

Funktionale versus objektorientierte Programmierung

Programmierparadigmen

Johannes Brauer

9. November 2019

Ziele

- Erkennen von Zusammenhängen zwischen verschiedenen Programmierparadigmen, insbesondere zwischen der funktionalen und der objektorientierten Programmierung
- Erkennen der Kernkonzepte der Paradigmen
- Verstehen von Dimensionen der Erweiterbarkeit von Programmen

Funktionale versus objektorientierte Programmierung

Ein wenig Historie

1940



Programmierung



20??

1936



ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO
THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

- Begriffe: Unterprogramm, Makro, code is data

1945

- Turing schreibt Code.
 - in Binärform
 - benutzt Integer-Zahlen (add) und boolesche Operationen
 - erfindet Unterprogramme einschl. Stack
 - erfindet Gleitkommazahlen
- Anzahl der Computer weltweit: $O(1)$.
- Anzahl der Programmierer weltweit: $O(1)$.

1953 Fortran

```

C AREA OF A TRIANGLE WITH A STANDARD SQUARE ROOT FUNCTION
C INPUT - TAPE READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPLAY ERROR OUTPUT CODE 1 IN JOE CONTROL LISTING STATEMENT
      READ INPUT TAPE 5, 501, IA, IB, IC
 501 FORMAT (3I5)
C IA, IB, AND IC MAY NOT BE NEGATIVE
C FURTHERMORE, THE SUM OF TWO SIDES OF A TRIANGLE
C IS GREATER THAN THE THIRD SIDE, SO WE CHECK FOR THAT, TOO
      IF (IA) 777, 701
    701 IF (IB) 777, 702
    702 IF (IC) 777, 777, 703
    703 IF (IA+IB-IC) 777,777,704
    704 IF (IA+IC-IB) 777,777,705
    705 IF (IB+IC-IA) 777,777,799
    777 STOP 1
C USING HERON'S FORMULA WE CALCULATE THE
C AREA OF THE TRIANGLE
    799 S = FLOATF (IA + IB + IC) / 2.0
      AREA = SQRT( S * (S - FLOATF(IA)) * (S - FLOATF(IB)) *
      + (S - FLOATF(IC)))
      WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
 601 FORMAT (4H A= ,I5,5H B= ,I5,5H C= ,I5,8H AREA= ,F10.2,
      +          13H SQUARE UNITS)
      STOP
      END

```

Lisp, 1958

```
(DEFUN FAC (N)
  (IF (= N 0)
      1
      (* N (FAC (- N 1)))))
```

- Value-oriented
- Garbage collection
- Lambda functions



John McCarthy
1927 - 2011

Funktionale Programmierung

1960

- Anzahl der Computer weltweit: $O(1E2)$.
- Anzahl der Programmierer weltweit: $O(1E3)$.

1965

- Anzahl der Computer weltweit: $O(1E4)$.

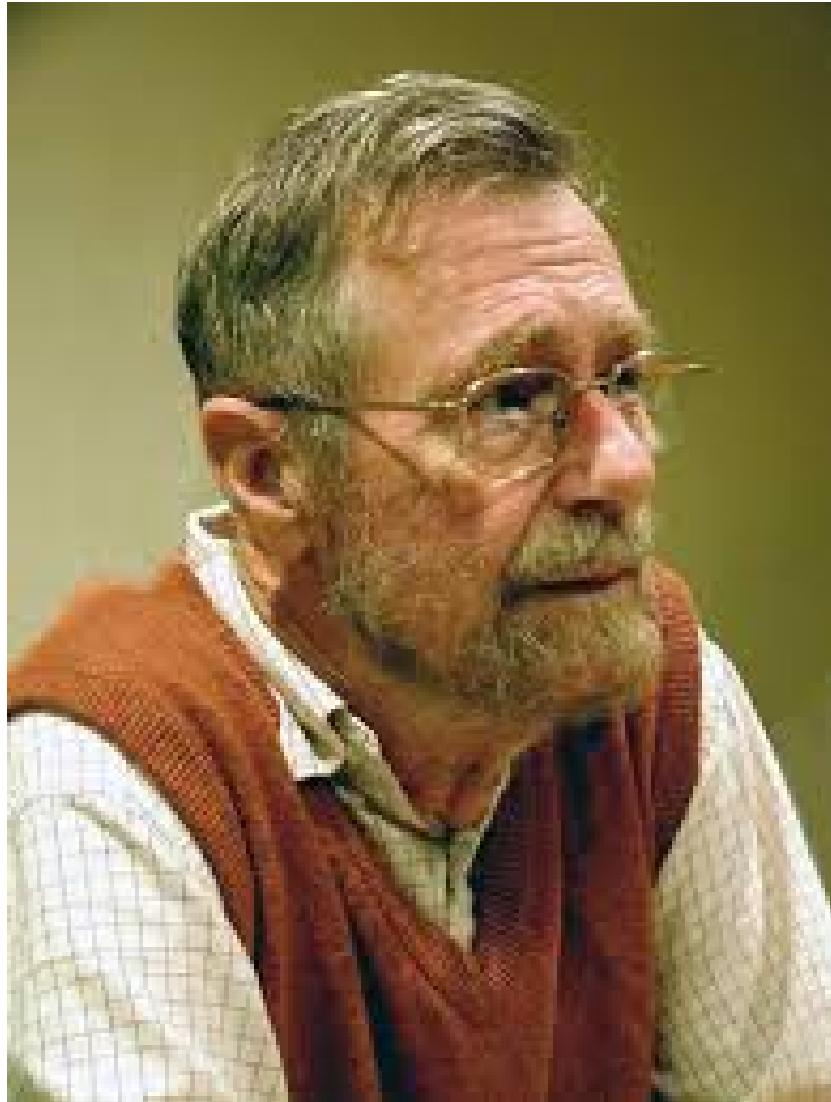
- Anzahl der Programmierer weltweit: $O(1E5)$.

1966 Simula 1967

Ole-Johan Dahl, Kristen Nygard:
./Abbildungen/simuladahlnygard.gif
Objektorientierte Programmierung

1968

Edsger Dijkstra:



Go To Statement Considered Harmful

Key Words and Phrases: go to statement, jump instruction, branch instruction, conditional clause, alternative clause, repetitive clause, program intelligibility, program sequencing
CR Categories: 4.22, 5.23, 5.24

EDITOR:

For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of **go to** statements in the programs they produce. More recently I discovered why the use of the **go to** statement has such disastrous effects, and I became convinced that the **go to** statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

Strukturierte Programmierung

Drei Paradigmen

- 1968 – Strukturierte Programmierung. Edsger Dijkstra zeigt, wie der Gebrauch von GOTO-Anweisungen durch **if/then/else** und **while** ersetzt werden kann.
- 1966 – Objektorientierte Programmierung. Ole-Johan Dahl und Kristen Nygaard entwickeln die erste OO-Sprache: *Simula-67*. Durch die Einführung der Polymorphie konnten die **function pointers** eliminiert werden.
- 1957 – Funktionale Programmierung. John McCarthy erfindet Lisp. Lisp basiert auf dem Lambda-Kalkül, entwickelt von Alonzo Church in den 1930ern. Ein wesentliches Merkmal: Es gibt keine Zuweisung.

Drei Paradigmen, drei Einschränkungen

1970

- Anzahl der Computer weltweit: $O(1E5)$.
- Anzahl der Programmierer weltweit: $O(1E6)$.

Was bringt die funktionale Programmierung?

- Worin besteht der Kerngedanke der funktionalen Programmierung?

Fehlen der Mutation

- Wie kann das „Fehlen von Etwas“ ein wichtiges Feature eines Programmierparadigmas sein?

- Das Fehlen von Zuweisungen ermöglicht das Schreiben von Programmcode ohne befürchten zu müssen, dass anderer Code den eigenen negativ beeinflusst.
- Nicht änderbare Datenstrukturen sind das wichtigste „non-feature“, das eine Programmiersprache haben kann.
- Auf einen wichtigen Vorteil soll hier besonders eingegangen werden: Sharing und Aliasing werden irrelevant.

Beispiel: Sortieren eines Vektors

```
(def sort-pair
  (fn [pair]
    (if (< (pair 0) (pair 1))
        pair
        [(pair 1) (pair 0)])))
```

- Die Funktion liefert im `else`-Fall einen neu erzeugten Vektor.
- Aber was passiert im `then`-Fall?
 - Erhalten wir eine Kopie des an die Funktion übergebenen Vektors oder
 - gibt die Funktion ein Alias auf denselben Vektor zurück?
- Bei Fehlen von Mutation ist die Frage irrelevant.

Beispiel: Funktion append

```
(def append
  (fn [xs ys]
    (cond
      (empty? xs) ys
      :else (cons (first xs) (append (rest xs) ys)))))
```

- Frage: Teilt die zurückgegebene Liste irgendwelche Elemente mit den Argumentlisten?
- Antwort 1: Es spielt keine Rolle, solange kein Aufrufer das feststellen kann.
- Antwort 2: Ja, die neue Liste „wiederverwendet“ alle Elemente von `ys`. Das spart Speicherplatz und Rechenzeit.
- Ein Problem entstünde nur dann, wenn nachträglich die Liste `ys` verändert werden könnte.
- Unveränderliche Daten erleichtern das Schreiben eleganter Algorithmen erheblich.
- Auch die Funktion `rest` teilt „dankenswerterweise“ ihr Ergebnis mit ihrer Argumentliste (bis auf das erste Element).
- In der objektorientierten Programmierung, wo intensiv von Verweissemantik und Mutation Gebrauch gemacht wird, muss die Programmiererin sich mit den Fragen beschäftigen, ob Verweise auf alte Objekte genutzt oder neue Objekte erzeugt werden.

Beispiel: Ein Java-Programm

- Ein inzwischen behobenes Sicherheitsproblem in einer Java library:

```
class ProtectedResource {  
    private Resource theResource = ...;  
    private String[] allowedUsers = ...;  
    public String[] getAllowedUsers() {  
        return allowedUsers;  
    }  
    public String currentUser() { ... }  
    public void useTheResource() {  
        for(int i=0; i < allowedUsers.length; i++) {  
            if(currentUser().equals(allowedUsers[i])) {  
                ... // access allowed: use it  
                return;  
            }  
        }  
        throw new IllegalAccessException();  
    }  
}
```

- Problem: `getAllowedUsers` gibt einen Verweis auf das Array `allowedUsers` zurück.
- Damit ist `getAllowedUsers()[0] = currentUser()` möglich.
- Abhilfe?

Integration von Daten und Funktionen – Übergang zum Objektbegriff

(bereits in EidP behandelt!)

Funktionen mit Funktionen als Resultat

Beispiele aus dem letzten Semester:

- Was liefert der Ausdruck `(inc 3)`, wenn die Funktion `inc` wie folgt definiert ist?

```
(def inc  
  (fn [n] (fn [z] (+ z n))))
```

- Welches Resultat liefert eine Anwendung der Funktion `dot`?

```
(def dot  
  (fn [f g] (fn [x] (f (g x)))))
```

Was ist ein Objekt?

- Objekte im Sinne der „objektorientierten Programmierung“ sind nichts weiter als Funktionen mit „internen“ Daten.
- So gesehen liefert ein Ausdruck der Art

```
(def inc-obj (inc 3))
```

ein Objekt (eine Funktion) mit dem internen Datum (Zustand) **3**.

- Die Funktion bezeichnet man in der objektorientierten Programmierung auch als *Methode*.
- Durch den Ausdruck (inc-obj 7) wird die Methode aktiviert (die Funktion aufgerufen). Sie verlangt ein Argument.
- inc-obj ist ein sehr primitives Objekt mit nur einer (unveränderlichen) Zustandsgrößen und einer Methode.

Ein etwas interessanteres Objekt

```
(def person
  (fn [name vorname]
    (let [get-name      (fn [] name)
          get-vorname   (fn [] vorname)
          vollname     (fn [] (str vorname " " name))
          gruss        (fn [g] (str g " " (vollname)))]
      (fn [nachricht]
        (cond
          (= nachricht 'get-name) get-name
          (= nachricht 'get-vorname) get-vorname
          (= nachricht 'get-vollname) vollname
          (= nachricht 'gruss) gruss
          :else (throw (Exception. "unbekannte Nachricht"))))))))
```

- Ein Aufruf der Funktion person liefert ein „Objekt“ mit
 - zwei Zustandsgrößen (`name`, `vorname`) und
 - vier Methoden: `get-name`, `get-vorname`, `vollname` und `gruss`
- „In Wirklichkeit“ liefert ein Aufruf der Funktion person eine Verteilfunktion, die anhand ihres Parameters `nachricht` ermittelt, welche der Methoden aufzurufen ist.

Anwendungen des person-Objekts

```
; ; Objekterzeugung:
(define p1 (person "Gans" "Gustav"))
((p1 'get-vorname))      ;;= "Gustav"
((p1 'get-name))         ;;= "Gans"
((p1 'get-vollname))     ;;= "Gustav Gans"
((p1 'gruss) "hallo")    ;;= "hallo Gustav Gans"
(p1 'x)                  ;; error
```

Java-Klasse Person

```
public class Person
{
    String vorname, name;

    public Person (String n, String v)
    {
        vorname = v;
        name = n;
    }

    public String get_name()
    {
        return name;
    }
    public String get_vorname()
    {
        return vorname;
    }
    public String vollname()
    {
        return vorname + " " + name;
    }
    public String gruss(String gruss)
    {
        return gruss + " " + this.vollname();
    }
}
```

Zusammenfassung

- Für die objektorientierte Programmierung bedarf es keiner speziellen Programmiersprachen.
- Objektorientierung besteht nur aus einer Reihe von Konventionen zum Umgang mit Daten.
- Diese Konventionen werden von objektorientierten Sprachen explizit und implizit unterstützt. Z. B. muss die Verteilfunktion nicht selbst programmiert werden.
- Objekte sind nichts weiter als
 - eine Menge von Name->Wert-Abbildungen
 - eine Reihe von Funktionen, die solche Abbildungen als erstes Argument akzeptieren und
 - eine Verteilfunktion, die ermittelt, welche dieser Funktion aufzurufen ist.

Erweiterbarkeit von Programmen

Identifizierung des Problems

Objektorientierte Programmierung ...

- ... ist gekennzeichnet durch dynamische Bindung und Polymorphie.
- Polymorphie-Ansatz der meisten OO-Sprachen:
 - Methoden werden innerhalb von Klassen definiert.
 - Methodenaktivierung durch „Senden von Nachrichten“: Die zu aktivernde Methode wird anhand der Klassenzugehörigkeit des Empfängerobjekts bestimmt.

```
class Rechteck {  
    int breite, hoehe;  
    int flaeche() { ... }  
  
    class Kreis {  
        int radius;  
        int flaeche() { ... }  
    }  
}
```

- Jeder „Datentyp“ definiert seine Operationen.
- **Problem:** Datentypen können ohne Zugriff auf den Quellcode nicht erweitert werden.

Funktionale Programmierung ...

- ... ist gekennzeichnet durch Funktionen als Werte erster Ordnung und die Abwesenheit von Seiteneffekten.
 - Funktionen enthalten Fallunterscheidungen für Datentypen

```
(def figur-flaeche  
  (fn [f]  
    (cond  
      (kreis? f) (kreis-flaeche f)  
      (rechteck? f) (rechteck-flaeche f))))
```

- **Problem:** Hinzufügen eines neuen Datentyps bedeutet: Alle vorhanden Figur-Funktionen müssen angepasst werden.

Objektorientierte versus funktionale Zerlegung

- Rückgriff auf das Problem der Verarbeitung einer kleinen „Sprache“ über arithmetische Ausdrücke (vgl. Aufgabe Mustervergleich).
- In der funktionalen Programmierung werden üblicherweise Programme durch Zerlegung in Funktionen für die auszuführenden Operationen zerlegt.

- In der objektorientierten Programmierung werden Programme in Klassen zerlegt, die Verhalten für bestimmte Arten von Daten bereitstellen.
- Die folgenden Betrachtungen sollen einen Eindruck vermitteln, welche Probleme mit der Erweiterung von Programmen verbunden sein können und inwieweit diese mit dem verwendeten Paradigma in Verbindung stehen.

Problemspezifikation

- Die arithmetischen Ausdrücke bestehen aus ganzzahligen Werten, Negationen und Additionen.
- Für diese Ausdrücke sollen die Operationen
 - Auswertung (Bestimmung des Zahlenwerts)
 - Umwandlung in eine Zeichenkettendarstellung
 - Bestimmung, ob der Ausdruck die Konstante 0 enthält
- Die Konstellation kann durch diese Matrix zusammengefasst werden.

	eval	toString	hasZero
Constant			
Add			
Negation			

- Gleichgültig welches Paradigma benutzt werden soll, es muss für jedes der neun Felder in der Matrix das geeignete Verhalten bestimmt werden.

Funktionale Lösung in ML

- Die drei Datenarten werden an einer Stelle definiert.
- Die Implementierung der Operationen erfolgt „spaltenweise“.

```

datatype exp =
  Constant of int
  | Negate of exp
  | Add of exp * exp

exception BadResult of string

fun eval e =
  case e of
    Constant _ => e
    | Negate e1 => (case eval e1 of
        Constant i => Constant (~i)
      | _ => raise BadResult "non-int in negation")
    | Add(e1,e2) => (case (eval e1, eval e2) of
        (Constant i, Constant j) => Constant (i+j)
      | _ => raise BadResult "non-ints in addition")

fun toString e =

```

```

    case e of
Constant i => Int.toString i
| Negate e1 => "-(" ^ (toString e1) ^ ")"
| Add(e1,e2) => "(" ^ (toString e1) ^ " + " ^ (toString e2) ^ ")"

fun hasZero e =
  case e of
Constant i      => i=0
| Negate e1     => hasZero e1
| Add(e1,e2)   => (hasZero e1) orelse (hasZero e2)

```

Eine objektorientierte Lösung ...

- ... sollte die folgenden Regeln beachten:
 - Definition einer Klasse für Ausdrücke mit je einer abstrakten Methode für jede Operation
 - Definition einer Unterklasse für jede Datenvariante
 - In jeder der Unterklassen Definition einer Methode für jede Operation; Gemeinsamkeiten könnten in der Oberklasse definiert werden.
 - Die Implementierung erfolgt „zeilenweise“.
- Teil 1 von Aufgabe (OOP versus FP)

Implementieren Sie das geschilderte Problem in einer objektorientierten Sprache Ihrer Wahl!

Zwischenbilanz

- Welche Lösung ist die bessere?
- Oft eine Frage der persönlichen Sichtweise, ob es „natürlicher“ erscheint, das Problem zeilen- oder spaltenweise zu zerlegen
- Für das Problem der Ausdrücke ist der funktionale Ansatz möglicherweise übersichtlicher. Die verschiedenen Fälle der `eval`-Funktion beieinander zu haben ist leichter überschaubar als die Operationen für `ADD` oder `Negate`.
- Für die Entwicklung einer graphischen Benutzeroberfläche ist die objektorientierte Sichtweise vielleicht besser: Hier möchte man eher die Operationen einer bestimmten Datenart, wie z. B. einer Menüleiste, beieinander zu haben, als alle bei einem Mausklick zu berücksichtigenden Fälle.

Erweiterung des Programms um neue Datenvarianten und Operationen

- Die Vor- und Nachteile der betrachteten Paradigmen werden deutlicher, wenn das Programm erweitert werden muss.
- Teil 2 von Aufgabe (OOP versus FP)

1. Erweitern Sie beide Varianten des Programms um die neue Operation `noNegConstants`, die alle negativen Konstanten in positive verwandelt werden. (Fügen Sie also der Matrix eine neue Spalte hinzu.)
2. Erweitern Sie beide Varianten des Programms um die neue Datenvariante `Mult` (in Analogie zu `Add`). (Fügen Sie also der Matrix eine neue Zeile hinzu.)
3. Dokumentieren Sie den Implementierungsaufwand für beide Erweiterungen.

Metamorphose einer Beispielanwendung

- von einer prozeduralen zu einer objektorientierten und einer funktionalen Lösung
- unter Benutzung der hybriden Sprache Javascript
- unter freundlicher Verwendung der Materialien von Rafal Dittwald

Prozeduren, Methoden, Funktionen

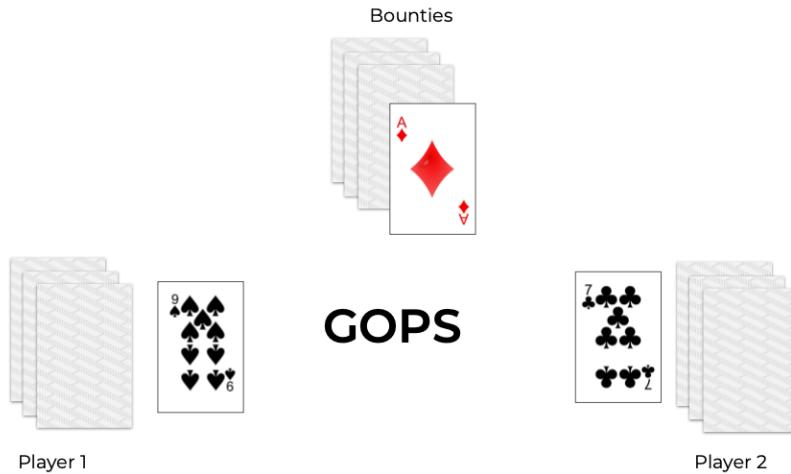
Prozedur Unterprogramm

Methode Prozedur innerhalb eines Objekts

Funktion Prozedur, die

- dynamisch erzeugt werden kann,
- als Wert betrachtet werden kann (first class values),
- keinen Namen haben muss,
- idealerweise keine Nebeneffekte, keine Zustandsänderungen hervorruft (pur),
- aufgerufen mit den gleichen Argumenten immer das gleiche Resultat liefert (pur)

Wir spielen GOPS



Prozedural-imperative Lösung

```

function runGame() {
    let turn = 0;
    let bountyCards = [1, 2, 3, 4, 5, 6, 7, 8];
    let playerCards = [[1, 2, 3, 4, 5, 6, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8]];
    let playerScores = [0, 0];

    while(bountyCards.length > 0) {
        const bountyCard = popRandom(bountyCards);
        const card0 = playRandomStrategy(playerCards[0], bountyCard);
        const card1 = playEqualStrategy(playerCards[1], bountyCard);
        turn += 1;

        if(card0 > card1) {
            playerScores[0] += bountyCard;
        } else if(card1 > card0) {
            playerScores[1] += bountyCard;
        }
    }

    console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

    if (playerScores[0] == playerScores[1]) {
        console.log("Players Tie!");
    } else if (playerScores[0] > playerScores[1]) {
        console.log("Player 0 Wins!");
    } else {
        console.log("Player 1 Wins!");
    }
}

runGame();

```

```

function popRandom(arr) {
    const index = Math.floor(Math.random() * arr.length);
    const value = arr[index];
    arr.splice(index, 1);
    return value;
}

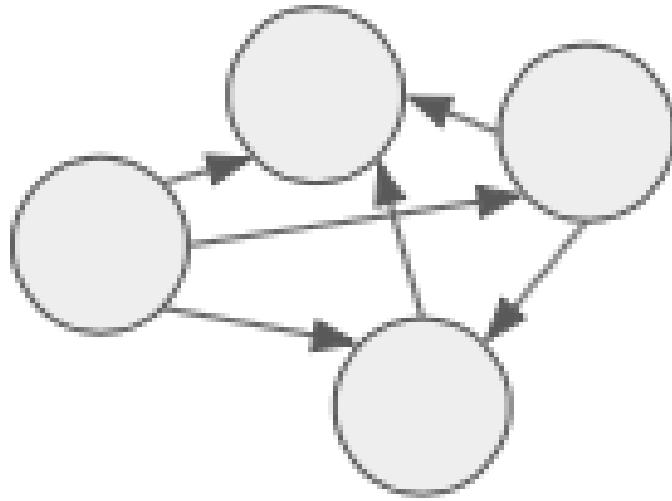
function playRandomStrategy(playerCards, bountyCard) {
    const card = popRandom(playerCards);
    console.log(`\tPlayer 0 plays: ${card}`);
    return card;
}

function playEqualStrategy(playerCards, bountyCard) {
    playerCards.splice(playerCards.indexOf(bountyCard), 1);
    console.log(`\tPlayer 1 plays: ${bountyCard}`);
    return bountyCard;
}

```

Objektorientierte Lösung

- Zustandsgrößen stiften Verwirrung, verstecken wir sie in Objekten.
- Ein objektorientiertes Programm ist ein System interagierender Objekte:



Wo sehen Sie hier Objekte?

```

function runGame() {
  let turn = 0;
  let bountyCards = [1, 2, 3, 4, 5, 6, 7, 8];
  let playerCards = [[1, 2, 3, 4, 5, 6, 7, 8],
    [1, 2, 3, 4, 5, 6, 7, 8]];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = popRandom(bountyCards);
    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);
    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    turn += 1;

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

  if (playerScores[0] == playerScores[1]) {
    console.log("Players Tie!");
  } else if (playerScores[0] > playerScores[1]) {
    console.log("Player 0 Wins!");
  } else {
    console.log("Player 1 Wins!");
  }
}

runGame();

```

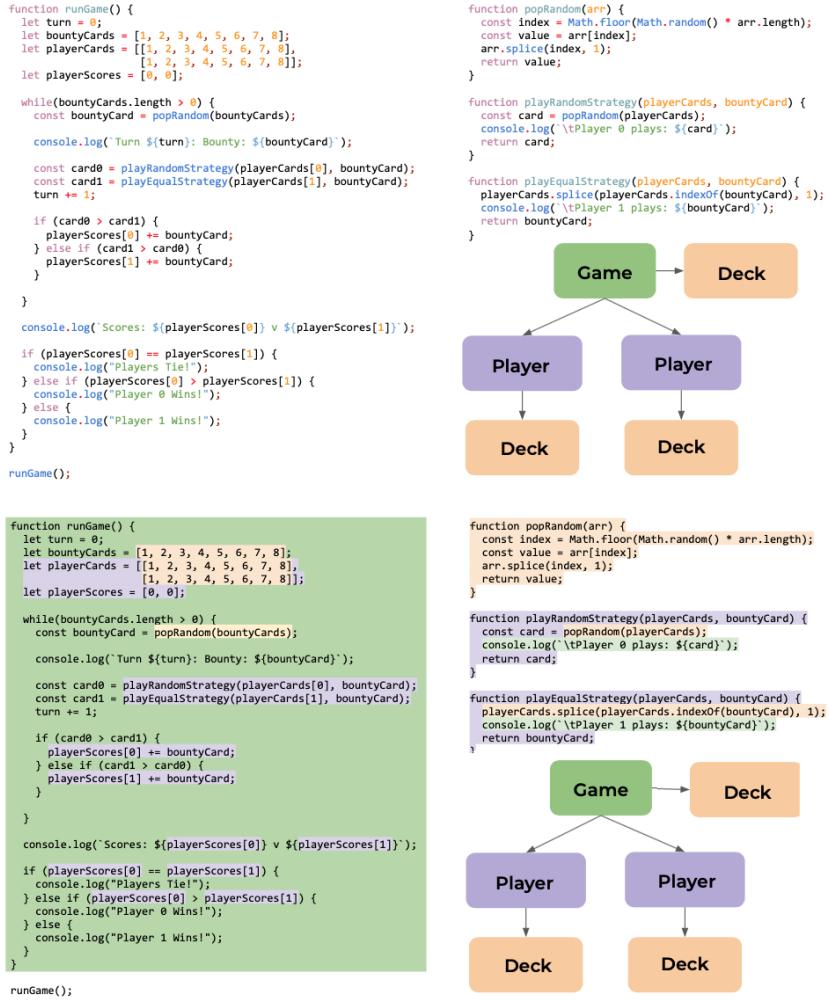
```

function popRandom(arr) {
  const index = Math.floor(Math.random() * arr.length);
  const value = arr[index];
  arr.splice(index, 1);
  return value;
}

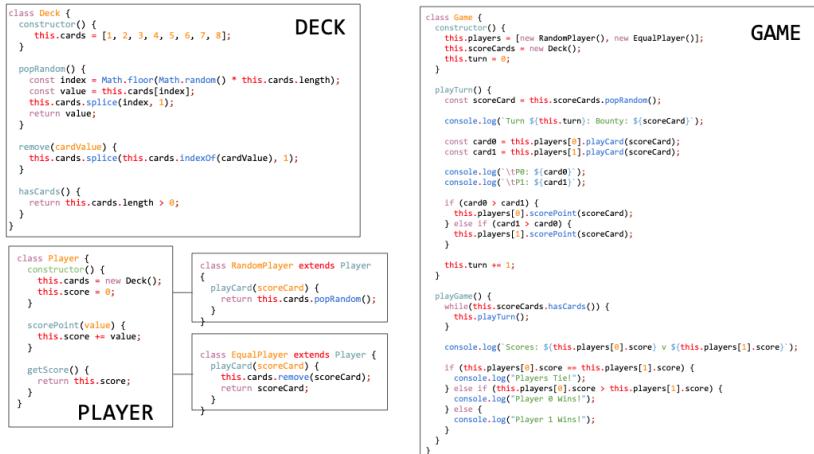
function playRandomStrategy(playerCards, bountyCard) {
  const card = popRandom(playerCards);
  console.log(`\tPlayer 0 plays: ${card}`);
  return card;
}

function playEqualStrategy(playerCards, bountyCard) {
  playerCards.splice(playerCards.indexOf(bountyCard), 1);
  console.log(`\tPlayer 1 plays: ${bountyCard}`);
  return bountyCard;
}

```

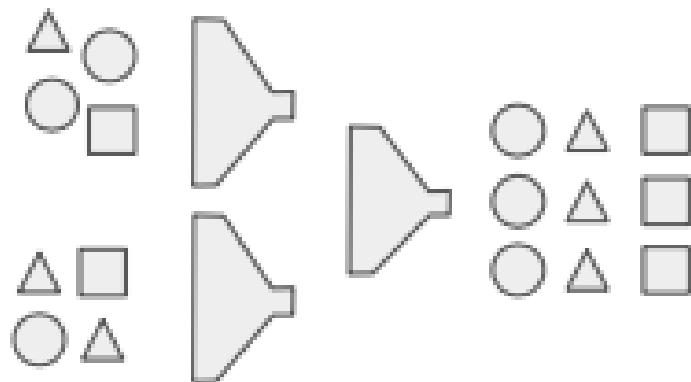


Definition von Klassen



Funktionale Lösung

- Vermeidung änderbarer Zustände (*mutable states*)
- Vermeidung von Seiteneffekten
- ein funktionales Programm beschreibt einen Datenfluss von der Eingabe zur Ausgabe:



Vermeidungsstrategien

Wie vermeiden wir...

	Zustände	Änderungen	Seiteneffekte
minimieren	unveränderbare Daten	kopieren statt ändern	
	Ableiten von Werten	verwende map, reduce, etc. verwende Lambda-Ausdrücke verwende natürliche Rekursion	
konzentrieren	An einer oder einigen wenigen Stellen zusammenfassen		
verlagern	An den Rand des Systems oder nach außerhalb bewegen		

Zustände, Änderungen, Seiteneffekte in der OO-Lösung

The diagram illustrates the state, changes, and side effects in an object-oriented solution for a card game. It shows two main classes: `Deck` and `Game`.

Deck Class:

```

class Deck {
  constructor() {
    this.cards = [1,2,3,4,5,6,7,8];
  }
  popRandom() {
    const index = Math.floor(Math.random() * this.cards.length);
    const value = this.cards[index];
    this.cards.splice(index, 1);
    return value;
  }
  remove(cardValue) {
    this.cards.splice(this.cards.indexOf(cardValue), 1);
  }
  hasCards() {
    return this.cards.length > 0;
  }
}

```

Player Class:

```

class Player {
  constructor() {
    this.cards = new Deck();
    this.score = 0;
  }
  scorePoint(value) {
    this.score += value;
  }
  getScore() {
    return this.score;
  }
}

```

RandomPlayer Class:

```

class RandomPlayer extends Player {
  playCard(scoreCard) {
    return this.cards.popRandom();
  }
}

```

EqualPlayer Class:

```

class EqualPlayer extends Player {
  playCard(scoreCard) {
    this.cards.remove(scoreCard);
    return scoreCard;
  }
}

```

Game Class:

```

class Game {
  constructor() {
    this.players = [new RandomPlayer(), new EqualPlayer()];
    this.scoreCards = new Deck();
    this.turn = 0;
  }
  playTurn() {
    const scoreCard = this.scoreCards.popRandom();
    console.log(`Turn ${this.turn}: Bounty: ${scoreCard}`);
    const card0 = this.players[0].playCard(scoreCard);
    const card1 = this.players[1].playCard(scoreCard);
    console.log(`\tP0: ${card0}`);
    console.log(`\tP1: ${card1}`);
    if (card0 > card1) {
      this.players[0].scorePoint(scoreCard);
    } else if (card1 > card0) {
      this.players[1].scorePoint(scoreCard);
    }
    this.turn += 1;
  }
  playGame() {
    while(this.scoreCards.hasCards()) {
      this.playTurn();
    }
    console.log(`Scores: ${this.players[0].score} v ${this.players[1].score}`);
    if (this.players[0].score == this.players[1].score) {
      console.log(`Players Tie!`);
    } else if (this.players[0].score > this.players[1].score) {
      console.log(`Player 0 Wins!`);
    } else {
      console.log(`Player 1 Wins!`);
    }
  }
}

```

Legend:

- Änderbare Zustände (Blue)
- Änderungen (Orange)
- Seiteneffekte (Green)

Zustände, Änderungen, Seiteneffekte in der prozeduralen Lösung

The diagram illustrates the state, changes, and side effects in a procedural solution for the same card game. It shows a single function `runGame()`.

```

function runGame() {
  let turn = 0;
  let bountyCards = [1, 2, 3, 4, 5, 6, 7, 8];
  let playerCards = [[1, 2, 3, 4, 5, 6, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8]];
  let playerScore = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = popRandom(bountyCards);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    turn += 1;

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }

    console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

    if (playerScores[0] == playerScores[1]) {
      console.log(`Players Tie!`);
    } else if (playerScores[0] > playerScores[1]) {
      console.log(`Player 0 Wins!`);
    } else {
      console.log(`Player 1 Wins!`);
    }
  }
}

function runGame();

```

Legend:

- Änderbare Zustände (Blue)
- Änderungen (Orange)
- Seiteneffekte (Green)

Migrationsschritte

```

function runGame() {
  let turn = 0;
  let bountyCards = [1, 2, 3, 4, 5, 6, 7, 8];
  let playerCards = [[1, 2, 3, 4, 5, 6, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8]];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = popRandom(bountyCards);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    turn += 1;

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

  if (playerScores[0] == playerScores[1]) {
    console.log("Players Tie!");
  } else if (playerScores[0] > playerScores[1]) {
    console.log("Player 0 Wins!");
  } else {
    console.log("Player 1 Wins!");
  }
}

runGame();

```

function popRandom(arr) {
 const index = Math.floor(Math.random() * arr.length);
 const value = arr[index];
 arr.splice(index, 1);
 return value;
}

 function playRandomStrategy(playerCards, bountyCard) {
 const card = popRandom(playerCards);
 console.log(`\tPlayer 0 plays: \${card}`);
 return card;
}

 function playEqualStrategy(playerCards, bountyCard) {
 playerCards.splice(playerCards.indexOf(bountyCard), 1);
 console.log(`\tPlayer 1 plays: \${bountyCard}`);
 return bountyCard;
}

verschiebe console.logs


```

function runGame() {
  let turn = 0;
  let bountyCards = [1, 2, 3, 4, 5, 6, 7, 8];
  let playerCards = [[1, 2, 3, 4, 5, 6, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8]];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = popRandom(bountyCards);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    turn += 1;

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

  if (playerScores[0] == playerScores[1]) {
    console.log("Players Tie!");
  } else if (playerScores[0] > playerScores[1]) {
    console.log("Player 0 Wins!");
  } else {
    console.log("Player 1 Wins!");
  }
}

runGame();

```

function popRandom(arr) {
 const index = Math.floor(Math.random() * arr.length);
 const value = arr[index];
 arr.splice(index, 1);
 return value;
}

 function playRandomStrategy(playerCards, bountyCard) {
 return popRandom(playerCards);
}

 function playEqualStrategy(playerCards, bountyCard) {
 playerCards.splice(playerCards.indexOf(bountyCard), 1);
 return bountyCard;
}

verschiebe console.logs ✓


```

function runGame() {
  let turn = 0;
  let bountyCards = [1, 2, 3, 4, 5, 6, 7, 8];
  let playerCards = [[1, 2, 3, 4, 5, 6, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8]];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = popRandom(bountyCards);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    turn += 1;

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

  if (playerScores[0] == playerScores[1]) {
    console.log("Players Tie!");
  } else if (playerScores[0] > playerScores[1]) {
    console.log("Player 0 Wins!");
  } else {
    console.log("Player 1 Wins!");
  }
}

runGame();

```

function popRandom(arr) {
 const index = Math.floor(Math.random() * arr.length);
 const value = arr[index];
 arr.splice(index, 1);
 return value;
}

 function playRandomStrategy(playerCards, bountyCard) {
 return popRandom(playerCards);
}

 function playEqualStrategy(playerCards, bountyCard) {
 playerCards.splice(playerCards.indexOf(bountyCard), 1);
 return bountyCard;
}

bereinige popRandom()

```

function runGame() {
  let turn = 0;
  let bountyCards = [1, 2, 3, 4, 5, 6, 7, 8];
  let playerCards = [[1, 2, 3, 4, 5, 6, 7, 8],  

                     [1, 2, 3, 4, 5, 6, 7, 8]];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = selectRandom(bountyCards); ←
    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);

    turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

  if (playerScores[0] == playerScores[1]) {
    console.log("Players Tie!");
  } else if (playerScores[0] > playerScores[1]) {
    console.log("Player 0 Wins!");
  } else {
    console.log("Player 1 Wins!");
  }
}

runGame();

```



```

function selectRandom(arr) {
  const index = Math.floor(Math.random() * arr.length);
  return arr[index];
}

function playRandomStrategy(playerCards, bountyCard) {
  return selectRandom(playerCards);
}

function playEqualStrategy(playerCards, bountyCard) {
  playerCards.splice(playerCards.indexOf(bountyCard), 1);
  return bountyCard;
}

```

bereinige popRandom()

umbenennen: selectRandom()
entferne Zustandsänderung
...die aber weiterhin notwendig


```

function runGame() {
  let turn = 0;
  let bountyCards = [1, 2, 3, 4, 5, 6, 7, 8];
  let playerCards = [[1, 2, 3, 4, 5, 6, 7, 8],  

                     [1, 2, 3, 4, 5, 6, 7, 8]];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = selectRandom(bountyCards);
    bountyCards = without(bountyCards, bountyCard); ←
    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    playerCards[0] = without(playerCards[0], card0);
    turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

  if (playerScores[0] == playerScores[1]) {
    console.log("Players Tie!");
  } else if (playerScores[0] > playerScores[1]) {
    console.log("Player 0 Wins!");
  } else {
    console.log("Player 1 Wins!");
  }
}

runGame();

```



```

function selectRandom(arr) {
  const index = Math.floor(Math.random() * arr.length);
  return arr[index];
}

function playRandomStrategy(playerCards, bountyCard) {
  return selectRandom(playerCards);
}

function playEqualStrategy(playerCards, bountyCard) {
  playerCards.splice(playerCards.indexOf(bountyCard), 1);
  return bountyCard;
}

function without(arr, value) {
  const index = arr.indexOf(value);
  return [...arr.slice(0, index), ...arr.slice(index+1)];
}

```

bereinige popRandom()

umbenennen: selectRandom()
entferne Zustandsänderung
... die aber weiterhin notwendig
... deswegen nach runGame
verschieben


```

function runGame() {
  let turn = 0;
  let bountyCards = [1, 2, 3, 4, 5, 6, 7, 8];
  let playerCards = [[1, 2, 3, 4, 5, 6, 7, 8],  

                     [1, 2, 3, 4, 5, 6, 7, 8]];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = selectRandom(bountyCards);
    bountyCards = without(bountyCards, bountyCard); ←
    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    playerCards[0] = without(playerCards[0], card0);
    turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

  if (playerScores[0] == playerScores[1]) {
    console.log("Players Tie!");
  } else if (playerScores[0] > playerScores[1]) {
    console.log("Player 0 Wins!");
  } else {
    console.log("Player 1 Wins!");
  }
}

runGame();

```



```

function selectRandom(arr) {
  const index = Math.floor(Math.random() * arr.length);
  return arr[index];
}

function playRandomStrategy(playerCards, bountyCard) {
  return selectRandom(playerCards);
}

function playEqualStrategy(playerCards, bountyCard) {
  playerCards.splice(playerCards.indexOf(bountyCard), 1);
  return bountyCard;
}

function without(arr, value) {
  const index = arr.indexOf(value);
  return [...arr.slice(0, index), ...arr.slice(index+1)];
}

```

bereinige popRandom()

umbenennen: selectRandom()
entferne Zustandsänderung
... die aber weiterhin notwendig
... deswegen nach runGame
Verschieben, und
... Hilfsfunktion without hinzufügen

```

function runGame() {
  let turn = 0;
  let bountyCards = [1, 2, 3, 4, 5, 6, 7, 8];
  let playerCards = [[1, 2, 3, 4, 5, 6, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8]];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = selectRandom(bountyCards);
    bountyCards = without(bountyCards, bountyCard);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    playerCards[0] = without(playerCards[0], card0);
    turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

  if (playerScores[0] == playerScores[1]) {
    console.log("Players Tie!");
  } else if (playerScores[0] > playerScores[1]) {
    console.log("Player 0 Wins!");
  } else {
    console.log("Player 1 Wins!");
  }
}

runGame();

```

bereinige popRandom()

Math.random() nicht strikt funktional


```

function runGame() {
  let turn = 0;
  let bountyCards = shuffle([1, 2, 3, 4, 5, 6, 7, 8]);
  let playerCards = [shuffle([1, 2, 3, 4, 5, 6, 7, 8]), shuffle([1, 2, 3, 4, 5, 6, 7, 8])];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = selectRandom(bountyCards);
    bountyCards = without(bountyCards, bountyCard);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    playerCards[0] = without(playerCards[0], card0);
    turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

  if (playerScores[0] == playerScores[1]) {
    console.log("Players Tie!");
  } else if (playerScores[0] > playerScores[1]) {
    console.log("Player 0 Wins!");
  } else {
    console.log("Player 1 Wins!");
  }
}

runGame();

```

bereinige popRandom()

Math.random() nicht strikt funktional

Mischen der Karten beim Start


```

function runGame() {
  let turn = 0;
  let bountyCards = shuffle([1, 2, 3, 4, 5, 6, 7, 8]);
  let playerCards = [shuffle([1, 2, 3, 4, 5, 6, 7, 8]), shuffle([1, 2, 3, 4, 5, 6, 7, 8])];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = first(bountyCards); ←
    bountyCards = without(bountyCards, bountyCard);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    playerCards[0] = without(playerCards[0], card0);
    turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

  if (playerScores[0] == playerScores[1]) {
    console.log("Players Tie!");
  } else if (playerScores[0] > playerScores[1]) {
    console.log("Player 0 Wins!");
  } else {
    console.log("Player 1 Wins!");
  }
}

runGame();

```

bereinige popRandom() ✓

Math.random() nicht strikt funktional

Mischen der Karten beim Start und aus selectRandom() wird first()

```

function runGame() {
  let turn = 0;
  let bountyCards = shuffle([1, 2, 3, 4, 5, 6, 7, 8]);
  let playerCards = [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    shuffle([1, 2, 3, 4, 5, 6, 7, 8])];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = first(bountyCards);
    bountyCards = without(bountyCards, bountyCard);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    playerCards[0] = without(playerCards[0], card0);
    turn += 1;

    console.log(`\nPlayer 0 plays: ${card0}`);
    console.log(`\nPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

  if (playerScores[0] == playerScores[1]) {
    console.log("Players Tie!");
  } else if (playerScores[0] > playerScores[1]) {
    console.log("Player 0 Wins!");
  } else {
    console.log("Player 1 Wins!");
  }
}

runGame();

```

```

function first(arr) {
  return arr[0];
}

function playRandomStrategy(playerCards, bountyCard) {
  return first(playerCards);
}

function playEqualStrategy(playerCards, bountyCard) {
  playerCards.splice(playerCards.indexOf(bountyCard), 1);
  return bountyCard;
}

function without(arr, value) {
  const index = arr.indexOf(value);
  return [...arr.slice(0, index), ...arr.slice(index+1)];
}

function shuffle(arr) {...}

```

bereinige playEqualStrategy()

```

function runGame() {
  let turn = 0;
  let bountyCards = shuffle([1, 2, 3, 4, 5, 6, 7, 8]);
  let playerCards = [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    shuffle([1, 2, 3, 4, 5, 6, 7, 8])];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = first(bountyCards);
    bountyCards = without(bountyCards, bountyCard);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    playerCards[0] = without(playerCards[0], card0);
    playerCards[1] = without(playerCards[1], card1); ←

    turn += 1;

    console.log(`\nPlayer 0 plays: ${card0}`);
    console.log(`\nPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

  if (playerScores[0] == playerScores[1]) {
    console.log("Players Tie!");
  } else if (playerScores[0] > playerScores[1]) {
    console.log("Player 0 Wins!");
  } else {
    console.log("Player 1 Wins!");
  }
}

runGame();

```

```

function first(arr) {
  return arr[0];
}

function playRandomStrategy(playerCards, bountyCard) {
  return first(playerCards);
}

function playEqualStrategy(playerCards, bountyCard) {
  return bountyCard;
}

function without(arr, value) {
  const index = arr.indexOf(value);
  return [...arr.slice(0, index), ...arr.slice(index+1)];
}

function shuffle(arr) {...}

```

bereinige playEqualStrategy() ✓

Gib einfach die Karte zurück
verändere playerCards[]
(vorerst)

Zwischenstand

```

function runGame() {
  let turn = 0;
  let bountyCards = shuffle([1, 2, 3, 4, 5, 6, 7, 8]);
  let playerCards = [shuffle([1, 2, 3, 4, 5, 6, 7, 8]), shuffle([1, 2, 3, 4, 5, 6, 7, 8])];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = first(bountyCards);
    bountyCards = without(bountyCards, bountyCard);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    playerCards[0] = without(playerCards[0], card0);
    playerCards[1] = without(playerCards[1], card1);
    turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

  if (playerScores[0] == playerScores[1]) {
    console.log("Players Tie!");
  } else if (playerScores[0] > playerScores[1]) {
    console.log("Player 0 Wins!");
  } else {
    console.log("Player 1 Wins!");
  }
}

runGame();

```

Weitere Migrationsschritte

```

function runGame() {
  let turn = 0;
  let bountyCards = shuffle([1, 2, 3, 4, 5, 6, 7, 8]);
  let playerCards = [shuffle([1, 2, 3, 4, 5, 6, 7, 8]), shuffle([1, 2, 3, 4, 5, 6, 7, 8])];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = first(bountyCards);
    bountyCards = without(bountyCards, bountyCard);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    playerCards[0] = without(playerCards[0], card0);
    playerCards[1] = without(playerCards[1], card1);
    turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);

  if (playerScores[0] == playerScores[1]) {
    console.log("Players Tie!");
  } else if (playerScores[0] > playerScores[1]) {
    console.log("Player 0 Wins!");
  } else {
    console.log("Player 1 Wins!");
  }
}

runGame();

```

extrahiere winMessage()



```

function runGame() {
  let turn = 0;
  let bountyCards = shuffle([1, 2, 3, 4, 5, 6, 7, 8]);
  let playerCards = [shuffle([1, 2, 3, 4, 5, 6, 7, 8]), shuffle([1, 2, 3, 4, 5, 6, 7, 8])];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = first(bountyCards);
    bountyCards = without(bountyCards, bountyCard);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    playerCards[0] = without(playerCards[0], card0);
    playerCards[1] = without(playerCards[1], card1);
    turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);
  console.log(winMessage(playerScores));
}

runGame();

```

extrahiere winMessage()

```

function first(arr) { ... }
function playRandomStrategy(playerCards, bountyCard) { ... }
function playEqualStrategy(playerCards, bountyCard) { ... }
function without(arr, value) { ... }
function shuffle(arr) { ... }

function winMessage(playerScores) {
  if (playerScores[0] == playerScores[1]) {
    return "Players Tie!";
  } else if (playerScores[0] > playerScores[1]) {
    return "Player 0 Wins!";
  } else {
    return "Player 1 Wins!";
  }
}

```

```

function runGame() {
  let turn = 0;
  let bountyCards = shuffle([1, 2, 3, 4, 5, 6, 7, 8]);
  let playerCards = [shuffle([1, 2, 3, 4, 5, 6, 7, 8]), shuffle([1, 2, 3, 4, 5, 6, 7, 8])];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = first(bountyCards);
    bountyCards = without(bountyCards, bountyCard);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    playerCards[0] = without(playerCards[0], card0);
    playerCards[1] = without(playerCards[1], card1);
    turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(`Scores: ${playerScores[0]} v ${playerScores[1]}`);
  console.log(winMessage(playerScores));
}

runGame();

```

Zusammenfassung der Endnachrichten

```

function runGame() {
  let turn = 0;
  let bountyCards = shuffle([1, 2, 3, 4, 5, 6, 7, 8]);
  let playerCards = [shuffle([1, 2, 3, 4, 5, 6, 7, 8]), shuffle([1, 2, 3, 4, 5, 6, 7, 8])];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = first(bountyCards);
    bountyCards = without(bountyCards, bountyCard);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);
    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    playerCards[0] = without(playerCards[0], card0);
    playerCards[1] = without(playerCards[1], card1);
    turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(endMessage(playerScores));
}

runGame();

```

Zusammenfassung der Endnachrichten ✓

Zwischenstand 2

```

function runGame() {
  let turn = 0;
  let bountyCards = shuffle([1, 2, 3, 4, 5, 6, 7, 8]);
  let playerCards = [shuffle([1, 2, 3, 4, 5, 6, 7, 8]), shuffle([1, 2, 3, 4, 5, 6, 7, 8])];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = first(bountyCards);
    bountyCards = without(bountyCards, bountyCard);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);
    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    playerCards[0] = without(playerCards[0], card0);
    playerCards[1] = without(playerCards[1], card1);
    turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(endMessage(playerScores));
}

runGame();

```

Weitere Migrationsschritte

```

function runGame() {
  let turn = 0;
  let bountyCards = shuffle([1, 2, 3, 4, 5, 6, 7, 8]);
  let playerCards = [shuffle([1, 2, 3, 4, 5, 6, 7, 8]), shuffle([1, 2, 3, 4, 5, 6, 7, 8])];
  let playerScores = [0, 0];

  while(bountyCards.length > 0) {
    const bountyCard = first(bountyCards);
    bountyCards = without(bountyCards, bountyCard);

    console.log(`Turn ${turn}: Bounty: ${bountyCard}`);
    const card0 = playRandomStrategy(playerCards[0], bountyCard);
    const card1 = playEqualStrategy(playerCards[1], bountyCard);
    playerCards[0] = without(playerCards[0], card0);
    playerCards[1] = without(playerCards[1], card1);
    turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      playerScores[1] += bountyCard;
    }
  }

  console.log(endMessage(playerScores));
}

runGame();

```

Zusammenfassung aller Zustandsvariablen in einer

```

function runGame() {
  let state = {turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]), shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0];
  }

  while(state.bountyCards.length > 0) {
    const bountyCard = first(state.bountyCards);
    state.bountyCards = without(state.bountyCards, bountyCard);

    console.log(`Turn ${state.turn}: Bounty: ${bountyCard}`);
    const card0 = playRandomStrategy(state.playerCards[0], bountyCard);
    const card1 = playEqualStrategy(state.playerCards[1], bountyCard);
    playerCards[0] = without(state.playerCards[0], card0);
    playerCards[1] = without(state.playerCards[1], card1);
    state.turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      state.playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      state.playerScores[1] += bountyCard;
    }
  }

  console.log(endMessage(state.playerScores));
}

runGame();

```

Zusammenfassung aller Zustandsvariablen in einer ✓

```

function runGame() {
  let state = {turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]), shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0];
  }

  while(state.bountyCards.length > 0) {
    const bountyCard = first(state.bountyCards);
    state.bountyCards = without(state.bountyCards, bountyCard);

    console.log(`Turn ${state.turn}: Bounty: ${bountyCard}`);
    const card0 = playRandomStrategy(state.playerCards[0], bountyCard);
    const card1 = playEqualStrategy(state.playerCards[1], bountyCard);
    playerCards[0] = without(state.playerCards[0], card0);
    playerCards[1] = without(state.playerCards[1], card1);
    state.turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      state.playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      state.playerScores[1] += bountyCard;
    }
  }

  console.log(endMessage(state.playerScores));
}

runGame();

```

Definition einer Funktion nextState()

```

function runGame() {
  let state = {turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]), shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0];
  }

  while(state.bountyCards.length > 0) {
    const bountyCard = first(state.bountyCards);
    state.bountyCards = without(state.bountyCards, bountyCard);

    console.log(`Turn ${state.turn}: Bounty: ${bountyCard}`);
    const card0 = playRandomStrategy(state.playerCards[0], bountyCard);
    const card1 = playEqualStrategy(state.playerCards[1], bountyCard);
    playerCards[0] = without(state.playerCards[0], card0);
    playerCards[1] = without(state.playerCards[1], card1);
    state.turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      state.playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      state.playerScores[1] += bountyCard;
    }
  }

  console.log(endMessage(state.playerScores));
}

runGame();

```

```

function first(arr) {}
function playRandomStrategy(playerCards, bountyCard) {}
function playEqualStrategy(playerCards, bountyCard) {}
function without(arr, value) {}
function shuffle(arr) {}
function winMessage(playerScores) {}
function scoreMessage(playerScores) {}
function endMessage(playerScores) {}

function nextState(state) {
  return {turn: ?, bountyCards: ?, playerCards: ?, playerScores: ?}
}

```

Definition einer Funktion nextState()

```

function runGame() {
  let state = {turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
      shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]};

  while(state.bountyCards.length > 0) {
    const bountyCard = first(state.bountyCards);
    state.bountyCards = without(state.bountyCards, bountyCard);

    console.log(`Turn ${state.turn}: Bounty: ${bountyCard}`);

    const card0 = playRandomStrategy(state.playerCards[0], bountyCard);
    const card1 = playEqualStrategy(state.playerCards[1], bountyCard);
    playerCards[0] = without(state.playerCards[0], card0);
    playerCards[1] = without(state.playerCards[1], card1);
    state.turn += 1;

    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);

    if (card0 > card1) {
      state.playerScores[0] += bountyCard;
    } else if (card1 > card0) {
      state.playerScores[1] += bountyCard;
    }
  }

  console.log(endMessage(state.playerScores));
}

runGame();

```

Definition einer Funktion nextState()

```

function runGame() {
  let state = {turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
      shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]};

  while(state.bountyCards.length > 0) {
    state = nextState(state);

    console.log(`Turn ${state.turn}: Bounty: ${bountyCard}`);
    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);
  }

  console.log(endMessage(state.playerScores));
}

runGame();

```

Definition einer Funktion nextState()

```

function runGame() {
  let state = {turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
      shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]};

  while(state.bountyCards.length > 0) {
    state = nextState(state);

    console.log(`Turn ${state.turn}: Bounty: ${bountyCard}`);
    console.log(`\tPlayer 0 plays: ${card0}`);
    console.log(`\tPlayer 1 plays: ${card1}`);
  }

  console.log(endMessage(state.playerScores));
}

runGame();

```

Definition einer Funktion nextState()

```

function runGame() {
  let state = {turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
      shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]};

  while(state.bountyCards.length > 0) {
    state = nextState(state);

    console.log(`Turn ${state.turn}: Bounty: ${state.lastBountyCard}`);
    console.log(`\tPlayer 0 plays: ${state.lastPlayedCards[0]}`);
    console.log(`\tPlayer 1 plays: ${state.lastPlayedCards[1]}`);
  }

  console.log(endMessage(state.playerScores));
}

runGame();

```

```

function first(arr) {...}
function playRandomStrategy(playerCards, bountyCard) {...}
function playEqualStrategy(playerCards, bountyCard) {...}
function without(arr, value) {...}
function shuffle(arr) {...}
function winMessage(playerScores) {...}
function scoreMessage(playerScores) {...}
function endMessage(playerScores) {...}

function nextState(state) {
  const bountyCard = first(state.bountyCards);
  const card0 = playRandomStrategy(state.playerCards[0], bountyCard);
  const card1 = playEqualStrategy(state.playerCards[1], bountyCard);

  return {turn: state.turn + 1,
    lastBountyCard: bountyCard,
    lastPlayedCards: [card0, card1],
    bountyCards: without(state.bountyCards, bountyCard),
    playerCards: [without(state.playerCards[0]), without(state.playerCards[1])],
    playerScores: newScores(state.playerScores, [card0, card1])}
}

function newScores(playerScores, playedCards) {
  if(playedCards[0] > playedCards[1]){
    return [playerScores[0]+1, playerScores[1]];
  } else if (playedCards[1] > playedCards[0]){
    return [playerScores[0], playerScores[1]+1];
  }
}

```

Definition einer Funktion nextState() ✓

Zwischenstand 3

```

function runGame() {
  let state = {turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
      shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]};

  while(state.bountyCards.length > 0) {
    state = nextState(state);

    console.log(`Turn ${state.turn}: Bounty: ${state.lastBountyCard}`);
    console.log(`\tPlayer 0 plays: ${state.lastPlayedCards[0]}`);
    console.log(`\tPlayer 1 plays: ${state.lastPlayedCards[1]}`);
  }

  console.log(endMessage(state.playerScores));
}

runGame();

```

```

function first(arr) {...}
function playRandomStrategy(playerCards, bountyCard) {...}
function playEqualStrategy(playerCards, bountyCard) {...}
function without(arr, value) {...}
function shuffle(arr) {...}
function winMessage(playerScores) {...}
function scoreMessage(playerScores) {...}
function endMessage(playerScores) {...}

function nextState(state) {
  const bountyCard = first(state.bountyCards);
  const card0 = playRandomStrategy(state.playerCards[0], bountyCard);
  const card1 = playEqualStrategy(state.playerCards[1], bountyCard);

  return {turn: state.turn + 1,
    lastBountyCard: bountyCard,
    lastPlayedCards: [card0, card1],
    bountyCards: without(state.bountyCards, bountyCard),
    playerCards: [without(state.playerCards[0]), without(state.playerCards[1])],
    playerScores: newScores(state.playerScores, [card0, card1])}
}

function newScores(playerScores, playedCards) {
  if(playedCards[0] > playedCards[1]){
    return [playerScores[0]+1, playerScores[1]];
  } else if (playedCards[1] > playedCards[0]){
    return [playerScores[0], playerScores[1]+1];
  }
}

```

Letzte Migrationsschritte

```

function runGame() {
  let state = {turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
      shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]};

  while(state.bountyCards.length > 0) {
    state = nextState(state);

    console.log(`Turn ${state.turn}: Bounty: ${state.lastBountyCard}`);
    console.log(`\tPlayer 0 plays: ${state.lastPlayedCards[0]}`);
    console.log(`\tPlayer 1 plays: ${state.lastPlayedCards[1]}`);
  }

  console.log(endMessage(state.playerScores));
}

runGame();

```

```

function first(arr) {...}
function playRandomStrategy(playerCards, bountyCard) {...}
function playEqualStrategy(playerCards, bountyCard) {...}
function without(arr, value) {...}
function shuffle(arr) {...}
function winMessage(playerScores) {...}
function scoreMessage(playerScores) {...}
function endMessage(playerScores) {...}
function nextState(state) {...}
function newScores(playerScores, playedCards) {...}

```

Definition einer turnMessage()

```

function runGame() {
  let state = {turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
      shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]};
  while(state.bountyCards.length > 0) {
    state = nextState(state);
    console.log(turnMessage(state));
  }
  console.log(endMessage(state));
}
runGame();

```

```

function first(arr) {...}
function playRandomStrategy(playerCards, bountyCard) {...}
function playEqualStrategy(playerCards, bountyCard) {...}
function without(arr, value) {...}
function shuffle(arr) {...}
function winMessage(playerScores) {...}
function scoreMessage(playerScores) {...}
function endMessage(playerScores) {...}
function nextState(state) {...}
function newScores(playerScores, playedCards) {...}
function turnMessage(state) {
  return `Turn ${state.turn}: Bounty: ${state.lastBountyCard}
  + ${'\n' * player 0 plays: ${state.lastPlayedCards[0]}}
  + ${'\n' * player 1 plays: ${state.lastPlayedCards[1]}}`;
}

```

Definition einer turnMessage() ✓

```

function runGame() {
  let state = {turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
      shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]};
  while(state.bountyCards.length > 0) {
    state = nextState(state);
    console.log(turnMessage(state));
  }
  console.log(endMessage(state));
}
runGame();

```

```

function first(arr) {...}
function playRandomStrategy(playerCards, bountyCard) {...}
function playEqualStrategy(playerCards, bountyCard) {...}
function without(arr, value) {...}
function shuffle(arr) {...}
function winMessage(playerScores) {...}
function scoreMessage(playerScores) {...}
function endMessage(playerScores) {...}
function nextState(state) {...}
function newScores(playerScores, playedCards) {...}
function turnMessage(state) {...}

```

Speichern der Zustände in Array

```

function runGame() {
  let states = [{turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
      shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]}];
  while(state.bountyCards.length > 0) {
    states = states.concat(nextState(state));
    console.log(turnMessage(last(states)));
  }
  console.log(endMessage(last(states)));
}
runGame();

```

```

function first(arr) {...}
function playRandomStrategy(playerCards, bountyCard) {...}
function playEqualStrategy(playerCards, bountyCard) {...}
function without(arr, value) {...}
function shuffle(arr) {...}
function winMessage(playerScores) {...}
function scoreMessage(playerScores) {...}
function endMessage(playerScores) {...}
function nextState(state) {...}
function newScores(playerScores, playedCards) {...}
function turnMessage(state) {...}
function last(arr) {
  return arr[arr.length - 1];
}

```

Speichern der Zustände in Array ✓

und definiere last()

```

function runGame() {
  let states = [ {turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
      shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]}];
  while(state.bountyCards.length > 0) {
    states = states.concat(nextState(state));
    console.log(turnMessage(last(states)));
  }
  console.log(endMessage(last(states)));
}
runGame();

```

```

function first(arr) {...}
function playRandomStrategy(playerCards, bountyCard) {...}
function playEqualStrategy(playerCards, bountyCard) {...}
function without(arr, value) {...}
function shuffle(arr) {...}
function nextState(state, playerScores) {...}
function scoreMessage(playerScores) {...}
function endMessage(playerScores) {...}
function nextState(state) {...}
function newScores(playerScores, playedCards) {...}
function turnMessage(state) {...}
function last(arr) {...}

```

aufspalten in runGame() und report()

```

function runGame() {
  let states = [ {turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
      shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]}];
  while(state.bountyCards.length > 0) {
    states = states.concat(nextState(state));
    console.log(turnMessage(last(states)));
  }
  console.log(endMessage(last(states)));
  return states;
}
runGame();

```

```

function first(arr) {...}
function playRandomStrategy(playerCards, bountyCard) {...}
function playEqualStrategy(playerCards, bountyCard) {...}
function without(arr, value) {...}
function shuffle(arr) {...}
function winMessage(playerScores) {...}
function scoreMessage(playerScores) {...}
function endMessage(playerScores) {...}
function nextState(state) {...}
function newScores(playerScores, playedCards) {...}
function turnMessage(state) {...}
function last(arr) {...}

```

runGame() returniert states

aufspalten in runGame() und report()

```

function runGame() {
  let states = [ {turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
      shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]}];
  while(state.bountyCards.length > 0) {
    states = states.concat(nextState(state));
    console.log(turnMessage(last(states)));
  }
  console.log(endMessage(last(states)));
  return states;
}
runGame();

```

```

function first(arr) {...}
function playRandomStrategy(playerCards, bountyCard) {...}
function playEqualStrategy(playerCards, bountyCard) {...}
function without(arr, value) {...}
function shuffle(arr) {...}
function winMessage(playerScores) {...}
function scoreMessage(playerScores) {...}
function endMessage(playerScores) {...}
function nextState(state) {...}
function newScores(playerScores, playedCards) {...}
function turnMessage(state) {...}
function last(arr) {...}

```

definiere report()

aufspalten in runGame() und report()

```

function runGame() {
  let states = [{turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]}];
  while(state.bountyCards.length > 0) {
    states = states.concat(nextState(state));
    console.log(turnMessage(last(states)));
  }
  console.log(endMessage(last(states)));
  return states;
}
runGame();

function report(states, onTurn, onEnd) {
  return states.map(onTurn).join("\n")
  + "\n" + onEnd(last(states));
}

console.log(report(runGame(), turnMessage, endMessage)); ← states an report übergeben

```

aufspalten in runGame() und report()

```

function runGame() {
  let states = [{turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]}];
  while(state.bountyCards.length > 0) {
    states = states.concat(nextState(state));
    console.log(turnMessage(last(states))); ← nicht mehr gebraucht
  }
  console.log(endMessage(last(states))); ← nicht mehr gebraucht
  return states;
}
runGame(); ← states an report übergeben

function report(states, onTurn, onEnd) {
  return states.map(onTurn).join("\n")
  + "\n" + onEnd(last(states));
}

console.log(report(runGame(), turnMessage, endMessage)); ← states an report übergeben

```

aufspalten in runGame() und report()

```

function runGame() {
  let states = [{turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]}];
  while(state.bountyCards.length > 0) {
    states = states.concat(nextState(state));
  }
  return states;
}

function report(states, onTurn, onEnd) {
  return states.map(onTurn).join("\n")
  + "\n" + onEnd(last(states));
}

console.log(report(runGame(), turnMessage, endMessage));

```

aufspalten in runGame() und report() ✓

```

function runGame() {
  let states = [{turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
      shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]}];
  while(state.bountyCards.length > 0) {
    states = states.concat(nextState(state));
  }
  return states;
}

```

```

function first(arr) {...}
function playRandomStrategy(playerCards, bountyCard) {...}
function playEqualStrategy(playerCards, bountyCard) {...}
function without(arr, value) {...}
function shuffle(arr) {...}
function winMessage(playerScores) {...}
function scoreMessage(playerScores) {...}
function endMessage(playerScores) {...}
function nextState(state) {...}
function newScores(playerScores, playedCards) {...}
function turnMessage(state) {...}
function last(arr) {...}

```

```

function report(states, onTurn, onEnd) {
  return states.map(onTurn).join("\n")
    + "\n" + onEnd(last(states));
}

console.log(report(runGame(), turnMessage, endMessage));

```

Rekursion anstelle von while

```

function runGame() {
  let states = [{turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
      shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]}];
  while(state.bountyCards.length > 0) {
    states = states.concat(nextState(state));
  }
  return states;
}

```

recur() Hilfsfunktion

```

function first(arr) {...}
function playRandomStrategy(playerCards, bountyCard) {...}
function playEqualStrategy(playerCards, bountyCard) {...}
function without(arr, value) {...}
function shuffle(arr) {...}
function winMessage(playerScores) {...}
function scoreMessage(playerScores) {...}
function endMessage(playerScores) {...}
function nextState(state) {...}
function newScores(playerScores, playedCards) {...}
function turnMessage(state) {...}
function last(arr) {...}

```

```

function report(states, onTurn, onEnd) {
  return states.map(onTurn).join("\n")
    + "\n" + onEnd(last(states));
}

console.log(report(runGame(), turnMessage, endMessage));

```

Rekursion anstelle von while

```

function runGame() {
  const states = [{turn: 0,
    bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
    playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8]),
      shuffle([1, 2, 3, 4, 5, 6, 7, 8])],
    playerScores: [0, 0]}];
  return recur(states, nextState,
    function(state) { return state.bountyCards.length > 0 });
}

function report(states, onTurn, onEnd) {
  return states.map(onTurn).join("\n")
    + "\n" + onEnd(last(states));
}

console.log(report(runGame(), turnMessage, endMessage));

```

benutze recur()

```

function first(arr) {...}
function playRandomStrategy(playerCards, bountyCard) {...}
function playEqualStrategy(playerCards, bountyCard) {...}
function without(arr, value) {...}
function shuffle(arr) {...}
function winMessage(playerScores) {...}
function scoreMessage(playerScores) {...}
function endMessage(playerScores) {...}
function nextState(state) {...}
function newScores(playerScores, playedCards) {...}
function turnMessage(state) {...}
function last(arr) {...}

```

```

function recur(states, stateChange, endCondition) {
  if(endCondition(last(states))) {
    return states;
  } else {
    return recur(
      states.concat(stateChange(last(states))),
      stateChange,
      endCondition
    );
  }
}

function recur(states, stateChange, endCondition) {
  if(endCondition(last(states))) {
    return states;
  } else {
    return recur(
      states.concat(stateChange(last(states))),
      stateChange,
      endCondition
    );
  }
}

```

Rekursion anstelle von while



Fazit

```
function runGame(seed) {
    const states = [{turn: 0,
        bountyCards: shuffle([1, 2, 3, 4, 5, 6, 7, 8], seed),
        playerCards: [shuffle([1, 2, 3, 4, 5, 6, 7, 8], seed/2),
            shuffle([1, 2, 3, 4, 5, 6, 7, 8], seed/3)],
        playerScores: [0, 0]}];
    return recur(states, nextState,
        function(state) { return state.bountyCards.length > 0 });
}

function first(arr) {...}
function playRandomStrategy(playerCards, bountyCard) {...}
function playEqualStrategy(playerCards, bountyCard) {...}
function without(arr, value) {...}
function shuffle(arr, seed) {...}
function winMessage(playerScores) {...}
function scoreMessage(playerScores) {...}
function turnMessage(state) {...}
function last(arr) {...}
function recur(states, stateChange, endCondition) {...}
```

```
function report(states, onTurn, onEnd) {
    return states.map(onTurn).join("\n")
    + "\n" + onEnd(last(states));
}

console.log(report(runGame(Math.random()), turnMessage, endMessage));
```

Was ist gewonnen?

Antworten

- tatsächlich etwas mehr Code aber nahezu ausschließlich aus reinen Funktionen bestehend
- Diese Art des Refactorings (Betrachtung von Zuständen, Zustandsänderungen und Seiteneffekten) liefert
 - klare semantische Trennung
 - einfachste Testbarkeit
 - einfache „Argumentierbarkeit“ (reason about)
 - * reine Funktionen können des Rest des Universums ignorieren
 - * Referenztransparenz: Funktionsaufrufe können durch ihren Wert ersetzt werden.
 - Reine Funktionen können trivialerweise memoisiert werden.
 - Reine Funktionen können einfach parallelisiert werden.

Funktionale Entwurfsmuster

Vorbemerkungen

- Welche Entwurfsmuster kennen Sie?
- Wozu dienen Entwurfsmuster?
- Das berühmte Buch: [?]
- These:

Jedes Entwurfsmuster offenbart einen Mangel der verwendeten Programmiersprache.

So fand Peter Norvig heraus, dass 16 der 23 Entwurfsmuster aus dem oben genannten Buch in Sprachen wie Lisp verschwinden bzw. sich stark vereinfachen lassen ([?]).

- Mehrere Entwurfsmuster behandeln bzw. implizieren Zustandsänderungen und sind daher nicht auf die funktionale Programmierung übertragbar.

- Viele (objektorientierte) Entwurfsmuster sind in funktionalen Sprachen Idiome. So wird das Iterator-Muster in Sprachen wie LISP oder ML nicht benötigt.
- Ein Bild, das in der Community kursiert (z. B. hier):

OO pattern/principle	FP pattern/principle
• Single Responsibility Principle	• Functions
• Open/Closed principle	• Functions
• Dependency Inversion Principle	• Functions, also
• Interface Segregation Principle	• Functions
• Factory pattern	• Yes, functions
• Strategy pattern	• Oh my, functions again!
• Decorator pattern	• Functions
• Visitor pattern	• Functions ☺

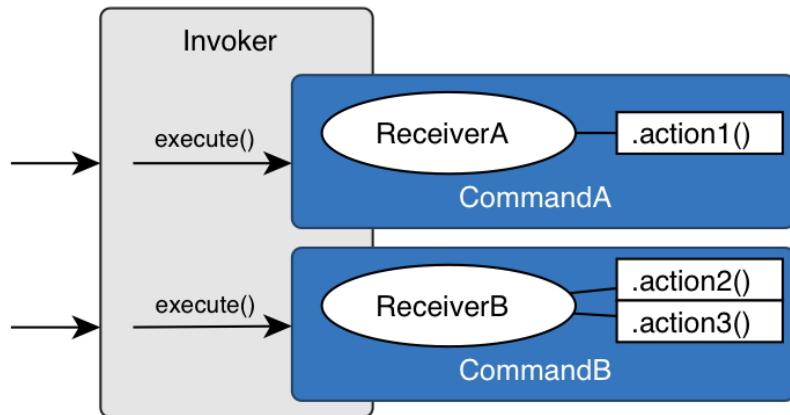
- Eine Kategorie der Entwurfsmuster sind die Verhaltensmuster (behavioral patterns). Charakteristisch für die Muster ist die Beobachtung, dass sie Verben (Aktivitäten, Funktionen) in Substantive (Objekte) verwandelt. Funktion werden in Objekte „verpackt“.
- Damit werden häufig natürliche Konzepte der funktionalen Programmierung für die Objektorientierung passend gemacht, häufig auf Kosten von Lesbarkeit und Prägnanz.

Konkrete Muster

Das Befehlsmuster (command pattern)

Zitiert aus Philipp Hauer: Design Pattern:

Das Command Design Pattern ermöglicht die Modularisierung von Befehlen und Aufrufen. Auf elegante Weise können Befehle rückgängig gemacht, protokolliert oder in einer Warteschlange gelegt werden.



Beispiel

- Es werden verschiedene Commands betrachtet, die eine Nachricht verarbeiten.
- Basis: ein Interface

```

interface Command {
    void run();
}

```

- Verschiedene Implementierungen des Interface:

1. für das Logging

```

public class Logger implements Command {
    public final String message;

    public Logger( String message ) {
        this.message = message;
    }

    @Override
    public void run() {
        System.out.println("Logging: " + message);
    }
}

```

2. Speichern der Nachricht

```

public class FileSaver implements Command {
    public final String message;

    public FileSaver( String message ) {
        this.message = message;
    }
}

```

```

@Override
public void run() {
    System.out.println("Saving: " + message);
}
}

```

3. Versenden per Email

```

public class Mailer implements Command {
    public final String message;

    public Mailer( String message ) {
        this.message = message;
    }

    @Override
    public void run() {
        System.out.println("Sending: " + message);
    }
}

```

- Ein Objekt, das diese Commands ausführen kann:

```

public class Executor {
    public void execute(List<Command> tasks) {
        for (Command task : tasks) {
            task.run();
        }
    }
}

```

- Ausführung einer Liste von Commands durch den Executor

```

List<Command> tasks = new ArrayList<>();
tasks.add(new Logger( "Hi" ));
tasks.add(new FileSaver( "Cheers" ));
tasks.add(new Mailer( "Bye" ));

new Executor().execute( tasks );

```

- Wie in [?] gefordert, werden Funktionen (die auszuführenden Aktionen) in Objekte verpackt (die Commands, die diese Aktionen ausführen).
- Diese Indirektion hat keinen Vorteil, außer Funktionen in strikt objektorientierter Weise behandeln zu können.

Die funktionale Version

- Durch die Einführung von Lambda-Ausdrücken in **Java 8** ist es möglich, beide Paradigmen (OO und FP) nahtlos miteinander zu verknüpfen.

- Das **Command**-Interface wird nicht benötigt, stattdessen kann das bekannte **Runnable**-Interface benutzt werden.
- Die drei **Command**-Implementierungen können in Java 8 prägnant durch drei als statische Methoden implementierte Funktionen geschrieben werden:

```
public static void log(String message) {
    System.out.println("Logging: " + message);
}

public static void save(String message) {
    System.out.println("Saving: " + message);
}

public static void send(String message) {
    System.out.println("Sending: " + message);
}
```

- Das „Signal-Rausch-Verhältnis“ des Codes wird deutlich verbessert, wenn man als Signal den Rumpf der Funktionen und als Rauschen den Code, der zusätzlich benötigt wird, um aus den Funktionen Methoden zu machen, betrachtet.
- Auch die **Executor**-Klasse kann durch eine statische Methode implementiert werden, die eine Liste von **Runnables** als Argument akzeptiert:

```
public static void execute(List<Runnable> tasks ) {
    tasks.forEach( Runnable::run );
}
```

- Die Funktionen können nun wieder in eine Liste gepackt und wie zuvor ausgeführt werden:

```
List<Runnable> tasks = new ArrayList<>();
tasks.add(() -> log("Hi"));
tasks.add(() -> save("Cheers"));
tasks.add(() -> send("Bye"));

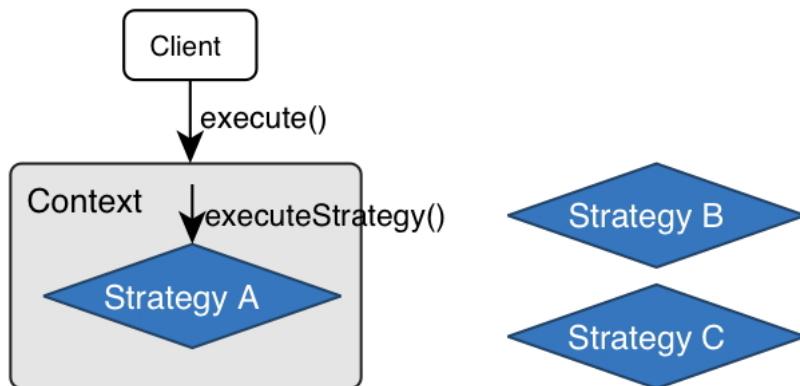
execute( tasks );
```

- Der Java-Compiler übersetzt die Lambda-Ausdrücke ohne Argumente, die die statischen Methoden (`log`, `save`, `send`) ausführen, in anonyme Implementierungen des **Runnable**-Interface, wodurch sie in eine Liste von **Runnables** gepackt werden können.

Das Strategiemuster (strategy pattern)

Zitiert aus Philipp Hauer: Design Pattern:

Das Strategy Design Pattern ermöglicht einer Klasse das flexible Wechseln von alternativen Verhalten.



Beispiel

- Das Ziel des Strategiemusters besteht in der Verallgemeinerung einer Prozedur durch Verwendung verschiedener austauschbarer Algorithmen.
- Hier soll eine Prozedur verallgemeinert werden, die
 - einen Text verarbeitet,
 - ihn nach gegebenen Kriterien filtert und
 - nach einer Formatierung/Transformation ausdrückt.
- Es wird also die Verallgemeinerung von zwei Verhaltensweisen benötigt:
 - Filtern des Textes
 - Transformation des Textes
- Im ersten Schritt werden Verhaltensweisen durch ein Interface abstrahiert:

```

interface TextFormatter {
    boolean filter(String text);
    String format(String text);
}

```

- Danach ist es möglich, eine Klasse zu entwerfen, die eine allgemeine Prozedur für die Publikation eines Textes implementiert.
- Ihr wird ein Exemplar eines `TextFormatter` Interface (die Strategie) übergeben, das die Details, wie der Text gefiltert und formatiert werden soll, kapselt:

```

public class TextEditor {
    private final TextFormatter textFormatter;

    public TextEditor(TextFormatter textFormatter) {
        this.textFormatter = textFormatter;
    }
}

```

```

        public void publishText(String text) {
            if (textFormatter.filter( text )) {
                System.out.println( textFormatter.format( text ) );
            }
        }
    }
}

```

- Drei verschiedene Strategie-Implementierungen:

1. die einfachste Variante: der Text wird unverändert ausgedruckt.

```

public class PlainTextFormatter implements TextFormatter {

    @Override
    public boolean filter( String text ) {
        return true;
    }

    @Override
    public String format( String text ) {
        return text;
    }
}

```

2. Der Text wird Zeile für Zeile nach Fehlermeldungen durchsucht. Es werden nur die Zeilen, die mit "ERROR" beginnen, in Großbuchstaben ausgedruckt.

```

public class ErrorTextFormatter implements TextFormatter {

    @Override
    public boolean filter( String text ) {
        return text.startsWith( "ERROR" );
    }

    @Override
    public String format( String text ) {
        return text.toUpperCase();
    }
}

```

3. Text wird in Kleinbuchstaben ausgedruckt, sofern er kürzer als zwanzig Zeichen ist.

```

public class ShortTextFormatter implements TextFormatter {

    @Override
    public boolean filter( String text ) {
        return text.length() < 20;
    }

    @Override

```

```

    public String format( String text ) {
        return text.toLowerCase();
    }
}

```

- Nun kann ein `TextEditor`-Exemplar angelegt und dabei mit einem bestimmten `TextFormatter`-Exemplar versehen werden:

```

TextEditor textEditor = new TextEditor( new ErrorTextFormatter() );
textEditor.publishText( "ERROR - something bad happened" );
textEditor.publishText( "DEBUG - I'm here" );

```

- Es stellt sich erneut die Frage, ob die Lösung nicht wortreicher ist, als sie sein sollte.
- Das einzige relevante Verhalten (das „Signal“) besteht in der Logik der `publishText`-Methode der `Textformatter`-Klasse. Der Rest ist „Rauschen“.

Die funktionale Version

- Die beiden durch den `Textformatter` definierten Verhaltensweisen können problemlos in Form von zwei Funktionen an die `publishText`-Methode übergeben werden:

- ein Prädikat (`Predicate`) für die Filterung des Textes
- eine Transformationsfunktion `UnaryOperator`, die ein Objekt in ein anderes desselben Typs verwandelt

```

public static void publishText( String text,
                               Predicate<String> filter,
                               UnaryOperator<String> format) {
    if (filter.test( text )) {
        System.out.println( format.apply( text ) );
    }
}

```

- Benutzung von `publishText` als `PlainTextFormatter`:

```
publishText( "DEBUG - I'm here", s -> true, s -> s );
```

- Benutzung von `publishText` als `ErrorTextFormatter`:

```

publishText( "ERROR - something bad happened",
            s -> s.startsWith( "ERROR" ),
            String::toUpperCase );

```

- **Möglicher Einwand:** Die Implementierung des `TextFormatter` als Klasse ermöglicht einfacher, eine Bibliothek von Strategien zu realisieren, anstatt ihre Implementierung bei jedem Aufruf erneut zu implementieren.

- Funktionales Gegenmodell:

```

public class TextUtil {
    public boolean acceptAll(String text) {
        return true;
    }

    public String noFormatting(String text) {
        return text;
    }

    public boolean acceptErrors(String text) {
        return text.startsWith( "ERROR" );
    }

    public String formatError(String text) {
        return text.toUpperCase();
    }
}

```

- Benutzung der Klasse `TextUtil`: Anstelle der anonymen Funktionen werden nun die dort definierten Methoden/Funktionen verwendet; z. B.:

```

publishText( "DEBUG - I'm here", TextUtil::acceptAll, TextUtil::noFormatting );

publishText( "ERROR - something bad happened",
            TextUtil::acceptErrors, TextUtil::formatError );

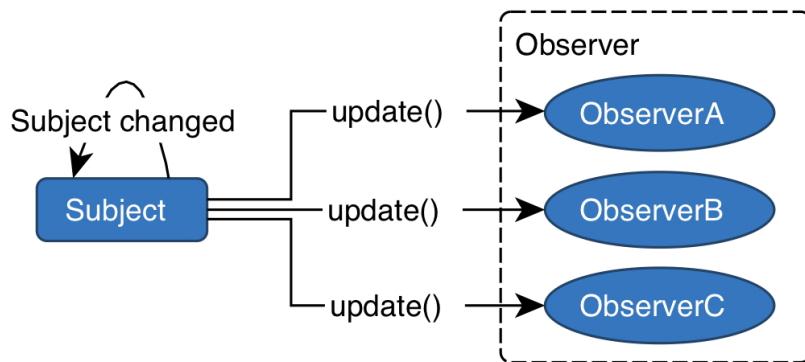
```

- Diese Funktionen besitzen eine feinere Granularität als eine Strategie-Klasse.
- Damit ergibt sich eine bessere Wiederverwendbarkeit.

Das Beobachtermuster (observer pattern)

Zitiert aus Philipp Hauer: Design Pattern:

Das Observer Entwurfsmuster ist für Situationen geeignet, in denen ein oder mehrere Objekte benachrichtigt werden müssen, sobald sich der Zustand eines bestimmten Objekts ändert.



Beispiel

- Das Entwurfsmuster erlaubt einem oder mehreren Objekten, über auftretende Ereignisse informiert zu werden.
- Die elementare Abstraktion ist das **Listener**-Interface:

```
interface Listener {  
    void onEvent(Object event);  
}
```

- Ein Objekt, das auf das Auftreten des Ereignisses (**event**) reagieren soll, muss lediglich dieses Interface implementieren und in der **onEvent**-Methode die erforderliche Reaktion festlegen.
- Das Gegenstück zu einem beobachtenden Objekt ist das beobachtete (beobachtbare, zu beobachtende): ein **Observable**-Objekt.
- Ein Objekt, das an alle registrierten Beobachter (listeners) eine Nachricht über ein aufgetretenes Ereignis sendet:

```
public class Observable {  
    private final Map<Object, Listener> listeners = new ConcurrentHashMap<>();  
  
    public void register(Object key, Listener listener) {  
        listeners.put(key, listener);  
    }  
  
    public void unregister(Object key) {  
        listeners.remove(key);  
    }  
  
    public void sendEvent(Object event) {  
        for (Listener listener : listeners.values()) {  
            listener.onEvent( event );  
        }  
    }  
}
```

- Zwei (herkömmliche) Varianten, ein **Observer**-Objekt zu implementieren:

1. durch eine anonyme, innere Klasse:

```
public class Observer1 {  
    Observer1(Observable observable) {  
        observable.register( this, new Listener() {  
            @Override  
            public void onEvent( Object event ) {  
                System.out.println(event);  
            }  
        } );  
    }  
}
```

2. durch die direkte Implementierung des Listener-Interface:

```
public class Observer2 implements Listener {
    Observer2(Observable observable) {
        observable.register( this, this );
    }
    @Override
    public void onEvent( Object event ) {
        System.out.println(event);
    }
}
```

- Beide Varianten können in der gleichen Weise benutzt werden:

```
Observable observable = new Observable();
new Observer1( observable );
new Observer2( observable );
observable.sendEvent( "Hello World!" );
```

- Auch hier wird das Kernproblem eines großen Teils der GoF-Muster deutlich: Verben, die Aktionen beschreiben, müssen in Substantive verwandelt werden: Klassen, die die Aktionen einwickeln.

Die funktionale Version

- Schritt 1: Ersetzen des Listener-Interface durch das semantisch äquivalente Consumer-Interface:

```
public static class Observable {
    private final Map<Object, Consumer<Object>> listeners = new ConcurrentHashMap<>();

    public void register(Object key, Consumer<Object> listener) {
        listeners.put(key, listener);
    }

    public void unregister(Object key) {
        listeners.remove(key);
    }

    public void sendEvent(Object event) {
        listeners.values().forEach( listener -> listener.accept( event ) );
    }
}
```

- Schritt 2:

- Die Implementierung der Listener durch eigene Klassen ist nicht mehr notwendig.
- Die spezifische Reaktion auf das Eintreten eines Ereignisses kann durch einen Lambda-Ausdruck oder einen Methoden-Verweis kodiert werden:

```
Observable observable = new Observable();
observable.register( "key1", e -> System.out.println(e) );
observable.register( "key2", System.out::println );
observable.sendEvent( "Hello World!" );
```

Literaturverzeichnis

- [GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.
Design Patterns: Elements of Reusable Object-Oriented Software.
Addison-Wesley, 1995.
- [Nor98] Peter Norvig. Design patterns in dynamic languages, 1998. zuletzt
aufgerufen am 10.09.2018.