Programmierung 2 VU 051020

Übungseinheit 10

SS 2020

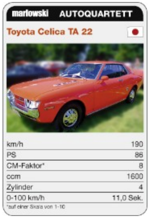
Projekt Setup

- Der Packagename für Ihr Projekt lautet **aMatrikelnummer.nachname**
- Instanzvariablen sind private zu deklarieren
- Die Signaturen der vorgegeben Methoden dürfen nicht verändert werden
- Java Version am almighty ist Java 11; verwenden Sie diese um Import-Probleme beim Abschlusstest zu vermeiden!

Autoquartett – Überblick

Ziel: Erstellung einer vereinfachten Simulation des Kartenspiels Autoquartett/Supertrumpf

- Ein Spieler hat ein Deck von Karten.
- Karten haben Kategorien mit Werten.
- Spieler können andere Spieler herausfordern.
- Es gibt spezielle Glitzerkarten (Foil), die mächtiger sind.

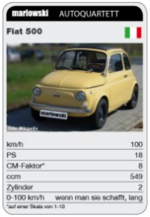


marlowski AUTOQUARTETT

Toyota Celica TA 22

km/h	190
PS	86
CM-Faktor*	8
ccm	1600
Zylinder	4
0-100 km/h	11,0 Sek.

*auf einer Skala von 1-10

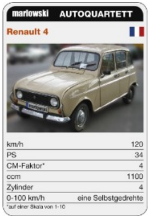


marlowski AUTOQUARTETT

Fiat 500

km/h	100
PS	18
CM-Faktor*	8
ccm	549
Zylinder	2
0-100 km/h	wenn man sie schafft, lang

*auf einer Skala von 1-10



marlowski AUTOQUARTETT

Renault 4

km/h	120
PS	34
CM-Faktor*	4
ccm	1100
Zylinder	4
0-100 km/h	eine Selbstgedrehte

*auf einer Skala von 1-10

[1]

[1]Quelle Bilder: http://www.marlowski-magazin.de/auto-quartett.html

Autoquartett – Überblick cont.

Folgende Klassen werden für den Abschlusstest benötigt

- class VehicleCard.Category $\rightarrow$ Kategorien der Karten
- class VehicleCard $\rightarrow$ Kartenklasse
- class FoilVehicleCard $\rightarrow$ Erbt von VehicleCard
- class Player $\rightarrow$ Besitzt ein Kartendeck und kann andere Spieler herausfordern.

Klasse: VehicleCard – Category

```
public class VehicleCard implements Comparable<VehicleCard> {
  public enum Category {
    //values:
    //PRICE_EUR, CYLINDER_CAPACITY_CM3, ENGINE_POWER_HP, ACCELERATION_SEC,
            VELOCITY_KMH, CONSUMPTION_L;

    // ("Preis",1), ("Hubraum",5), ("Leistung",4), ("Beschleunigung",3), ("Geschwindigkeit",2), ("Verbrauch",0)
    final private String categoryName;
    final private int factor;

    private Category(String categoryName,int factor) {/*throws IllegalArgumentException if categoryName null
            or empty or if factor less than 0 */}

    public int bonus(Double value) {
        /* return int(factor times value) */
        // must be overriden for CONSUMPTION_L. returns int(value + factor).
    }

    public int getFactor() {/*...*/}

    @Override
    public String toString() { /*categoryName*/ }
  }
  ...
}
```

Klasse: VehicleCard

```
public class VehicleCard implements Comparable<VehicleCard> {
  public enum Category {...}

  private String name;
  private Map<Category, Double> categories;

  public VehicleCard(String name, Map<Category, Double> categories) {
   // throws IllegalArgumentException if name is null or empty.
   // throws IllegalArgumentException if categories is null not every Category exists in categories.
   // throws IllegalArgumentException if categories contains any null value or values less than 0.
   // set member variables
  }

  // getters for_immutable class, no setters (!)

  @Override
  public int compareTo(VehicleCard other) {
    // compare by totalBonus
  }

  protected int getBonus(Category category) {
    // return Bonus value for category
  }
  public int totalBonus(){
    /* returns total bonus of this card which is computed as the sum of the bonusValues (getBonus) of all the
           categories assigned to this card. */
  }
  ...
```

Klasse: VehicleCard cont.

```
    ...
    public static Map<Category, Double> newCategoriesMap(double price, double capa, double pwr, double acc,
            double velo, double cons) {...}

    protected String categoryToString(Category category) { /*returns category.toString() */}

    @Override
    public String toString() {/* "- <name>(totalBonus) -> {<categories>}" e.g.:
      - Audi TT RS Roadster(73032) -> {Preis=58650.0, Hubraum=2480.0, Leistung=350.0, Beschleunigung
            =4.6, Geschwindigkeit=280.0, Verbrauch=9.2}
      use categoryToString for representation of Category values */
    }
}
```

Klasse: FoilVehicleCard

```
public class FoilVehicleCard extends VehicleCard {

  private List<Category> specials;

  public FoilVehicleCard(String name, Map<Category, Double> categories,List<Category> specials) {
   // throws IllegalArgumentException if specials contains more than 3 items or is null
   // set member variables
  }

  @Override
  protected int getBonus(Category category) {
    /* returns twice the bonus value of the base class if category is a special
    category, and the bonus value of the base class otherwise */
  }

  @Override
  protected String categoryToString(Category category) {
    /* returns the same string as the base class, but prefixes and suffixes it with an asterisk if category is a
            special category, e. g. *Preis* */
  }
}
```

Klasse: Player

```
public class Player implements Comparable<Player>{
  private String name;
  private Queue<VehicleCard> deck = new ArrayDeque();

  public Player(String name) {
    // throw IllegalArgumentException if name is null or empty
  }

  public String getName() {...}

  public void addCards(Collection<VehicleCard> cards) {//add cards to end}
  public void addCard(VehicleCard card) {//add card to end}
  public void clearDeck() {...}
  public VehicleCard playNextCard() { /*poll next card from deck*/ }
  public int totalBonus(){
    /* returns total bonus of this player which is computed as the sum of the bonusValues (totalBonus) of all his
           cards. */
  }
  public int compareTo(Player other) {
    // compare by name[case insensitive]
  }
  @Override
  public int hashCode() { /*hash(name[case insensitive])*/}
  @Override
  public boolean equals(Object obj) { /* auto generate but cmp name case insensitive */ }
  ...
```

Klasse: Player cont.

```
    ....

    public boolean challengePlayer(Player p) {
      //throws IllegalArgumentException if p is null or p is this.
      //playNextCard from this and p.
      //Either scores are different (1) or draw (2)
      //If (1):
      //(a) Player who has higher scoring card, adds both of them to the end of his deck. Order is not important.
      //(b) Player who has lower scoring card, loses card.
      //If (2), repeat until winner is found and
      //(a) Winner gets all cards played. Order is not important.
      //(b) Loser loses all cards played.
      //(c) If either deck is empty before winner is found, cards are returned to the original deck.
      //Returns true if this wins. Else false.
    }

    public static Comparator<Player> compareByBonus() {....}
    public static Comparator<Player> compareByDeckSize() {....}

    @Override
    public String toString() { /*contains: Player.name(totalBonus), one card per line, e.g.:
      Maria(73214):
        – Porsche 911(73054) –> {Preis=<val>, Hubraum=<val>, ...}
        – Renault Clio(160) –> {...}*/
    }
  }
```