# Dominion Deck Generator (DDG)

Dominion ist ein Strategiespiel, bei dem die Spieler versuchen im Verlaufe der Partie möglichst viele Punkte zu sammeln. Jeder Spieler startet mit demselben Stapel an Karten. Ausgehend davon stehen in jeder Partie zehn weitere Königreichkarten zum Kauf, mit denen der Spieler seinen Stapel erweitern kann. Durch die Auflage mehrerer Erweiterungssets stehen mittlerweile eine Vielzahl von Königreichkarten zur Verfügung. Jede Karte hat andere Eigenschaften, die Auswirkungen auf das Spiel haben. Somit muss man sich für jede Konstellation von Karten eine neue Strategie überlegen.

Mehr Informationen zum Spiel können Sie im Dominionblog nachlesen. Dort gibt es auch eine Übersicht aller verfügbaren Karten mit einer Beschreibung ihrer Eigenschaften.

In dieser Aufgabe wird ein Programm entwickelt, mit dem man neue Partien zufällig erzeugen und die so generierten Partien verwalten kann. Darüber hinaus soll es möglich sein Einschränkungen für die Generierung festzulegen und die Daten in ein XML-Format zu exportieren und davon zu laden.

**Kenntnisse über den Ablauf des Spiels sind für diese Aufgabe nicht erforderlich.**

## Das Modell

Eine Sammlung von zehn Karten für eine Partie wird in Dominion auch als Deck bezeichnet. Zunächst sollen die verschiedenen Kartentypen und das Deck mit Schnittstellen und Klassen modelliert werden. Einige Eigenschaften sind alle Karten unabhängig von ihrem Typ gemein. Diese Eigenschaften werden in der Schnittstelle ICard im Paket de.uniwue.jpp.deckgen.model zusammengefasst.

### Die Schnittstelle *ICard*

Legen Sie die Schnittstelle *ICard* im Paket *de.uniwue.jpp.deckgen.model* an. In Dominion existieren unterschiedliche Typen von Karten, in ICard sind die gemeinsamen Eigenschaften aller Karten modelliert:

- **name:** Der Name der Karte.
- **description:** Eine Beschreibung der Eigenschaften der Karte.
- **extension:** Der Name der Erweiterung zu der die Karte gehört.
- **cost:** Der Preis der Karte.

```
package de.uniwue.jpp.deckgen.model;

/**
 * Common behaviour for a Dominion card.
 */
public interface ICard {

    /**
     * Gets the name of the card, e.g. 'Thief'.
     *
     * @return the name.
     */
    String getName();


    /**
     * Gets the description of the card.
     *
     * @return the description.
     */
    String getDescription();


    /**
     * Gets the name of the extension this card belongs to, e.g. 'Prosperity'
     *
     * @return the extension.
     */
    String getExtension();


    /**
     * Gets the cost of this card.
     *
     * @return the cost of the card.
     */
    int getCost();
}
```

Grundsätzlich lassen sich die Königreichkarten in drei Kategorien einordnen:

- Aktionskarten
- Siegkarten
- Geldkarten

Außerdem kann eine Karte auch gleichzeitig mehreren Kategorien angehören. Legen Sie jetzt die Schnittstellen *IActionCard*, *IVictoryCard*, *IMoneyCard* im Paket *de.uniwue.jpp.deckgen.model* an, die von *ICard* abgeleitet werden.

## Die Schnittstelle *IActionCard*

Aktionskarten führen keine zusätzlichen Eigenschaften ein, daher handelt es sich hier um eine leere Schnittstelle.

```
/**
 * Specifies behaviour of an action card.
 */
public interface IActionCard extends ICard {}
```

## Die Schnittstelle *IVictoryCard*

Siegkarten besitzen als zusätzliche Eigenschaft noch eine Anzahl von Siegpunkten.

```
/**
 * Specifies behaviour for a victory card.
 */
public interface IVictoryCard extends ICard {
   /**
    * Gets the victory points of this card.
    *
    * @return the victory points.
    */
    int getVictoryPoints();
}
```

## Die Schnittstelle *IMoneyCard*

Geldkarten zeichnen sich durch ihren Wert aus.

```
/**
 * Specifies behaviour for a money card.
 */
public interface IMoneyCard extends ICard {

  /**
   * Gets the value of this card.
   *
   * @return the value.
   */
  int getValue();
}
```

## Die abstrakte Klasse *AbstractCard*

Legen Sie nun die abstrakte Klasse *AbstractCard* im Paket *de.uniwue.jpp.deckgen.model* an. *AbstractCard* soll die Schnittstelle *ICard* vollständig implementieren, damit Sie die in *ICard* spezifizierten Methoden nicht für jeden konkreten Kartentyp neu implementieren müssen. Die abstrakte Klasse soll außerdem folgenden Konstruktor bereitstellen und die Methoden *equals* und *hashCode* überschreiben.

- **protected AbstractCard(String name, String description, String extension, int cost)**
  Initialisiert eine abstrakte Karte mit den gegebenen Werten. Behandeln Sie folgende Ausnahmen:
  - Einer der Parameter name, description, extension hat den Wert: null
    Werfen Sie eine java.lang.NullPointerException.
  - Einer der Parameter name, description, extension ist der leere String.
    Implementieren Sie eine eigene aussagekräftige von java.lang.RuntimeException abgeleitete Exception, die in diesem Fall geworfen wird.
  - Der Parameter cost hat einen negativen Wert.
    Implementieren Sie eine eigene aussagekräftige von java.lang.RuntimeException abgeleitete Exception, die in diesem Fall geworfen wird.

- **public boolean equals(Object that)**
  überschreiben Sie die Methode gemäß des in der Java-API spezifizierten Vertrages so, dass zwei abstrakte Karten gleich sind, wenn der

- name

gleich ist.

- **public int hashCode()**
  überschreiben Sie die Methode gemäß des in der Java-API spezifizierten Vertrages.

## Konkrete Karten

Nun können Sie die konkreten Kartentypen implementieren. Legen Sie dazu die folgenden Klassen im Paket *de.uniwue.jpp.deckgen.model* an.

**ActionCard**
Repräsentiert eine Aktionskarte. Sie soll von *AbstractCard* erben und die Schnittstelle *IActionCard* implementieren.
**MoneyCard**
Repräsentiert eine Geldkarte. Sie soll von *AbstractCard* erben und die Schnittstelle *IMoneyCard* implementieren. Implementieren Sie folgenden Konstruktor:

- **public MoneyCard(String name, String description, String extension, int cost, int value)**
  Initialisiert eine Instanz von MoneyCard. Delegieren Sie an den Konstruktor von AbstractCard! Behandeln Sie folgende Ausnahme:
  - Der Parameter value hat einen negativen Wert.
    Implementieren Sie eine eigene aussagekräftige von java.lang.RuntimeException abgeleitete Exception, die in diesem Fall geworfen wird.

**VictoryCard**
Repräsentiert eine Siegkarte. Sie soll von *AbstractCard* erben und die Schnittstelle *IVictoryCard* implementieren. Implementieren Sie folgenden Konstruktor:

- **public VictoryCard(String name, String description, String extension, int cost, int victoryPoints)**
  Initialisiert eine Instanz von VictoryCard. Delegieren Sie an den Konstruktor von AbstractCard! Behandeln Sie folgende Ausnahme:
  - Der Parameter victoryPoints hat einen negativen Wert.
    Implementieren Sie eine eigene aussagekräftige von java.lang.RuntimeException abgeleitete Exception, die in diesem Fall geworfen wird.

**MoneyAndVictoryCard**
Repräsentiert eine Karte, die sowohl Geld- als auch Siegkarte ist. Sie soll von *AbstractCard* erben und die Schnittstellen *IMoneyCard* und *IVictoryCard* implementieren. Implementieren Sie folgenden Konstruktor:

- **public MoneyAndVictoryCard(String name, String description, String extension, int cost, int value, int victoryPoints)**
  Initialisiert eine Instanz von MoneyAndVictoryCard. Delegieren Sie an den Konstruktor von AbstractCard! Behandeln Sie folgende Ausnahme:
  - Einer der Parameter value, victoryPoints hat einen negativen Wert.
    Implementieren Sie eine eigene aussagekräftige von java.lang.RuntimeException abgeleitete Exception, die in diesem Fall geworfen wird.

**ActionAndVictoryCard**
Repräsentiert eine Karte, die sowohl Aktions- als auch Siegkarte ist. Sie soll von *AbstractCard* erben und die Schnittstellen *IActionCard* und *IVictoryCard* implementieren. Implementieren Sie folgenden Konstruktor:

- **public VictoryCard(String name, String description, String extension, int cost, int victoryPoints)**
  Initialisiert eine Instanz von ActionAndVictoryCard. Delegieren Sie an den Konstruktor von AbstractCard! Behandeln Sie folgende Ausnahme:
  - Der Parameter victoryPoints hat einen negativen Wert.
    Implementieren Sie eine eigene aussagekräftige von java.lang.RuntimeException abgeleitete Exception, die in diesem Fall geworfen wird.

Betrachten Sie nocheinmal die Schnittstellen der Karten. Ihnen sollte auffallen, dass nur sogenannte "Getter" spezifiziert wurden. Über die Schnittstelle ist also nur ein lesender Zugriff auf die Eigenschaften der Instanzen möglich. Da sich die Eigenschaften der Karten nicht ändern können, macht dies auch Sinn. Klassen, deren Eigenschaften nach der Instanziierung nicht mehr modifzierbar sind, nennt man auch **nicht veränderbar**, bzw. **immutable**. Nicht veränderbare Klassen sind vor allem in der nebenläufigen Programmierung interessant, da sie keine Seiteneffekte zulassen.

## Die Klasse Deck

Für jede Partie in Dominion werden genau 10 Karten aus der Menge aller Karten ausgewählt. Eine solche Auswahl nennt man auch Deck. Zur Beschreibung soll ein Deck zusätzlich zur Menge der Karten, noch die folgenden Eigenschaften besitzen:

**title**
Ein das Deck charakterisierender Titel.
**comment**
Ein Kommentar zu den Spieleigenschaften des Decks.

Implementieren Sie als letztes nun noch die Klasse *Deck* im Paket *de.uniwue.jpp.deckgen.model* mit den folgenden Konstruktoren und Methoden:

- **public Deck(String title, String comment, java.util.Set<ICard> cards)**
  Initialisiert ein neues Objekt vom Typ Deck, mit den gegebenen Eigenschaften. Behandeln Sie folgende Ausnahmen:
  - Einer der Parameter title, comment, cards hat den Wert: null
    Werfen Sie eine java.lang.NullPointerException.
  - Einer der Parameter title, comment ist der leere String.
    Implementieren Sie eine eigene aussagekräftige von java.lang.RuntimeException abgeleitete Exception, die in diesem Fall geworfen wird.
  - Dem Parameter cards wird ein Set übergeben, das **nicht** genau zehn Karten enthält.
    Implementieren Sie eine eigene aussagekräftige von java.lang.RuntimeException abgeleitete Exception, die in diesem Fall geworfen wird.

- **public String getTitle()**
  Gibt den bei der Initialisierung gesetzten Titel zurück.

- **public String getComment()**
  Gibt den bei der Initialisierung gesetzten Kommentar zurück.

- **public java.util.Set<ICard> getCards()**
  Gibt die Menge der Karten dieses Deck zurück. Veränderungen an der zurückgegebenen Menge, dürfen die in der Instanz enthaltene Menge nicht verändern.

- **public boolean equals(Object that)**
  überschreiben Sie die Methode gemäß des in der Java-API spezifizierten Vertrages so, dass zwei Decks gleich sind, wenn sie aus den gleichen Karten besteht.

- **public int hashCode()**
  überschreiben Sie die Methode gemäß des in der Java-API spezifizierten Vertrages.

# Verwaltung der Karten und Decks

latex facingMultiCoreBeamer.tex latex facingMultiCoreBeamer.tex
Die Menge der Karten und Decks muss in einer Datenstruktur verwaltet werden. Es gibt viele Möglichkeiten die Daten zu verwalten. Beispiele sind: Collections, Datenbanken, Xml-Dateien, eigene Datenformate, ein Webservice, ... Um unabhängig von einer konkreten Implementierung zu sein, sollen die konkreten Klassen hinter zwei Schnittstellen versteckt werden.

## Die Schnittstelle *ICardRepository*

Die Schnittstelle *ICardRepository* bietet Zugriff auf die Menge der vorhandenen Karten. Erstellen Sie die Schnittstelle *ICardRepository* im Paket *de.uniwue.jpp.deckgen.repository*.

```
package de.uniwue.jpp.deckgen.repository;

import de.uniwue.jpp.deckgen.model.ICard;
import java.util.Set;


public interface ICardRepository {

    /**
     * Adds a card to the repository. A card should only be once in the repository.
     *
     * @param card - card that should be inserted.
     */
    void add(ICard card);

    /**
     * Retrieves all cards in the repository.
     *
     * @return a Set of all cards.
     */
    Set<ICard> getAll();

    /**
     * Finds a card with the given name.
     *
     * @param name the name.
     *
     * @return the card with the name or <code>null</code> if not found.
     */
    ICard find(String name);

}
```

## Die Schnittstelle *IDeckRepository*

Die Schnittstelle *IDeckRepository* bietet den Zugriff auf alle bereits erstellten und gespeicherten Decks. Erstellen Sie nun die Schnittstelle *IDeckRepository* im Paket *de.uniwue.jpp.deckgen.repository*.

```
package de.uniwue.jpp.deckgen.repository;

import de.uniwue.jpp.deckgen.model.Deck;
import java.util.Set;

public interface IDeckRepository {

    /**
     * Adds a deck to the repository. A deck should only be once in the repository.
```

```
     *
     * @param deck - deck that should be inserted.
     */
    void add(Deck deck);

    /**
     * Retrieves all decks in the repository.
     *
     * @return a set of all decks.
     */
    Set<Deck> getAll();

}
```

## Implementierungen der Schnittstellen

Im folgenden sollen Sie zwei einfache Implementierungen für die zuvor definierten Schnittstellen anfertigen. Dabei wird auf *java.util.Set* als Datenstruktur zur Verwaltung der Instanzen zurückgegriffen.

## Die Klasse CardRepository

Implementieren Sie die Klasse *CardRepository* im Paket *de.uniwue.jpp.deckgen.repository*. Benutzen Sie eine Implementierung der Schnittstelle *java.util.Set* als zugrundeliegende Datenstruktur. Die Klasse soll die Schnittstelle *ICardRepository* implementieren und außerdem folgende Konstruktoren zur Verfügung stellen:

- **public CardRepository()**
  Initialisiert ein leeres Repository.

- **public CardRepository(Collection<ICard> cards)**
  Initialisiert ein leeres Repository und fügt dann die gegebenen Karten hinzu. Behandeln Sie folgende Ausnahmen:
  - Der Parameter cards hat den Wert: null
    Werfen Sie eine java.lang.NullPointerException.

## Die Klasse DeckRepository

Implementieren Sie die Klasse *DeckRepository* im Paket *de.uniwue.jpp.deckgen.repository*. Benutzen Sie wieder eine Implementierung der Schnittstelle *java.util.Set* als zugrundeliegende Datenstruktur. Die Klasse soll die Schnittstelle *IDeckRepository* implementieren und außerdem folgende Konstruktoren zur Verfügung stellen:

- **public DeckRepository()**
  Initialisiert ein leeres Repository.

- **public DeckRepository(Collection<Deck> decks)**
  Initialisiert ein leeres Repository und fügt dann die gegebenen Decks hinzu. Behandeln Sie folgende Ausnahmen:
  - Der Parameter decks hat den Wert: null
    Werfen Sie eine java.lang.NullPointerException.

# Import/Export für Karten und Decks

Da beim Verwalten der Karten und Decks keine Datenbank oder etwas ähnliches zum Einsatz kommt (siehe letzter Abschnitt), beginnt man bei jedem Start des Programms von vorne neue Decks zu generieren. Bereits generierte Decks gehen verloren. Außerdem müssen irgendwie die vorhandenen Karten geladen werden.
Es soll nun eine Importfunktion für Karten und eine Import- und Exportfunktion für Decks implementiert werden.

Als Format für den Import und Export soll XML verwendet werden. Passende XML-Schema für die beiden Formate finden Sie in ihrem Repository:

**cards.xsd**
Beschreibt das Format der verfügbaren Karten.
**decks.xsd**
Beschreibt das Format der generierten Decks.
**cards.xml**
Beispieldatei.
**decks.xml**
Beispieldatei.

Darüber hinaus finden Sie auch Beispieldateien, die Sie für eigene Tests nutzen können.
Für das Generieren und Parsen der XML-Dateien soll die in der Java Standard-Bibliothek vorhandene Implementierung der DOM-Schnittstelle verwendet werden. Näheres zum Thema finden Sie zum Beispiel hier.

## Die Klasse CardImportService

Implementieren Sie nun zunächst die Klasse *CardImportService* im Paket *de.uniwue.jpp.deckgen.io*. Die Klasse soll mindestens folgende Konstruktoren und Methoden enthalten:

- **public CardImportService()**
  Initialisiert eine neue Instanz.

- **public java.util.Set<ICard> importFrom(InputStream in) throws ImportException**
  Liest das XML mit den Karten vom Eingabestrom *in*, erstellt daraus die entsprechenden Objekte vom Typ *ICard* und gibt Sie als *java.util.Set* zurück. Der konkrete Typ der einzelnen Karten ist im XML als Attribut *"type"* angegeben und entspricht dem Wert von *obj.getClass().getCanonicalName();*. Erstellen Sie die Exception *ImportException* im Paket *de.uniwue.jpp.deckgen.io*, die von java.lang.RuntimeException abgeleitet und in einem Fehlerfall geworfen wird.

  **Tipp:**
  Der Typ der Karte ist als Attribut des Elements: *card* codiert. Da sich die Kartentypen in den vorhandenen Eigenschaften unterscheiden, ist es etwas aufwändiger immer die korrekten Instanzen zu erstellen. Hier kann Ihnen das Entwurfsmuster: **Fabrik** helfen.

## Die Klasse DeckImportExportService

Nun soll die Klasse *DeckImportExportService* im Paket *de.uniwue.jpp.deckgen.io* implementiert werden. Die Klasse soll mindestens folgende Konstruktoren und Methoden bereitstellen:

- **public DeckImportExportService(ICardRepository cardRepository)**
  Initialisiert eine neue Instanz. Behandeln Sie folgende Ausnahmen:
  - Der Parameter cardRepository hat den Wert: null
    Werfen Sie eine java.lang.NullPointerException.

- **public java.util.Set<Deck> importFrom(InputStream in) throws ImportException**
  Liest das XML mit den Decks vom Eingabestrom *in*, erstellt daraus die entsprechenden Objekte vom Typ *Deck* und gibt Sie als *java.util.Set* zurück.
  Erstellen Sie die Exception *ImportException* im Paket *de.uniwue.jpp.deckgen.io*, die von java.lang.RuntimeException abgeleitet und in einem Fehlerfall geworfen wird.

- **public void exportTo(java.util.Set<Deck> decks, OutputStream out) throws ExportException**
  Nimmt die übergebenen *decks*, erzeugt eine XML-Darstellung wie in *decks.xsd* angegebenen und schreibt diese auf den Ausgabestrom *out*. Erstellen Sie die Exception *ExportException* im Paket *de.uniwue.jpp.deckgen.io*, die von java.lang.RuntimeException abgeleitet und in einem Fehlerfall geworfen wird.

# Generierung neuer Decks

Nun da es möglich ist Karten und Decks zu laden und zu speichern, soll der Generator für neue Decks geschrieben werden.

Ein Deck besteht aus zehn unterschiedlichen Karten, die aus der Menge aller Karten (zufällig) ausgewählt werden. Damit der Benutzer Einfluss auf die automatische Auswahl der Karten nehmen kann, soll es möglich sein Einschränkungen zu definieren, die für ein generiertes Deck gelten müssen. Damit handelt es sich um ein *Constraint Satisfaction Problem* siehe Artificial Intelligence. Zunächst wollen wir die benötigten Schnittstellen erstellen:

## Die Schnittstelle: IConstraint

Die Schnittstelle beschreibt ganz allgemein eine Einschränkung, die für eine Menge von Karten definiert ist. Sie liefert genau dann den Wahrheitswert *true*, wenn die Einschränkung für die Menge der Karten gilt, sonst liefert sie *false*. Erstellen Sie nun die Schnittstelle *IConstraint* im Paket *de.uniwue.jpp.deckgen.service.csp*:

```
package de.uniwue.jpp.deckgen.service.csp;

import de.uniwue.jpp.deckgen.model.ICard;
import java.util.Set;

/**
 * A constraint that can be imposed on a set of cards.
 */
public interface IConstraint {

    /**
     * Checks whether or not a set of cards satisfies a constraint.
     *
     * @param cards the set of cards.
     *
     * @return <code>true</code> if the set satisfies the constraint, <code>false</code> otherwise.
     */
    public boolean isSatisfied(Set<ICard> cards);
}
```

## Konkrete Einschränkungen

Es sind viele Arten von Einschränkungen denkbar, es sollen aber hier nur zwei besonders einfache implementiert werden.

**ExcludeCardConstraint**
Ist genau dann erfüllt, wenn die definierte Karte nicht im Deck enthalten ist.
**ExcludeExtensionConstraint**
Ist genau dann erfüllt, wenn keine der Karten im Deck aus der definierten Erweiterung stammt.

Implementieren Sie nun die Klasse *ExcludeCardConstraint* im Paket *de.uniwue.jpp.deckgen.service.csp.constraints*. Die Klasse soll die Schnittstelle *IConstraint* implementieren und mindestens folgende Konstruktoren und Methoden zur Verfügung stellen:

- **public ExcludeCardConstraint(ICard card)**
  Initialisiert eine neue Instanz der Einschränkung. Behandeln Sie folgende Ausnahmen:
  - Der Parameter card hat den Wert: null
    Werfen Sie eine java.lang.NullPointerException.

- **public boolean isSatisfied(java.util.Set<ICard> cards)**
  Gibt genau dann *true* zurück, wenn die Karte nicht in der Menge der Karten enthalten ist. Sonst gibt Sie *false* zurück.

Implementieren Sie nun die Klasse *ExcludeExtensionConstraint* im Paket *de.uniwue.jpp.deckgen.service.csp.constraints*. Die Klasse soll die Schnittstelle *IConstraint* implementieren und mindestens folgende Konstruktoren und Methoden zur Verfügung stellen:

- **public ExcludeExtensionConstraint(String extension)**
  Initialisiert eine neue Instanz der Einschränkung. Behandeln Sie folgende Ausnahmen:
  - Der Parameter extension hat den Wert: null
    Werfen Sie eine java.lang.NullPointerException.
  - Der Parameter extension ist der leere String.
    Implementieren Sie eine eigene aussagekräftige von java.lang.RuntimeException abgeleitete Exception, die in diesem Fall geworfen wird.

- **public boolean isSatisfied(java.util.Set<ICard> cards)**
  Gibt genau dann *true* zurück, wenn in der Menge keine Karte mit der definierten Extension vorhanden ist. Sonst gibt Sie *false* zurück.

## Die Schnittstelle: ISolver

*ISolver* dient als Schnittstelle für einen Lösungsalgorithmus. Legen Sie nun die Schnittstelle *ISolver* im Paket *de.uniwue.jpp.deckgen.service.csp* an:

```
package de.uniwue.jpp.deckgen.service.csp;

import java.util.Set;
import de.uniwue.jpp.deckgen.model.ICard;

/**
 * Interface for implementations that can find a deck that satisfies certain constraints.
 */
public interface ISolver {

    /**
     * Finds a configuration of ten distinct cards that satisfies all the constraints.
     *
     * @param allCards a set of all available cards.
     * @param constraints a set of constraints.
     *
     * @return a set of ten cards satisfying the constraints.
     */
    Set<ICard> solve(Set<ICard> allCards, Set<IConstraint> constraints) throws NoSolutionFoundException;
}
```

## Finden einer Lösung mit Backtracking

Implementieren Sie nun einen konkreten Lösungsalgorithmus. Verwenden Sie zur Lösung Backtracking. Der Algorithmus soll dabei schrittweise ausgehend von einer leeren Menge von Karten eine gültige Lösung aufbauen. Dazu werden der Menge zufällig Karten hinzugefügt und geprüft ob die Menge noch alle Einschränkungen erfüllt. Sind nicht alle Einschränkungen erfüllt, geht man einen Schritt zurück und versucht eine andere Karte. Der Algorithmus endet, wenn eine Menge von 10 Karten, die die Einschränkungen erfüllt gefunden wurde, oder alle Karten ausprobiert wurden.
Implementieren Sie nun die Klasse *BacktrackingSolver* im Paket *de.uniwue.jpp.deckgen.service.csp*, die die Schnittstelle *ISolver* implementiert und mindestens die folgenden Konstruktoren und Methoden bereitstellt:

- **public BacktrackingSolver()**
  Initialisiert eine neue Instanz.

- **public Set<ICard> solve(Set<ICard> allCards, Set<IConstraint> constraints)**
  Findet mit Hilfe von Backtracking ein Deck mit 10 Karten, das den Einschränkungen genügt. Die nächste Karte soll dabei zufällig ausgewählt werden, sodass stets andere Decks entstehen. Kann kein gültiges Deck gefunden werden, so soll eine *NoSolutionFoundException* geworfen werden. Erstellen Sie dazu die von *RuntimeException* abgeleitete Exception *NoSolutionFoundException* im Paket *de.uniwue.jpp.deckgen.service.csp*. Die Exception sollte folgende Konstruktoren und Methoden haben:

- **public NoSolutionFoundException(String message, Set<IConstraint> constraints)**
  Initialisiert eine neue Instanz.

- **public Set<IConstraint> getConstraints()**
  Liefert die Menge von Einschränkungen, für die keine Lösung gefunden werden konnte.

# Die Swing GUI

Als letztes sollen Sie nun noch eine graphischen Benutzeroberfläche mit Swing implementieren. Diese Oberfläche wird von PABS nicht automatisch getestet, sondern später von den Betreuern beurteilt. Die Benutzeroberfläche soll **mindestens** folgende Funktionen bieten und möglichst übersichtlich gestaltet sein:

**Wichtig: Erstellen Sie die Klasse *Dominion* im Paket *de.uniwue.jpp.deckgen* mit einer Main-Methode, die ihre GUI startet!**

Beschreibung der Benutzeroberfläche:

1. **Das Startfenster**

   Hier soll es möglich sein die Kartendaten aus einer XML-Datei einzulesen. Die Karten sollen dann dem *CardRepository* hinzugefügt werden. Sie können dazu einen *JButton* und einen *JFileChooser* verwenden.

2. **Die Übersicht**

   Hier sollen alle generierten Decks angezeigt werden. Dazu können Sie z.B. eine *JList* verwenden. Außerdem soll es möglich sein ein Deck auszuwählen, dessen Details dann sichtbar werden, z.B. in einem separaten Panel, dass auf Auswahlereignisse der JList wartet. Zu den Details gehört der Titel, der Kommentar und die Liste von Karten. Außerdem sollen folgende Aktionen möglich sein:

   1. **Load**

      Lädt eine Liste von Decks aus einer XML-Datei und fügt Sie dem *DeckRepository* hinzu. Achten Sie darauf, dass die Decks anschließend angezeigt werden.

   2. **Save**

      Speichert alle im *DeckRepository* vorhandenen Decks in eine XML-Datei.

   3. **New**

      Öffnet eine neue Ansicht, in der ein neues Deck generiert werden kann.

3. **Der Generierungsdialog**

   Die Generierungsansicht zeigt zunächst eine leere Liste von Einschränkungen. Es sollen folgende Aktionen möglich sein:

   1. **Add Constraint**

      Fügt eine Einschränkung hinzu, die bei der Suche nach einem neuen Deck berücksichtigt wird. Die Einschränkung soll in der Liste angezeigt werden. Es ist hier leichter ein JPanel mit entsprechendem LayoutManager zu verwenden. Eine JList entsprechend anzupassen ist schwieriger, aber nicht unmöglich.

   2. **Find Deck**

      Sucht nach einem neuen Deck unter Berücksichtigung der konfigurierten Einschränkungen. Das gefundene Deck soll anschließend angezeigt werden. Außerdem sollen Eingabefelder für *Titel* und *Kommentar* vorhanden sein. Folgende Aktion soll möglich sein:

      1. **Save**

         Fügt das neu generierte Deck dem *DeckRepository* hinzu. Anschließend wird die Übersicht wieder angezeigt.

**Verfügbar ab:** Montag, 20. August 2012, 12:00
**Abgabetermin:** Montag, 1. Oktober 2012, 13:00
**Repository:** https://s225698@pabs.uni-wuerzburg.de:443/svn/WS12_I_PP/s225698/dominion/code

## Korrektur-Status: Keine endgültige Abgabe gefunden.

✗ SVN-Revision 14130 vom 20. September 2012, 23:07:11 – richtige Bedingungen

Tests zeigen grobe Mängel, Abgabe wird **nicht** akzeptiert.

Commit-Message:

```
richtige Bedingungen
```

Reports

Model | Compile

```
     [echo] Compiling Model tests...
    [javac] Compiling 9 source files to /tmp/reactor-a2e1d3fe-4722-4a8b-b9be-322c50f3a168/build/cla
sses
     [echo] Executing Model Tests...
```

```
[junit] Running tests.required.model.ActionAndVictoryCardTest
[junit] Testsuite: tests.required.model.ActionAndVictoryCardTest
[junit] Tests run: 19, Failures: 0, Errors: 0, Time elapsed: 0.037 sec
[junit] Tests run: 19, Failures: 0, Errors: 0, Time elapsed: 0.037 sec
[junit]
[junit] Testcase: testGetVictoryPoints took 0.003 sec
[junit] Testcase: testConstructorNameIsNull took 0 sec
[junit] Testcase: testConstructorNameIsEmptyString took 0 sec
[junit] Testcase: testConstructorDescriptionIsNull took 0.001 sec
[junit] Testcase: testConstructorDescriptionIsEmptyString took 0 sec
[junit] Testcase: testConstructorExtensionIsNull took 0 sec
[junit] Testcase: testConstructorExtensionIsEmptyString took 0 sec
[junit] Testcase: testConstructorCostIsNegative took 0 sec
[junit] Testcase: testConstructorPointsAreNegative took 0.001 sec
[junit] Testcase: testGetName took 0 sec
[junit] Testcase: testGetDescription took 0 sec
[junit] Testcase: testGetCost took 0 sec
[junit] Testcase: testGetExtenstion took 0 sec
[junit] Testcase: testEqualToSelf took 0 sec
[junit] Testcase: testEqualToCopy took 0.001 sec
[junit] Testcase: testEqualNameOnly took 0.001 sec
[junit] Testcase: testHashCodeSelf took 0 sec
[junit] Testcase: testHashCodeCopy took 0 sec
[junit] Testcase: testHashCodeNameOnly took 0 sec
[junit] Running tests.required.model.ActionCardTest
[junit] Testsuite: tests.required.model.ActionCardTest
[junit] Tests run: 17, Failures: 0, Errors: 0, Time elapsed: 0.013 sec
[junit] Tests run: 17, Failures: 0, Errors: 0, Time elapsed: 0.013 sec
[junit]
[junit] Testcase: testConstructorNameIsNull took 0 sec
[junit] Testcase: testConstructorNameIsEmptyString took 0 sec
[junit] Testcase: testConstructorDescriptionIsNull took 0 sec
[junit] Testcase: testConstructorDescriptionIsEmptyString took 0 sec
[junit] Testcase: testConstructorExtensionIsNull took 0 sec
[junit] Testcase: testConstructorExtensionIsEmptyString took 0 sec
[junit] Testcase: testConstructorCostIsNegative took 0 sec
[junit] Testcase: testGetName took 0 sec
[junit] Testcase: testGetDescription took 0 sec
[junit] Testcase: testGetCost took 0 sec
[junit] Testcase: testGetExtenstion took 0 sec
[junit] Testcase: testEqualToSelf took 0 sec
[junit] Testcase: testEqualToCopy took 0 sec
[junit] Testcase: testEqualNameOnly took 0 sec
[junit] Testcase: testHashCodeSelf took 0.001 sec
[junit] Testcase: testHashCodeCopy took 0 sec
[junit] Testcase: testHashCodeNameOnly took 0 sec
[junit] Running tests.required.model.DeckTest
[junit] Testsuite: tests.required.model.DeckTest
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck hello erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck hello erstellt
[junit] Tests run: 18, Failures: 2, Errors: 0, Time elapsed: 0.025 sec
[junit] Output:
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck title erstellt
[junit] Deck hello erstellt
```

```
    [junit] Deck title erstellt
    [junit] Deck title erstellt
    [junit] Deck title erstellt
    [junit] Deck hello erstellt
    [junit]
    [junit] Tests run: 18, Failures: 2, Errors: 0, Time elapsed: 0.025 sec
    [junit] ------------- Standard Output ---------------
    [junit] Deck title erstellt
    [junit] Deck title erstellt
    [junit] Deck title erstellt
    [junit] Deck title erstellt
    [junit] Deck title erstellt
    [junit] Deck title erstellt
    [junit] Deck title erstellt
    [junit] Deck title erstellt
    [junit] Deck title erstellt
    [junit] Deck title erstellt
    [junit] Deck hello erstellt
    [junit] Deck title erstellt
    [junit] Deck title erstellt
    [junit] Deck title erstellt
    [junit] Deck hello erstellt
    [junit] ------------- ---------------- ---------------
    [junit]
    [junit] Testcase: testEqualToSelf took 0.002 sec
    [junit] Testcase: testEqualToCopy took 0.004 sec
    [junit]     FAILED
    [junit] An instance of type Deck should be equal to an exact copy of itself.
    [junit] Expected: <de.uniwue.jpp.deckgen.model.Deck@41b70ad9>
    [junit]      got: <de.uniwue.jpp.deckgen.model.Deck@41b70ad9>
    [junit]
    [junit] junit.framework.AssertionFailedError: An instance of type Deck should be equal to an ex
act copy of itself.
    [junit] Expected: <de.uniwue.jpp.deckgen.model.Deck@41b70ad9>
    [junit]      got: <de.uniwue.jpp.deckgen.model.Deck@41b70ad9>
    [junit]
    [junit]     at tests.required.model.DeckTest.testEqualToCopy(DeckTest.java:101)
    [junit]
    [junit] Testcase: testHashCodeSelf took 0.001 sec
    [junit] Testcase: testConstructorTitleIsNull took 0 sec
    [junit] Testcase: testConstructorTitleIsEmptyString took 0 sec
    [junit] Testcase: testConstructorCommentIsNull took 0.001 sec
    [junit] Testcase: testConstructorCommentIsEmptyString took 0 sec
    [junit] Testcase: testConstructorCardsIsNull took 0 sec
    [junit] Testcase: testConstructorSizeOfCardsIsNot10Size0 took 0 sec
    [junit] Testcase: testConstructorSizeOfCardsIsNot10Size3 took 0 sec
    [junit] Testcase: testConstructorSizeOfCardsIsNot10Size9 took 0 sec
    [junit] Testcase: testGetTitle took 0 sec
    [junit] Testcase: testGetComment took 0.001 sec
    [junit] Testcase: testGetCards took 0 sec
    [junit] Testcase: testGetCardsDoesNotGiveReferenceToInstanceVariable took 0 sec
    [junit] Testcase: testEqualCardsOnly took 0 sec
    [junit]     FAILED
    [junit] An instance of type Deck should be equal to another deck that has the same cards.
    [junit] Expected: <de.uniwue.jpp.deckgen.model.Deck@41b70ad9>
    [junit]      got: <de.uniwue.jpp.deckgen.model.Deck@41b70ad9>
    [junit]
    [junit] junit.framework.AssertionFailedError: An instance of type Deck should be equal to anoth
er deck that has the same cards.
    [junit] Expected: <de.uniwue.jpp.deckgen.model.Deck@41b70ad9>
    [junit]      got: <de.uniwue.jpp.deckgen.model.Deck@41b70ad9>
    [junit]
    [junit]     at tests.required.model.DeckTest.testEqualCardsOnly(DeckTest.java:108)
    [junit]
    [junit] Testcase: testHashCodeToCopy took 0.001 sec
    [junit] Testcase: testHashCodeCardsOnly took 0 sec
    [junit] TEST tests.required.model.DeckTest FAILED
    [junit] Running tests.required.model.MoneyAndVictoryCardTest
    [junit] Testsuite: tests.required.model.MoneyAndVictoryCardTest
    [junit] Tests run: 21, Failures: 0, Errors: 0, Time elapsed: 0.021 sec
    [junit] Tests run: 21, Failures: 0, Errors: 0, Time elapsed: 0.021 sec
    [junit]
    [junit] Testcase: testGetVictoryPoints took 0 sec
    [junit] Testcase: testConstructorNameIsNull took 0 sec
    [junit] Testcase: testConstructorNameIsEmptyString took 0.001 sec
```

```
    [junit] Testcase: testConstructorDescriptionIsNull took 0 sec
    [junit] Testcase: testConstructorDescriptionIsEmptyString took 0 sec
    [junit] Testcase: testConstructorExtensionIsNull took 0 sec
    [junit] Testcase: testConstructorExtensionIsEmptyString took 0 sec
    [junit] Testcase: testConstructorCostIsNegative took 0 sec
    [junit] Testcase: testConstructorPointsAreNegative took 0 sec
    [junit] Testcase: testGetValue took 0 sec
    [junit] Testcase: testConstructorValueIsNegative took 0 sec
    [junit] Testcase: testGetName took 0 sec
    [junit] Testcase: testGetDescription took 0.001 sec
    [junit] Testcase: testGetCost took 0 sec
    [junit] Testcase: testGetExtenstion took 0 sec
    [junit] Testcase: testEqualToSelf took 0 sec
    [junit] Testcase: testEqualToCopy took 0 sec
    [junit] Testcase: testEqualNameOnly took 0 sec
    [junit] Testcase: testHashCodeSelf took 0 sec
    [junit] Testcase: testHashCodeCopy took 0 sec
    [junit] Testcase: testHashCodeNameOnly took 0.001 sec
    [junit] Running tests.required.model.MoneyCardTest
    [junit] Testsuite: tests.required.model.MoneyCardTest
    [junit] Tests run: 19, Failures: 0, Errors: 0, Time elapsed: 0.033 sec
    [junit] Tests run: 19, Failures: 0, Errors: 0, Time elapsed: 0.033 sec
    [junit]
    [junit] Testcase: testConstructorNameIsNull took 0 sec
    [junit] Testcase: testConstructorNameIsEmptyString took 0.016 sec
    [junit] Testcase: testConstructorDescriptionIsNull took 0.001 sec
    [junit] Testcase: testConstructorDescriptionIsEmptyString took 0 sec
    [junit] Testcase: testConstructorExtensionIsNull took 0 sec
    [junit] Testcase: testConstructorExtensionIsEmptyString took 0 sec
    [junit] Testcase: testConstructorCostIsNegative took 0 sec
    [junit] Testcase: testGetValue took 0 sec
    [junit] Testcase: testConstructorValueIsNegative took 0 sec
    [junit] Testcase: testGetName took 0 sec
    [junit] Testcase: testGetDescription took 0 sec
    [junit] Testcase: testGetCost took 0 sec
    [junit] Testcase: testGetExtenstion took 0 sec
    [junit] Testcase: testEqualToSelf took 0 sec
    [junit] Testcase: testEqualToCopy took 0 sec
    [junit] Testcase: testEqualNameOnly took 0 sec
    [junit] Testcase: testHashCodeSelf took 0 sec
    [junit] Testcase: testHashCodeCopy took 0 sec
    [junit] Testcase: testHashCodeNameOnly took 0 sec
    [junit] Running tests.required.model.VictoryCardTest
    [junit] Testsuite: tests.required.model.VictoryCardTest
    [junit] Tests run: 19, Failures: 0, Errors: 0, Time elapsed: 0.028 sec
    [junit] Tests run: 19, Failures: 0, Errors: 0, Time elapsed: 0.028 sec
    [junit]
    [junit] Testcase: testGetVictoryPoints took 0 sec
    [junit] Testcase: testConstructorNameIsNull took 0 sec
    [junit] Testcase: testConstructorNameIsEmptyString took 0 sec
    [junit] Testcase: testConstructorDescriptionIsNull took 0 sec
    [junit] Testcase: testConstructorDescriptionIsEmptyString took 0 sec
    [junit] Testcase: testConstructorExtensionIsNull took 0 sec
    [junit] Testcase: testConstructorExtensionIsEmptyString took 0 sec
    [junit] Testcase: testConstructorCostIsNegative took 0 sec
    [junit] Testcase: testConstructorPointsAreNegative took 0 sec
    [junit] Testcase: testGetName took 0 sec
    [junit] Testcase: testGetDescription took 0 sec
    [junit] Testcase: testGetCost took 0 sec
    [junit] Testcase: testGetExtenstion took 0 sec
    [junit] Testcase: testEqualToSelf took 0 sec
    [junit] Testcase: testEqualToCopy took 0 sec
    [junit] Testcase: testEqualNameOnly took 0 sec
    [junit] Testcase: testHashCodeSelf took 0 sec
    [junit] Testcase: testHashCodeCopy took 0 sec
    [junit] Testcase: testHashCodeNameOnly took 0 sec
    [junit] Tests FAILED

BUILD FAILED

/tmp/reactor-a2e1d3fe-4722-4a8b-b9be-322c50f3a168/build.xml:209: Model tests failed...
        at org.apache.tools.ant.taskdefs.Exit.execute(Exit.java:164)
        at org.apache.tools.ant.UnknownElement.execute(UnknownElement.java:291)
        at sun.reflect.GeneratedMethodAccessor4.invoke(Unknown Source)
```

```
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:601)
        at org.apache.tools.ant.dispatch.DispatchUtils.execute(DispatchUtils.java:106)
        at org.apache.tools.ant.Task.perform(Task.java:348)
        at org.apache.tools.ant.taskdefs.Sequential.execute(Sequential.java:68)
        at net.sf.antcontrib.logic.IfTask.execute(IfTask.java:197)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:601)
        at org.apache.tools.ant.dispatch.DispatchUtils.execute(DispatchUtils.java:106)
        at org.apache.tools.ant.TaskAdapter.execute(TaskAdapter.java:154)
        at org.apache.tools.ant.UnknownElement.execute(UnknownElement.java:291)
        at sun.reflect.GeneratedMethodAccessor4.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:601)
        at org.apache.tools.ant.dispatch.DispatchUtils.execute(DispatchUtils.java:106)
        at org.apache.tools.ant.Task.perform(Task.java:348)
        at org.apache.tools.ant.Target.execute(Target.java:392)
        at org.apache.tools.ant.Target.performTasks(Target.java:413)
        at org.apache.tools.ant.Project.executeSortedTargets(Project.java:1399)
        at org.apache.tools.ant.Project.executeTarget(Project.java:1368)
        at org.apache.tools.ant.helper.DefaultExecutor.executeTargets(DefaultExecutor.java:41)
        at org.apache.tools.ant.Project.executeTargets(Project.java:1251)
        at org.apache.tools.ant.Main.runBuild(Main.java:811)
        at org.apache.tools.ant.Main.startAnt(Main.java:217)
        at org.apache.tools.ant.launch.Launcher.run(Launcher.java:280)
        at org.apache.tools.ant.launch.Launcher.main(Launcher.java:109)
```

Quellen

Resources

Libraries

SVN-Revision 14122 vom 20. September 2012, 22:53:23 – richtiger hashCode in Deck

SVN-Revision 14116 vom 20. September 2012, 22:42:55 – richtige Bedingungen

SVN-Revision 14057 vom 20. September 2012, 19:48:20 – equals-Methode in AbstractCard korrigiert, Dominio...

SVN-Revision 13260 vom 20. September 2012, 8:12:42 – first commit without gui

SVN-Revision 11783 vom 17. September 2012, 14:11:43 – Copying template for user s225698 and assignment: ...