Westfälische Hochschule

Fachbereich Informatik und Kommunikation
Prof. Dr. Marcel Luis

OPR: Praktische Aufgabe „Operatoren“

Der Zweck dieser Aufgabe ist es, die wichtigen Konzepte dieses Kapitels – Hierarchie von Klassen, Überschreiben von Methoden, abstrakte Klassen und Methoden, Template-Methoden – an einem möglichst kleinen, einfachen Beispiel zu üben. Es geht in dieser Aufgabe um Operatoren. Ein Operator kann genau zwei Dinge:

- Er kann auf ein Argument angewendet werden und liefert einen Wert.
- Er kann eine textuelle Information über sich zurückgeben.

Realisieren Sie im Paket `operator` ein abstrakte Klasse `Operator` mit folgenden Methoden:

- Eine Methode `public abstract double wendeAn(double argument)`, um den Operator auf das übergebene Argument anzuwenden.
- Eine (Template-)Methode `public final String gibInfo()`, durch die der Operator eine textuelle Information über sich gibt. Diese Methode ist eine Template-Methode. Sie ist nicht abstrakt, sondern wird konkret implementiert. Sie kann wegen des Modifikators `final` in Unterklassen nicht überschrieben werden.

  Das Format der textuellen Information ist wie folgt[1]:

  name Beispielaufruf: *Konstruktorausdruck*.wendeAn(*Beispielargument*) = *Wert*

  Hierbei stehen die kursiv dargestellten Textbestandteile für folgende Informationen (s. auch Testablauf am Ende der Aufgabenbeschreibung):

  - $name$ ist der Name des Operators, z. B. `Fakultät`.
  - $Konstruktorausdruck$ ist ein Ausdruck, durch den das Objekt, das die `gibInfo`-Methode ausführt, erzeugt werden kann, z. B. `new Fakultaet()`.
  - $Beispielargument$ ist ein beliebiges Argument, auf das der Operator sinnvoll angewendet werden kann.
  - $Wert$ ist der Wert, den der Operator für das Beispielargument liefert.

[1] Sie müssen nicht den Sinn von und hinterfragen. Es gibt keinen tieferen Sinn. Es handelt sich einfach um Bestandteile des textuellen Formats, das die Methode erzeugen soll.

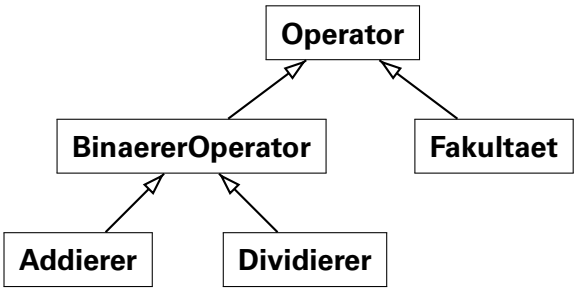
Realisieren Sie im Paket `operator` darüber hinaus eine Klasse `Fakultaet` als direkte Unterklasse von `Operator`. Sie dürfen bei der Implementierung der Methode `wendeAn` davon ausgehen, dass die Methode nur auf *sinnvolle* Argumente angewendet wird, also nur auf natürliche Zahlen einschließlich 0. Weitere Hinweise auf die Realisierung dieser Klasse ergeben sich aus dem unten dargestellten Testablauf.

Realisieren Sie im Paket `operator` ebenfalls eine Klasse `Addierer`. Addition scheint nicht zum Konzept der Klasse `Operator` zu passen, denn die Methode `wendeAn` besitzt nur *einen* Parameter, wogegen man bei Addition an die Verknüpfung von *zwei* Zahlen denkt. Keine Sorge! Es passt, wenn man den Operator zur Addition bei der Erzeugung mit einem Parameter versieht. Durch `new Addierer(2.5)` erzeugt man einen Addierer, der bei `wendeAn` zum übergebenen Argument 2.5 addiert. Beispiel: `new Addierer(2.5).wendeAn(1.3)` liefert den Wert 3.8. Durch `new Addierer(2.5)` erzeugt man somit einen Addierer, der bei `wendeAn` immer 2.5 hinzuaddiert. Realisieren Sie in der Klasse `Addierer` einen entsprechenden Konstruktor.

Da man dieses Prinzip natürlich auch auf andere binäre Operatoren (Multiplikation, Division, ...) übertragen kann, realisieren Sie im Paket `operator` außerdem eine abstrakte Klasse `BinaererOperator` als direkte Unterklasse von `Operator`.
Die Klasse `Addierer` ist direkte Unterklasse von `BinaererOperator`.

Realisieren Sie im Paket `operator` ebenfalls eine Klasse `Dividierer`.

Zusammenfassend bilden alle beschriebenen Klassen folgende Klassenhierarchie:



Realisieren Sie schließlich im gleichen Paket eine Klasse `OperatorTest` mit einer `main`-Methode, die mindestens folgende Anweisungen enthält:

```
Operator op = new Addierer(5);
System.out.println(op.gibInfo());

op = new Dividierer(5);
System.out.println(op.gibInfo());

op = new Fakultaet();
System.out.println(op.gibInfo());
```

Dabei wird folgende Ausgabe erwartet:

```
<p>Addition</p><p>Beispielaufruf: new Addierer(5.0).wendeAn(10.0) = 15.0</p>
<p>Division</p><p>Beispielaufruf: new Dividierer(5.0).wendeAn(3.0) = 0.6</p>
<p>Fakultät</p><p>Beispielaufruf: new Fakultaet().wendeAn(10.0) = 3628800.0</p>
```

Hinweise

- Sie dürfen in den einzelnen Klassen weitere Methode realisieren. Ich könnte auch sagen: Sie *müssen* es sogar. Welche Methoden dies sind, finden Sie schnell heraus, wenn Sie das Konzept von Template-Methoden verstanden haben. Eine Template-Methode realisiert ein grundsätzliches Verhalten, zu dem jedoch noch Details fehlen, die in den Unterklassen festgelegt werden.
- Verwenden Sie nur den Vorlesungsstoff bis einschließlich zum Kapitel „Klassenhierarchie und Polymorphie".
- Erstellen Sie für die Prüfung durch ARCTERN im Abgabebereich der Aufgabe ein Verzeichnis für das Java-Paket und laden dorthin Ihre Klassen hoch.