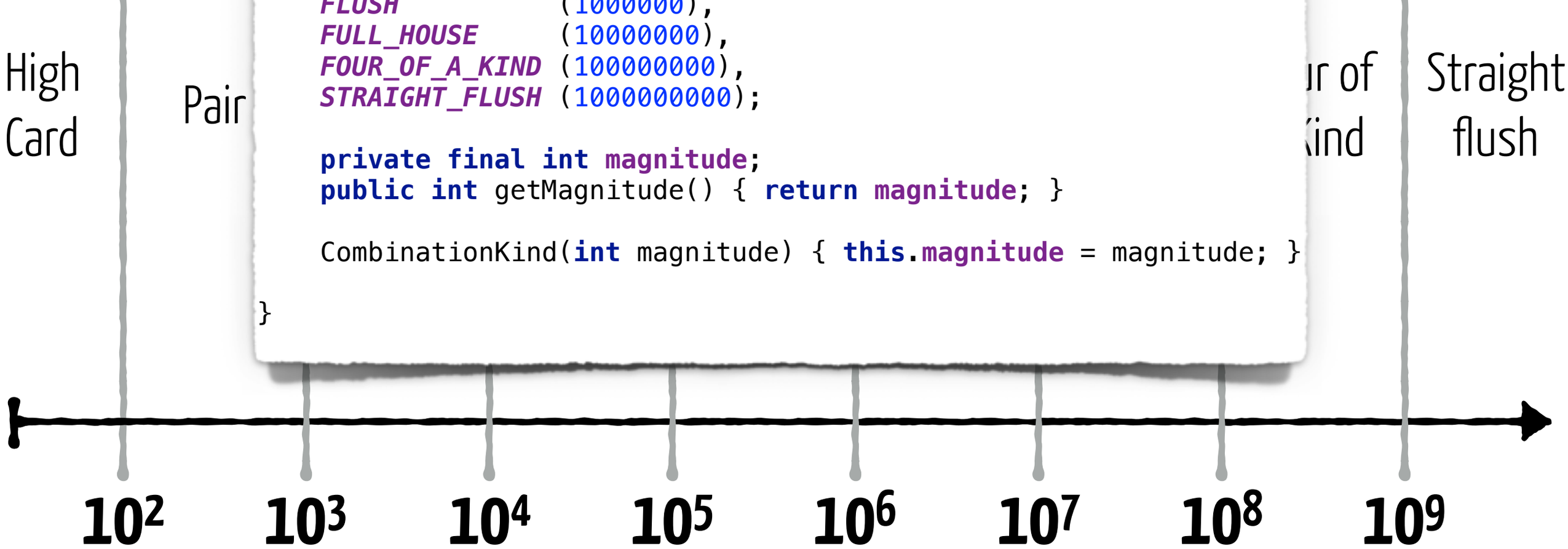
Highest card, Pair, Three of a ...

Scoring Combination instead of Hands



Scoring Combination instead of Hands

```
public enum CombinationKind {

    HIGH_CARD       (0),
    PAIR            (100),
    TWO_PAIRS       (1000),
    THREE_OF_A_KIND(10000),
    STRAIGHT        (100000),
    FLUSH           (1000000),
    FULL_HOUSE      (10000000),
    FOUR_OF_A_KIND (100000000),
    STRAIGHT_FLUSH (1000000000);

    private final int magnitude;
    public int getMagnitude() { return magnitude; }

    CombinationKind(int magnitude) { this.magnitude = magnitude; }

}
```



```java
public class Combination {

    private CombinationKind kind;
    private Set<Card> involvedCards = new HashSet<>();

    public int score() {
        Card[] sorted = involvedCards.toArray(new Card[involvedCards.size()]);
        Arrays.sort(sorted);
        int involved = 0;
        int powerOfTen = 1;
        for(int i = 0; i < sorted.length; i++) {
            Card current = sorted[i];
            involved += current.getFace().getValue() * powerOfTen;
            powerOfTen *= 10;
        }
        return kind.getMagnitude() + involved;
    }
}
```

```java
public class Combination {

    private CombinationKind kind;
    private Set<Card> involvedCards = new HashSet<>();

    public int score() {
        Card[] sorted = involvedCards.toArray(n          d[involvedCards.size()]);
        Arrays.sort(sorted);
        int involved = 0;
        int powerOfTen = 1;
        for(int i = 0; i < so          ngth; i++) {
            Card current          ed[i];
            involved          rent.getFace().getValue() * powerOfTen;
            powe          *= 10;
        }
        return kind.getMagnitude() + involved;
    }
}
```

Comparing cards => include remaining cards

Magic trick: Method extraction

```
public void fillWithRe
    for(Card c: hand.g
        if(! this.invo
            this.remai
}

public int score() {
    Card[] sorted = in
    Arrays.sort(sorted
    int involved = 0;
    int powerOfTen = 1
    for(int i = 0
        Card curr
        involved
        powerOfTe
    }
    return kind.g
}

@Override
public int compar
    Integer thisS
    Integer thatS
    int raw = thi
    if (raw != 0)
        return ra
```

Variable... ⌥⌘V
Constant... ⌥⌘C
Field... ⌥⌘F
Parameter... ⌥⌘P

Functional Parameter... ⌥⇧⌘P
Functional Variable...
Parameter Object...

Method... ⌥⌘M
Type Parameter...
Method Object...

Delegate...
Interface...
Superclass...
Subquery as CTE

Find Usages ⌥F7
Refactor ▶

Folding ▶

Safe Delete... ⌘⌫

Extract ▶
Inline... ⌥⌘N
Find and Replace Code Duplicates...
Invert Boolean...

Pull Members Up...
Push Members Down...
Use Interface Where Possible...
Replace Inheritance with Delegation...
Remove Middleman...
Wrap Method Return Value...

Convert Anonymous to Inner...
Encapsulate Fields...
Replace Temp with Query...
Replace Constructor with Factory Method...
Replace Constructor with Builder...

Generify...
Migrate...

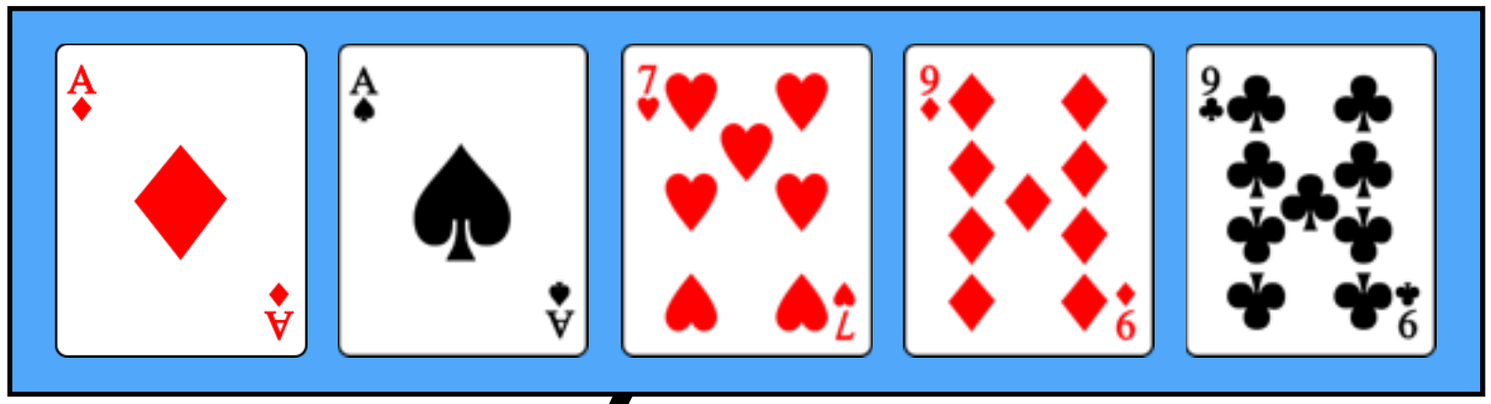
```java
public int score() {
    int involved = getIntegerValue(involvedCards);
    return kind.getMagnitude() + involved;
}

@Override public int compareTo(Combination that) {
    Integer thisScore = this.score();
    Integer thatScore = that.score();
    int raw = thisScore.compareTo(thatScore);
    if (raw != 0) { // Different combination !
        return raw;
    } else {        // Same Combination => using remaining cards
        thisScore = getIntegerValue(remainingCards);
        thatScore = getIntegerValue(that.remainingCards);
        return thisScore.compareTo(thatScore);
    }
}
```

```java
private int getIntegerValue(Collection<Card> cards) {
    Card[] sorted = cards.toArray(new Card[cards.size()]);
    Arrays.sort(sorted);
    int result = 0;
    int powerOfTen = 1;
    for(int i = 0; i < sorted.length; i++) {
        Card current = sorted[i];
        result += current.getFace().getValue() * powerOfTen;
        powerOfTen *= 10;
    }
    return result;
}
```

DRY!

```
private Combination c1;
@Before public void initCombination1() {
    Hand h = new Hand("p1", factory.transform("QD JH 5C 2H 7D"));
    c1 = new Combination(CombinationKind.HIGH_CARD);
    c1.addInvolvedCards(Arrays.asList(new Card(QUEEN, DIAMONDS)));
    c1.fillWithRemainingCards(h);
}

@Test public void highCardCombination() {

    Hand h2 = new Hand("p2", factory.transform("KC JH 5C 2H 7D"));
    Combination c2 = new Combination(CombinationKind.HIGH_CARD);
    c2.addInvolvedCards(Arrays.asList(new Card(KING, CLUBS)));
    c2.fillWithRemainingCards(h2);

    int comparison = c1.compareTo(c2);
    assertTrue(comparison < 0);

    int reverse = c2.compareTo(c1);
    assertTrue(reverse > 0);

    assertEquals(0, c1.compareTo(c1));
    assertEquals(0, c2.compareTo(c2));
}
```

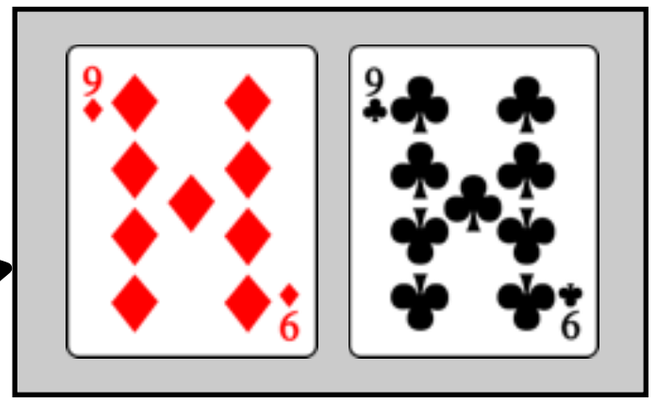
Abstraction: “Checking Rules”

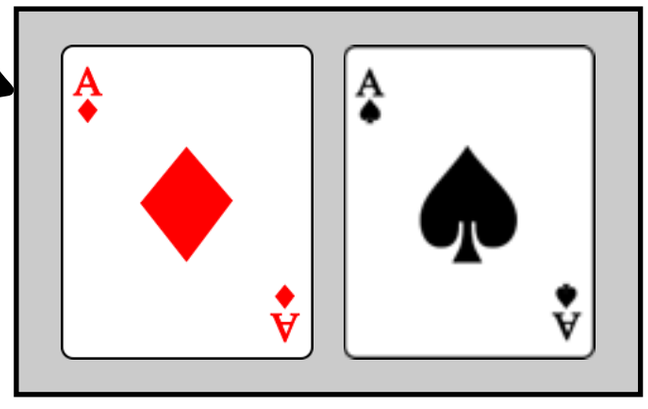


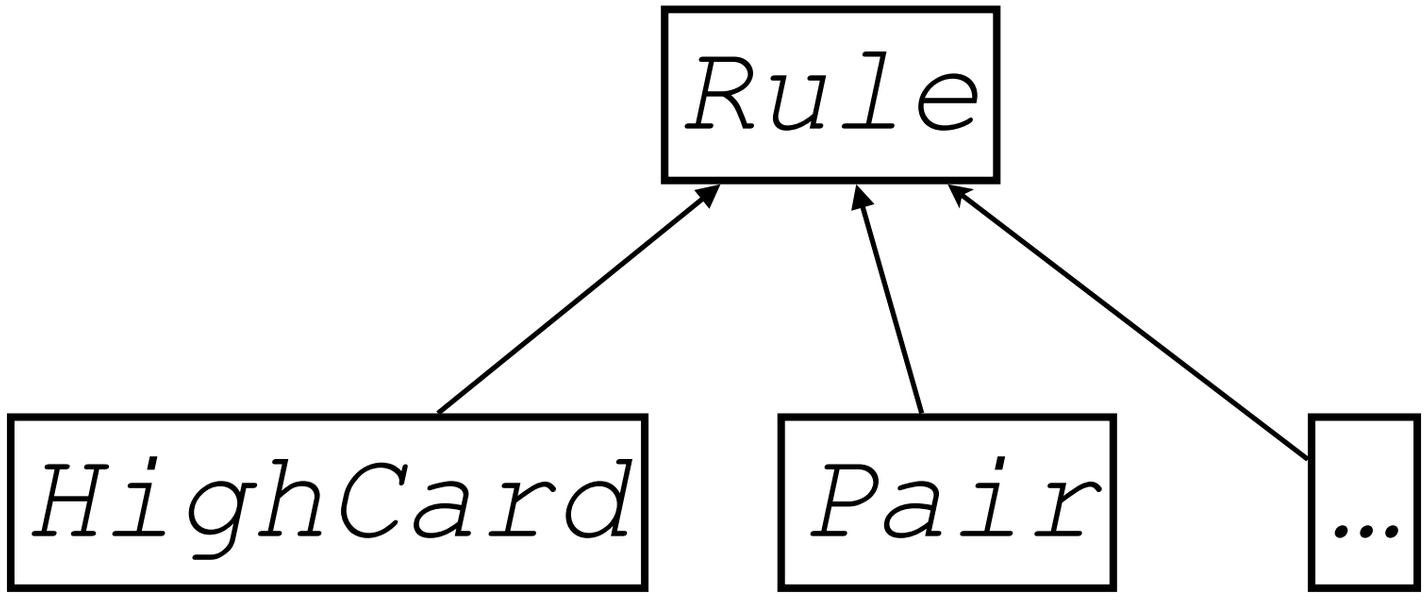
Hand

Rule for
Pair detection

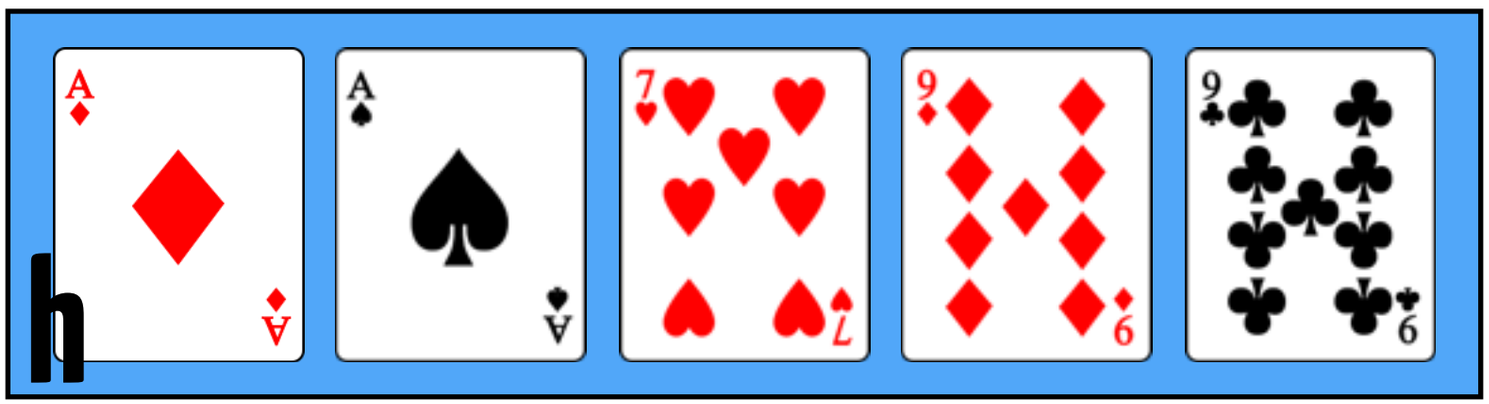
Combination



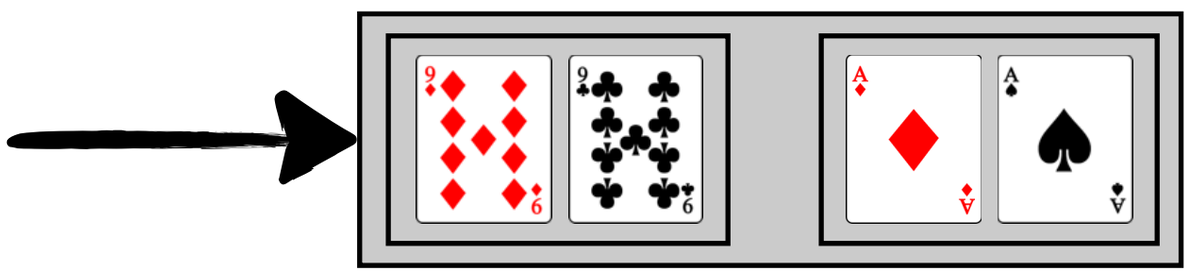


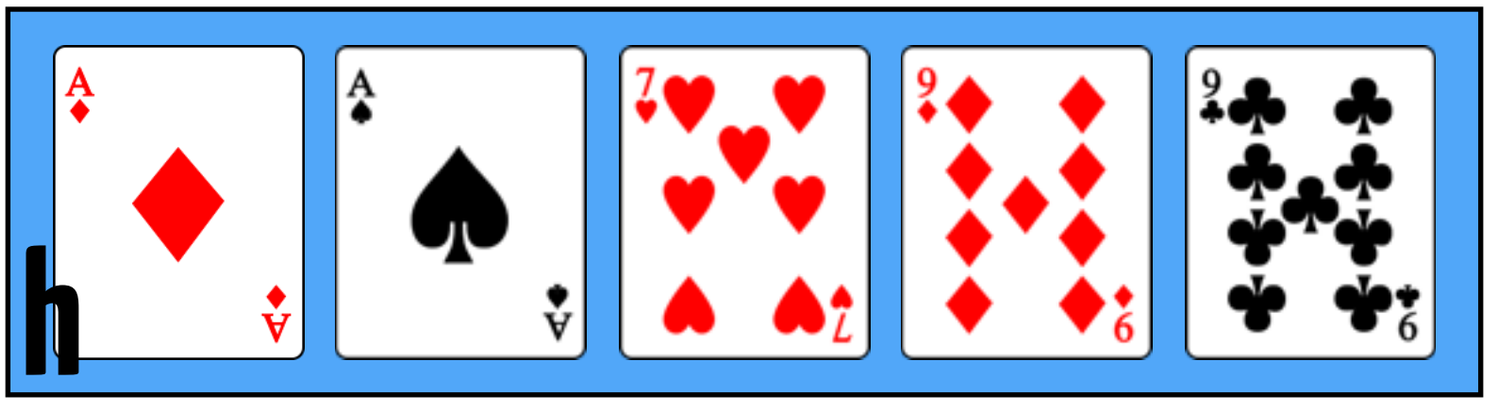



```
public interface Rule {
    public Set<Combination> apply(Hand h);
}
```

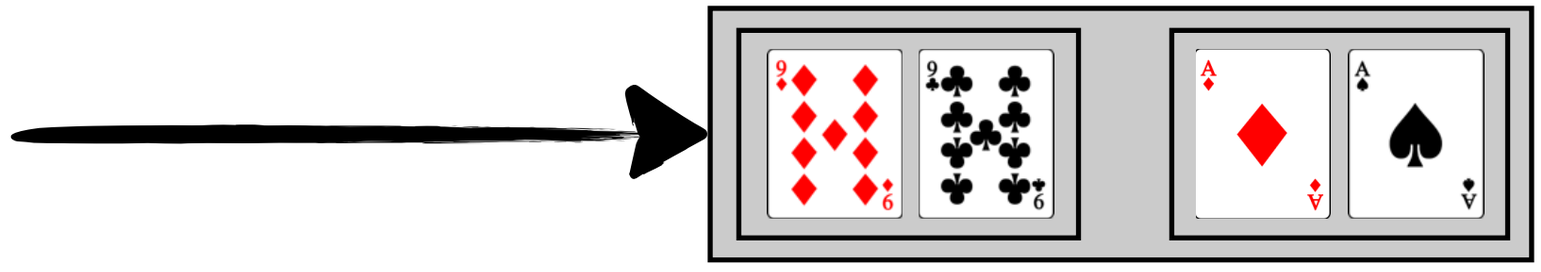



```
Rule r = new Pair();
r.apply(h);
```



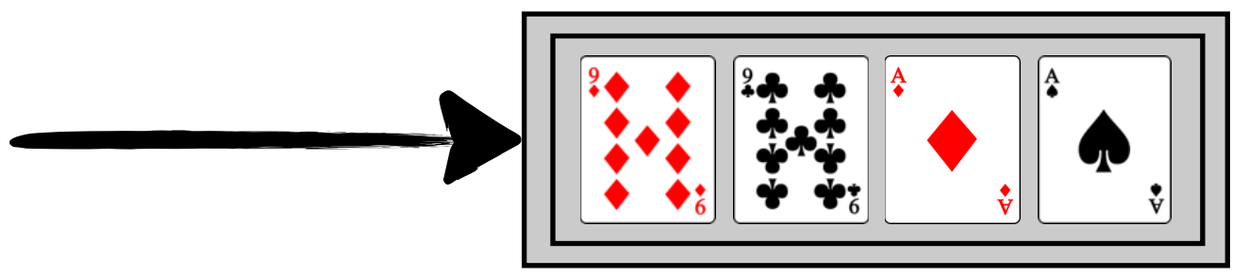



```
Rule r = new Pair();
r.apply(h);
```

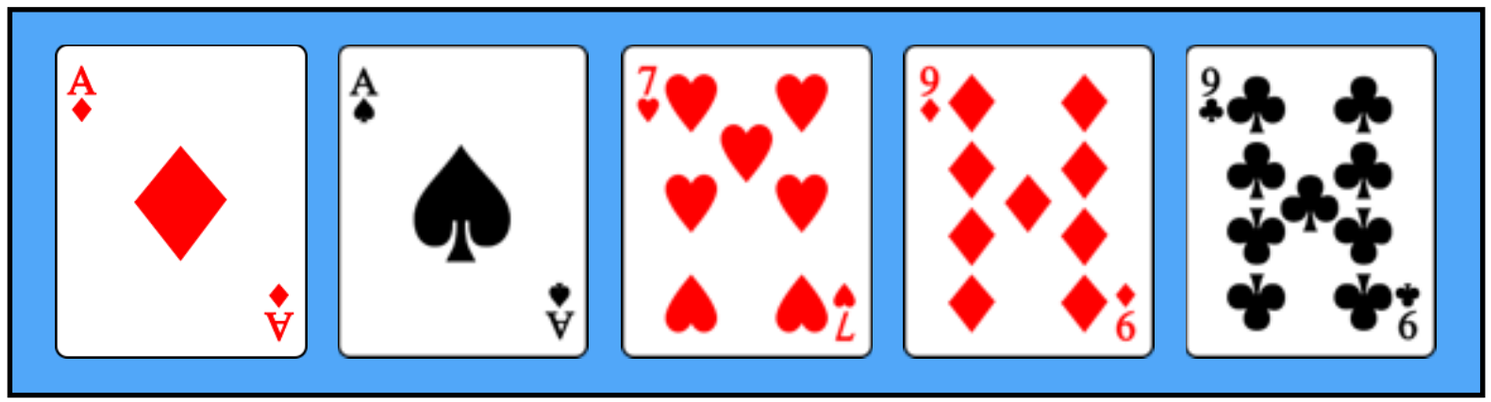


```
Rule r = newThreeOfAKind();
r.apply(h);
```

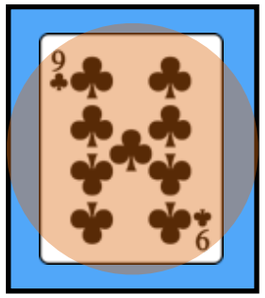

```
Rule r = new DoublePair();
r.apply(h);
```

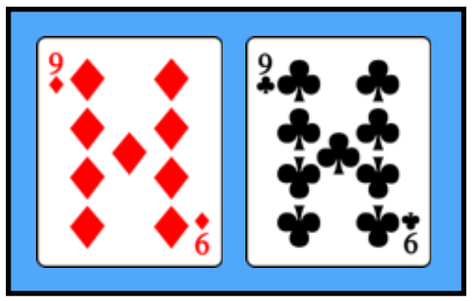


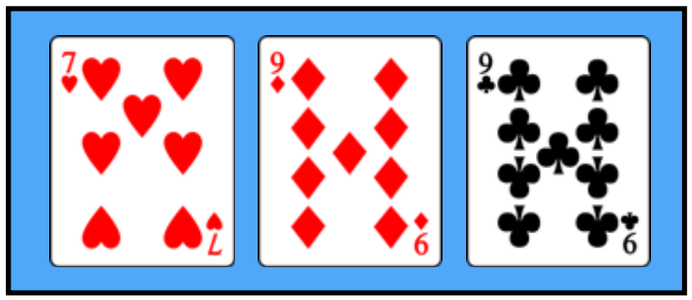
Rule Implementation

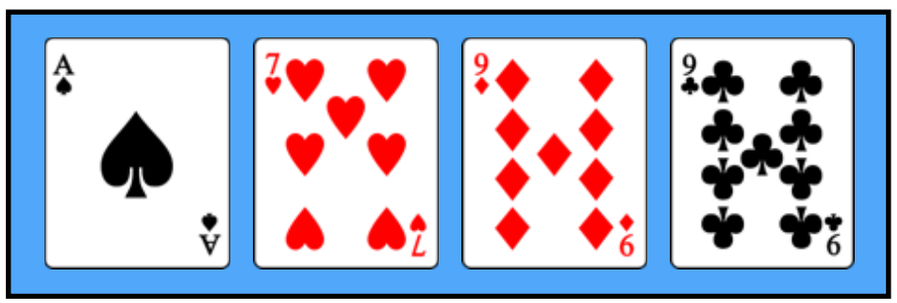


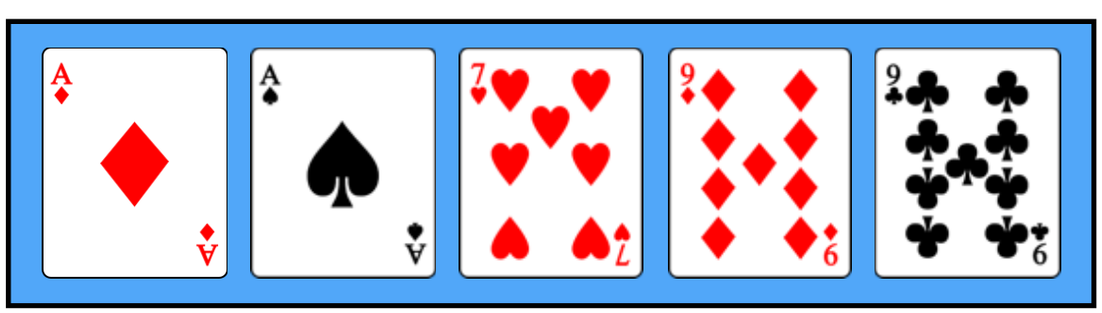
```
|cards| < 2 => No pairs

|cards| ≥ 2 => {
        card[0] = cards[1] => Pair!
         + detectPair(cards[1..n])
}
```

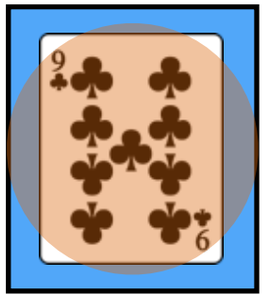




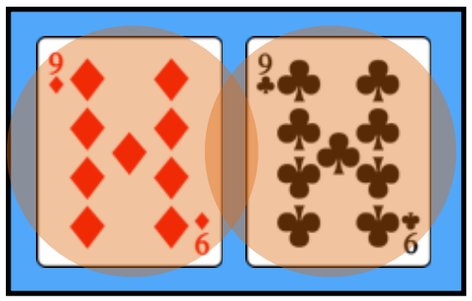




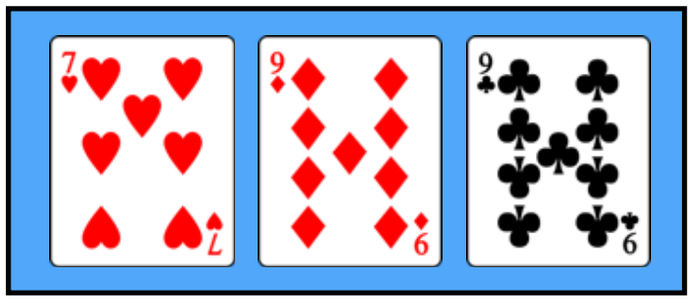


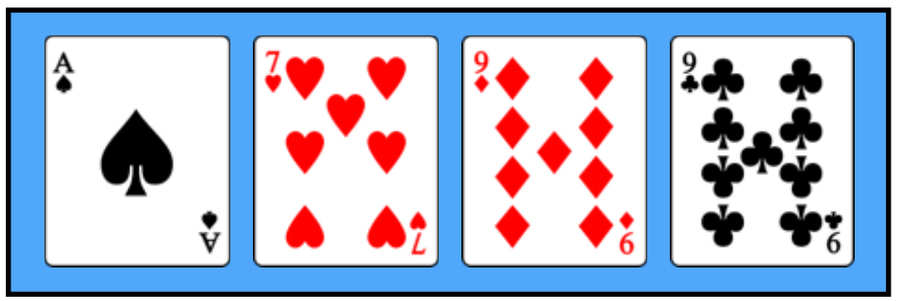


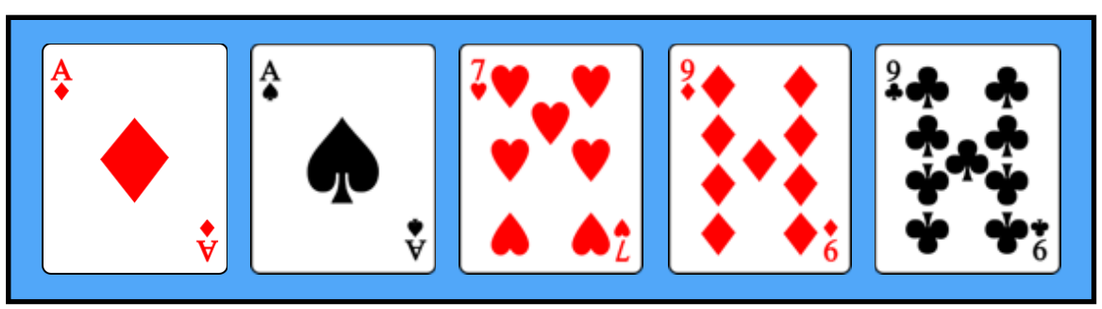
detect => Ø

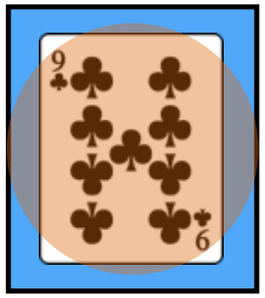


detect => (9,9) + Ø

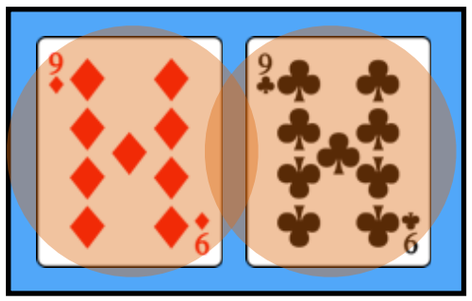




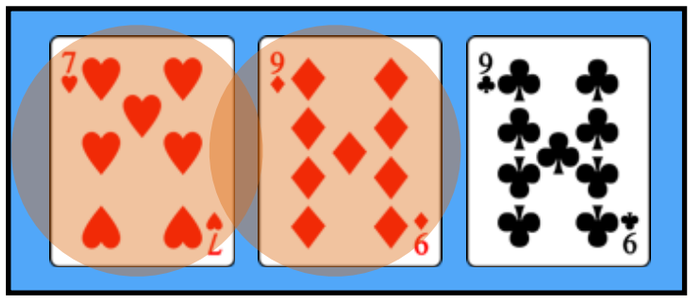




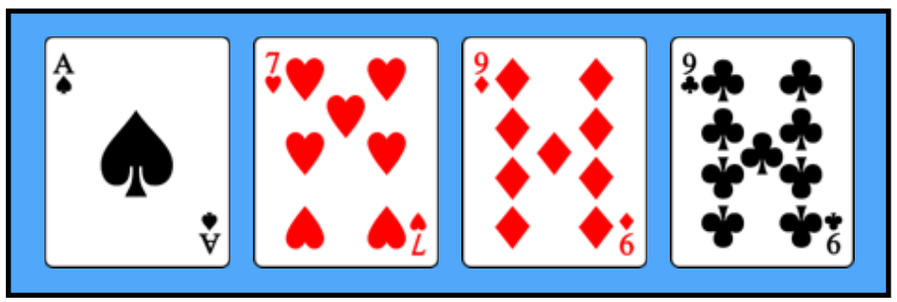
detect => ø

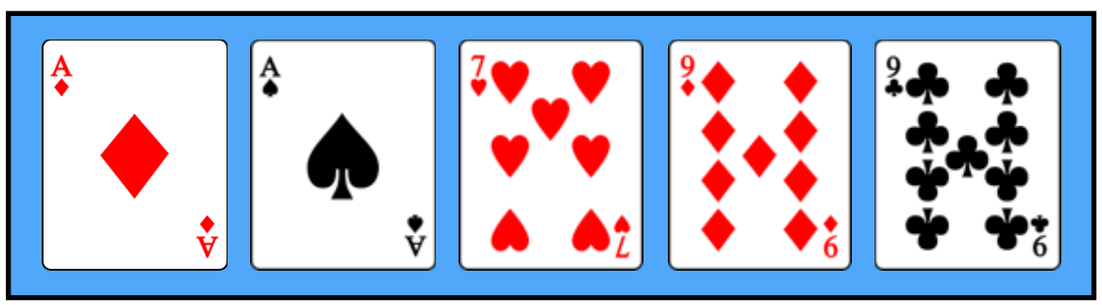


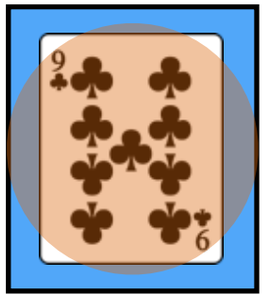
detect => (9,9) + ø



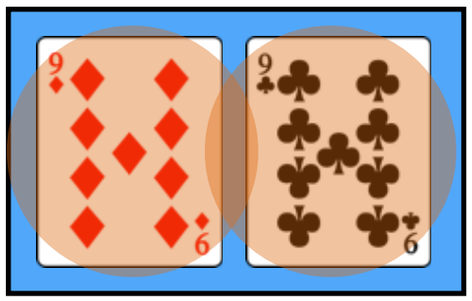
detect => (9,9) + ø



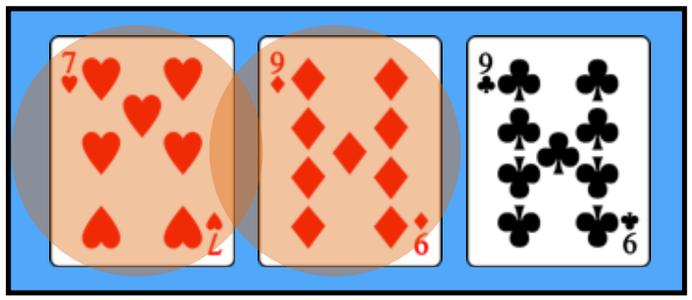




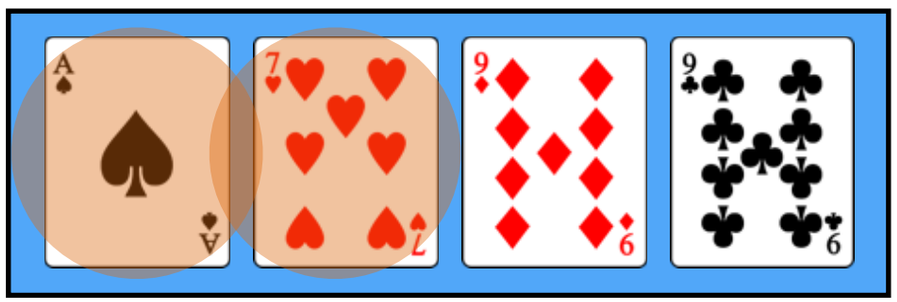
detect => ø



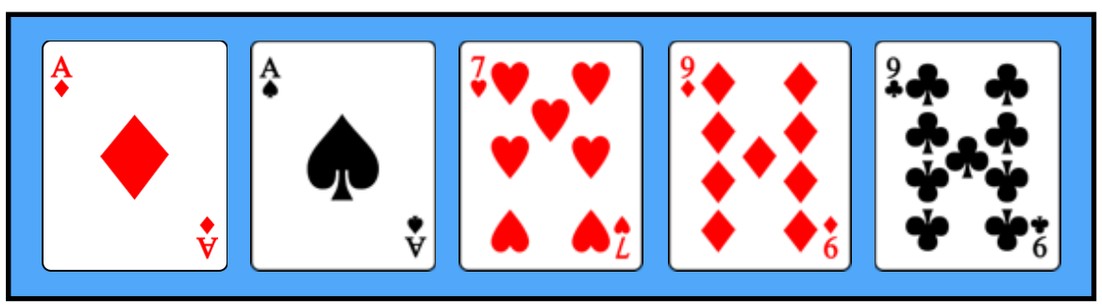
detect => (9,9) + ø

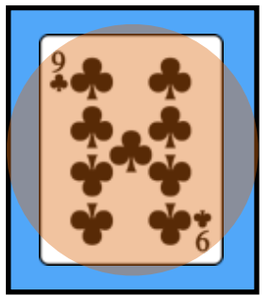


detect => (9,9) + ø

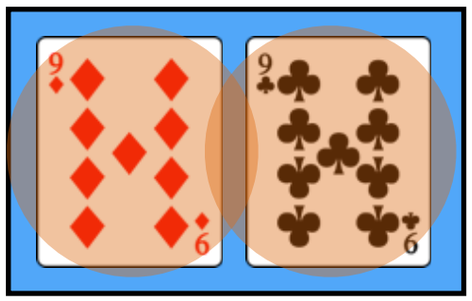


detect => (9,9) + ø

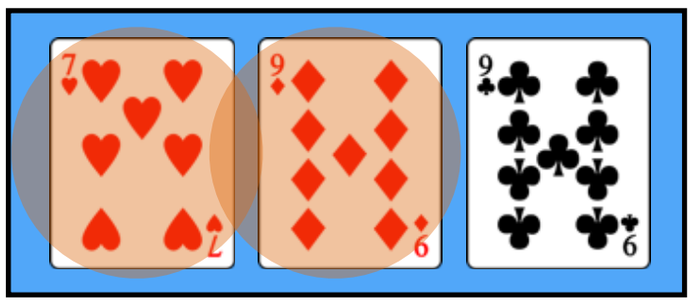




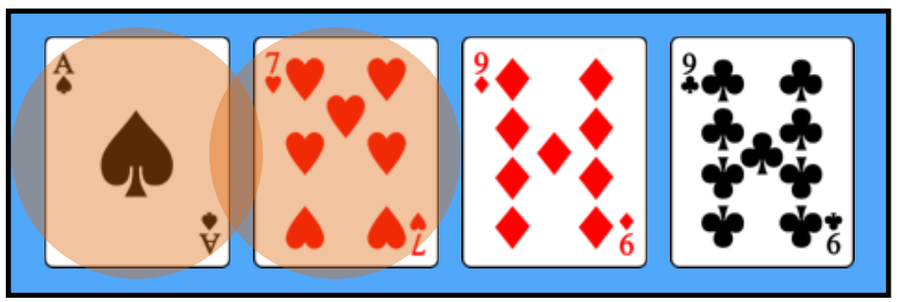
detect => ø



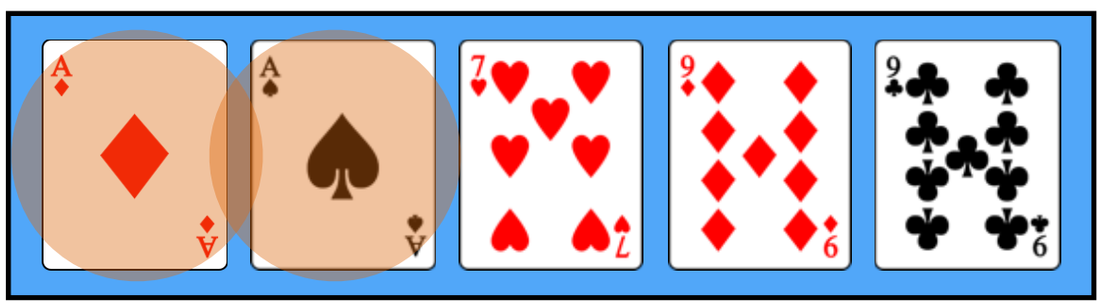
detect => (9,9) + ø



detect => (9,9) + ø



detect => (9,9) + ø



detect => (A,A) +(9,9) + ø

```java
public class Pair implements Rule {

    @Override public Set<Combination> apply(Hand h) {
        List<Card> cards = h.getOrderedCards();
        return collectPairs(cards);
    }

    private Set<Combination> collectPairs(List<Card> cards) {
        if (cards.size() < 2)
            return new HashSet<>();
        Set<Combination> detected = collectPairs(cards.subList(1,cards.size()));
        Card first = cards.get(0);
        Card second = cards.get(1);
        if (first.getFace() == second.getFace()) {
            Combination c = new Combination(CombinationKind.PAIR);
            c.addInvolvedCards(Arrays.asList(first,second));
            c.fillWithRemainingCards(cards);
            detected.add(c);
        }
        return detected;
    }

}
```

Referee's logic

```
public class Referee {

    private List<Rule> rules;

    public Referee() {
        this.rules = new LinkedList<>();
        rules.add(new HighCard());
        rules.add(new Pair());
    }

    public Combination findBest(Hand h){
        Set<Combination> detected = new HashSet<>();
        for(Rule r: rules)
            detected.addAll(r.apply(h));

        Combination result = null;
        for(Combination c: detected) {
            if (result == null)
                result = c;
            else if (result.compareTo(c) < 0)
                result = c;
        }
        return result;
    }
```

```
public GameResult decide(Hand left, Hand right) {
    Combination lc = findBest(left);
    Combination rc = findBest(right);
    int comparison = lc.compareTo(rc);
    if (comparison < 0)
        return new Victory(right, rc);
    else if (comparison == 0) {
        return new Tie(left, right, lc);
    } else {
        return new Victory(left, lc);
    }
}
```