Vorbemerkung zum 5.Aufgabenzettel und allen weiteren Aufgabenzetteln

- Ein **wichtiges (Lern-)Ziel der folgenden Aufgaben** ist auch, dass Sie die richtige Datenstruktur an der richtigen Stelle verwenden. Also diejenige Datenstruktur, die für den Zweck optimal geeignet ist und die nötigen darauf wirkenden Operationen optimal unterstützt – im Sinne von zu möglichst einfachen Methoden führt.
  Achtung! Wir haben verschiedene ADT, Collections, Map, … mehr oder weniger intensiv kennengelernt. Auch dies sind Datenstrukturen.

Aufgabe A5.0 Vorbereitungsaufgabe: N gleicher Farbe

Diese Aufgabe ist eine Vorbereitungsaufgabe. Auch wenn die Bearbeitung dieser Aufgabe Pflicht ist, wird Ihre Bearbeitung (vermutlich) nicht im Labor kontrolliert. Sie können aber gern Verständnisfragen im Labor stellen.

Lesen Sie zunächst den Methodenkopfkommentar der Methode `drawSameColour()` durch. Schauen Sie sich dann die Implementierung an und verstehen Sie diese.

Aufgabe A5.1 In umgekehrter Reihenfolge ausgeben

Schreiben Sie eine Methode **`reverseOrder()`**, die zunächst solange Karten von einem gegebenen Kartenstapel zieht bis eine gewünschte Karte gezogen wurde und danach alle bisher gezogenen Karten in umgekehrter Reihenfolge ausgibt.
Also, die letzte gezogene Karte (das ist die gewünschte Karte) zuerst ausgeben und die erste gezogene Karte zuletzt ausgeben. (Ja, wir denken "LIFO" bzw. Last In First Out ;-)
Es darf vorausgesetzt werden, dass der Kartenstapel 52 Karten enthält bzw. die gewünschte Karte auch im Kartenstapel vorhanden ist.
Die Methode soll 3 Parameter aufweisen, die in der nachfolgenden Reihenfolge:

- den gegebenen Kartenstapel,
- die gewünschte Karte und
- einen Wahrheitswert für eine optionale Ausgabe der jeweils gezogenen Karte unmittelbar nach der Ziehung entgegen nehmen.

Siehe hierzu auch den gestellten TestFrame.

Aufgabe A5.2 Keine doppelten Karten

Schreiben Sie eine Methode **`removeDuplicates()`**, die beliebig viele Karten entgegen nimmt. Aus diesen Karten sollen die Doppelten entfernt werden. Die so bereinigten Karten sind als Ergebnis (konkret Array über Karten) zurückzugeben.
Siehe hierzu auch den gestellten TestFrame.

Aufgabe A5.3 Mit Comparator sortieren lassen

Schreiben Sie einen Comparator **`UsualOrder`**, der Karten vergleicht und es Ihnen ermöglicht eine Liste über Karten (List<Card>) mit Collections.sort() zu sortieren.
Die Karten sollen nach Aufruf von Collections.sort() nach folgender Ordnung sortiert sein.
Mit **erste**r **Priorität** nach **Ränge**n (Rank) und zwar Ass vor König, König vor Dame, Dame vor Bube, Bube vor 10, 10 vor 9, 9 vor 8, 8 vor 7, 7 vor 6, 6 vor 5, 5 vor 4, 4 vor 3 und 3 vor 2.
Mit **zweite**r **Priorität** nach **Farbe**n (Suit) und zwar Kreuz vor Pik, Pik vor Herz, Herz vor Karo.
Siehe hierzu auch den gestellten TestFrame.

Aufgabe A5.4 Resistance Net

Im Rahmen dieser Aufgabe sollen Sie u.a. geeignete Referenztypen für **ComposedResistor**, **ParallelResistor**, **Potentiometer**, **ResistanceNet**, **Resistor** und **SeriesResistor** implementieren.

Widerstände (engl. "resistor") sind elektrische Bauteile mit einem Widerstandswert, gemessen in der Einheit Ohm mit dem Einheitenzeichen Ω. Aus Widerständen lassen sich Widerstandsnetze zusammensetzen. Hierfür gelten die folgenden Konstruktionsregeln:

- Ein einzelner Widerstand mit dem Widerstandswert R ist ein Widerstandsnetz, wenn auch ein sehr einfaches.
- Zwei oder mehr Netze mit den Widerstandswerten R_1 bis R_n können in Reihe, das heißt hintereinander, geschaltet werden (siehe hierzu beispielsweise http://de.wikipedia.org/wiki/Reihenschaltung). Die Kombination ist ein neues Netz mit dem Gesamt-Widerstandswert R, der bestimmt ist durch:
  $$R = R_1 + \ldots + R_n$$
- Zwei oder mehr Netze mit den Widerstandswerten R_1 bis R_n können parallel, das heißt nebeneinander, geschaltet werden (siehe hierzu beispielsweise http://de.wikipedia.org/wiki/Parallelschaltung). Die Kombination ist ein neues Netz mit einem Gesamt-Widerstandswert R, der bestimmt ist durch:
  $$R = 1 / (1/R_1 + \ldots + 1/R_n) \quad \text{bzw.: } 1/R = 1/R_1 + \ldots + 1/R_n$$

Implementieren Sie geeignete Referenztypen zur Repräsentation derartiger Schaltungen:

- Definieren Sie einen geeigneten Referenztypen für Widerstandsnetze mit Namen: **ResistanceNet**. Objekte, die diesem Typ genügen müssen zumindest auch die folgenden Methoden aufweisen:
  - **`double getResistance()`**
    Liefert den Gesamtwiderstand des Netzes.
  - **`int getNumberOfResistors()`**
    Liefert die Anzahl an einfachen Widerständen im Netz (in nachfolgender Abbildung1 z.B. 6).
  - **`String getCircuit()`**
    Liefert eine Beschreibung der Schaltung als String. Siehe nachfolgende Abbildung1 für Beispiel.

- Definieren Sie einen geeigneten Referenztypen für "einfache" Widerstände mit Namen: **Resistor**. Achtung! Jeder Widerstand ist auch ein Widerstandsnetz. **Resistor** muss einen Konstruktor mit der Signatur **`Resistor( String, double )`** aufweisen. Hierbei ist der 1.Parameter der Name des Widerstandes und der 2.Parameter der Widerstandswert gemessen in Ohm. Der Widerstandswert muss für Objekte des Typs **Resistor** unveränderlich sein (keine setter-Methode), für abgeleitete Referenztypen soll es jedoch eine Zugriffsmöglichkeit geben.
  Die zugehörige **`getCircuit()`**-Methode soll nur den Namen des Widerstands liefern.

- Definieren Sie einen geeigneten Referenztypen für zusammengesetzte Widerstandsnetze mit Namen: **ComposedResistor**. Zusammengesetzte Widerstandsnetze sind Widerstandsnetze, aber keine "einfachen" Widerstände. Objekte, die diesem Typ genügen müssen zumindest auch die folgenden Methoden aufweisen:
  - **`ResistanceNet[] getSubNets()`**
    Liefert ein Array der Widerstandsnetze aus denen das zusammengesetzte Widerstandsnetz unmittelbar zusammengesetzt ist in der "Original-Reihenfolge" - also genau in der gleichen Reihenfolge in der diese an den Konstruktor übergeben wurden.
- Definieren Sie einen geeigneten Referenztypen für Reihen-Schaltungen von Widerstandsnetzen (bzw. serielle Widerstandsnetze) mit Namen: **SeriesResistor**.

  Die **`getCircuit()`**-Methode eines seriellen Widerstandsnetzes soll alle unmittelbar enthaltenen Widerstandsnetze als String durch "+" getrennt liefern. Der String / das gesamte Widerstandsnetz ist von Runden Klammern umrahmt:

  Beispiel: (Widerstandsnetz$_1$ + Widerstandsnetz$_2$ + Widerstandsnetz$_3$)

  Achtung! Um mögliche Rundungsfehler bei der Berechnung des Gesamt-Widerstands $R = R_1 + \ldots + R_n$ zu minimieren, müssen Sie in geeigneter Reihenfolge die Werte verknüpfen bzw. an der "entscheidenden Stelle" beginnend mit dem jeweils kleinesten zum größten Wert hin in aufsteigender Reihenfolge aufaddieren. Die ursprüngliche im Konstruktor übergebene Reihenfolge darf jedoch nicht verloren gehen.

- Definieren Sie einen geeigneten Referenztypen für Parallel-Schaltungen von Widerstandsnetzen (bzw. parallele Widerstandsnetze) mit Namen: **ParallelResistor**.

  Die **`getCircuit()`**-Methode eines parallele Widerstandsnetzes soll alle unmittelbar enthaltenen Widerstandsnetze als String durch "|" getrennt liefern. Der String / das gesamte Widerstandsnetz ist von Runden Klammern umrahmt:

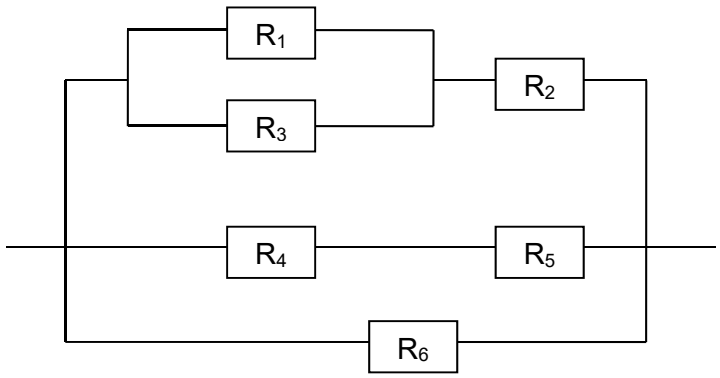
  Beispiel: (Widerstandsnetz$_1$ | Widerstandsnetz$_2$ | Widerstandsnetz$_3$)

  Achtung! Um mögliche Rundungsfehler zu minimieren, sollten Sie in geeigneter Reihenfolge die Werte verknüpfen bzw. an der "entscheidenden Stelle" beginnend mit dem jeweils kleinesten zum größten Wert hin in aufsteigender Reihenfolge aufaddieren. Die ursprüngliche im Konstruktor übergebene Reihenfolge darf jedoch nicht verloren gehen.

- Sowohl serielle Widerstandsnetze (**SeriesResistor**) wie auch parallele Widerstandsnetze (**ParallelResistor**) sind zusammengesetzte Widerstandsnetze (**ComposedResistor**). Die Konstruktoren mit denen Objekte zusammengesetzter Widerstandsnetze erzeugt werden können, müssen beliebig viele Teil-Widerstandsnetze (**ResistanceNet**) aus denen sich die zusammengesetzten Widerstandsnetze zusammensetzen `T` als Argumente akzeptieren (Tipp: "varargs-Parameter") und speichern diese.
  Beispiel: **`new ParallelResistor(r1, r3)`**

- Schreiben Sie eine Anwendung, die das nachfolgend abgebildet Widerstandsnetz aufbaut.

  Abbildung1: Widerstandsnetz

  Die Widerstände R_1 bis R_6 haben die Werte 100 Ω, 200 Ω, ..., 600 Ω.

  Für das links abgebildete Widerstandsnetz gilt dann:

  - Der Gesamtwiderstand beträgt:
    (etwa) 155.91 Ω
  - **`getNumberOfResistors()`** liefert:
    6 Widerstände
  - **`getCircuit()`** liefert z.B. den String:
    `(((R1|R3)+R2)|(R4+R5)|R6)`
    Achtung! Die Anzahl Klammern hängt vom Netzaufbau ab.

- Unterstützen Sie auch einen Spezialfall eines einzelnen Widerstands in Form eines Potenziometers. Potenziometer sind einfache Widerstände mit einem regelbaren Widerstandswert (spezielle Eigenschaft). Jedes Potenziometer ist ein Widerstand. Definieren Sie einen geeigneten Referenztypen für Potenziometer mit Namen: **Potentiometer**, der auch einen Konstruktor mit der Signatur **`Potentiometer( String )`** aufweist sowie zusätzlich eine Setter-Methode **`setResistance( double )`** für den zugehörigen Widerstandswert bietet.

- Ersetzen Sie in der oben skizzierten Schaltung den Widerstand R_4 durch ein Potenziometer. Schreiben Sie eine neue Anwendung, die eine Liste der Widerstandswerte der modifizierten Schaltung ausgibt, wenn das Potenziometer in Schritten von 400 von 0 bis auf 4000 Ohm hochgeregelt wird.

- Erstellen Sie ein UML-Klassendiagramm **bevor** Sie zu implementieren anfangen.
  Dieses Klassendiagramm ist bei der Abnahme vorzulegen.

- Leider waren Packages noch nicht Thema in der Vorlesung und werden auch von BlueJ nicht optimal unterstützt. Das nachfolgende (Sub-)Package **component** ist also eine freiwillige Zusatzaufgabe, die Sie mit dem Vorlesungswissen nicht lösen können. Aber die Aufgabe Resistance Net "schreit" ganz laut nach einem solchen Package und allzu schwierig ist es auch nicht. Sofern Sie diese freiwillige Zusatzaufgabe lösen möchten, legen Sie die Referenztypen: **ComposedResistor**, **ParallelResistor**, **Potentiometer**, **ResistanceNet**, **Resistor** und **SeriesResistor** (sowie mögliche sinnvolle(!) Hilfs-Referenztypen, die Sie bei der Implementierung unterstützen) in einem Sub-Package **component** ab.
  Ihr Test bzw. die "Anwendung der Komponenten" muss sich außerhalb dieses (Sub-)Packages befinden.