

# Lamb.da - Das Spiel

## Entwurfsdokument

Farid El-Haddad, Florian Fervers, Kai Fieger,  
Robert Hochweiß, Kay Schmitteckert

13. Januar 2015



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>5</b>
<b>2</b>	<b>Grobentwurf</b>	<b>6</b>
<b>3</b>	<b>Feinentwurf</b>	<b>7</b>
3.1	package <code>lambda</code> . . . . .	7
3.1.1	public class <b>Observable</b> <Observer> . . . . .	7
3.2	package <code>lambda.model.lambdaterm</code> . . . . .	8
3.2.1	public abstract class <b>LambdaTerm</b> . . . . .	8
3.2.2	public interface <b>LambdaTermObserver</b> . . . . .	10
3.2.3	public class <b>LambdaApplication</b> extends <code>LambdaTerm</code> . . . . .	11
3.2.4	public abstract class <b>LambdaValue</b> extends <code>LambdaTerm</code> . . . . .	13
3.2.5	public class <b>LambdaAbstraction</b> extends <code>LambdaValue</code> . . . . .	14
3.2.6	public class <b>LambdaVariable</b> extends <code>LambdaValue</code> . . . . .	15
3.2.7	public class <b>LambdaRoot</b> extends <code>LambdaTerm</code> implements <code>Observable</code> < <code>LambdaTermObserver</code> > . . . . .	16
3.2.8	public final class <b>LambdaUtils</b> . . . . .	18
3.3	package <code>lambda.model.lambdaterm.visitor</code> . . . . .	18
3.3.1	public interface <b>LambdaTermVisitor</b> <R> . . . . .	18
3.3.2	public class <b>AlphaConversionVisitor</b> implements <code>LambdaTermVisitor</code> < <code>Object</code> > . . . . .	20
3.3.3	public class <b>ColorCollectionVisitor</b> implements <code>LambdaTermVisitor</code> < <code>Set</code> < <code>Color</code> >> . . . . .	21
3.3.4	public class <b>IsColorBoundVisitor</b> implements <code>LambdaTermVisitor</code> < <code>Boolean</code> > . . . . .	22
3.3.5	public class <b>ApplicationVisitor</b> implements <code>LambdaTermVisitor</code> < <code>LambdaTerm</code> > . . . . .	24
3.3.6	public class <b>CopyVisitor</b> implements <code>LambdaTermVisitor</code> < <code>LambdaTerm</code> > . . . . .	26
3.3.7	public class <b>RemoveTermVisitor</b> implements <code>LambdaTermVisitor</code> < <code>Object</code> > . . . . .	28
3.3.8	public abstract class <b>BetaReductionVisitor</b> implements <code>LambdaTermVisitor</code> < <code>LambdaTerm</code> > . . . . .	29
3.4	package <code>lambda.model.lambdaterm.visitor.strategy</code> . . . . .	31
3.4.1	public class <b>ReductionStrategyNormalOrder</b> extends <code>BetaReductionVisitor</code> . . . . .	31
3.4.2	public class <b>ReductionStrategyApplicativeOrder</b> extends <code>BetaReductionVisitor</code> . . . . .	32

3.4.3	public class <b>ReductionStrategyCallByValue</b> extends BetaReductionVisitor .....	32
3.4.4	public class <b>ReductionStrategyCallByName</b> extends BetaReductionVisitor .....	33
3.5	package lambda.model.profiles .....	34
3.5.1	public interface <b>ProfileModelObserver</b> .....	34
3.5.2	public class <b>ProfileModel</b> implements Observable<ProfileModelObserver> .....	35
3.5.3	public interface <b>ProfileManagerObserver</b> .....	37
3.5.4	public class <b>ProfileManager</b> implements Observable<ProfileManagerObserver> .....	38
3.5.5	public interface <b>ProfileEditLangObserver</b> .....	40
3.5.6	public class <b>ProfileEditLangModel</b> implements Observable<ProfileEditLangObserver> .....	41
3.5.7	public interface <b>ProfileEditAvatarObserver</b> .....	42
3.5.8	public class <b>ProfileEditAvatarModel</b> implements Observable<ProfileEditAvatarObserver> .....	42
3.6	package lambda.model.settings .....	43
3.6.1	public interface <b>SettingsModelObserver</b> .....	43
3.6.2	public class <b>SettingsModel</b> implements Observable<SettingsModelObserver> .....	44
3.7	package lambda.model.achievements .....	47
3.7.1	public interface <b>AchievementModelObserver</b> .....	47
3.7.2	public abstract class <b>AchievementModel</b> implements Observable<AchievementModelObserver> .	47
3.7.3	public class <b>TimeAchievementModel</b> extends AchievementModel .....	51
3.7.4	public class <b>LevelAchievementModel</b> extends AchievementModel .....	52
3.7.5	public class <b>GemsEnchantedAchievementModel</b> extends AchievementModel .....	52
3.7.6	public class <b>LambsEnchantedAchievementModel</b> extends AchievementModel .....	53
3.7.7	public class <b>GemsPlacedAchievementModel</b> extends AchievementModel .....	54
3.7.8	public class <b>LambsPlacedAchievementModel</b> extends AchievementModel .....	55
3.7.9	public class <b>HintsAchievementModel</b> extends AchievementModel .....	56
3.7.10	public abstract class <b>PerLevelAchievementModel</b> extends AchievementModel .....	56
3.7.11	public class <b>GemsEnchantedPerLevelAchievementModel</b> extends PerLevelAchievementModel .....	57

3.7.12	public class <b>LambsEnchantedPerLevelAchievementModel</b> extends PerLevelAchievementModel . . . . .	58
3.7.13	public class <b>GemsPlacedPerLevelAchievementModel</b> extends PerLevelAchievementModel . . . . .	58
3.7.14	public class <b>LambsPlacedPerLevelAchievementModel</b> extends PerLevelAchievementModel . . . . .	59
3.7.15	public class <b>AchievementManager</b> . . . . .	60
3.8	package lambda.viewcontroller.lambdaterm . . . . .	63
3.8.1	public class <b>LambdaTermViewController</b> extends scene2d.Group implements LambdaTermObserver . . . .	63
3.8.2	public abstract class <b>LambdaNodeViewController</b> extends scene2d.Actor . . . . .	68
3.8.3	public class <b>LambdaAbstractionViewController</b> extends LambdaNodeViewController . . . . .	71
3.8.4	public class <b>LambdaApplicationViewController</b> extends LambdaNodeViewController . . . . .	72
3.8.5	public class <b>LambdaVariableViewController</b> extends LambdaNodeViewController . . . . .	73
3.9	package lambda.viewcontroller.lambdaterm.visitor . . . .	75
3.9.1	public class <b>ViewInsertionVisitor</b> implements LambdaTermVisitor<Object> . . . . .	75
3.10	package lambda.viewcontroller.achievements . . . . .	77
3.10.1	public class <b>AchievementMenuViewController</b> extends Controller implements AchievementModelObserver . . . . .	77
3.11	package lambda.model.shop . . . . .	79
3.11.1	public class <b>ShopModel</b> . . . . .	79
3.11.2	public abstract class <b>ShopItemModel</b> . . . . .	80
3.11.3	public class <b>MusicModel</b> extends ShopItemModel . . .	82
3.11.4	public class <b>BackgroundImageModel</b> extends ShopItemModel . . . . .	83
3.11.5	public class <b>SpriteModel</b> extends ShopItemModel . .	83
3.11.6	public class <b>ShopItemTypeModel&lt;T&gt;</b> . . . . .	84
3.12	package lambda.model.level . . . . .	85
3.12.1	public class <b>LevelModel</b> . . . . .	85
3.13	package lambda.viewcontroller.shop . . . . .	87
3.13.1	public class <b>ShopViewController1</b> extends Controller . . . . .	87
3.13.2	public class <b>ShopItemViewController1</b> extends Controller . . . . .	88
4	<b>Datenstrukturen</b>	90
5	<b>Dynamische Modelle</b>	91

<b>6</b>	<b>Projektplan</b>	<b>92</b>
<b>7</b>	<b>Glossar</b>	<b>93</b>
<b>8</b>	<b>Anhang</b>	<b>94</b>

## 1 Einleitung

## 2 Grobentwurf

## 3 Feinentwurf

### 3.1 package lambda

#### 3.1.1 public class **Observable**<Observer>

##### Beschreibung

Repräsentiert ein Objekt, das von Beobachtern überwacht werden kann. Dabei informiert das Objekt alle Beobachter, sobald Änderungen an ihm vorgenommen werden.

##### Typ-Parameter

- <Observer>  
Der Typ eines Beobachters.

##### Attribute

- private List<Observer> **observers**  
Die Liste der Beobachter dieses Objektes.

##### Konstruktoren

- public **Observable**()  
Instanziert ein Objekt dieser Klasse.

##### Methoden

- public void **addObserver**(Observer o)  
Fügt den gegebenen Beobachter diesem Objekt hinzu, sodass dieser bei Änderungen informiert wird.

##### Parameter

- Observer o  
Der neue Beobachter.

##### Exceptions

- NullPointerException  
Falls o == null ist.

- public void **removeObserver**(Observer o)  
Entfernt den Beobachter aus der Liste, falls dieser darin existiert, sodass dieser nicht mehr bei Änderungen informiert wird.



#### Parameter

- `Observer o`  
Der zu entfernende Beobachter.

#### Exceptions

- `NullPointerException`  
Falls `o == null` ist.

- `public void notify(Consumer<Observer> notifier)`  
Ruft die gegebene Funktion auf allen Beobachtern auf. Wird benutzt, um Beobachter über Änderungen am Objekt zu informieren.

#### Parameter

- `Consumer<Observer> notifier`  
Die Funktion, die auf allen Beobachtern ausgeführt wird.

#### Exceptions

- `NullPointerException`  
Falls `notifier == null` ist.

## 3.2 package `lambda.model.lambdaterm`

### 3.2.1 public abstract class `LambdaTerm`

#### Beschreibung

Repräsentiert einen Term im Lambda-Kalkül bzw. ein Knoten in der Baumstruktur eines Lambda-Terms.

#### Attribute

- `private LambdaTerm parent`  
Der Elternknoten dieses Terms. Kann auch `null` sein, falls der Knoten eine Wurzel ist.
- `private boolean locked`  
Gibt an, ob dieser Knoten im Editor verändert werden kann.

#### Konstruktoren

- `public LambdaTerm(LambdaTerm parent, boolean locked)`  
Instanziert ein Objekt dieser Klasse mit dem gegebenen Elternknoten.

### Parameter

- `LambdaTerm parent`  
Der Elternknoten dieses Terms. Kann auch `null` sein, falls der Knoten eine Wurzel ist.
- `boolean locked`  
Gibt an, ob dieser Knoten im Editor verändert werden kann.

### Methoden

- `public abstract <T> T accept(LambdaTermVisitor<T> visitor)`  
Nimmt den gegebenen Besucher entgegen und ruft dessen `visit`-Methode auf. Die Rückgabe des Besuchers wird auch von dieser Methode zurückgegeben.

### Typ-Parameter

- `<T>`  
Der Typ des Rückgabewertes des Besuchers. Wird benötigt, um verschiedene Rückgabewerte von verschiedenen Besucherklassen zu ermöglichen.

### Parameter

- `LambdaTermVisitor<T> visitor`  
Der Besucher, der entgegen genommen wird.

### Rückgabe

- Gibt den Rückgabewert des Besuchers zurück.

### Exceptions

- `NullPointerException`  
Falls `visitor == null` ist.

- `public void notifyRoot(Consumer<LambdaTermObserver> notifier)`  
Gibt die Nachricht weiter zur Wurzel, wo die Beobachter informiert werden.

### Parameter

- `Consumer<LambdaTermObserver> notifier`  
Die Funktion, die auf allen Beobachtern ausgeführt wird.

### Exceptions

- `NullPointerException`  
Falls `notifier == null` ist.

- `public boolean isValue()`  
Gibt zurück, ob dieser Term ein Wert - d.h. eine Abstraktion oder Variable - ist. Gibt in der Standard-Implementierung `false` zurück und wird von entsprechenden Unterklassen überschrieben.

#### Rückgabe

- Gibt zurück, ob dieser Term ein Wert ist.

- `public LambdaTerm getParent()`  
Gibt den Elternknoten dieses Knotens wieder oder `null`, falls dieser Knoten eine Wurzel ist.

#### Rückgabe

- Der Elternknoten dieses Knotens.

- `public boolean isLocked()`  
Gibt zurück, ob dieser Knoten im Editor verändert werden kann.

#### Rückgabe

- Gibt zurück, ob dieser Knoten im Editor verändert werden kann.

- `public boolean equals(Object o)`  
Gibt zurück, ob dieses und das gegebene Element gleich sind.

#### Rückgabe

- Gibt zurück, ob dieses und das gegebene Element gleich sind.

### 3.2.2 `public interface LambdaTermObserver`

#### Beschreibung

Repräsentiert einen Beobachter eines Lambda-Terms, welcher über Änderungen am Term informiert wird.

#### Methoden

- `public void replaceTerm(LambdaTerm old, LambdaTerm new)`  
Wird aufgerufen um dem Beobachter mitzuteilen, dass der gegebene alte Term durch den gegebenen neuen ersetzt wird. Einer von beiden Parametern kann `null` sein, niemals aber beide.

#### Parameter

– LambdaTerm old  
Der ersetzte Term.

– LambdaTerm new  
Der neue Term.

- public void **setColor**(LambdaValue term, Color color)  
Wird aufgerufen um dem Beobachter mitzuteilen, dass die Farbe des gegebenen Terms durch die gegebene neue Farbe ersetzt wird.

#### Parameter

– LambdaValue term  
Der veränderte Term.

– Color color  
Die neue Farbe des Terms.

**3.2.3** public class **LambdaApplication** extends LambdaTerm

#### Beschreibung

Repräsentiert eine Applikation im Lambda-Kalkül.

#### Attribute

- private LambdaTerm **first**  
Linker bzw. erster Kindknoten der Applikation.
- private LambdaTerm **second**  
Rechter bzw. zweiter Kindknoten der Applikation.

#### Konstruktoren

- public **LambdaApplication**(LambdaTerm parent, boolean locked)  
Instanziert ein Objekt dieser Klasse mit dem gegebenen Elternknoten.

#### Parameter

– LambdaTerm parent  
Der Elternknoten dieses Terms. null ist erlaubt, resultiert aber in einem ungültigen Lambda-Term.

- boolean `locked`  
Gibt an, ob dieser Knoten im Editor verändert werden kann.

## Methoden

- `public <T> T accept(LambdaTermVisitor<T> visitor)`  
Siehe `LambdaTerm.accept`
- `public void setFirst(LambdaTerm first)`  
Setzt den linken bzw. ersten Kindknoten dieser Applikation und informiert alle Beobachter über diese Änderung.

### Parameter

- `LambdaTerm first`  
Der neue linke Kindknoten. `null` ist erlaubt, resultiert aber in einem ungültigen Lambda-Term.
- `public LambdaTerm getFirst()`  
Gibt den linken bzw. ersten Kindknoten dieser Applikation zurück.

### Rückgabe

- Der linke Kindknoten dieser Applikation.
- `public void setSecond(LambdaTerm second)`  
Setzt den rechten bzw. zweiten Kindknoten dieser Applikation und informiert alle Beobachter über diese Änderung.

### Parameter

- `LambdaTerm second`  
Der neue rechte Kindknoten. `null` ist erlaubt, resultiert aber in einem ungültigen Lambda-Term.
- `public LambdaTerm getSecond()`  
Gibt den rechten bzw. zweiten Kindknoten dieser Applikation zurück.

### Rückgabe

- Der rechte Kindknoten dieser Applikation.
- `public boolean equals(Object o)`  
Gibt zurück, ob dieses und das gegebene Element gleich sind. Zwei Applikationen sind gleich, wenn beide rechte Kindknoten gleich und beide linke Kindknoten gleich sind.

### Rückgabe

- Gibt zurück, ob dieses und das gegebene Element gleich sind.

**3.2.4** `public abstract class LambdaValue extends LambdaTerm`

### Beschreibung

Repräsentiert einen Wert - d.h. Abstraktion oder Variable - im Lambda-Kalkül.

### Attribute

- `private Color color`  
Die Farbe dieses Wertes, äquivalent zum Variablennamen.

### Konstruktoren

- `public LambdaValue(LambdaTerm parent, Color color, boolean locked)`  
Instanziert ein Objekt dieser Klasse mit dem gegebenen Elternknoten und der gegebenen Farbe.

### Parameter

- `LambdaTerm parent`  
Der Elternknoten dieses Terms. `null` ist erlaubt, falls der Term eine Wurzel ist.
- `Color color`  
Die Farbe dieses Wertes.
- `boolean locked`  
Gibt an, ob dieser Knoten im Editor verändert werden kann.

### Exceptions

- `NullPointerException`  
Falls `color == null` ist.

### Methoden

- `public boolean isValue()`  
Gibt zurück, ob dieser Term ein Wert ist. Überschreibt die Funktion in `LambdaTerm` und gibt hier immer `true` zurück.

### Rückgabe

- Gibt zurück, ob dieser Term ein Wert ist.

- `public void setColor(Color color)`  
Setzt die Farbe dieses Wertes und informiert alle Beobachter über diese Änderung.

#### Parameter

- `Color color`  
Die neue Farbe.

#### Exceptions

- `NullPointerException`  
Falls `color == null` ist.

- `public Color getColor()`  
Gibt die Farbe dieses Wertes zurück.

#### Rückgabe

- Die Farbe dieses Wertes.

**3.2.5** `public class LambdaAbstraction extends LambdaValue`

#### Beschreibung

Repräsentiert eine Abstraktion im Lambda-Kalkül.

#### Attribute

- `private LambdaTerm inside`  
Der Term innerhalb der Applikation. Kann `null` sein, resultiert aber in einem ungültigen Term.

#### Konstruktoren

- `public LambdaAbstraction(LambdaTerm parent, Color color, boolean locked)`  
Instanziert ein Objekt dieser Klasse mit dem gegebenen Elternknoten und der gegebenen Farbe.

#### Parameter

- `LambdaTerm parent`  
Der Elternknoten dieses Terms. Kann `null` sein, falls der Term eine Wurzel ist.
- `Color color`  
Die Farbe der in dieser Abstraktion gebundenen Variable.

- boolean `locked`  
Gibt an, ob dieser Knoten im Editor verändert werden kann.

### Exceptions

- `NullPointerException`  
Falls `color == null` ist.

### Methoden

- `public <T> T accept(LambdaTermVisitor<T> visitor)`  
Siehe `LambdaTerm.accept`
- `public void setInside(LambdaTerm inside)`  
Setzt den Term innerhalb der Abstraktion und informiert alle Beobachter über diese Änderung.

### Parameter

- `LambdaTerm inside`  
Der neue innere Term. Kann null sein, resultiert aber in einem ungültigen Term.
- `public LambdaTerm getInside()`  
Gibt den Term innerhalb der Abstraktion zurück.

### Rückgabe

- Der innere Term.
- `public boolean equals(Object o)`  
Gibt zurück, ob dieses und das gegebene Element gleich sind. Zwei Abstraktionen sind gleich, wenn beide dieselbe Farbe haben und die Kindknoten gleich sind.

### Rückgabe

- Gibt zurück, ob dieses und das gegebene Element gleich sind.

### 3.2.6 `public class LambdaVariable extends LambdaValue`

#### Beschreibung

Repräsentiert eine Variable im Lambda-Kalkül.



## Konstrukturen

- `public LambdaVariable(LambdaTerm parent, Color color, boolean locked)`  
Instanziert ein Objekt dieser Klasse mit dem gegebenen Elternknoten und der gegebenen Farbe.

### Parameter

- `LambdaTerm parent`  
Der Elternknoten dieses Terms. Kann null sein, falls der Term eine Wurzel ist.
- `Color color`  
Die Farbe der Variable.
- `boolean locked`  
Gibt an, ob dieser Knoten im Editor verändert werden kann.

### Exceptions

- `NullPointerException`  
Falls `color == null` ist.

## Methoden

- `public <T> T accept(LambdaTermVisitor<T> visitor)`  
Siehe `LambdaTerm.accept`

`public boolean equals(Object o)`

Gibt zurück, ob dieses und das gegebene Element gleich sind. Zwei Variablen sind gleich, wenn beide dieselbe Farbe haben.

### Rückgabe

- Gibt zurück, ob dieses und das gegebene Element gleich sind.

**3.2.7** `public class LambdaRoot extends LambdaTerm implements Observable<LambdaTermObserver>`

## Beschreibung

Repräsentiert die Wurzel eines Lambda-Terms. Die Wurzel eines gültigen Terms muss immer eine Instanz dieser Klasse sein.

## Attribute

- `private LambdaTerm child`  
Kind der Wurzel der Applikation.

### Konstruktoren

- `public LambdaRoot()`  
Instanziert ein Objekt dieser Klasse ohne Elternknoten.

### Methoden

- `public <T> T accept(LambdaTermVisitor<T> visitor)`  
Siehe `LambdaTerm.accept`
- `public void notifyRoot(Consumer<LambdaTermObserver> notifier)`  
Überschreibt die Funktion von `LambdaTerm`, um die Nachricht vom Kindknoten entgegenzunehmen und `notify` damit aufzurufen.

#### Parameter

- `Consumer<LambdaTermObserver> notifier`  
Die Funktion, die auf allen Beobachtern ausgeführt wird.

#### Exceptions

- `NullPointerException`  
Falls `notifier == null` ist.

- `public void setChild(LambdaTerm child)`  
Setzt den Kindknoten dieser Wurzel und informiert alle Beobachter über diese Änderung.

#### Parameter

- `LambdaTerm child`  
Der neue Kindknoten. `null` ist erlaubt, resultiert aber in einem ungültigen `LambdaTerm`.

- `public LambdaTerm getChild()`  
Gibt den Kindknoten dieser Wurzel zurück.

#### Rückgabe

- Der Kindknoten dieser Wurzel.

- `public boolean equals(Object o)`  
Gibt zurück, ob dieses und das gegebene Element gleich sind. Zwei Wurzeln sind gleich, wenn beide Kindknoten gleich sind.

### Rückgabe

- Gibt zurück, ob dieses und das gegebene Element gleich sind.

## 3.2.8 `public final class LambdaUtils`

### Beschreibung

Liefert statische Methoden zum einfachen Bearbeiten eines Lambda-Terms.

### Konstruktoren

- `private LambdaUtils()`  
Um zu verhindern, dass diese Klasse instanziiert wird.

### Methoden

- `public static LambdaRoot split(LambdaTerm term)`  
Entfernt den gegebenen Knoten aus seinem Elternknoten und fügt ihn in eine neue Wurzel des Typs `LambdaRoot` ein. Gibt die neue Wurzel zurück.

### Parameter

- `LambdaTerm term`  
Der Term, der abgespalten werden soll.

### Rückgabe

- Der abgespaltene Term in einer neuen Wurzel.

### Exceptions

- `NullPointerException`  
Falls `term == null` ist.

## 3.3 `package lambda.model.lambdaterm.visitor`

### 3.3.1 `public interface LambdaTermVisitor<R>`

### Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur. Der Besucher kann Operationen an der Datenstruktur ausführen und hat optional einen Rückgabewert.

## Typ-Parameter

- `<R>`  
Der Typ des Rückgabewertes.

## Methoden

- `public void visit(LambdaRoot node)`  
Besucht die gegebene Wurzel.

### Parameter

- `LambdaRoot node`  
Die besuchte Wurzel. Ist nie null.

- `public void visit(LambdaApplication node)`  
Besucht die gegebene Applikation.

### Parameter

- `LambdaApplication node`  
Die besuchte Applikation. Ist nie null.

- `public void visit(LambdaAbstraction node)`  
Besucht die gegebene Abstraktion.

### Parameter

- `LambdaAbstraction node`  
Die besuchte Abstraktion. Ist nie null.

- `public void visit(LambdaVariable node)`  
Besucht die gegebene Variable.

### Parameter

- `LambdaVariable node`  
Die besuchte Variable. Ist nie null.

- `public R getResult()`  
Gibt das Resultat der Besucheroperation zurück. Wird nur nach einem Besuch ausgeführt. Gibt in der Standard-Implementierung `null` zurück.

### Rückgabe

- Das Resultat der Besucheroperation.

**3.3.2** `public class AlphaConversionVisitor implements  
LambdaTermVisitor<Object>`

### Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, welcher eine Alpha-Konversion auf ihr ausführt.

### Attribute

- `private Color old`  
Die zu ersetzende Farbe.
- `private Color new`  
Die neue Farbe.

### Konstruktoren

- `public AlphaConversionVisitor(Color old, Color new)`  
Instanziert ein Objekt dieser Klasse mit der gegebenen ersetzten und ersetzenden Farbe.

#### Parameter

- `Color old`  
Die zu ersetzende Farbe.
- `Color new`  
Die neue Farbe.

### Methoden

- `public void visit(LambdaRoot node)`  
Besucht die gegebene Wurzel und traversiert wenn möglich weiter zum Kindknoten.

#### Parameter

- `LambdaRoot node`  
Die besuchte Wurzel.

- `public void visit(LambdaApplication node)`  
Besucht die gegebene Applikation und traversiert wenn möglich weiter zu beiden Kindknoten.

#### Parameter

- `LambdaApplication node`  
Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`  
Besucht die gegebene Abstraktion. Dabei wird die Farbe wenn nötig ersetzt und wenn möglich weiter zum Kindknoten traversiert.

#### Parameter

- `LambdaAbstraction node`  
Die besuchte Abstraktion.

- `public void visit(LambdaVariable node)`  
Besucht die gegebene Variable und ersetzt die Farbe wenn nötig.

#### Parameter

- `LambdaVariable node`  
Die besuchte Variable.

**3.3.3** `public class ColorCollectionVisitor implements  
LambdaTermVisitor<Set<Color>>`

#### Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, der die Menge der benutzten Farben in diesem Term zurückgibt.

#### Attribute

- `private Set<Color> result`  
Die Menge aller benutzten Farben.

#### Konstruktoren

- `public ColorCollectionVisitor()`  
Instanziert ein Objekt dieser Klasse.

#### Methoden

- `public void visit(LambdaRoot node)`  
Besucht die gegebene Wurzel und traversiert wenn möglich weiter zum Kindknoten.

#### Parameter

- LambdaRoot node  
Die besuchte Wurzel.

- `public void visit(LambdaApplication node)`  
Besucht die gegebene Applikation und traversiert wenn möglich weiter zu beiden Kindknoten.

#### Parameter

- LambdaApplication node  
Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`  
Besucht die gegebene Abstraktion. Dabei wird die Farbe zur Menge hinzugefügt und wenn möglich weiter zum Kindknoten traversiert.

#### Parameter

- LambdaAbstraction node  
Die besuchte Abstraktion.

- `public void visit(LambdaVariable node)`  
Besucht die gegebene Variable und fügt die Farbe zur Menge hinzu.

#### Parameter

- LambdaVariable node  
Die besuchte Variable.

- `public Set<Color> getResult()`  
Gibt die Menge der Farben zurück, die in dem besuchten Term benutzt werden.

#### Rückgabe

- Die Menge der benutzten Farben.

**3.3.4** `public class IsColorBoundVisitor implements LambdaTermVisitor<Boolean>`

#### Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, der zurückgibt, ob eine Variable mit der gegebenen Farbe in diesem Term gebunden ist.

## Attribute

- `private Color color`  
Die zu überprüfende Farbe.
- `private boolean result`  
Der Rückgabewert des Besuchs.

## Konstruktoren

- `public IsColorBoundVisitor(Color color)`  
Instanziert ein Objekt dieser Klasse mit der zu überprüfenden Farbe.

### Parameter

- `Color color`  
Die zu überprüfende Farbe.

### Exceptions

- `NullPointerException`  
Falls `color == null` ist.

## Methoden

- `public void visit(LambdaRoot node)`  
Besucht die gegebene Wurzel und beendet die Traversierung hier.

### Parameter

- `LambdaRoot node`  
Die besuchte Wurzel.

- `public void visit(LambdaApplication node)`  
Besucht die gegebene Applikation und traversiert wenn möglich weiter zum Elternknoten.

### Parameter

- `LambdaApplication node`  
Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`  
Besucht die gegebene Abstraktion und überprüft, ob die Farbe hier gebunden ist. Traversiert wenn nötig und möglich weiter zum Elternknoten.

### Parameter



- `LambdaAbstraction node`  
Die besuchte Abstraktion.

- `public void visit(LambdaVariable node)`  
Besucht die gegebene Variable und traversiert weiter zum Elternknoten.

#### Parameter

- `LambdaVariable node`  
Die besuchte Variable.

- `public Boolean getResult()`  
Gibt zurück, ob die Variable mit der gegebenen Farbe im Term gebunden ist.

#### Rückgabe

- Gibt zurück, ob die Variable mit der gegebenen Farbe gebunden ist.

**3.3.5** `public class ApplicationVisitor implements  
LambdaTermVisitor<LambdaTerm>`

#### Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, welcher eine Applikation ausführt.

#### Attribute

- `private Color color`  
Die Farbe der zu ersetzenden Variablen.
- `private LambdaTerm applicant`  
Das Argument der Applikation.
- `private LambdaTerm result`  
Der Term nach der Applikation.
- `private boolean hasCheckedAlphaConversion`  
Initialisiert mit `false`. Speichert, ob bereits überprüft wurde, ob eine Alpha-Konversion vor der Applikation notwendig ist.

#### Konstruktoren

- `public ApplicationVisitor(Color color, LambdaTerm applicant)`  
Instanziert ein Objekt dieser Klasse mit der gegebenen Variablenfarbe und dem gegebenen Argument.

#### Parameter

- `Color color`  
Die Farbe der zu ersetzenden Variablen.
- `LambdaTerm applicant`  
Das Argument der Applikation.

#### Exceptions

- `NullPointerException`  
Falls `color == null` oder `applicant == null` ist.

#### Methoden

- `public void visit(LambdaRoot node)`  
Kann nie aufgerufen werden, da der besuchte Knoten keinen Elternknoten hat, von wo aus eine Applikation ausgeführt werden könnte.

#### Parameter

- `LambdaRoot node`  
Die besuchte Wurzel.

- `public void visit(LambdaApplication node)`  
Besucht die gegebene Applikation und traversiert weiter zu beiden Kindknoten. Dabei werden die Kindknoten auf die Rückgabewerte beider Besuche gesetzt. Speichert als Rückgabewert den besuchten Term.

#### Parameter

- `LambdaApplication node`  
Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`  
Besucht die gegebene Abstraktion und traversiert weiter zum Kindknoten. Dabei wird der Kindknoten auf den Rückgabewert des Besuchs gesetzt. Speichert als Rückgabewert den besuchten Term.

#### Parameter

- `LambdaAbstraction node`  
Die besuchte Abstraktion.

- `public void visit(LambdaVariable node)`  
Besucht die gegebene Variable und speichert wenn nötig als Rückgabewert `applicant`.

#### Parameter

- `LambdaVariable node`  
Die besuchte Variable.

- `public LambdaTerm getResult()`  
Gibt den Term nach der Applikation zurück.

#### Rückgabe

- Der besuchte Term.

- `private void checkAlphaConversion()`  
Überprüft, ob eine Alpha-Konversion notwendig ist, falls dies noch nicht getan wurde, und führt diese wenn nötig aus. Entfernt danach das Argument der Applikation aus dem `LambdaTerm`.

**3.3.6** `public class CopyVisitor implements  
LambdaTermVisitor<LambdaTerm>`

#### Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, welcher die Datenstruktur kopiert und die Kopie zurückgibt.

#### Attribute

- `private LambdaTerm result`  
Die Kopie.

#### Konstruktoren

- `public CopyVisitor()`  
Instanziert ein Objekt dieser Klasse.

#### Methoden

- `public void visit(LambdaRoot node)`  
Besucht die gegebene Wurzel und erstellt eine Kopie. Traversiert zum Kindknoten und speichert den Rückgabewert dieses Besuchs im Kindknoten der Kopie.

#### Parameter

- `LambdaRoot node`  
Die besuchte Wurzel.

- `public void visit(LambdaApplication node)`  
Besucht die gegebene Applikation und erstellt eine Kopie. Traversiert zu beiden Kindknoten und speichert die Rückgabewerte dieser Besuche in den Kindknoten der Kopie.

#### Parameter

- `LambdaApplication node`  
Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`  
Besucht die gegebene Abstraktion und erstellt eine Kopie. Traversiert zum Kindknoten und speichert den Rückgabewert dieses Besuchs im Kindknoten der Kopie.

#### Parameter

- `LambdaAbstraction node`  
Die besuchte Abstraktion.

- `public void visit(LambdaVariable node)`  
Besucht die gegebene Variable und speichert als Rückgabewert eine Kopie dieser Variable.

#### Parameter

- `LambdaVariable node`  
Die besuchte Variable.

- `public LambdaTerm getResult()`  
Gibt die Kopie zurück.

#### Rückgabe

- Die Kopie.

**3.3.7** `public class RemoveTermVisitor implements  
LambdaTermVisitor<Object>`

### Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, welcher den besuchten Term aus der Datenstruktur entfernt.

### Attribute

- `private LambdaTerm removed`  
Der zu entfernende Term. Initialisiert mit `null`.

### Konstruktoren

- `public RemoveTermVisitor()`  
Instanziert ein Objekt dieser Klasse.

### Methoden

- `public void visit(LambdaRoot node)`  
Falls ein zu entfernender Term - Kindknoten der Wurzel - gespeichert ist, setze den Kindknoten auf `null`.

#### Parameter

- `LambdaRoot node`  
Die besuchte Wurzel.

- `public void visit(LambdaApplication node)`  
Besucht die gegebene Applikation. Falls noch kein zu entfernender Term gespeichert ist, speichere diese Applikation und traversiere zum Elternknoten, falls dieser nicht `null` ist. Ansonsten ist der Term bereits aus der Baumstruktur entfernt. Falls ein zu entfernender Term - Kindknoten in der Applikation - gespeichert ist, ersetze diesen durch `null`.

#### Parameter

- `LambdaApplication node`  
Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`  
Besucht die gegebene Abstraktion. Falls noch kein zu entfernender Term gespeichert ist, speichere diese Abstraktion und traversiere zum Elternknoten, falls dieser nicht `null` ist. Ansonsten ist der Term bereits aus der Baumstruktur entfernt. Falls ein zu entfernender Term - Kindknoten der Abstraktion - gespeichert ist, ersetze diesen durch `null`.

#### Parameter

- `LambdaAbstraction node`  
Die besuchte Abstraktion.

- `public void visit(LambdaVariable node)`  
Speichere die Variable als zu entfernenden Term und traversiere zum Elternknoten, falls dieser nicht `null` ist. Ansonsten ist der Term bereits aus der Baumstruktur entfernt.

#### Parameter

- `LambdaVariable node`  
Die besuchte Variable.

**3.3.8** `public abstract class BetaReductionVisitor implements LambdaTermVisitor<LambdaTerm>`

#### Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, der eine einzelne Beta-Reduktion gemäß einer Reduktionsstrategie durchführt. Dabei sind Strategien durch Unterklassen dieses Besuchers gegeben.

#### Attribute

- `protected LambdaTerm result`  
Der Term nach der Beta-Reduktion.
- `protected boolean hasReduced`  
Speichert, ob von diesem Besucher bereits eine Reduktion durchgeführt wurde. Initialisiert mit `false`.
- `protected LambdaTerm applicant`  
Falls der Elternknoten des aktuell besuchten Knotens eine Applikation ist, speichert diese Variable das Argument der Applikation. Initialisiert mit `null`.

#### Konstruktoren

- `public BetaReductionVisitor()`  
Instanziert ein Objekt dieser Klasse.

#### Methoden

- `public void visit(LambdaRoot node)`  
Traversiere weiter zum Kindknoten und setze diesen auf das Resultat des Besuchs. Speichere als Rückgabewert die besuchte Wurzel.

#### Parameter

- `LambdaRoot node`  
Die besuchte Wurzel.

- `public abstract void visit(LambdaApplication node)`  
Implementiert von der Reduktionsstrategie. Führt entsprechende Operationen zur Reduktion am Term aus (siehe Unterklassen). Gibt in der Standard-Implementierung nur den besuchten Knoten zurück.

#### Parameter

- `LambdaApplication node`  
Die besuchte Applikation.

- `public abstract void visit(LambdaAbstraction node)`  
Implementiert von der Reduktionsstrategie. Führt entsprechende Operationen zur Reduktion am Term aus (siehe Unterklassen). Gibt in der Standard-Implementierung nur den besuchten Knoten zurück.

#### Parameter

- `LambdaAbstraction node`  
Die besuchte Abstraktion.

- `public abstract void visit(LambdaVariable node)`  
Implementiert von der Reduktionsstrategie. Führt entsprechende Operationen zur Reduktion am Term aus (siehe Unterklassen). Gibt in der Standard-Implementierung nur den besuchten Knoten zurück.

#### Parameter

- `LambdaVariable node`  
Die besuchte Variable.

- `public LambdaTerm getResult()`  
Gibt das Resultat der Reduktion zurück.

#### Rückgabe

- Der reduzierte Term.

### 3.4 package `lambda.model.lambdaterm.visitor.strategy`

3.4.1 public class **ReductionStrategyNormalOrder** extends  
BetaReductionVisitor

#### Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, der eine einzelne Beta-Reduktion gemäß der Normal-Order Strategie durchführt.

#### Konstruktoren

- public **ReductionStrategyNormalOrder**()  
Instanziert ein Objekt dieser Klasse.

#### Methoden

- public void **visit**(LambdaApplication node)  
Falls noch keine Applikation ausgeführt wurde, traversiert erst zum linken Kind mit rechtem Kind als Argument und dann, falls dort keine Applikation ausgeführt wurde, zum rechten Kind ohne Argument. Rückgabewert ist der linke Kindknoten, falls dort die Applikation ausgeführt wurde, ansonsten der besuchte Knoten.

#### Parameter

- LambdaApplication node  
Die besuchte Applikation.

- public void **visit**(LambdaAbstraction node)  
Falls bereits eine Applikation ausgeführt wurde, gibt nur den besuchten Knoten zurück. Ansonsten, falls ein Argument gegeben ist, führt damit eine Applikation auf dieser Abstraktion aus. Rückgabewert ist das Resultat der Applikation. Traversiert ansonsten zum Kindknoten und gibt den besuchten Knoten zurück.

#### Parameter

- LambdaAbstraction node  
Die besuchte Abstraktion.



**3.4.2** `public class ReductionStrategyApplicativeOrder extends  
BetaReductionVisitor`

### Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, der eine einzelne Beta-Reduktion gemäß der Applicative-Order Strategie durchführt.

### Konstruktoren

- `public ReductionStrategyApplicativeOrder()`  
Instanziert ein Objekt dieser Klasse.

### Methoden

- `public void visit(LambdaApplication node)`  
Falls bereits eine Applikation ausgeführt wurde, gibt den besuchten Knoten zurück. Traversiert ansonsten erst zum linken Kind mit rechtem Kind als Argument und dann, falls dort keine Applikation ausgeführt wurde, zum rechten Kind ohne Argument. Rückgabewert ist der linke Kindknoten, falls dort die Applikation ausgeführt wurde, ansonsten der besuchte Knoten.

#### Parameter

- `LambdaApplication node`  
Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`  
Falls bereits eine Applikation ausgeführt wurde, gibt den besuchten Knoten zurück. Traversiert ansonsten zum Kindknoten und gibt den besuchten Knoten zurück. Falls danach noch keine Applikation ausgeführt wurde und ein Argument gegeben ist, führt damit eine Applikation auf dieser Abstraktion aus. Rückgabewert ist dann das Resultat der Applikation.

#### Parameter

- `LambdaAbstraction node`  
Die besuchte Abstraktion.

**3.4.3** `public class ReductionStrategyCallByValue extends  
BetaReductionVisitor`

### Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, der eine einzelne Beta-Reduktion gemäß der Call-By-Value Strategie durchführt.

## Konstruktoren

- `public ReductionStrategyCallByValue()`  
Instanziert ein Objekt dieser Klasse.

## Methoden

- `public void visit(LambdaApplication node)`  
Falls bereits eine Applikation ausgeführt wurde, gibt den besuchten Knoten zurück. Traversiert ansonsten erst zum rechten Kind ohne Argument und dann, falls dort keine Applikation ausgeführt wurde, zum linken Kind mit rechtem Kind als Argument. Rückgabewert ist der linke Kindknoten, falls dort die Applikation ausgeführt wurde, ansonsten der besuchte Knoten.

### Parameter

- `LambdaApplication node`  
Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`  
Falls bereits eine Applikation ausgeführt wurde, gibt den besuchten Knoten zurück. Falls ansonsten ein Argument gegeben und ein Wert - d.h. Abstraktion oder Variable - ist, führt damit eine Applikation auf dieser Abstraktion aus. Rückgabewert ist dann das Resultat der Applikation, ansonsten der besuchte Knoten.

### Parameter

- `LambdaAbstraction node`  
Die besuchte Abstraktion.

**3.4.4** `public class ReductionStrategyCallByName extends BetaReductionVisitor`

## Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, der eine einzelne Beta-Reduktion gemäß der Call-By-Name Strategie durchführt.

## Konstruktoren

- `public ReductionStrategyCallByName()`  
Instanziert ein Objekt dieser Klasse.

## Methoden

- `public void visit(LambdaApplication node)`  
 Falls noch keine Applikation ausgeführt wurde, traversiert erst zum linken Kind mit rechtem Kind als Argument und dann, falls dort keine Applikation ausgeführt wurde, zum rechten Kind ohne Argument. Rückgabewert ist der linke Kindknoten, falls dort die Applikation ausgeführt wurde, ansonsten der besuchte Knoten.

#### Parameter

- `LambdaApplication node`  
 Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`  
 Falls bereits eine Applikation ausgeführt wurde, gibt nur den besuchten Knoten zurück. Ansonsten, falls ein Argument gegeben ist, führt damit eine Applikation auf dieser Abstraktion aus. Rückgabewert ist das Resultat der Applikation. Traversiert nicht weiter zum Kindknoten.

#### Parameter

- `LambdaAbstraction node`  
 Die besuchte Abstraktion.

## 3.5 package `lambda.model.profiles`

### 3.5.1 public interface `ProfileModelObserver`

#### Beschreibung

Stellt einen Beobachter eines ProfileModels dar, welcher über Änderungen informiert wird.

#### Methoden

- default `public void changedAvatar()`  
 Wird aufgerufen um dem Beobachter mitzuteilen, dass sich der Key des Avatars, durch den sich das Avatarbild unter den Assets finden lässt, geändert hat. Die Standard-Implementierung ist leer.

**3.5.2** `public class ProfileModel implements  
Observable<ProfileModelObserver>`

### Beschreibung

Repräsentiert ein komplettes Benutzerprofil.

### Attribute

- `private String name`  
Gibt den Name des Profils an, wodurch es eindeutig zu identifizieren ist.
- `private String avatar`  
Gibt den Key des Avatars an, durch den sich das Avatarbild unter den Assets finden lässt.
- `private String language`  
Gibt den Key der Sprache an, durch den sich das Sprachpaket unter den Assets finden lässt.
- `private int levelIndex`  
Gibt die Nummer des ersten, noch nicht bestandenen Levels an.
- `private SettingsModel settings`  
Stellt eine Referenz zu den, zum Profil gehörenden, Einstellungen dar.
- `private ShopModel shop`  
Stellt eine Referenz zu dem, zum Profil gehörenden, Shop dar.
- `private AchievementManager achievements`  
Stellt eine Referenz zu dem, zum Profil gehörenden, Achievement-System dar.
- `private StatisticsModel statistics`  
Stellt eine Referenz zu den, zum Profil gehörenden, Statistiken dar.

### Konstruktoren

- `public ProfileModel(String name)`  
Instanziert ein Objekt dieser Klasse.

#### Parameter

- `String name`  
Der Name des neuen Profils. `null` ist erlaubt, resultiert in einem -String als Name.

## Methoden

- `public String getName()`  
Gibt den Profil-Name zurück.

### Rückgabe

- Gibt den Profil-Name zurück.

- `public String getAvatar()`  
Gibt den Key des Avatars zurück.

### Rückgabe

- Gibt den Key des Avatars zurück.

- `public void setAvatar(String avatar)`  
Setzt den Avatar-Key neu und informiert alle Beobachter über diese Änderung.

### Parameter

- `String avatar`  
Der Key unter dem der neue Avatar zu finden ist.

- `public String getLanguage()`  
Gibt den Key des Sprachpakets zurück.

### Rückgabe

- Gibt den Key des Sprachpakets zurück.

- `public void setLanguage(String language)`  
Setzt den Sprachpaket-Key neu und setzt die Spiel-Sprache, falls dieses Profil das vom ProfileManager aktuell ausgewählte Profil ist.

### Parameter

- `String language`  
Der Key unter dem das neue Sprachpaket zu finden ist.

- `public int getLevelIndex()`  
Gibt die Nummer des ersten, noch nicht bestandenen Levels zurück.

### Rückgabe

- Gibt die Nummer des ersten, noch nicht bestandenen Levels zurück.

- `public void setLevelIndex(int levelIndex)`  
Setzt die Nummer des ersten, noch nicht bestandenen Levels und informiert alle Beobachter über diese Änderung.

#### Parameter

- `int levelIndex`  
Den Wert auf den der Level-Index gesetzt werden soll.

#### Exceptions

- `IllegalArgumentException`  
Falls `levelIndex < 1` ist.

- `public SettingsModel getSettings()`  
Gibt die Referenz auf die Einstellungen zurück.

#### Rückgabe

- Gibt die Referenz auf die Einstellungen zurück.

- `public ShopModel getShop()`  
Gibt die Referenz auf den Shop zurück.

#### Rückgabe

- Gibt die Referenz auf den Shop zurück.

- `public AchievementManager getAchievements()`  
Gibt die Referenz auf das Achievement-System zurück.

#### Rückgabe

- Gibt die Referenz auf das Achievement-System zurück.

- `public StatisticsModel getStatistics()`  
Gibt die Referenz auf die Statistiken zurück.

#### Rückgabe

- Gibt die Referenz auf die Statistiken zurück.

### 3.5.3 `public interface ProfileManagerObserver`

#### Beschreibung

Stellt einen Beobachter eines ProfileManagers dar, welcher über Änderungen infor-

miert wird.

## Methoden

- default public void **changedProfile()**  
Wird aufgerufen um dem Beobachter mitzuteilen, dass der ProfileManager ein anderes Profil ausgewählt hat. Die Standard-Implementierung ist leer.
- default public void **changedLanguage()**  
Wird aufgerufen um dem Beobachter mitzuteilen, dass sich die Spielsprache geändert hat. Die Standard-Implementierung ist leer.
- default public void **changedNames()**  
Wird aufgerufen um dem Beobachter mitzuteilen, dass es eine Änderung der Profil-Namen gab. Die Standard-Implementierung ist leer.

**3.5.4** public class **ProfileManager** implements  
Observable<ProfileManagerObserver>

## Beschreibung

Verwaltet alle Profile des Spiels und die Spielsprache.

## Attribute

- private static ProfileManager **manager**  
Stellt die einzige Instanz dar, die vom ProfileManager gleichzeitig existieren darf.
- private ProfileModel **currentProfile**  
Gibt das momentan im Spiel ausgewählte Profil an.
- private I18NBundle **language**  
Stellt das aktuelle Sprachpaket des Spiels dar.
- private List<ProfileModel> **profiles**  
Stellt eine Liste aller im Spiel vorhandenen Profile dar.

## Konstruktoren

- private **ProfileManager()**  
Instanziert ein Objekt dieser Klasse und lädt alle gespeicherten ProfileModels.

## Methoden

- `public static ProfileManager getManager()`  
Nimmt die existierende ProfileManager-Instanz oder erstellt eine Neue und gibt diese zurück.

#### Rückgabe

- Nimmt die existierende ProfileManager-Instanz oder erstellt eine Neue und gibt diese zurück.

- `public ProfileModel getCurrentProfile()`  
Gibt das ausgewählte Profil zurück.

#### Rückgabe

- Gibt das ausgewählte Profil zurück.

- `public boolean setCurrentProfile(String name)`  
Setzt das ausgewählte Profil neu und informiert alle Beobachter über diese Änderung.

#### Parameter

- `String name`  
Name des neuen Profils.

#### Rückgabe

- Gibt zurück, ob der ProfileManager das gegebene Profil finden konnte.

- `public List<String> getNames()`  
Gibt eine Liste aller Profil-Namen zurück.

#### Rückgabe

- Gibt eine Liste aller Profil-Namen zurück.

- `public ProfileModel createProfile()`  
Erstellt ein neues Profil mit einem leeren String als Namen, gibt dieses zurück und informiert alle Beobachter über diese Änderung.

#### Rückgabe

- Neues Profil. `null` falls die maximale Profil-Anzahl erreicht wurde oder als Name schon vorkommt, was nicht passieren sollte.

- `public boolean changeName(String old, String new)`  
Ersetzt das Profil mit Namen `old` durch ein Profil mit dem Namen `new`, aber



sonst identischen Werten und informiert alle Beobachter über diese Änderung.

#### Rückgabe

- Gibt zurück, ob die Methode erfolgreich war.

- `public void save(String name)`  
Sichert das angegebene Profil als Datei.

#### Parameter

- `String name`  
Der Name des Profils, das gespeichert werden soll.

- `public void delete(String name)`  
Löscht das angegebene Profil komplett (auch Datei) und informiert alle Beobachter über diese Änderung.

#### Parameter

- `String name`  
Der Name des Profils, das gelöscht werden soll.

- `public void setLanguage(String lang)`  
Setzt die Spielsprache neu und informiert alle Beobachter über diese Änderung. Wird beim Profilwechsel und vom aktuellen Profil aufgerufen.

#### Parameter

- `String lang`  
Der Key unter dem das gewünschte Sprachpaket gefunden werden kann.

- `public I18NBundle getLanguage()`  
Gibt das aktuelle Sprachpaket des Spiels zurück.

#### Rückgabe

- Gibt das aktuelle Sprachpaket des Spiels zurück.

### 3.5.5 `public interface ProfileEditLangObserver`

#### Beschreibung

Stellt einen Beobachter eines `ProfileEditLangModels` dar, welcher über Änderungen der ausgewählten Sprache informiert wird.

## Methoden

- default public void **changedLang()**  
Wird aufgerufen um dem Beobachter mitzuteilen, dass vom ProfileEditLang-Model eine andere Sprache ausgewählt wurde. Die Standard-Implementierung ist leer.

**3.5.6** public class **ProfileEditLangModel** implements  
Observable<ProfileEditLangObserver>

## Beschreibung

Repräsentiert die Logik hinter der Sprachauswahl

## Attribute

- private List<String> **lang**  
Stellt eine Liste an Keys dar, durch die alle Sprachpakete des Spiels gefunden werden können.
- private List<String> **langpic**  
Stellt eine, zu **lang** passende, Liste an Keys dar, durch die alle Bilder der Landesflaggen gefunden werden können.
- private int **selectedLang**  
Die aktuelle Position in beiden Listen.

## Konstruktoren

- public **ProfileEditLangModel()**  
Instanziert ein Objekt dieser Klasse und dessen Attribute.

## Methoden

- public void **setLang**(String lang)  
Setzt die Sprachauswahl neu.

### Parameter

- String name  
Key der entsprechenden Sprache.

- public void **nextLang()**  
Wählt die nächste Sprache aus und informiert alle Beobachter über diese Änderung.

- `public void previousLang()`  
Wählt die vorherige Sprache aus und informiert alle Beobachter über diese Änderung.
- `public String getLang()`  
Gibt den Key des Sprachpakets der gewählten Sprache zurück.

#### Rückgabe

- Gibt den Key des Sprachpakets der gewählten Sprache zurück.

- `public String getLang()`  
Gibt den Key des Bildes der gewählten Sprache zurück.

#### Rückgabe

- Gibt den Key des Bildes der gewählten Sprache zurück.

### 3.5.7 `public interface ProfileEditAvatarObserver`

#### Beschreibung

Stellt einen Beobachter eines `ProfileEditAvatarModels` dar, welcher über Änderungen des ausgewählten Avatars informiert wird.

#### Methoden

- `default public void changedAvatar()`  
Wird aufgerufen um dem Beobachter mitzuteilen, dass vom `ProfileEditAvatar-Model` ein anderer Avatar ausgewählt wurde. Die Standard-Implementierung ist leer.

### 3.5.8 `public class ProfileEditAvatarModel implements Observable<ProfileEditAvatarObserver>`

#### Beschreibung

Repräsentiert die Logik hinter der Avataorauswahl

#### Attribute

- `private List<String> avatar`  
Stellt eine Liste an Keys dar, durch die alle Avatarbilder des Spiels gefunden werden können.

- `private int selectedAvatar`  
Die aktuelle Position in der **avatar**-Liste.

### Konstruktoren

- `public ProfileEditAvatarModel()`  
Instanziert ein Objekt dieser Klasse und dessen Attribute.

### Methoden

- `public void setAvatar(String avatar)`  
Setzt die Avataorauswahl neu.

#### Parameter

- `String avatar`  
Key des entsprechenden Avatars.

- `public void nextAvatar()`  
Wählt den nächsten Avatar aus und informiert alle Beobachter über diese Änderung.
- `public void previousAvatar()`  
Wählt den vorherige Avatar aus und informiert alle Beobachter über diese Änderung.
- `public String getAvatar()`  
Gibt den gewählten Avatar-Key zurück.

#### Rückgabe

- Gibt den gewählten Avatar-Key zurück.

## 3.6 package `lambda.model.settings`

### 3.6.1 public interface **SettingsModelObserver**

#### Beschreibung

Stellt einen Beobachter eines SettingModels dar, welcher über geänderte Einstellungen informiert wird.

#### Methoden

- default public void **changedMusicOn()**  
Wird aufgerufen um dem Beobachter mitzuteilen, dass sich der Status, ob Musik gespielt werden soll oder nicht, geändert hat. Die Standard-Implementierung ist leer.
- default public void **changedMusicVolume()**  
Wird aufgerufen um dem Beobachter mitzuteilen, dass sich die Musiklautstärke geändert hat. Die Standard-Implementierung ist leer.
- default public void **changedSoundVolume()**  
Wird aufgerufen um dem Beobachter mitzuteilen, dass sich die Geräuschlautstärke geändert hat. Die Standard-Implementierung ist leer.
- default public void **changedTeacher()**  
Wird aufgerufen um dem Beobachter mitzuteilen, dass der Lehrermodus aktiviert oder deaktiviert wurde. Die Standard-Implementierung ist leer.
- default public void **changedColorblind()**  
Wird aufgerufen um dem Beobachter mitzuteilen, dass der Farbenblindenmodus aktiviert oder deaktiviert wurde. Die Standard-Implementierung ist leer.

**3.6.2** public class **SettingsModel** implements  
Observable<SettingsModelObserver>

### Beschreibung

Repräsentiert Einstellungen eines Profils bezüglich Tonausgabe, Lehrer- und Farbenblindenmodus.

### Attribute

- private boolean **musicOn**  
Gibt an, ob die Hintergrundmusik laufen soll oder nicht.
- private float **musicVolume**  
Gibt die Musiklautstärke in Prozent an. (0 bis 100)
- private float **soundVolume**  
Gibt die Lautstärke von Geräuschen in Prozent an. (0 bis 100)
- private boolean **teacher**  
Gibt an, ob der Lehrermodus aktiviert ist.

- `private boolean colorblind`  
Gibt an, ob der Farbenblindenmodus aktiviert ist.

## Konstrukturen

- `public SettingsModel()`  
Instanziert ein Objekt dieser Klasse und initialisiert Attribute mit Standardwerten.

## Methoden

- `public boolean isMusicOn()`  
Gibt zurück, ob Musik abgespielt werden soll.

### Rückgabe

- Gibt zurück, ob Musik abgespielt werden soll.

- `public void setMusicOn(boolean musicOn)`  
Setzt den Status, ob Musik abgespielt werden soll und informiert alle Beobachter über diese Änderung.

### Parameter

- `boolean musicOn`  
Gibt an, ob Musik abgespielt werden soll.

- `public float getMusicVolume()`  
Gibt die Musiklautstärke in Prozent zurück.

### Rückgabe

- Gibt die Musiklautstärke in Prozent zurück.

- `public void setMusicVolume(float musicVolume)`  
Setzt die Musiklautstärke und informiert alle Beobachter über diese Änderung.

### Parameter

- `float musicVolume`  
Die neue Lautstärke der Musik. Sollte zwischen 0 und 100 liegen.

- `public float getSoundVolume()`  
Gibt die Geräuschlautstärke in Prozent zurück.

### Rückgabe

- Gibt die Geräuschlautstärke in Prozent zurück.

- `public void setSoundVolume(float soundVolume)`  
Setzt die Geräuschlautstärke und informiert alle Beobachter über diese Änderung.

#### Parameter

- `float soundVolume`  
Die neue Lautstärke der Geräusche. Sollte zwischen 0 und 100 liegen.

- `public boolean isTeacher()`  
Gibt zurück, ob der Lehrermodus aktiviert ist.

#### Rückgabe

- Gibt zurück, ob der Lehrermodus aktiviert ist.

- `public void setTeacher(boolean teacher)`  
Aktiviert bzw. deaktiviert den Lehrermodus und informiert alle Beobachter über diese Änderung.

#### Parameter

- `boolean teacher`  
Gibt an, ob der Lehrermodus aktiviert oder deaktiviert werden soll.

- `public boolean isColorblind()`  
Gibt zurück, ob der Farbenblindenmodus aktiviert ist.

#### Rückgabe

- Gibt zurück, ob der Farbenblindenmodus aktiviert ist.

- `public void setTeacher(boolean teacher)`  
Aktiviert bzw. deaktiviert den Farbenblindenmodus und informiert alle Beobachter über diese Änderung.

#### Parameter

- `boolean teacher`  
Gibt an, ob der Farbenblindenmodus aktiviert oder deaktiviert werden soll.

### 3.7 package `lambda.model.achievements`

#### 3.7.1 public interface **AchievementModelObserver**

##### Beschreibung

Stellt einen Beobachter eines AchievementModels dar, welcher über Änderungen informiert wird.

##### Methoden

- default public void **changedLockedState**(int id)  
Wird aufgerufen um dem Beobachter mitzuteilen, dass sich der Zustand des Erfolgs, also ob dieser freigeschaltet ist oder nicht, geändert hat. Die Standard-Implementierung ist leer.

##### Parameter

- int id  
Die ID des Erfolgs mit dem geänderten Zustand.

#### 3.7.2 public abstract class **AchievementModel** implements `Observable<AchievementModelObserver>`

##### Beschreibung

Repräsentiert einen Erfolg.

##### Attribute

- private int **id**  
Die kennzeichnende ID des Erfolgs.
- private int **index**  
Der Index des Erfolgs der dessen Platz in einer geordneten Auflistung bestimmt.
- private String **description**  
Die Beschreibung des freigeschalteten Erfolgs.
- private String **requirementDescription**  
Die Beschreibung der Bedingungen des Erfolgs.
- private String **iconPathAchievementLocked**



Der Pfad zum Piktogramm des nicht freigeschalteten Erfolgs, der beim Laden des jeweiligen Piktogramms hilft.

- `private String iconPathAchievementUnlocked`  
Der Pfad zum Piktogramm der nicht freigeschalteten Erfolgs, der beim Laden des jeweiligen Piktogramms hilft.
- `private boolean locked`  
Gibt an, ob die Bedingungen des Erfolgs erfüllt sind und der Erfolgs damit freigeschaltet ist oder nicht.

### Konstrukturen

- `public AchievementModel()`  
Instanziert ein Objekt dieser Klasse. Setzt `locked` auf `false`.

### Methoden

- `public abstract void initialize()`  
Initialisiert den Erfolg.
- `public abstract boolean meetRequirements(StatisticModel statistic)`  
Prüft anhand der übergebenen Statistik, ob die Bedingungen dieses Erfolgs erfüllt sind.

### Parameter

- `StatisticModel statistic`  
Die Statistik anhand derer die Überprüfung geschieht.

### Rückgabe

- Gibt `true` zurück, falls die Bedingungen dieses Erfolgs erfüllt sind und ansonsten `false`.

### Exceptions

- `NullPointerException`  
Falls `statistic == null` ist.

- `public int getId()`  
Gibt die ID dieses Erfolgs zurück.

### Rückgabe

- Die ID des Erfolgs.

- `public int getIndex()`  
Gibt den Index des Erfolgs zurück.

#### **Rückgabe**

- Der Index des Erfolgs.

- `public String getDescription()`  
Gibt die Beschreibung für den freigeschalteten Erfolg zurück.

#### **Rückgabe**

- Die Beschreibung des freigeschalteten Erfolgs.

- `public String getRequirementsDescription()`  
Gibt die Beschreibung der Bedingungen dieses Erfolgs zurück.

#### **Rückgabe**

- Die Beschreibung der Bedingungen dieses Erfolgs.

- `public String getIconPathAchievementLocked()`  
Gibt den Pfad zum Piktogramm des nicht freigeschalteten Erfolgs zurück.

#### **Rückgabe**

- Der Pfad zum Piktogramm des nicht freigeschalteten Erfolgs.

- `public String getIconPathAchievementUnlocked()`  
Gibt den Pfad zum Piktogramm des freigeschalteten Erfolgs zurück.

#### **Rückgabe**

- Der Pfad zum Piktogramm des freigeschalteten Erfolgs.

- `public boolean isLocked()`  
Gibt an, ob der Erfolg freigeschaltet ist oder nicht.

#### **Rückgabe**

- Gibt `true` zurück, falls der Erfolg nicht freigeschaltet ist und `false` falls der Erfolg freigeschaltet ist.

- `public void setId(int id)`  
Setzt eine neue ID für diesen Erfolg.

#### Parameter

- int id  
Die neue ID.

- public void **setIndex**(int index)  
Setzt einen neuen Index für diesen Erfolg

#### Parameter

- int id  
Der neue Index.

- public void **setDescription**(String description)  
Setzt eine neue Beschreibung für den freigeschalteten Erfolg.

#### Parameter

- String description  
Die neue Beschreibung des freigeschalteten Erfolgs.

- public void **setRequirementsDescription**(String requirementsDescription)  
Setzt eine neue Beschreibung für die Bedingungen dieses Erfolgs.

#### Parameter

- String requirementsDescription  
Die neue Beschreibung für die Bedingungen dieses Erfolgs.

- public void **setIconPathAchievementLocked**(String iconPathAchievementLocked)  
Setzt einen neuen Pfad zum Piktogramm des nicht freigeschalteten Erfolgs.

#### Parameter

- String iconPathAchievementLocked  
Der neue Pfad zum Piktogramm des nicht freigeschalteten Erfolgs.

- public void **setIconPathAchievementUnlocked**(String iconPathAchievementUnlocked)  
Setzt einen neuen Pfad zum Piktogramm des freigeschalteten Erfolgs.

#### Parameter

- String iconPathAchievementUnlocked  
Der neue Pfad zum Piktogramm des freigeschalteten Erfolgs.

- public void **setLocked**(boolean locked)  
Setzt diesen Erfolg auf den Zustand freigeschaltet oder nicht freigeschaltet.

#### Parameter

- boolean `locked`  
Bei `true` wird der Erfolg auf nicht freigeschaltet und bei `false` auf freigeschaltet gesetzt.

**3.7.3** `public class TimeAchievementModel extends AchievementModel`

#### Beschreibung

Repräsentiert einen Erfolg, der nach dem Spielen einer bestimmten Zeitspanne freigeschaltet wird.

#### Konstruktoren

- `public TimeAchievementModel()`  
Instanziert ein Objekt dieser Klasse. Setzt `locked` auf `false`.

#### Methoden

- `public void initialize()`  
Initialisiert den Erfolg.
- `public boolean meetRequirements(StatisticModel statistic)`  
Prüft anhand der übergebenen Statistik, ob die Bedingungen dieses Erfolgs erfüllt sind.

#### Parameter

- `StatisticModel statistic`  
Die Statistik anhand derer die Überprüfung geschieht.

#### Rückgabe

- Gibt `true` zurück, falls die Bedingungen dieses Erfolgs erfüllt sind und ansonsten `false`.

#### Exceptions

- `NullPointerException`  
Falls `statistic == null` ist.

**3.7.4** `public class LevelAchievementModel extends AchievementModel`

### Beschreibung

Repräsentiert einen Erfolg, der nach dem erfolgreichen ersten Abschluss einer bestimmten Mindestanzahl von Level freigeschaltet wird.

### Konstruktoren

- `public LevelAchievementModel()`  
Instanziert ein Objekt dieser Klasse. Setzt `locked` auf `false`.

### Methoden

- `public void initialize()`  
Initialisiert den Erfolg.
- `public boolean meetRequirements(StatisticModel statistic)`  
Prüft anhand der übergebenen Statistik, ob die Bedingungen dieses Erfolgs erfüllt sind.

### Parameter

- `StatisticModel statistic`  
Die Statistik anhand derer die Überprüfung geschieht.

### Rückgabe

- Gibt `true` zurück, falls die Bedingungen dieses Erfolgs erfüllt sind und ansonsten `false`.

### Exceptions

- `NullPointerException`  
Falls `statistic == null` ist.

**3.7.5** `public class GemsEnchantedAchievementModel extends AchievementModel`

### Beschreibung

Repräsentiert einen Erfolg, der nach dem Verzaubern einer bestimmten Mindestanzahl von Edelsteinen freigeschaltet wird.

## Konstrukturen

- `public GemsEnchantedAchievementModel()`  
Instanziert ein Objekt dieser Klasse. Setzt `locked` auf `false`.

## Methoden

- `public void initialize()`  
Initialisiert den Erfolg.
- `public boolean meetRequirements(StatisticModel statistic)`  
Prüft anhand der übergebenen Statistik, ob die Bedingungen dieses Erfolgs erfüllt sind.

### Parameter

- `StatisticModel statistic`  
Die Statistik anhand derer die Überprüfung geschieht.

### Rückgabe

- Gibt `true` zurück, falls die Bedingungen dieses Erfolgs erfüllt sind und ansonsten `false`.

### Exceptions

- `NullPointerException`  
Falls `statistic == null` ist.

**3.7.6** `public class LambsEnchantedAchievementModel extends AchievementModel`

## Beschreibung

Repräsentiert einen Erfolg, der nach dem Verzaubern einer bestimmten Mindestanzahl von Lämmern freigeschaltet wird.

## Konstrukturen

- `public LambsEnchantedAchievementModel()`  
Instanziert ein Objekt dieser Klasse. Setzt `locked` auf `false`.

## Methoden

- `public void initialize()`  
Initialisiert den Erfolg.

- `public boolean meetRequirements(StatisticModel statistic)`  
Prüft anhand der übergebenen Statistik, ob die Bedingungen dieses Erfolgs erfüllt sind.

#### Parameter

- `StatisticModel statistic`  
Die Statistik anhand derer die Überprüfung geschieht.

#### Rückgabe

- Gibt `true` zurück, falls die Bedingungen dieses Erfolgs erfüllt sind und ansonsten `false`.

#### Exceptions

- `NullPointerException`  
Falls `statistic == null` ist.

**3.7.7** `public class GemsPlacedAchievementModel extends AchievementModel`

#### Beschreibung

Repräsentiert einen Erfolg, der nach dem Platzieren einer bestimmten Mindestanzahl von Edelsteinen auf dem Spielfeld freigeschaltet wird.

#### Konstruktoren

- `public GemsPlacedAchievementModel()`  
Instanziert ein Objekt dieser Klasse. Setzt `locked` auf `false`.

#### Methoden

- `public void initialize()`  
Initialisiert den Erfolg.
- `public boolean meetRequirements(StatisticModel statistic)`  
Prüft anhand der übergebenen Statistik, ob die Bedingungen dieses Erfolgs erfüllt sind.

#### Parameter

- `StatisticModel statistic`  
Die Statistik anhand derer die Überprüfung geschieht.

### Rückgabe

- Gibt `true` zurück, falls die Bedingungen dieses Erfolgs erfüllt sind und ansonsten `false`.

### Exceptions

- `NullPointerException`  
Falls `statistic == null` ist.

**3.7.8** `public class LambsPlacedAchievementModel extends AchievementModel`

### Beschreibung

Repräsentiert einen Erfolg, der nach dem Platzieren einer bestimmten Mindestanzahl von Lämmern auf dem Spielfeld freigeschaltet wird.

### Konstruktoren

- `public LambsPlacedAchievementModel()`  
Instanziert ein Objekt dieser Klasse. Setzt `locked` auf `false`.

### Methoden

- `public void initialize()`  
Initialisiert den Erfolg.
- `public boolean meetRequirements(StatisticModel statistic)`  
Prüft anhand der übergebenen Statistik, ob die Bedingungen dieses Erfolgs erfüllt sind.

### Parameter

- `StatisticModel statistic`  
Die Statistik anhand derer die Überprüfung geschieht.

### Rückgabe

- Gibt `true` zurück, falls die Bedingungen dieses Erfolgs erfüllt sind und ansonsten `false`.

### Exceptions

- `NullPointerException`  
Falls `statistic == null` ist.



**3.7.9** `public class HintsAchievementModel extends AchievementModel`

### Beschreibung

Repräsentiert einen Erfolg, der freigeschaltet wird, nachdem eine bestimmte Mindestanzahl von Level erfolgreich ohne Nutzung des Hinweises abgeschlossen wurden.

### Konstruktoren

- `public HintsAchievementModel()`  
Instanziert ein Objekt dieser Klasse. Setzt `locked` auf `false`.

### Methoden

- `public void initialize()`  
Initialisiert den Erfolg.
- `public boolean meetRequirements(StatisticModel statistic)`  
Prüft anhand der übergebenen Statistik, ob die Bedingungen dieses Erfolgs erfüllt sind.

### Parameter

- `StatisticModel statistic`  
Die Statistik anhand derer die Überprüfung geschieht.

### Rückgabe

- Gibt `true` zurück, falls die Bedingungen dieses Erfolgs erfüllt sind und ansonsten `false`.

### Exceptions

- `NullPointerException`  
Falls `statistic == null` ist.

**3.7.10** `public abstract class PerLevelAchievementModel extends AchievementModel`

### Beschreibung

Repräsentiert einen Erfolg, dessen Freischaltung eine bestimmte Mindestanzahl von Ereignissen in einem Level erfordert.

## Konstruktor

- `public PerLevelAchievementModel()`  
Instanziert ein Objekt dieser Klasse. Setzt `locked` auf `false`.

**3.7.11** `public class GemsEnchantedPerLevelAchievementModel extends PerLevelAchievementModel`

## Beschreibung

Repräsentiert einen Erfolg, der freigeschaltet wird, falls eine bestimmte Mindestanzahl von Edelsteinen in einem Level verzaubert werden.

## Konstruktor

- `public GemsEnchantedPerLevelAchievementModel()`  
Instanziert ein Objekt dieser Klasse. Setzt `locked` auf `false`.

## Methoden

- `public void initialize()`  
Initialisiert den Erfolg.
- `public boolean meetRequirements(StatisticModel statistic)`  
Prüft anhand der übergebenen Statistik, ob die Bedingungen dieses Erfolgs erfüllt sind.

### Parameter

- `StatisticModel statistic`  
Die Statistik anhand derer die Überprüfung geschieht.

### Rückgabe

- Gibt `true` zurück, falls die Bedingungen dieses Erfolgs erfüllt sind und ansonsten `false`.

### Exceptions

- `NullPointerException`  
Falls `statistic == null` ist.

**3.7.12** `public class LambsEnchantedPerLevelAchievementModel  
extends PerLevelAchievementModel`

### Beschreibung

Repräsentiert einen Erfolg, der freigeschaltet wird, wenn eine bestimmte Mindestanzahl Lämmern in einem Level verzaubert werden.

### Konstruktoren

- `public LambsEnchantedPerLevelAchievementModel()`  
Instanziert ein Objekt dieser Klasse. Setzt `locked` auf `false`.

### Methoden

- `public void initialize()`  
Initialisiert den Erfolg
- `public boolean meetRequirements(StatisticModel statistic)`  
Prüft anhand der übergebenen Statistik, ob die Bedingungen dieses Erfolgs erfüllt sind.

### Parameter

- `StatisticModel statistic`  
Die Statistik anhand derer die Überprüfung geschieht.

### Rückgabe

- Gibt `true` zurück, falls die Bedingungen dieses Erfolgs erfüllt sind und ansonsten `false`.

### Exceptions

- `NullPointerException`  
Falls `statistic == null` ist.

**3.7.13** `public class GemsPlacedPerLevelAchievementModel extends  
PerLevelAchievementModel`

### Beschreibung

Repräsentiert einen Erfolg, der freigeschaltet wird, falls eine bestimmte Mindestanzahl von Edelsteinen in einem Level auf dem Spielfeld platziert wird.

## Konstrukturen

- `public GemsPlacedPerLevelAchievementModel()`  
Instanziert ein Objekt dieser Klasse. Setzt `locked` auf `false`.

## Methoden

- `public void initialize()`  
Initialisiert den Erfolg.
- `public boolean meetRequirements(StatisticModel statistic)`  
Prüft anhand der übergebenen Statistik, ob die Bedingungen dieses Erfolgs erfüllt sind.

### Parameter

- `StatisticModel statistic`  
Die Statistik anhand derer die Überprüfung geschieht.

### Rückgabe

- Gibt `true` zurück, falls die Bedingungen dieses Erfolgs erfüllt sind und ansonsten `false`.

### Exceptions

- `NullPointerException`  
Falls `statistic == null` ist.

**3.7.14** `public class LambsPlacedPerLevelAchievementModel extends PerLevelAchievementModel`

## Beschreibung

Repräsentiert einen Erfolg, der freigeschaltet wird, wenn eine bestimmte Mindestanzahl von Lämmern in einem Level auf dem Spielfeld platziert wird.

## Konstrukturen

- `public LambsPlacedPerLevelAchievementModel()`  
Instanziert ein Objekt dieser Klasse. Setzt `locked` auf `false`.

## Methoden

- `public void initialize()`  
Initialisiert den Erfolg.

- `public boolean meetRequirements(StatisticModel statistic)`  
Prüft anhand der übergebenen Statistik, ob die Bedingungen dieses Erfolgs erfüllt sind.

#### Parameter

- `StatisticModel statistic`  
Die Statistik anhand derer die Überprüfung geschieht.

#### Rückgabe

- Gibt `true` zurück, falls die Bedingungen dieses Erfolgs erfüllt sind und ansonsten `false`.

#### Exceptions

- `NullPointerException`  
Falls `statistic == null` ist.

### 3.7.15 `public class AchievementManager`

#### Beschreibung

Diese Klasse dient zur Verwaltung aller Erfolge.

#### Attribute

- `private Map<int, AchievementModel> unlockedAchievements`  
Ansammlung aller freigeschalteten Erfolge.
- `private Map<int, AchievementModel> lockedAchievements`  
Ansammlung aller nicht freigeschalteten Erfolge.

#### Konstruktoren

- `public AchievementManager()`  
Instanziert ein Objekt dieser Klasse.

#### Methoden

- `public void initializeAchievements()`  
Initialisiert alle Erfolge.
- `public void checkUnlockedAchievements(StatisticModel statistic)`  
Überprüft alle nicht freigeschalteten Erfolge anhand der übergebenen und aktuellen Statistik, ob deren Bedingungen erfüllt sind und sortiert sie gegebenenfalls in die Ansammlung der freigeschalteten Erfolge ein.

### Parameter

- `StatisticModel statistic`  
Die Statistik anhand derer die Überprüfung aller nicht freigeschalteter Erfolge geschieht.

### Exceptions

- `NullPointerException`  
Falls `statistic == null` ist.

- `public Map<int, AchievementModel> getUnlockedAchievements()`  
Gibt die Ansammlung aller freigeschalteter Erfolge zurück.

### Rückgabe

- Die Ansammlung aller freigeschalteter Erfolge.

- `public Map<int, AchievementModel> getLockedAchievements()`  
Gibt die Ansammlung aller nicht freigeschalteter Erfolge zurück.

### Rückgabe

- Die Ansammlung aller nicht freigeschalteten Erfolge.

- `public boolean addAchievement(AchievementModel achievement)`  
Überprüft, ob der Erfolg freigeschaltet ist oder nicht und fügt ihn anschließend der entsprechenden Ansammlung hinzu.

### Parameter

- `AchievementModel achievement`  
Der zu hinzufügende Erfolg.

### Rückgabe

- Gibt `true` zurück, falls das Hinzufügen in einer der Ansammlungen erfolgreich war oder `false`, falls der Erfolg schon in einer der Ansammlungen vorhanden ist und das Hinzufügen damit nicht erfolgreich war.

### Exceptions

- `NullPointerException`  
Falls `achievement == null` ist.

- `public boolean moveToUnlockedAchievements(int id)`  
Verschiebt den bereits vorhandenen Erfolg von der Ansammlung der nicht

freigeschalteten Erfolge zu der Ansammlung der freigeschalteten Erfolge.

#### Parameter

- `int id`  
Die ID des zu verschiebenden Erfolgs.

#### Rückgabe

- Gibt `true` zurück, falls das Verschieben des Erfolgs erfolgreich war und `false`, falls sich der Erfolg bereits in der Ansammlung der freigeschalteten Erfolge befand und damit das Verschieben nicht erfolgreich war.

#### Exceptions

- `IllegalArgumentException`  
Falls `id` in keiner der beiden Ansammlungen von Erfolgen vorhanden ist.

- `public boolean moveToLockedAchievements(int id)`  
Verschiebt die bereits vorhandenen Erfolge von der Ansammlung der freigeschalteten Erfolge zu der Ansammlung der nicht freigeschalteten Erfolge.

#### Parameter

- `int id`  
Die ID des zu verschiebenden Erfolgs.

#### Rückgabe

- Gibt `true` zurück, falls das Verschieben des Erfolgs erfolgreich war und `false`, falls sich der Erfolg bereits in der Ansammlung der nicht freigeschalteten Erfolge befand und damit das Verschieben nicht erfolgreich war.

#### Exceptions

- `IllegalArgumentException`  
Falls `id` in keiner der beiden Ansammlungen von Erfolgen vorhanden ist.

### 3.8 package `lambda.viewcontroller.lambdaterm`

3.8.1 public class **LambdaTermViewController** extends  
scene2d.Group implements LambdaTermObserver

#### Beschreibung

Kontrolliert die Darstellung von und Benutzerinteraktion mit einem Lambda-Term.

#### Attribute

- private scene2d.ClickListener **inputListener**  
Empfängt und bearbeitet UI-Events.
- private boolean **editable**  
Gibt an, ob Änderungen am Term durch den Benutzer zugelassen sind.
- private LambdaNodeViewController **selection**  
Enthält den Term, den der Benutzer per Drag&Drop-Geste auswählt. Initialisiert mit null.
- private Map<LambdaTerm, LambdaNodeViewController **nodeViewMap**  
Speichert alle View-Knoten als Wert zum verknüpften Lambda-Term als Schlüssel. Dabei wird die Identität der Schlüssel per Referenzvergleich anstatt deren inhaltlicher Gleichheit per `LambdaTerm.equals`-Vergleich zum Abbilden benutzt.
- private LambdaRoot **term**  
Der angezeigte Lambda-Term.

#### Konstruktoren

- public **LambdaTermViewController**(LambdaRoot root, boolean editable)  
Instanziert ein Objekt dieser Klasse mit dem gegebenen Lambda-Term. Fügt sich selber dem gegebenen Lambda-Term als Beobachter hinzu. Erstellt die Wurzel des View-Baumes vom Typ `LambdaNodeViewController` und fügt dann rekursiv alle Knoten des gegebenen Lambda-Terms per `ViewInsertionVisitor` dieser Wurzel hinzu.

#### Parameter

- LambdaRoot root  
Der dargestellte Lambda-Term.
- boolean editable



Gibt an, ob Änderungen am Term durch den Benutzer zugelassen sind.

### Exceptions

- `NullPointerException`  
Falls `root == null` ist.

### Methoden

- `public void replaceTerm(LambdaTerm old, LambdaTerm new)`  
Wird vom Lambda-Term aufgerufen um mitzuteilen, dass der gegebene alte Term durch den gegebenen neuen ersetzt wird. Löscht den View-Knoten zum alten Term per `ViewRemovalVisitor` aus dem View-Baum und fügt den View-Knoten des neuen Terms per `ViewInsertionVisitor` dem View-Baum hinzu. Wenn einer der beiden Terme null ist, wird der entsprechende Schritt übersprungen.

### Parameter

- `LambdaTerm old`  
Der ersetzte Term.
- `LambdaTerm new`  
Der neue Term.
- `public void setColor(LambdaValue term, Color color)`  
Wird vom Lambda-Term aufgerufen um mitzuteilen, dass die Farbe des gegebenen Terms durch die gegebene neue Farbe ersetzt wird. Setzt dabei die Farbe des View-Knotens zum gegebenen Term auf die gegebene Farbe.

### Parameter

- `LambdaValue term`  
Der veränderte Term.
- `Color color`  
Die neue Farbe des Terms.
- `protected LambdaNodeViewController getNodeView(LambdaTerm term)`  
Gibt den View-Knoten zum gegebenen Lambda-Term zurück oder null, falls zum Term kein View-Knoten existiert.

### Parameter

- `LambdaValue term`  
Der Lambda-Term.

### Rückgabe

- Der View-Knoten zum gegebenen Lambda-Term zurück oder null, falls zum Term kein View-Knoten existiert.

### Exceptions

- `NullPointerException`  
Falls `term == null` ist.

- `protected boolean hasNodeView(LambdaTerm term)`  
Gibt zurück, ob zum gegebenen Lambda-Term ein View-Knoten existiert.

### Parameter

- `LambdaValue term`  
Der Lambda-Term.

### Rückgabe

- Gibt zurück, ob zum gegebenen Lambda-Term ein View-Knoten existiert.

### Exceptions

- `NullPointerException`  
Falls `term == null` ist.

- `protected void addNodeView(LambdaNodeViewController nodeView)`  
Fügt den gegebenen View-Knoten zur `nodeViewMap` und zur `scene2d.Group` hinzu.

### Parameter

- `LambdaNodeViewController nodeView`  
Der View-Knoten, der hinzugefügt wird.

### Exceptions

- `NullPointerException`  
Falls `nodeView == null` ist.

- `protected void removeNodeView(LambdaNodeViewController nodeView)`  
Löscht den gegebenen View-Knoten aus der `nodeViewMap` und der `scene2d.Group`.

### Parameter

- `LambdaNodeViewController nodeView`  
Der View-Knoten, der gelöscht wird.

### Exceptions

- `NullPointerException`  
Falls `nodeView == null` ist.

- `public boolean isEditable()`  
Gibt zurück, ob Änderungen am Lambda-Term durch den Benutzer zugelassen sind.

### Rückgabe

- Gibt zurück, ob Änderungen am Lambda-Term durch den Benutzer zugelassen sind.

- `public void setSelection(LambdaTerm term)`  
Falls nicht `term == null` ist, erstellt einen neuen, nicht editierbaren `LambdaTermViewController` und speichert diesen in `selection`. Fügt den neuen `ViewController` der View-Hierarchie hinzu. Fügt außerdem dem neuen `ViewController` Event-Handler hinzu: Mit dem `touchUp` Event wird der ausgewählte Term an der aktuellen Zeigerposition mit Hilfe von `getParentFromPosition` und `getChildIndexFromPosition` eingefügt. Ansonsten wird der aktuell ausgewählte `ViewController` gelöscht und aus der View-Hierarchie entfernt.

### Parameter

- `LambdaTerm term`  
Der Term, zu dem ein View-Knoten erstellt wird.

- `public LambdaTermViewController getSelection()`  
Gibt den aktuell ausgewählten Knoten als `ViewController` zurück oder `null`, falls kein Knoten ausgewählt ist.

### Rückgabe

- Der aktuell ausgewählte View-Knoten oder `null`, falls kein Element ausgewählt ist.

- `public LambdaNodeViewController getParentFromPosition(float x, float y)`  
Hilfsfunktion um Elemente an der Zeigerposition einzufügen. Gibt dabei das Element zurück, welches der Elternknoten zum eingefügten Knoten wäre, falls das Element an der gegebenen Position eingefügt würde. Falls die Position über dem Wurzel-Knoten ist, wird die Wurzel zurückgegeben.

### Parameter

- `float x`  
Die X-Koordinate der Einfügeposition.

- float y  
Die Y-Koordinate der Einfügeposition.

### Rückgabe

- Der Elternknoten zur gegebenen Einfügeposition.

- public LambdaNodeViewController **getChildIndexFromPosition**(float x, float y)  
Hilfsfunktion um Elemente an der Zeigerposition einzufügen. Gibt den Kind-index zurück, den ein Knoten hätte, welcher an dieser Position in den Baum unter dem Elternknoten getParentFromPosition(x, y) eingefügt würde. Ein Kind an erster Stelle hat Index 0, ein Kind an letzter Stelle hat Index children.size().

### Parameter

- float x  
Die X-Koordinate der Einfügeposition.
- float y  
Die Y-Koordinate der Einfügeposition.

### Rückgabe

- Der Kindindex an der gegebenen Einfügeposition.

- public.gdx.math.Rectangle **getGapRectangle**(float x, float y)  
Gibt das Rechteck zurück, an welchem ein Knoten eingefügt wird, wenn der Zeiger an der gegebenen Position losgelassen wird. Die Breite des Rechtecks entspricht der Lücke zwischen zwei horizontal nebeneinanderliegenden Knoten. Dient zum Markieren der Stelle, an der ein Knoten eingefügt werden kann.

### Parameter

- float x  
Die X-Koordinate der Einfügeposition.
- float y  
Die Y-Koordinate der Einfügeposition.

### Rückgabe

- Das Einfügerechteck an der gegebenen Zeigerposition.

**3.8.2** `public abstract class LambdaNodeViewController extends scene2d.Actor`

### Beschreibung

Repräsentiert einen View-Knoten im View-Baum eines Lambda-Terms. Im Gegensatz zur Lambda-Term Datenstruktur kann ein View-Knoten beliebig viele Kindknoten haben.

### Attribute

- `private LambdaTerm linkedTerm`  
Der Lambda-Term Knoten, der durch diesen View-Knoten angezeigt wird.
- `private LambdaTermViewController viewController`  
Der ViewController, in dem dieser View-Knoten angezeigt wird.
- `private LambdaNodeViewController parent`  
Der View-Elternknoten dieses Knotens.
- `private List<LambdaNodeViewController> children`  
Die Liste der View-Kindknoten dieses Knotens.

### Konstruktoren

- `public LambdaNodeViewController(LambdaTerm linkedTerm, LambdaNodeViewC`  
Instanziert ein Objekt dieser Klasse, das den gegebenen Lambda-Term Knoten auf der gegebenen View anzeigt, mit dem gegebenen Elternknoten. Falls der ViewController durch den Spieler editierbar ist, werden diesem Actor Event-Handler hinzugefügt, die das Model entsprechend der Benutzerevents verändern: Mit dem `longPress` Event wird der angezeigte Lambda-Term mit Hilfe von `LambdaUtils.split` aus dem Baum entfernt im ViewController als aktuell ausgewählter Term gesetzt. Mit dem `tap` Event wird ein Popup zur Auswahl der Farbe für den angezeigten Lambda-Term aufgerufen.

### Parameter

- `LambdaTerm linkedTerm`  
Der Lambda-Term, der durch diesen View-Knoten angezeigt wird.
- `LambdaTermViewController viewController`  
Der ViewController, in dem dieser View-Knoten angezeigt wird.
- `LambdaNodeViewController parent`  
Der View-Elternknoten dieses Knotens.

## Exceptions

- `NullPointerException`  
Falls `linkedTerm == null` oder `view == null` ist.

## Methoden

- `public LambdaNodeViewController getParent()`  
Gibt den View-Elternknoten dieses View-Knotens zurück.

## Rückgabe

- Der View-Elternknoten dieses View-Knotens.

- `public boolean isRoot()`  
Gibt zurück, ob dieser View-Knoten eine Wurzel ist. Ein View-Knoten ist eine Wurzel, falls `parent == null` ist.

## Rückgabe

- Gibt zurück, ob dieser View-Knoten eine Wurzel ist.

- `public LambdaTerm getLinkedTerm()`  
Gibt den Lambda-Term Knoten zurück, der von diesem View-Knoten angezeigt wird.

## Rückgabe

- Gibt den Lambda-Term Knoten zurück, der von diesem View-Knoten angezeigt wird.

- `public void updateWidth()`  
Berechnet und setzt die eigene Breite mit Hilfe der Breiten seiner View-Kindknoten. Ruft rekursiv `updateWidth` des View-Elternknotens auf, falls dieser Knoten keine Wurzel ist. Im Falle einer Wurzel wird die Position mit Hilfe von `updatePosition` mit dem Ursprung als Argument aktualisiert.
- `public void updatePosition(float x, float y)`  
Setzt die eigene Position auf die gegebenen Koordinaten. Berechnet die Positionen der View-Kindknoten und ruft rekursiv deren `updatePosition` auf.

## Parameter

- `float x`  
Die neue X-Koordinate des View-Knotens.
- `float y`

Die neue Y-Koordinate des View-Knotens.

- `public abstract float getMinWidth()`  
Gibt die minimale Breite dieses View-Knotens zurück. Wird von Unterklassen überschrieben.

#### Rückgabe

- Die minimale Breite dieses View-Knotens.

- `public void insertChild(LambdaNodeViewController child, LambdaTerm rightSibling)`  
Fügt den gegebenen View-Kindknoten links neben dem Knoten, der den gegebenen Lambda-Term anzeigt, ein. Falls `rightSibling == null` ist, wird der Term an letzter Stelle in der Liste *ganzrechts* eingefügt. Teilt dem Lambda-Term ViewController über `addNodeView` mit, dass der gegebene View-Kindknoten neu hinzugefügt wurde. Ruft `updateWidth` des eigenen Knotens auf und animiert die Veränderung. Blockiert den Prozessfaden, bis die Animation beendet wurde.

#### Parameter

- `LambdaNodeViewController child`  
Der neue View-Kindknoten.
- `LambdaTerm rightSibling`  
Der Term, neben dem der neue Kindknoten links eingefügt wird.

#### Exceptions

- `NullPointerException`  
Falls `child == null` ist.

- `public void removeChild(LambdaNodeViewController child)`  
Entfernt den gegebenen View-Kindknoten.

#### Parameter

- `LambdaNodeViewController child`  
Der zu entfernende View-Kindknoten. Teilt dem Lambda-Term ViewController über `removeNodeView` mit, dass der gegebene View-Kindknoten entfernt wurde. Ruft `updateWidth` des eigenen Knotens auf und animiert die Veränderung. Blockiert den Prozessfaden, bis die Animation beendet wurde.

#### Exceptions

- `NullPointerException`  
Falls `child == null` ist.

**3.8.3** `public class LambdaAbstractionViewController extends  
LambdaNodeViewController`

### Beschreibung

Repräsentiert einen View-Knoten einer Lambda-Abstraktion im View-Baum eines Lambda-Terms.

### Attribute

- `private Color color`  
Die Farbe der Abstraktion.

### Konstruktoren

- `public LambdaAbstractionViewController(LambdaAbstraktion linkedTerm, I`  
Instanziert ein Objekt dieser Klasse, das den gegebenen Abstraktions-Knoten auf der gegebenen View anzeigt, mit dem gegebenen Elternknoten. Ruft dabei den Elternkonstruktor mit genau diesen Argumenten auf.

### Parameter

- `LambdaAbstraktion linkedTerm`  
Die Lambda-Abstraktion, die durch diesen View-Knoten angezeigt wird.
- `LambdaTermViewController viewController`  
Der ViewController, in dem dieser View-Knoten angezeigt wird.
- `LambdaNodeViewController parent`  
Der View-Elternknoten dieses Knotens.

### Exceptions

- `NullPointerException`  
Falls `linkedTerm == null` oder `view == null` ist. Wird vom Elternkonstruktor aus kontrolliert.

### Methoden

- `public void draw(Batch batch, float parentAlpha)`  
Zeichnet diesen View-Knoten auf dem gegebenen Batch mit dem gegebenen Alpha-Wert.



#### Parameter

- Batch batch  
Der Batch, auf dem gezeichnet wird.
- float parentAlpha  
Der Alpha-Wert, mit dem gezeichnet wird.

- public float **getMinWidth()**  
Gibt die minimale Breite zurück, die der View-Knoten einer Abstraktion haben kann.

#### Rückgabe

- Die minimale Breite dieses View-Knotens.

**3.8.4** public class **LambdaApplicationViewController** extends  
LambdaNodeViewController

#### Beschreibung

Repräsentiert einen View-Knoten einer Lambda-Applikation im View-Baum eines Lambda-Terms.

#### Konstruktoren

- public **LambdaApplicationViewController**(LambdaApplication linkedTerm, I  
Instanziert ein Objekt dieser Klasse, das den gegebenen Applikations-Knoten auf der gegebenen View anzeigt, mit dem gegebenen Elternknoten. Ruft dabei den Elternkonstruktor mit genau diesen Argumenten auf.

#### Parameter

- LambdaApplication linkedTerm  
Die Lambda-Applikation, die durch diesen View-Knoten angezeigt wird.
- LambdaTermViewController viewController  
Der ViewController, in dem dieser View-Knoten angezeigt wird.
- LambdaNodeViewController parent  
Der View-Elternknoten dieses Knotens.

#### Exceptions

- `NullPointerException`  
Falls `linkedTerm == null` oder `view == null` ist. Wird vom Elternkonstruktor aus kontrolliert.

## Methoden

- `public void draw(Batch batch, float parentAlpha)`  
Zeichnet diesen View-Knoten auf dem gegebenen Batch mit dem gegebenen Alpha-Wert.

### Parameter

- `Batch batch`  
Der Batch, auf dem gezeichnet wird.
- `float parentAlpha`  
Der Alpha-Wert, mit dem gezeichnet wird.

- `public float getMinWidth()`  
Gibt die minimale Breite zurück, die der View-Knoten einer Applikation haben kann.

### Rückgabe

- Die minimale Breite dieses View-Knotens.

**3.8.5** `public class LambdaVariableViewController extends LambdaNodeViewController`

## Beschreibung

Repräsentiert einen View-Knoten einer Lambda-Variable im View-Baum eines Lambda-Terms.

## Attribute

- `private Color color`  
Die Farbe der Variable.

## Konstruktoren

- `public LambdaVariableViewController(LambdaVariable linkedTerm, LambdaNodeViewController parentView)`  
Instanziert ein Objekt dieser Klasse, das den gegebenen Variablen-Knoten auf der gegebenen View anzeigt, mit dem gegebenen Elternknoten. Ruft dabei den Elternkonstruktor mit genau diesen Argumenten auf.

### Parameter

- `LambdaVariable linkedTerm`  
Die Lambda-Variable, die durch diesen View-Knoten angezeigt wird.
- `LambdaTermViewController viewController`  
Der ViewController, in dem dieser View-Knoten angezeigt wird.
- `LambdaNodeViewController parent`  
Der View-Elternknoten dieses Knotens.

### Exceptions

- `NullPointerException`  
Falls `linkedTerm == null` oder `view == null` ist. Wird vom Elternkonstruktor aus kontrolliert.

### Methoden

- `public void draw(Batch batch, float parentAlpha)`  
Zeichnet diesen View-Knoten auf dem gegebenen Batch mit dem gegebenen Alpha-Wert.

### Parameter

- `Batch batch`  
Der Batch, auf dem gezeichnet wird.
  - `float parentAlpha`  
Der Alpha-Wert, mit dem gezeichnet wird.
- `public float getMinWidth()`  
Gibt die minimale Breite zurück, die der View-Knoten einer Variable haben kann.

### Rückgabe

- Die minimale Breite dieses View-Knotens.

### 3.9 package `lambda.viewcontroller.lambdaterm.visitor`

**3.9.1** `public class ViewInsertionVisitor implements  
LambdaTermVisitor<Object>`

#### Beschreibung

Repräsentiert einen Besucher auf einer Lambda-Term Baumstruktur, welcher rekursiv View-Knoten eines gegebenen Lambda-Terms erstellt und in einen gegebenen Lambda-Term ViewController einfügt. Dabei traversiert der Besucher so lange nach oben, bis ein Elternknoten gefunden ist, zu dem ein View-Knoten im Lambda-Term ViewController existiert. Dort wird ein neuer View-Kindknoten erstellt und eingefügt.

#### Attribute

- `private LambdaTerm inserted`  
Der Lambda-Term, zu dem View-Knoten erstellt werden.
- `private LambdaTermViewController viewController`  
Der Lambda-Term ViewController, in den die erstellten View-Knoten eingefügt werden.
- `private LambdaTerm rightSibling`  
Der Knoten rechts neben dem eingefügten Knoten, falls der Elternknoten eine Applikation ist. Initialisiert mit `null`.
- `private LambdaTerm lastVisited`

#### Konstruktoren

- `public ApplicationVisitor(Color color, LambdaTerm applicant)`  
Instanziert ein Objekt dieser Klasse mit der gegebenen Variablenfarbe und dem gegebenen Argument.

#### Parameter

- `Color color`  
Die Farbe der zu ersetzenden Variablen.
- `LambdaTerm applicant`  
Das Argument der Applikation.

#### Exceptions

- `NullPointerException`  
Falls `color == null` oder `applicant == null` ist.

## Methoden

- `public void visit(LambdaRoot node)`  
Kann nie aufgerufen werden, da der besuchte Knoten keinen Elternknoten hat, von wo aus eine Applikation ausgeführt werden könnte.

### Parameter

- `LambdaRoot node`  
Die besuchte Wurzel.

- `public void visit(LambdaApplication node)`  
Besucht die gegebene Applikation und traversiert weiter zu beiden Kindknoten. Dabei werden die Kindknoten auf die Rückgabewerte beider Besuche gesetzt. Speichert als Rückgabewert den besuchten Term.

### Parameter

- `LambdaApplication node`  
Die besuchte Applikation.

- `public void visit(LambdaAbstraction node)`  
Besucht die gegebene Abstraktion und traversiert weiter zum Kindknoten. Dabei wird der Kindknoten auf den Rückgabewert des Besuchs gesetzt. Speichert als Rückgabewert den besuchten Term.

### Parameter

- `LambdaAbstraction node`  
Die besuchte Abstraktion.

- `public void visit(LambdaVariable node)`  
Besucht die gegebene Variable und speichert wenn nötig als Rückgabewert `applicant`.

### Parameter

- `LambdaVariable node`  
Die besuchte Variable.

- `public LambdaTerm getResult()`  
Gibt den Term nach der Applikation zurück.

## Rückgabe

- Der besuchte Term.

- private void **checkAlphaConversion()**  
Überprüft, ob eine Alpha-Konversion notwendig ist, falls dies noch nicht getan wurde, und führt diese wenn nötig aus. Entfernt danach das Argument der Applikation aus dem LambdaTerm.

## 3.10 package `lambda.viewcontroller.achievements`

3.10.1 public class **AchievementMenuViewController** extends  
Controller implements AchievementModelObserver

### Beschreibung

Kontrolliert und regelt die Darstellung des Erfolgsmenüs und damit der einzelnen Erfolge und die Benutzerinteraktion mit dem Menü.

### Attribute

- private Map<int, String> **renderAchievementsMap**  
Die Repräsentation aller Erfolge, die dargestellt werden. Gespeichert wird der Pfad zum Piktogramm des jeweiligen Erfolgs, abhängig davon, ob der Erfolg freigeschaltet ist oder nicht. Als Key wird der jeweilige Index verwendet, der auch die Anzeigereihenfolge bestimmt.
- private AchievementManager **achievementManager**  
Verwaltet und kontrolliert alle Erfolge.
- private scene2d.Stage **stage**  
2D-Scene-Graph, der die Hierarchie der gesamten grafischen Komponenten (Akteure mit Typ scene2d.Actor des Screens (der aktuell angezeigte Bildschirm) enthält.
- private InputMultiplexer **inputProcessor**  
Delegiert die Eingabe- Ereignisse an die geordnete Liste der InputProcessor, die die Ereignisse empfangen und weiterverarbeiten.

### Konstruktoren

- public **AchievementMenuViewController()**  
Instanziert ein Objekt dieser Klasse.

## Methoden

- `public void changedLockedState(int id)`  
Wird aufgerufen um dem Beobachter mitzuteilen, dass sich der Zustand des Erfolgs, also ob dieser freigeschaltet ist oder nicht, geändert hat.

### Parameter

- `int id`  
Die ID des Erfolgs mit dem geänderten Zustand.

### Exceptions

- `IllegalArgumentException`  
Falls `id` nicht bei den Erfolgen vorhanden ist.

- `public void update(int index)`  
Aktualisiert den entsprechenden repräsentierten Erfolg der den übergebenen Index besitzt in `renderAchievementMap`.

### Parameter

- `int index`  
Der Index des zu aktualisierenden Erfolgs.

### Exceptions

- `IllegalArgumentException`  
Falls `index` nicht in `renderAchievementMap` vorhanden ist.

- `public AchievementManager getAchievementManager()`  
Gibt eine Referenz auf `achievementManager` und damit auf die Verwaltung der Erfolge zurück.

### Rückgabe

- Der Manager der Erfolge.

- `public void dispose()`  
Wird aufgerufen, wenn der Screen all seine Ressourcen freigeben soll.
- `public void show()`  
Wird automatisch aufgerufen, wenn der Screen als aktueller Screen für das Spiel gesetzt wird.
- `public void hide()`  
Wird automatisch aufgerufen, wenn der Screen nicht mehr der aktuelle Screen

des Spiels ist.

- `public void resume()`  
Wird automatisch aufgerufen, wenn die Applikation nach einem pausierten Zustand fortgesetzt wird.
- `public void pause()`  
Wird automatisch aufgerufen, wenn die Applikation pausiert wird.
- `public void render(float delta)`  
Wird automatisch zum Zeichnen und Darstellen des Screens aufgerufen.

#### Parameter

- `float delta`  
Die Zeit in Sekunden seit dem letzten Aufruf dieser Methode.

- `public void resize(int width, int height)`  
Wird automatisch aufgerufen, wenn sich die Bildschirmgröße geändert hat.

#### Parameter

- `int width`  
Die neue Breite in Pixel.
- `int height`  
Die neue Höhe in Pixel.

### 3.11 `package lambda.model.shop`

#### 3.11.1 `public class ShopModel`

##### Beschreibung

Repräsentiert einen Shop, der es dem Benutzer ermöglicht seine erspielten Münzen gegen Belohnungen einzutauschen.

##### Attribute

- `private ShopItemTypeModel<Music> musics`  
Stellt im Shop die zu kaufenden Items des Typs Music dar.
- `private ShopItemTypeModel<Image> images`  
Stellt im Shop die zu kaufenden Items des Typs Image dar.



- `private ShopItemTypeModel<ElementUIContextFamily> elementUIContextFami`  
Stellt im Shop die zu kaufenden Items des Typs `ElementUIContextFamily` dar.

### Konstruktoren

- `public ShopModel()`  
Instanziert ein Objekt dieser Klasse.

### Methoden

- `public ShopItemTypeModel<Music> getMusics()`  
Gibt das Attribut `musics` zurück.

#### Rückgabe

– Gibt `musics` zurück.

- `public ShopItemTypeModel<Image> getImages()`  
Gibt das Attribut `images` zurück.

#### Rückgabe

– Gibt `images` zurück.

- `public ShopItemTypeModel<ElementUIContextFamily> getElementUIContextFa`  
Gibt das Attribut `elementUIContextFamilies` zurück.

#### Rückgabe

– Gibt `elementUIContextFamilies` zurück.

### 3.11.2 `public abstract class ShopItemModel`

#### Beschreibung

Repräsentiert ein allgemeines Item, welches man im Shop erwerben kann.

#### Attribute

- `private String ID`  
Gibt den eindeutigen Bezeichner des Items an.
- `private int price`  
Gibt an für wie viele Münzen dieses Item erworben werden kann.

- `private ShopModel shop`  
Hält die Referenz auf das ShopModel.
- `private ShopItemTypeModel shopItemType`  
Hält eine Referenz auf ein ShopItemTypeModel, um das aktivierte Item zu setzen.
- `private boolean purchased`  
Gibt an, ob das Item erworben wurde oder nicht und wird mit `false` initialisiert. Bei dem Wert `true` wurde das Item bereits erworben, bei `false` nicht.

### Konstruktoren

- `public ShopItemModel()`  
Instanziert ein Objekt dieser Klasse.

### Methoden

- `public String getID()`  
Gibt den Bezeichner des Items zurück.

#### Rückgabe

– Gibt ID zurück.

- `public int getPRICE()`  
Gibt zurück für welche Anzahl an Münzen dieses Item erworben werden kann.

#### Rückgabe

– Gibt PRICE zurück.

- `public int getShop()`  
Gibt das ShopModel zurück.

#### Rückgabe

– Gibt shop zurück.

- `public int getShopItemType()`  
Gibt das ShopItemTypeModel zurück.

#### Rückgabe

– Gibt shopItemType zurück.

- `public int getPurchased()`  
Gibt über den boolean zurück, ob das Item erworben wurde oder nicht (bei `true` erworben, bei `false` noch nicht erworben).
- `public void buy()`  
Vergleicht die Anzahl der Münzen des Benutzerprofils mit `price` und falls diese Anzahl größer oder gleich groß ist, wird `purchased` von `false` auf `true` gesetzt.
- `public void activate()`  
Prüft, ob das Item schon erworben wurde (`purchased == true`) und setzt dann, insofern dies geschehen ist, das Item als aktuell aktiviertes Item in der entsprechenden Kategorie.

#### Rückgabe

- Gibt `purchased` zurück.

### 3.11.3 `public class MusicModel extends ShopItemModel`

#### Beschreibung

Repräsentiert ein Musik-Item, welches im Shop erworben werden kann.

#### Attribute

- `private Music music`  
Enthält ein Musikstück, welches im Shop erworben werden kann.

#### Konstruktoren

- `public MusicModel()`  
Instanziert ein Objekt dieser Klasse.

#### Methoden

- `public Music getMusic()`  
Gibt das Musik-File zurück.

#### Rückgabe

- Gibt `music` zurück.

**3.11.4** `public class BackgroundImageModel extends ShopItemModel`

#### Beschreibung

Repräsentiert ein Hintergrundbild-Item, welches im Shop erworben werden kann.

#### Attribute

- `private Music music`  
Enthält ein Hintergrundbild, welches im Shop erworben werden kann.

#### Konstruktoren

- `public ImageModel()`  
Instanziert ein Objekt dieser Klasse.

#### Methoden

- `public Image getBackgroundImage()`  
Gibt das Image-File zurück.

#### Rückgabe

– Gibt image zurück.

**3.11.5** `public class SpriteModel extends ShopItemModel`

#### Beschreibung

Repräsentiert ein Sprite-Item, welches ein Teil eines `ElementUIContextFamily`-Objektes ist, welches im Shop erworben werden kann.

#### Attribute

- `private Sprite sprite`  
Enthält ein Sprite, welches Teil eines `ElementUIConext`-Objektes ist.

#### Konstruktoren

- `public ImageModel()`  
Instanziert ein Objekt dieser Klasse.

#### Methoden

- `public Image getSprite()`  
Gibt das Sprite-File zurück.

### Rückgabe

- Gibt sprite zurück.

### 3.11.6 public class **ShopItemTypeModel**<T>

#### Beschreibung

Repräsentiert eine ganze Kategorie von erwerbbaaren Items.

#### Typ-Parameter

- <Observer>  
Der Typ einer Kategorie.

#### Attribute

- private String **typeName**  
Gibt den Namen des Item-Typs an.
- private List<T> **items**  
Enthält alle Items vom Typ des Typ-Parameters.
- private T **activatedItem**  
Enthält das aktuell aktivierte Item vom Typ des Typ-Parameters.

#### Konstruktoren

- public **ShopItemTypeModel**()  
Instanziert ein Objekt dieser Klasse.

#### Methoden

- public String **getTypeName**()  
Gibt den Namen der Kategorie zurück.

### Rückgabe

- Gibt typeName zurück.

- public List<T> **getItems**()  
Gibt die Liste zurück, welche alle Items des gesetzten Typ-Parameters enthält.

### Rückgabe

- Gibt items zurück.

- `public T getActivatedItems()`  
Gibt das Item vom gesetzten Typ zurück, welches aktuell aktiviert ist.

#### Rückgabe

- Gibt `activatedItem` zurück.

### 3.12 `package lambda.model.level`

#### 3.12.1 `public class LevelModel`

##### Beschreibung

Repräsentiert einen Shop, der es dem Benutzer ermöglicht seine erspielten Münzen gegen Belohnungen einzutauschen.

##### Attribute

- `private int ID`  
Ist der eindeutige Identifizierer eines Levels.
- `private LambdaRoot start`  
Enthält den Startterm eines Levels.
- `private LambdaRoot goal`  
Enthält den Zielterm eines Levels.
- `private LambdaRoot hint`  
Enthält einen Lösungshinweis für ein Levels.
- `private List<TutorialMessage> tutorials`  
Enthält alle Tutorials, die für das Level benötigt werden.
- `private List<ReductionStrategy> availableRedStrats`  
Enthält alle Reduktionsstrategien, welche man für das Level verwenden darf.
- `private List<ElementType> usableElements`  
Enthält alle Elementtypen, die im Level in der Elementen-Leiste im Editormodus für das Level verfügbar sind.
- `private int difficulty`  
Beschreibt den Schwierigkeitsgrad des Levels aufsteigend von 1.

- `private Assets assets`  
Referenz zu den Assets.

## Konstrukturen

- `public LevelModel()`  
Instanziert ein Objekt dieser Klasse.

## Methoden

- `public int getID()`  
Gibt die ID des Levels zurück.

### Rückgabe

- Gibt ID zurück.

- `public LambdaRoot getStart()`  
Gibt den Starterterm des Levels zurück.

### Rückgabe

- Gibt start zurück.

- `public LambdaRoot getGoal()`  
Gibt den Zielterm des Levels zurück.

### Rückgabe

- Gibt goal zurück.

- `public LambdaRoot getHint()`  
Gibt einen Hinweis zum Level.

### Rückgabe

- Gibt hint zurück.

- `public List<TutorialMessage> getTutorials()`  
Gibt eine Liste mit allen Anleitungen zurück, welche zum Level gehören.

### Rückgabe

- Gibt tutorials zurück.

- `public List<ReductionStrategy> getAvailableRedStrats()`  
Gibt eine Liste zurück mit allen Reduktionsstrategien, welche man für dieses Level verwenden darf.

#### Rückgabe

– Gibt `availableRedStrats` zurück.

- `public List<ElementType> getUsableElements()`  
Gibt eine Liste mit Elementtypen zurück, welche zum Lösen des Levels benutzt werden dürfen.

#### Rückgabe

– Gibt `usableElements` zurück.

- `public int getDifficulty()`  
Gibt den Schwierigkeitsgrad in Form einer Zahl zurück.

#### Rückgabe

– Gibt `difficulty` zurück.

- `public Assets getAssets()`  
Gibt `assets` zurück

#### Rückgabe

– Gibt `assets` zurück.

### 3.13 package `lambda.viewcontroller.shop`

#### 3.13.1 `public class ShopViewController1 extends Controller`

##### Beschreibung

Repräsentiert einen Shop, der es dem Benutzer ermöglicht seine erspielten Münzen gegen Belohnungen einzutauschen.

##### Attribute

- `private ShopModel model`
- `private scene2d.Stage stage`
- `private InputMultiplexer InputProcessor`



- `private List<DropDownMenuViewController> items`

### Konstruktoren

- `public ShopViewController()`  
Instanziert ein Objekt dieser Klasse.

### Methoden

- `public List<DropDownMenuViewController> getItems()`  
Gibt eine Liste mit DropDownViewControllern zurück, welche für den Shop benötigt werden, um die Items anzuzeigen.

#### Rückgabe

– Gibt items zurück.

- `public void update()`  
.

**3.13.2** `public class ShopItemViewController1 extends Controller`

### Beschreibung

#### Attribute

- `private ShopItemModel model`
- `private scene2d.Stage stage`
- `private InputMultiplexer InputProcessor`

### Konstruktoren

- `public ShopItemViewController()`  
Instanziert ein Objekt dieser Klasse.

### Methoden

- `public void purchasedChanged()`

#### Parameter

– boolean purchased

- public void **activatedChanged()**  
.

#### Parameter

– boolean activated

## 4 Datenstrukturen

## 5 Dynamische Modelle

## 6 Projektplan

## 7 Glossar

## 8 Anhang