KLAUSUR

Modul:	FPA	**Semester:**	SS 2012	**Name, Vorname:**	
Dauer:	90 min	**Datum:**	05.07.2012	**Matrikelnummer:**	

Hinweis: Die Teilnahme an der Klausur ist nur bei Abschluss der Übung mit 4.0 oder besser zulässig. Sollte die Übung nicht bestanden sein, gilt die Klausur auch als „nicht bestanden".

Vor Beginn der Bearbeitungszeit, werden die Aufgaben erläutert. Die Zeitmessung beginnt erst danach. Das kurzzeitige Verlassen des Raumes ist nur mit An- und Abmeldung unter Angabe des Grundes zulässig.

Es sind keine Lehrunterlagen zugelassen. Während der Klausur darf nichts aus den mitgebrachten Taschen/Rucksäcken raus genommen werden.

Übungen 64,0%
Klausur 80,0%

Jeder Austausch mit anderen Personen wird als Täuschungsversuch gewertet und führt dazu, dass die Arbeiten aller beteiligten Studierenden als "nicht bestanden" gewertet werden.

72,0%

Individuelle Fragen werden beantworten, sofern es die Frage und die Ruhe im Saal erlaubt. Die letzten 15 Minuten werden keine Fragen mehr beantwortet.

2,7

Jedes über die Aufgabenblätter hinausgehende Blatt ist zu beschriften (Name, Vorname, Matrikelnummer, Seitennummer). Nicht beschriftete Zettel werden nicht gewertet.

Die Rückgabe der Klausur findet in der kommenden Woche zur Vorlesungszeit statt.

Bei Aufgaben, die Design Patterns behandeln, wird – sofern nicht anders angegeben - immer Bezug zu der Gang-of-Four Ausführung, die wir auch in der SU behandelt haben, genommen.

VIEL ERFOLG!

72 von 90

2.0

Bearbeiten Sie die folgenden Single – Choice Aufgaben. Es ist jeweils nur eine der möglichen Antworten korrekt. Falsche Antworten führen zu keinem Punktabzug. Kennzeichnen Sie Ihre Antwort eindeutig, in dem Sie ein Kreuz in dem Kästchen vor der Antwort zeichnen. Falls Sie ihre Antwort korrigieren wollen, machen Sie dies unmissverständlich deutlich. Unklare Antworten werden nicht gewertet!
(24 von 90 Punkten)

21/24

1. **Bei der Entwicklung von Software - Systemen streben wir nach ...**
   - ☐ niedriger Kohäsion und hoher Kopplung.
   - ☐ hoher Kohäsion und hoher Kopplung.
   - ☐ niedriger Kohäsion und niedriger Kopplung.
   - ☑ hoher Kohäsion und niedriger Kopplung.
2. **Welches der folgenden Design Patterns erlaubt es dem Klienten in einer baumartigen Struktur eine Liste von Objekten und individuelle Objekte gleich zu behandeln?**
   - ☐ Command Pattern
   - ☐ Decorator Pattern
   - ☐ Strategy Pattern
   - ☑ Composite Pattern
3. **Welche der folgenden Aussagen trifft zu?**
   - ☐ Vererbung erlaubt ein hohes Maß an Dynamik zur Laufzeit.
   - ☐ Lange Methodenrümpfe sind kleinen Methodenrümpfen vorzuziehen.
   - ☑ Eine Klasse sollte möglichst nur eine Zuständigkeit besitzen.
   - ☐ Attribute einer Klasse sollten stets "public" sein.
4. **Welche der folgenden Aussagen trifft zu?**
   - ☐ Eine Klasse sollte möglichst viele Aufgaben erledigen.
   - ☐ Hohe Kopplung erhöht die Wartbarkeit eines Softwaresystems.
   - ☑ Programmiere stets gegen Schnittstellen, nicht gegen Implementationen.
   - ☐ Niedrige Kohäsion erhöht die Erweiterbarkeit eines Softwaresystems.
5. **Die Gesamtheit eines Software Design Patterns besteht aus ...**
   - ☐ Name, Problemstellung, Vorgabe der Programmiersprache, Lösung und Konsequenzen.
   - ☐ Index, Problemstellung und Lösung.
   - ☑ Name, Problemstellung, Lösung und Konsequenzen.
   - ☐ Problemstellung und Lösung.

6. **Das State Pattern definiert die Akteure ...**
   - ☐ ConcreteStateFactory, Client und StateFactory
   - ☑ State, Context und ConcreteState.
   - ☐ StateInvoker, MakroState und StateUndo.

7. **"Kapsle einen Befehl als ein Objekt. Dies ermöglicht es, Klienten mit verschiedenen Anfragen zu parametrisieren, Operationen in eine Schlange zu stellen, ein Logbuch zu führen und Operationen rückgängig zu machen" gehört zu**
   - ☑ Command Pattern
   - ☐ Composite
   - ☐ Adapter Pattern

8. **Welches Design Pattern eignet sich am ehesten, wenn man eine existierende Klasse verwenden möchte, deren Schnittstelle aber nicht der benötigten Schnittstelle entspricht?**
   - ☐ Strategy
   - ☑ Adapter
   - ☐ Decorator
   - ☐ Command

9. **Welche der Aussagen trifft am ehesten zu? Ein eng gekoppeltes System ...**
   - ☐ kann die Wartbarkeit verringern..
   - ☐ ist immer performanter als ein lose gekoppeltes System.
   - ☐ führt immer zu schlecht lesbarem Code.
   - ☐ kann leicht modularisiert werden.

10. **Ein Decorator kann ...**
    - ☑ zu einzelnen Objekten einer Klasse Funktionalität hinzufügen und wieder entfernen.
    - ☐ zu allen Objekten einer Klasse Funktionalität hinzufügen, aber nicht mehr entfernen.

11. **Wenn eine Gruppen-SMS (eine SMS an mehrere Mobiltelefone) <u>asynchron</u> verschickt wird, wird beim Versenden der einzelnen SMS-Nachrichten ...**
    - ☑ eine Empfangsbestätigung der Mobiltelefone nicht abgewartet und somit die SMS an die anderen Mobiltelefone sofort verschickt.
    - ☐ immer eine Empfangsbestätigung abgewartet und erst danach die SMS an die anderen Mobiltelefone verschickt.

12. Das Decorator Design Pattern verwendet man am ehesten, wenn man ...

- ☐ das Verhalten meines Objekts von seinem Zustand abhängig ist, und es dieses Verhalten zur Laufzeit auf Basis dieses Zustands ändern können soll.
- ☑ ein Objekt dynamisch um Zuständigkeiten erweitern möchte.
- ☐ in einer baumartigen Struktur eine Liste von Objekten und individuelle Objekte gleich zu behandeln möchte.

13. Ein Modul kann charakterisiert werden durch ...

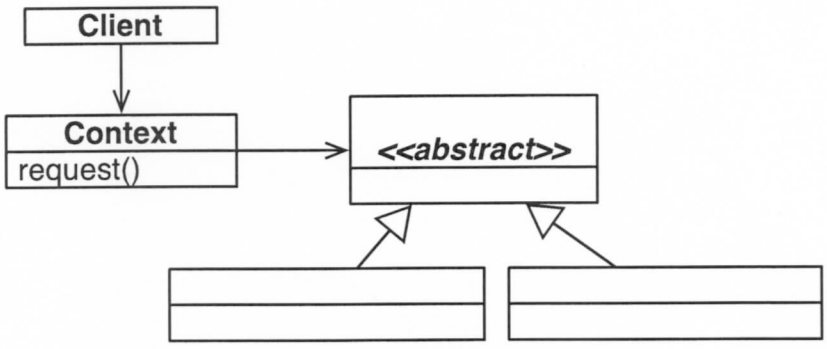
- ☑ eine Menge von logisch gekapselten Implementierungsklassen mit einer öffentlichen API auf Basis einer Teilmenge der Implementierungsklassen.
- ☐ eine Schablone zum Erstellen von Objekten und deren potenzielle Kommunikationsmöglichkeiten.
- ☐ eine Einheit zum modellieren von Objekten der realen Welt.

14. Beim Composite Pattern gibt es die zwei Implementierungsvarianten „Sicherheit" und „Transparenz". Bei der Transparenz-Variante ...

- ☐ muss der Client Typprüfungen und Castings durchführen, da er nicht alle Objekte gleich behandeln kann.
- ☐ sind die Methoden zum Hinzufügen und Entfernen von Komponenten (Component) in der Knoten (Compositum) Klasse.
- ☐ kann der Client alle Objekte im Baum gleich behandeln.

15. Das folgende Klassendiagramm ließe sich zu folgenden Design Patterns ergänzen:

- ☐ State und Adapter
- ☑ State und Strategy
- ☐ Singleton und Observer
- ☐ Dectorator und Observer



16. Das Command Design Pattern eignet sich um Objekte zu Baumstrukturen zusammenzusetzen und es dem Klienten zu ermöglichen, sowohl einzelne Objekte als auch Kompositionen von Objekten einheitlich zu behandeln.

- ☐ trifft zu
- ☑ trifft nicht zu

17. Ein Unterschied zwischen dem Adapter und dem Decorator ist, dass der Adapter Objekte der selben Schnittstelle kapselt, wohingegen der Decorator Objekte einer anderen/inkompatiblen Schnittstelle kapselt.

- ☐ trifft zu
- ☑ trifft nicht zu

18. Eines der Maßnahmen zum Überprüfen des Programmierstils ist ein Review des Quellcodes.

- ☑ trifft zu
- ☐ trifft nicht zu

19. Beim Strategy Pattern hat der Client eine Referenz auf die konkrete Strategie

- ☑ trifft zu
- ☐ trifft nicht zu

20. Mit dem Null-Object Pattern kann man ~~kann~~ im Command Pattern mehrere Kommandos zusammenfassen

- ☐ trifft zu
- ☑ trifft nicht zu

21. Software Design Patterns tragen auf jeden Fall zum besseren Verständnis eines Softwaresystems bei.

- ☐ trifft zu
- ☑ trifft nicht zu

22. Software Design Patterns tragen auf jeden Fall zum besseren Verständnis eines Softwaresystems bei.

- ☐ trifft zu
- ☑ trifft nicht zu

23. Bei der Datenübermittlungsart "Pull - Data" überträgt der Aufrufer keine Nutzdaten.

- ☐ trifft zu
- ☐ trifft nicht zu

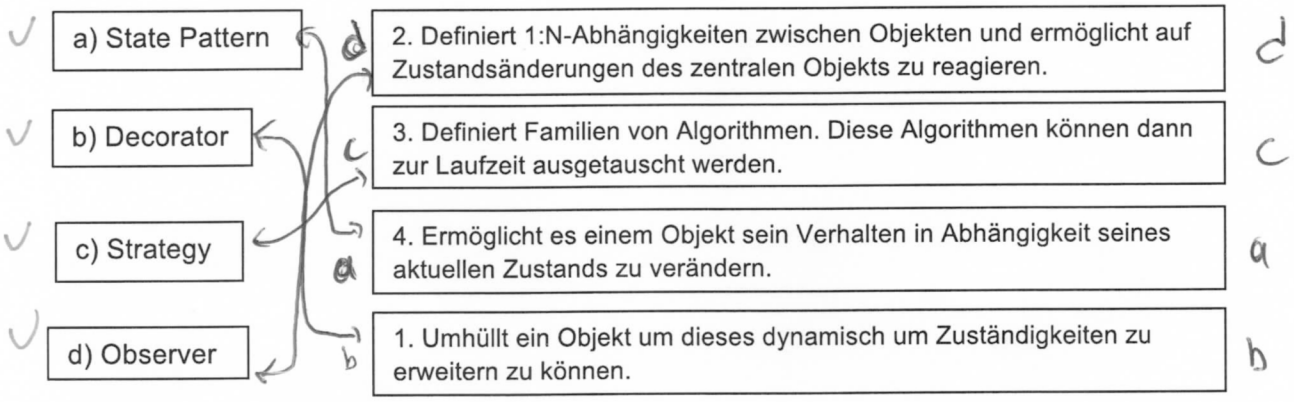
24. Patterns gibt es ausschließlich im Bereich der Softwareentwicklung.

- ☐ trifft zu
- ☑ trifft nicht zu

Patterns Ordnen

Ordnen Sie die folgenden Patterns einem Aussagesatz zu, indem Sie einen Pfeil vom Pattern zu dem betreffenden Aussagesatz zeichnen. Wahlweise können Sie auch die Kombination der Kennungen notieren.
(6 von 90 Punkten)

6/6



Erkennen von Design Patterns

Gegeben ist folgender Programmcode: Decorator Pattern

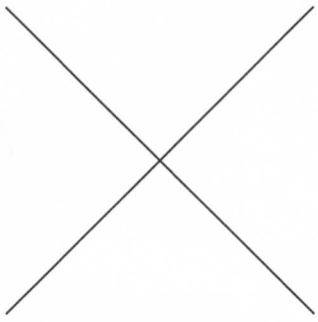
```
public class TextField extends Element {
  @Override
  public void draw() {
    System.out.println("drawing text field");
  }
}

public abstract class ExtendedElement extends Element {
  protected final Element delegate;

  public ExtendedElement (Element delegate) {
    this.delegate = delegate;
  }
}

public class Border extends ExtendedElement {
  public Border(Element delegate) {
    super(delegate);
  }
  @Override
  public void draw() {
    System.out.println("drawing border");
    delegate.draw();
  }
}
```

Bearbeiten Sie folgende Aufgaben:

1. Analysieren Sie den Programmcode und geben Sie an, um welches Patterns es sich dabei handelt. Schreiben Sie eine kurze Begründung zu Ihrer Antwort.
2. Vervollständigen Sie fehlende Klassen und Schnittstellen. Programmieren Sie, falls nötig, entsprechende Methoden aus.
3. Entwerfen Sie ein kleines Main-Programm, welches die Benutzung aller Klassen verdeutlicht.

(30 von 90 Punkten)

30/30



Designen einer Anwendung

Es soll eine Software zur Verwaltung von BAföG-Anträgen erstellt werden. Der BAföG-Verwaltung können Anträge eingereicht werden. Wenn ein Antrag eingereicht wird, so kann er geprüft werden. Wenn beim Prüfen das Jahreseinkommen der Eltern weniger oder gleich 20.000 € ist, wird der Antrag genehmigt. Wenn beim Prüfen dabei das Jahreseinkommen der Eltern höher als 20.000 € ist, wird der Antrag abgelehnt. Wurde ein Antrag abgelehnt, so kann dieser erneut eingereicht werden.

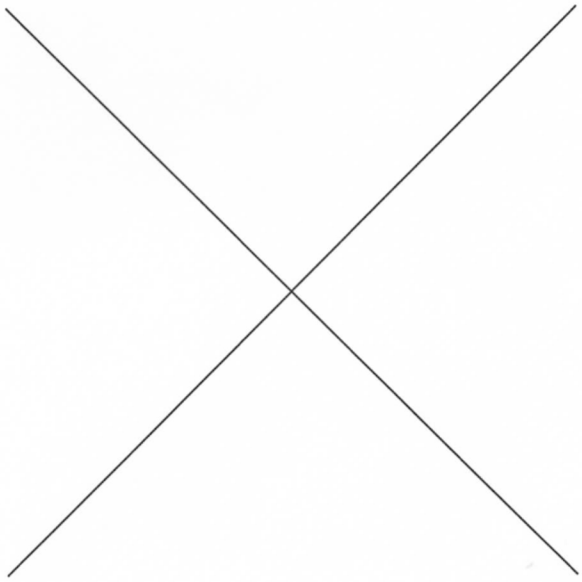
Weiterhin soll die BAföG-Verwaltung die Anträge auf verschiedene Arten sortieren können. Zunächst soll es möglich sein nach Datum oder nach Antragsstatus zu sortieren.

Bearbeiten Sie folgende Aufgaben:

1. Entwerfen Sie unter Verwendung geeigneter Patterns ein Klassendiagramm, welches die oberen Anforderungen erfüllt. Geben Sie die notwendigen Methoden und Attribute an, die für das Verständnis wichtig sind. Falls es zur Erklärung Ihres Designs dienlich ist, können Sie weitere Diagramme erstellen.
2. Beschreiben Sie die zentralen Aspekte Ihres Designs und nennen Sie die verwendeten Patterns. Begründen Sie, warum Sie diese Patterns verwendet haben. Verwenden Sie bei Bedarf ein weiteres Blatt.

(30 von 90 Punkten)

15/30



= Erkennen von Design Pattern =

– 05.07.2012 –

(1) Decorator Pattern handelt es sich um in dem gegebenen Programm.

Ein Objekt erzeugt von dem Typ von dem er geerbt hat. Und delegiert er dieser Objekte.

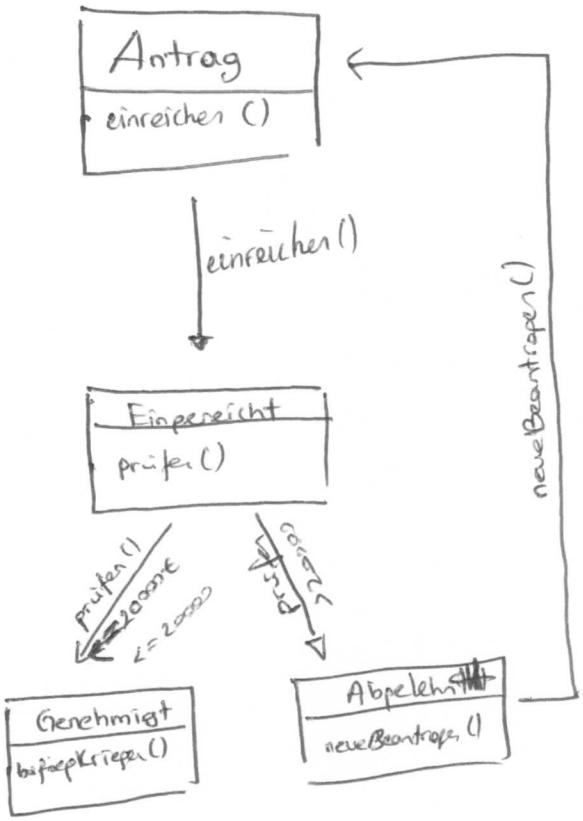
"Element" ✓ "delegate" :) = ~~delegation~~

(2)
```
public class Element {
    public Element() { }
    public void draw() {...}
}
```

(3)
```
public static void main(String[] args) {

    Element border = new Border(new TextField());

    border.draw();
}
```

~~Ausgabe~~ wäre "drawing border" oder "drawing text field"

Designen Eines Anwendung



Ich habe hier das State Pattern eingesetzt.

Weil je nach Zustand ~~ändert~~ sich die Verhalten des Programms ✓

Sortierung? -1,5