The “Poker Game” part 2

Philippe Collet - with slides from Sébastien Mosser

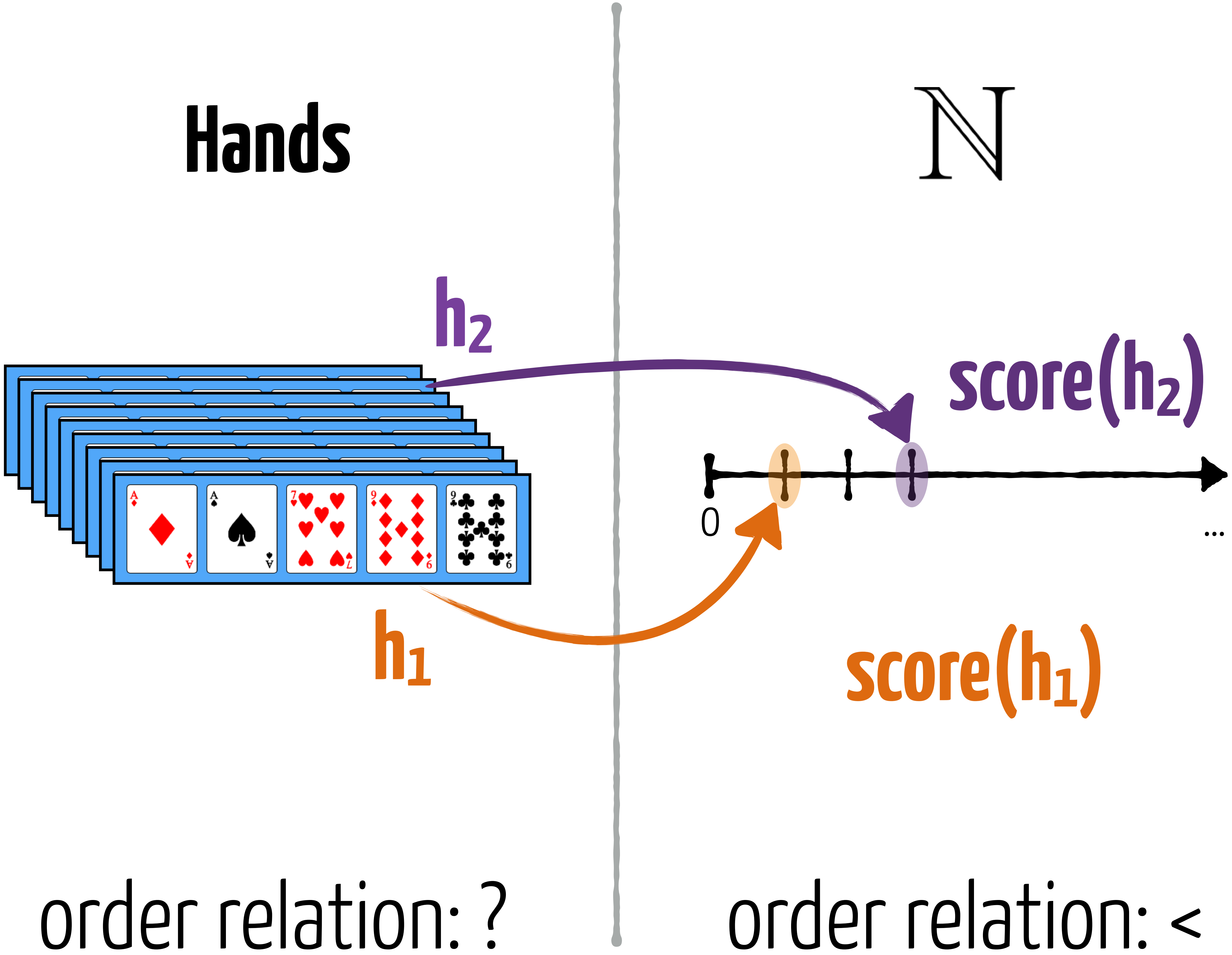
F3: Scoring Hands

How to compare two hands?

How to score "Hands" ?

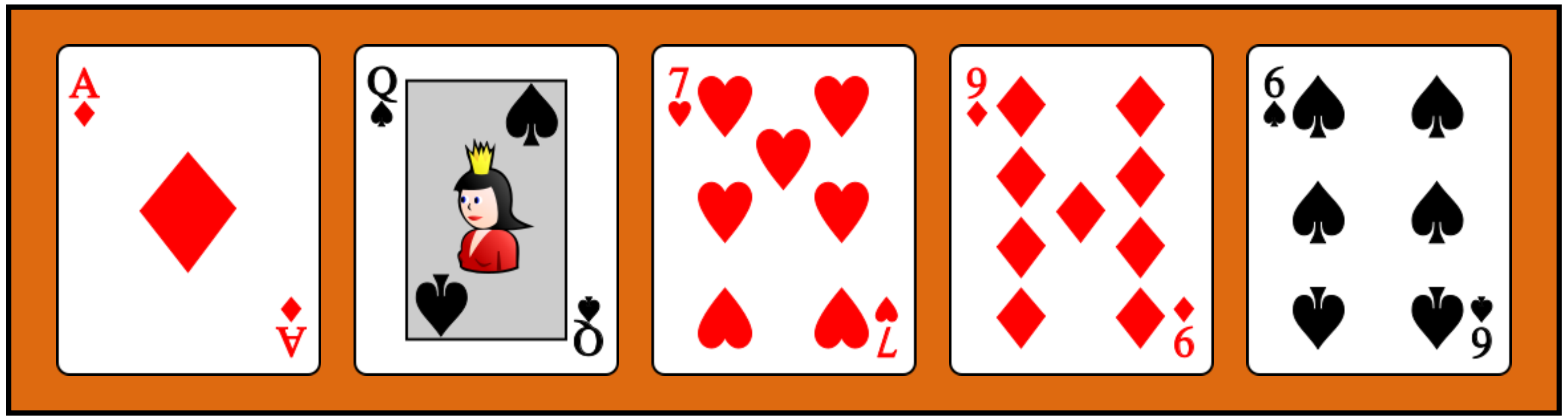
$|\text{Hands}| =$ **?**

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


Hands
$\mathbb{N}$
h_2
h_1
$\text{score}(h_2)$
$\text{score}(h_1)$
0
...
order relation: ?
order relation: <

Warning: all cards being different

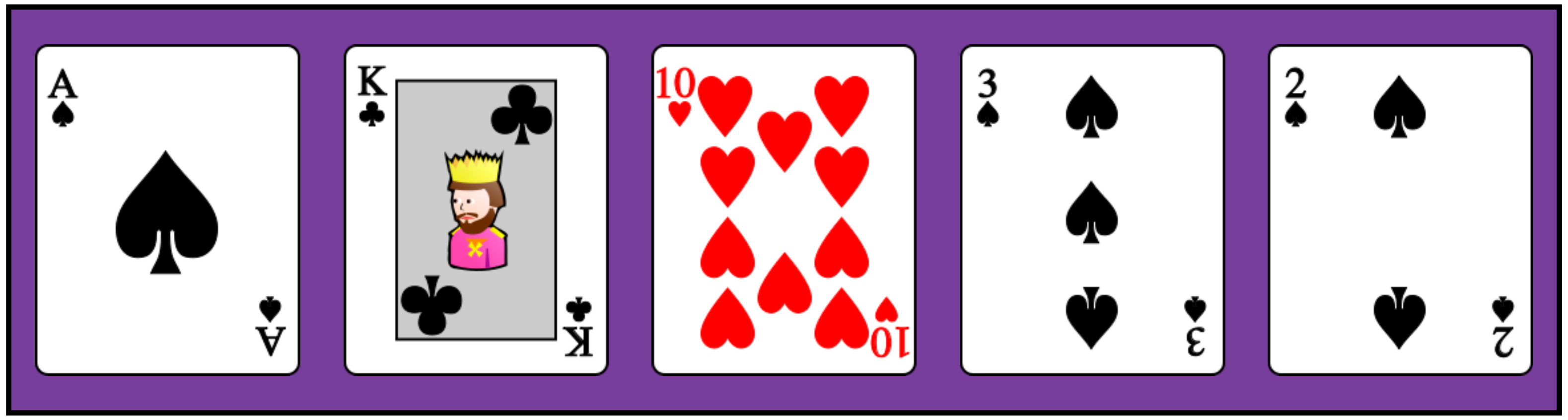
h_1



14 12 7 9 6

$\text{score}(h_1)$
$= 14+12+7+9+6$
$= 42$

h_2



14 13 10 3 2

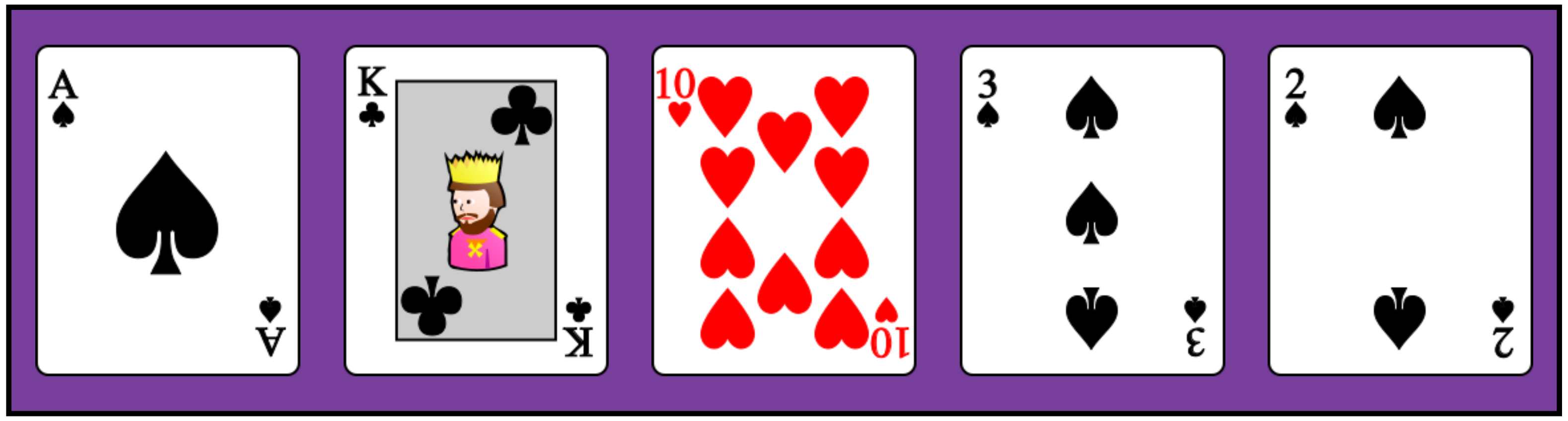
$\text{score}(h_2)$
$= 14+13+10+3+2$
$= 42$

$h_2 > h_1$

$\text{score}(h_1) = \text{score}(h_2)$

Warning: all cards being different

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
$+ 3 * 10$
$+ 10 * 100$
$+ 13 * 1000$
$+ 14 * 10000$
$= 154{,}032$

S€orTing 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

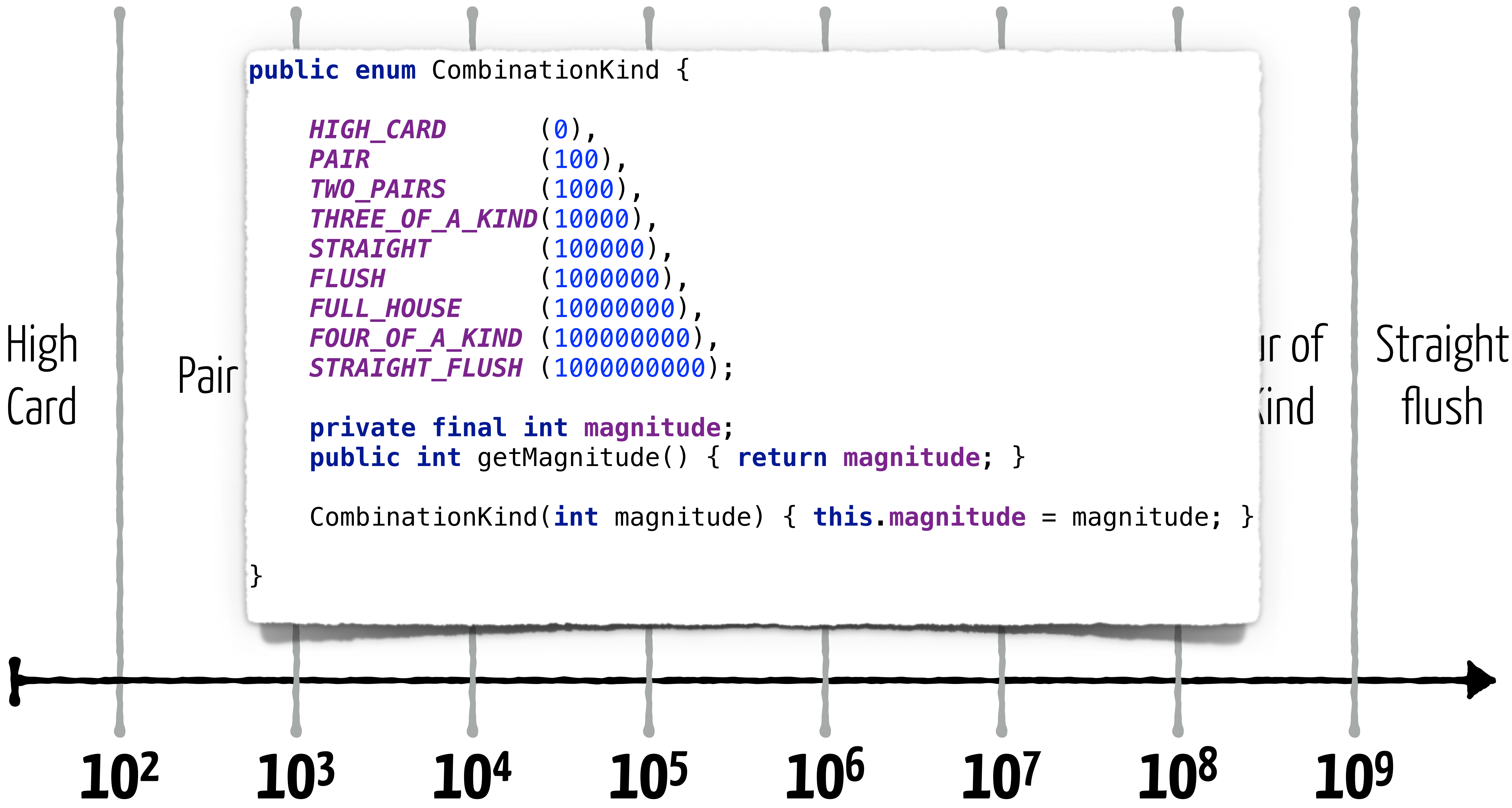
```java
public int score() {
    List<Card> sorted = getOrderedCards();
    return  sorted.get(0).getValue().getValue() +
            sorted.get(1).getValue().getValue() * 10 +
            sorted.get(2).getValue().getValue() * 100 +
            sorted.get(3).getValue().getValue() * 1000 +
          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

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

F$_4$: Detecting Combinations

Highest card, Pair, Three of a ...

Scoring Combination instead of Hands

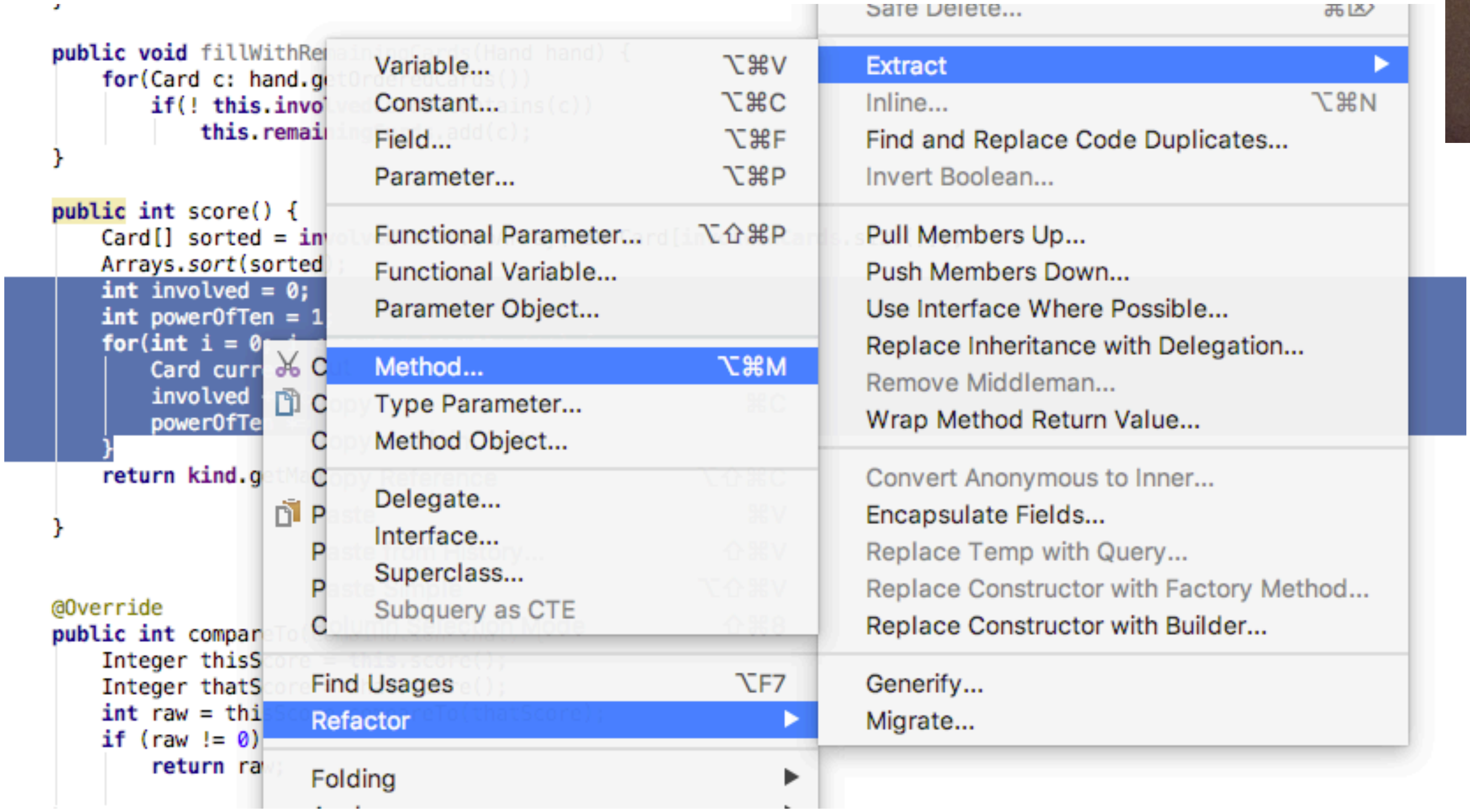


```
public enum CombinationKind {

    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; }

}
```

Comparing cards => include remaining cards

```java
public class Combination {

    private CombinationKind kind;
    private Set<Card> involvedCard[illegible] HashSet<>();

    public int score() {
        Card[] sorted [illegible]lvedCards.toArray(new Card[involvedCards.size()]);
        Arrays.so[illegible]ted);
        int i[illegible]ed = 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;
    }
}
```

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
```

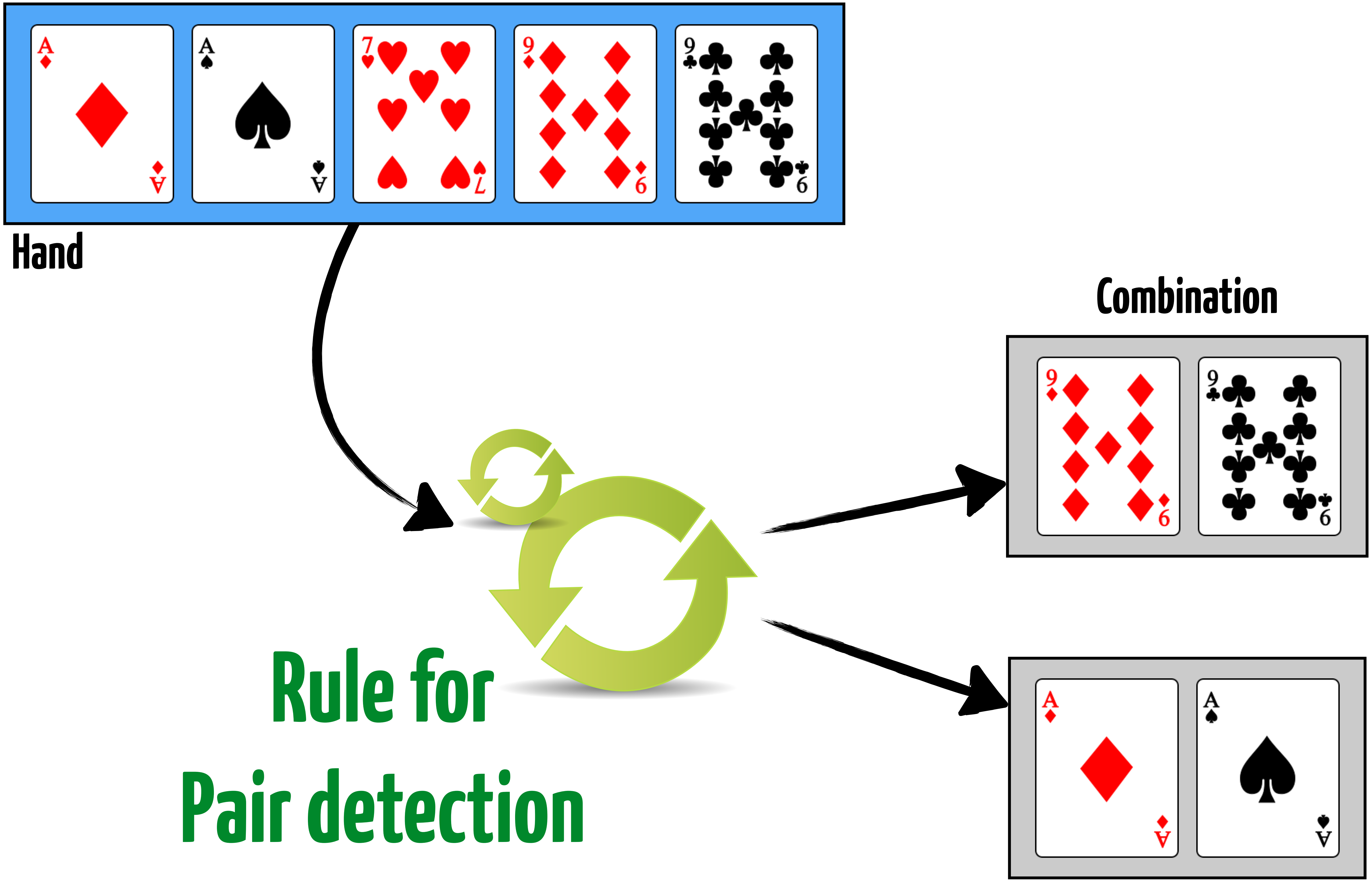
```
public int score() {
    int involved = getIntegerValue(involvedCards);
    return kind.getMagnitude() + involved;
}

@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);
    }
}
```

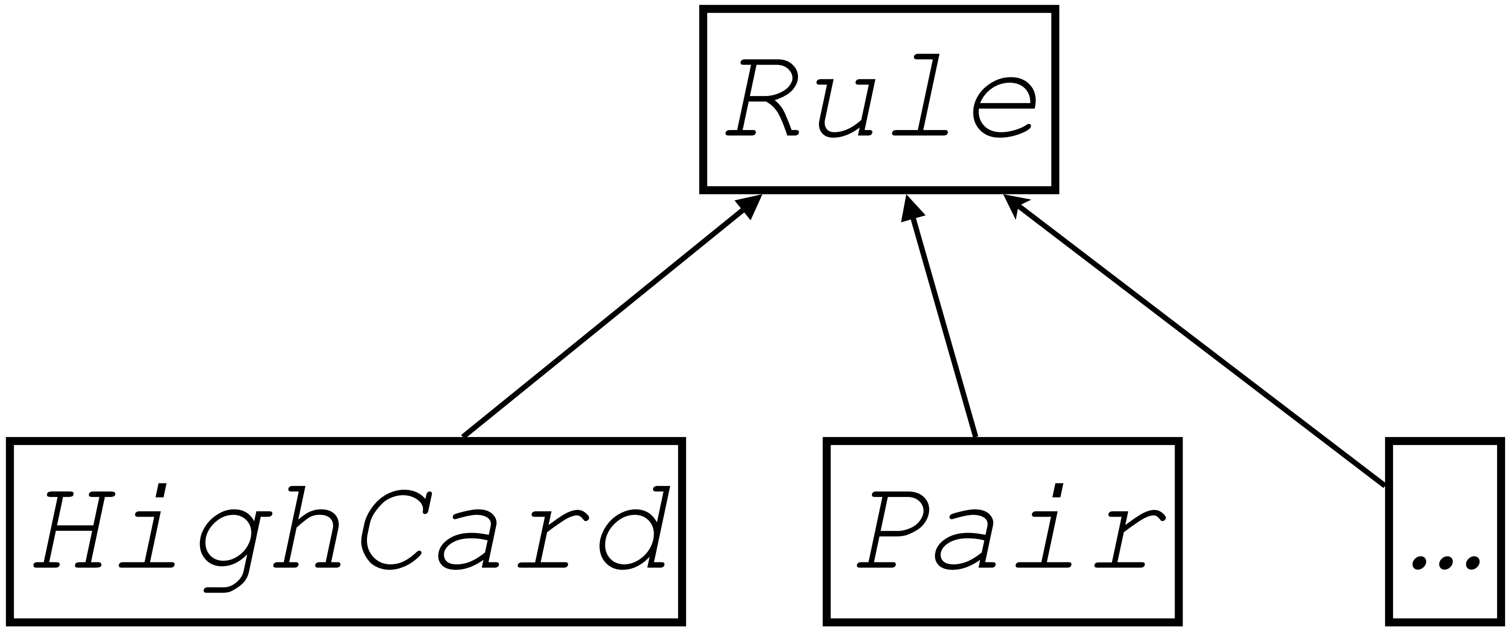
Works for 3 remaining cards

```
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;
}
```

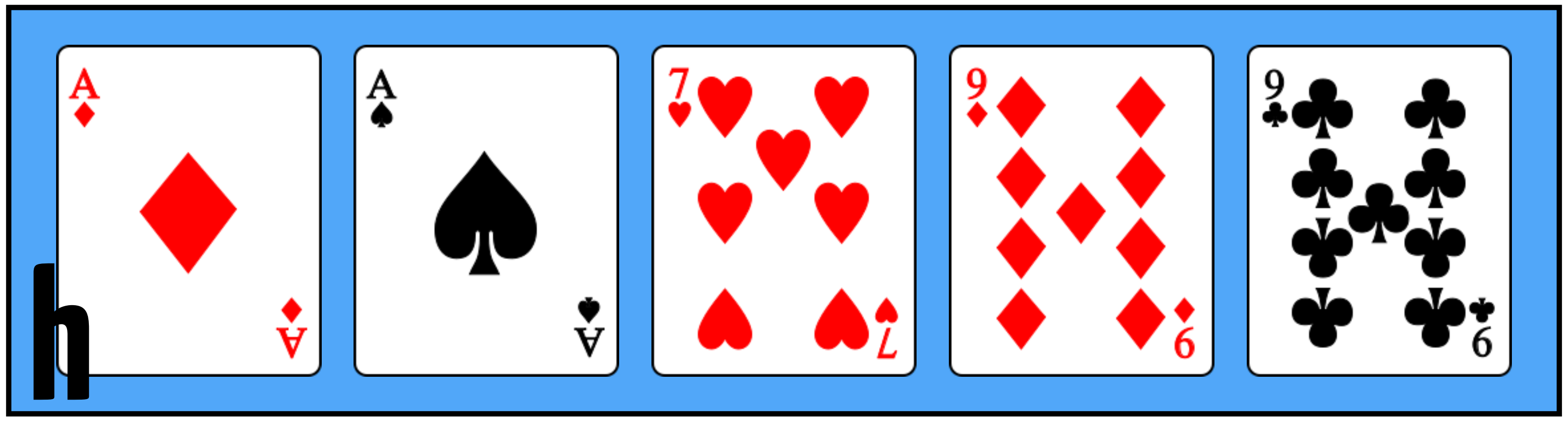
```java
private Combination c1;
@Before public void initCombination1() {
    Hand h = new Hand("p1", factory.transform("QD JH 5C 2H 7D"));
    c1 = new Combination(CombinationKind.HIGH_CARD);
    c1.addInvolvedCards(Arrays.asList(new Card(QUEEN, DIAMONDS)));
    c1.fillWithRemainingCards(h);
}

@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”

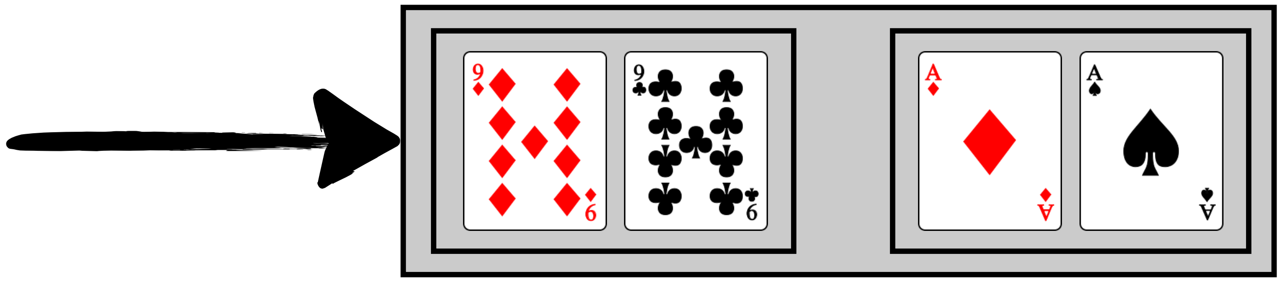


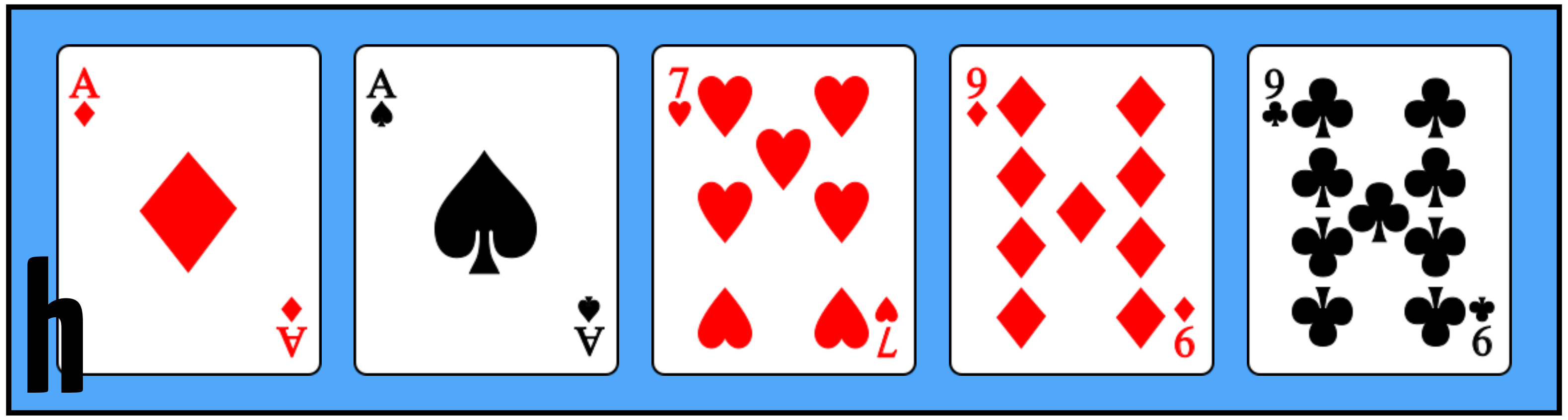


```
public interface Rule {
   public Set<Combination> apply(Hand h);
}
```

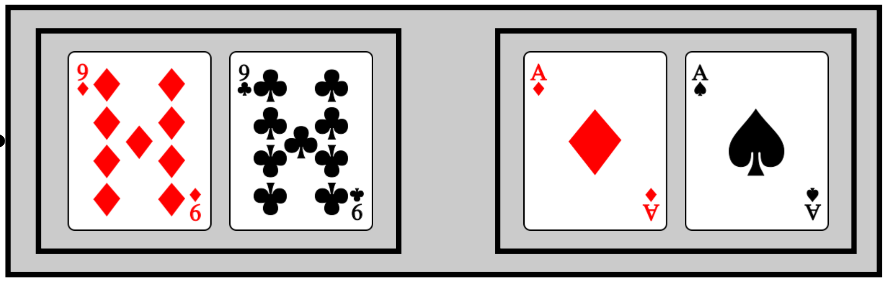


```
Rule r = new Pair();
r.apply(h);
```

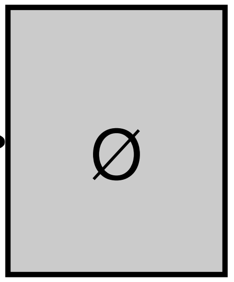




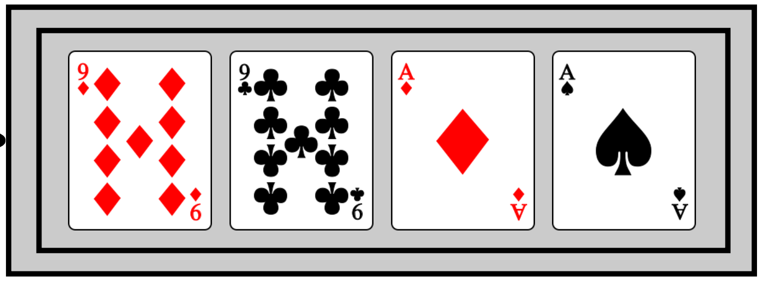
```
Rule r = new Pair();
r.apply(h);
```



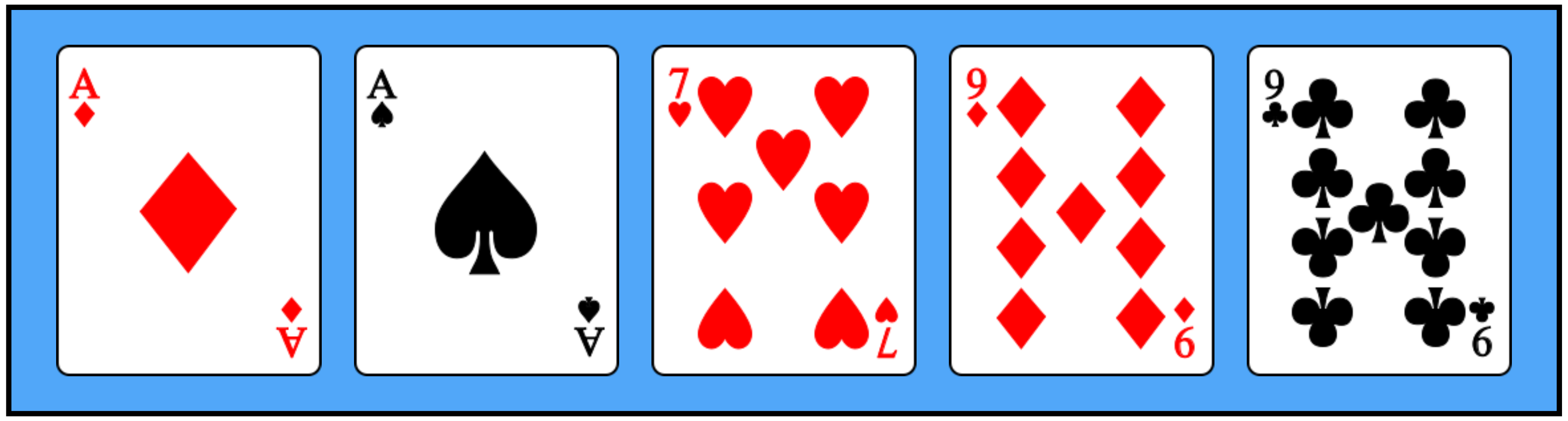
```
Rule r = newThreeOfAKind();
r.apply(h);
```



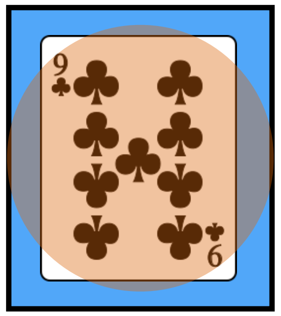
```
Rule r = new DoublePair();
r.apply(h);
```



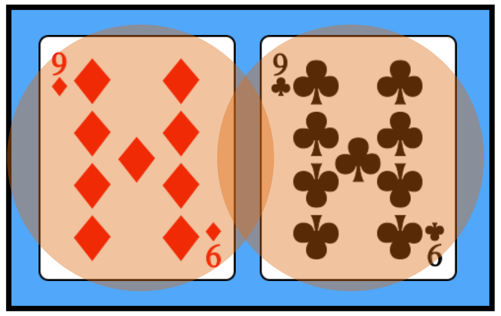
Rule Implementation



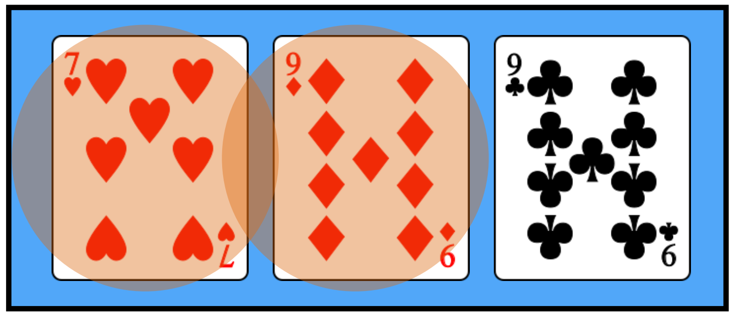
```
|cards| < 2 => No pairs
|cards| ≥ 2 => {
        card[0] = cards[1] => Pair!
         + detectPair(cards[1..n])
}
```



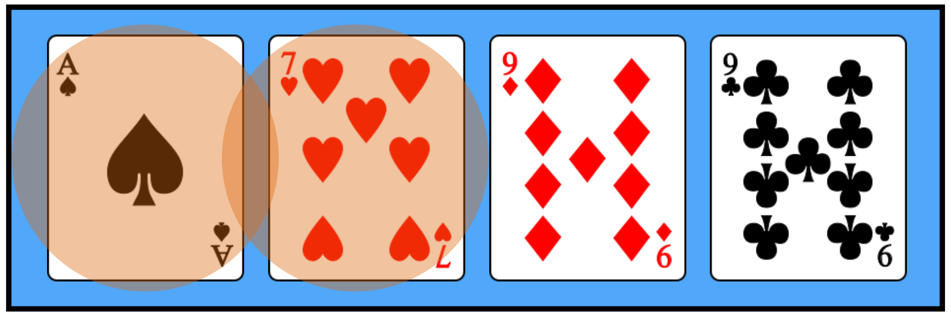
detect => Ø



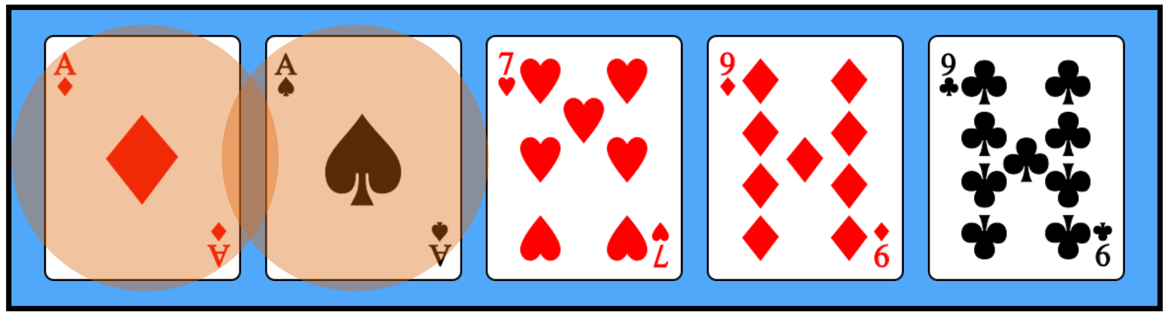
detect => (9,9) + Ø



detect => (9,9) + Ø



detect => (9,9) + Ø



detect => (A,A) +(9,9) + Ø

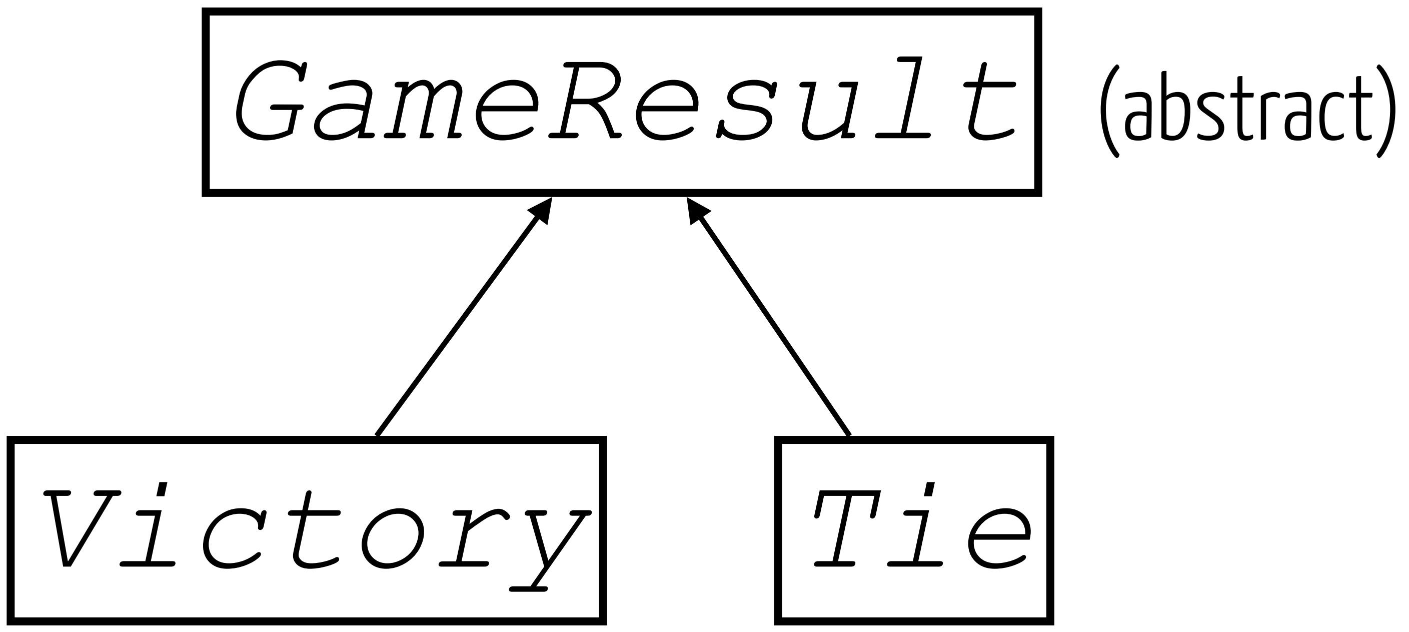
```java
public class Pair implements Rule {

    @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

```java
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;
    }
```

Could be optimized…

```java
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);
    }
}
```


GameResult
(abstract)
Victory
Tie