Prüfung
Software Engineering (SE)

Prof. Dr.-Ing. Christian Heller <christian.heller@ba-leipzig.de>

Student	
Vor- und Nachname	Ihre Daten werden von der Klausuraufsichtsperson schriftlich auf der Anwesenheitsliste festgehalten.
Matrikelnummer	
Studienrichtung und Jahr	CS 2018-2
Anmeldename	Das Login "klaus..." muss unbedingt schriftlich auf der Anwesenheitsliste festgehalten werden, da sonst keine Zuordnung des Logins zu Ihrem Namen und damit keine Korrektur der Klausur möglich ist!

Prüfung	
Datum	September 2020
Dauer [min]	100
Hilfsmittel	Dokumentation im lokalen Netzwerk (Intranet) sowie Recherche im Internet. NICHT gestattet: * Kommunikation in jeglicher Form * Anmeldung via SSH auf dem Rechner "fileserv" * Anmeldung mit Klausur-Login nach Ende der Prüfung Dies kann leicht geprüft werden (last cs16*, Server-Log-Dateien). Bitte unterlassen Sie also Täuschungsversuche in Ihrem eigenen Interesse.
Bemerkungen	Hinterlegen Sie alle Programme und Antworten in elektronischer Form! Es wird kein Papier angenommen. Möchten Sie Lösungen erläutern, so nutzen Sie Quelltext-Kommentare oder legen eine Text-Datei an. Speichern Sie sämtliche Daten im HOME-Verzeichnis des Nutzers, d. h. unter Windows auf Laufwerk H:\ (NICHT auf C:\ oder "Eigene Dateien")! Idealerweise legen Sie dort ein Unterverzeichnis namens "klausur" an. Lesen Sie die Aufgaben komplett durch, bevor Sie sie lösen! Die Reihenfolge der Lösung ist Ihnen überlassen. Probieren Sie immer, eine Aufgabe zu lösen, da auch auf richtige Teile nicht vollständiger Lösungen Punkte vergeben werden! Falls vom Prinzip her richtig, so werden auch alternative Lösungen akzeptiert. Sie dürfen beliebig viele Bildschirmausgaben von Werten in den Quelltext einbauen, um ein Programm besser nachvollziehen zu können. Bitte duplizieren Sie Ihre Quelltextdateien (workspace) NICHT, da beim Korrigieren dann beide durchsucht werden müssen, was sinnlosen Aufwand verursacht. Diese Aufgabenstellung in Papierform können Sie nach dem Ende der Klausur behalten.

Bewertung						
Aufgabe	1	2	3	4	5	Summe
Punkte	20	30	20	20	10	100

Im Rahmen der Klausur sind Aufgaben mit Bezug zur Unified Modeling Language (UML) zu lösen. Insofern modelliert werden soll, ist das BOUML-Werkzeug zu verwenden.

Aufgabe 1: Klassendiagramm „Postamt“ [20]

Zweck: Modellieren eines Sachverhaltes mit Strukturelementen und Beziehungen.

Hinweis: Soweit wie möglich sollen die in der Beschreibung benutzten Begriffe verwendet werden. Auch Kardinalitäten sind, wo zutreffend, anzugeben.

a) Ein Brief ist ein spezielles Postgut. Es gibt noch weitere Postgüter: Paket und Päckchen. [2]

b) Im Zusammenhang mit einem Postgut spielt genau ein Depot die Rolle des Abgabeorts. [2]

c) Es kann zu einem Postgut prinzipiell beliebig viele Verteilungsorte geben, die ebenfalls Depots sind. [2]

d) Ein Postgut hat immer einen Zielort, der gleichfalls ein Depot ist. [2]

e) Ein Postgut steht mit zwei Adressen in Verbindung. Eine notwendige Adresse beschreibt den Empfänger. Eine weitere Adresse kann (optional) der Absender sein. [2]

f) Die Ausmaße eines Postgutes werden durch Zahlen charakterisiert: Jedes Postgut hat genau eine Länge und genau eine Breite; zusätzlich kann es eine Höhe aufweisen. [2]

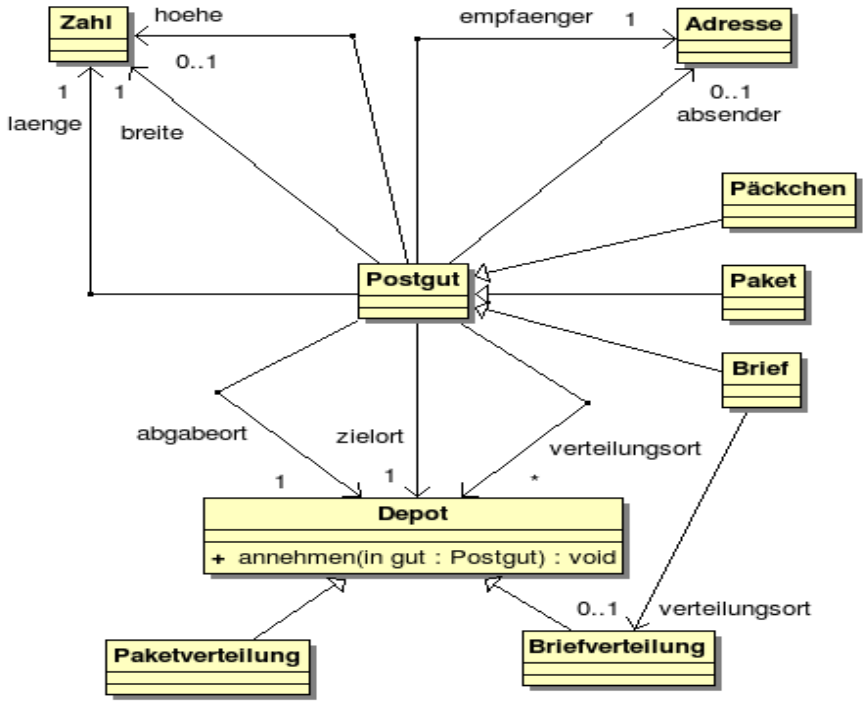
g) Ein Depot bietet die Dienstleistung des Annehmens (von Postgütern) an. [2]

h) Paketverteilung und Briefverteilung sind spezielle Depots, die bei den unterschiedlichen Postgütern eine Rolle spielen können. [2]

i) Der im Zusammenhang mit Briefen verwendete Verteilungsort ist die Briefverteilung. [2]

j) Passen Sie die Diagrammeinstellungen so an, dass alle Details von Klassenmitgliedern angezeigt werden (u.A. Sichtbarkeit; Name; Name, Typ und Richtung von Parametern; Typ des Rückgabewertes)! [2]

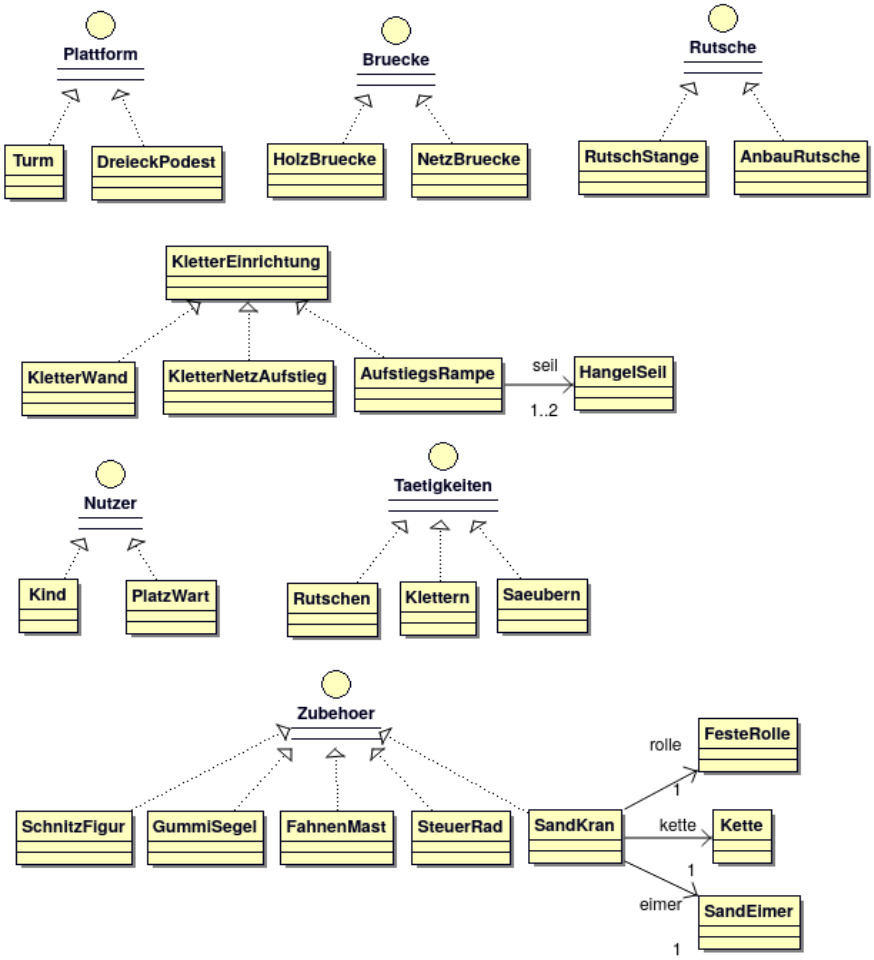
Lösung:



Aufgabe 2: Komponentendiagramm „Spielkombination“ [30]

Zweck: Konzipieren der Komponenten eines Holzschiffes für einen Spielplatz.

Zur Bearbeitung der Aufgaben sei ein Klassendiagramm gegeben.



a) Erstellen Sie ein Komponentendiagramm und fügen Sie eine Komponente namens SpielKombination ein! [2]

b) Erstellen Sie vier in SpielKombination eingebettete Komponenten: Bug, DecksAufbau, Heck, Kran! [2]

Hinweis: In der Baumstruktur linkerhand im BOUML-Werkzeug zu realisieren.

c) Zeichnen Sie sie im Diagramm ein und platzieren Sie sie als eingebettete Komponenten! [2]

d) Zeichnen Sie eine weitere Komponente namens Nutzer außerhalb der Komponente SpielKombination ein! [2]

e) Ändern Sie die Diagrammeigenschaften so, dass Komponenten in der Farbe „very light orange“ dargestellt werden! [2]

Hinweis: Erstellen Sie die im weiteren Verlauf nötigen Klassen (NICHT alle im obigen Klassendiagramm gezeigten) Schritt für Schritt! Fassen Sie sie als „provided interface“ in Komponenten zusammen, wie folgt:

f) Komponente Bug: KletterWand, Turm [2]

g) Komponente DecksAufbau: Bruecke, KletterNetzAufstieg, AufstiegsRampe [2]

h) Komponente Heck: AnbauRutsche, SteuerRad [2]

i) Komponente Kran: Kette [2]

j) Komponente SpielKombination: Klettern, Rutschen, Saeubern [2]

k) Verdeutlichen Sie über eine Schnittstelle, dass der DecksAufbau mit sowohl Bug als auch Heck über eine Bruecke verbunden ist! [2]

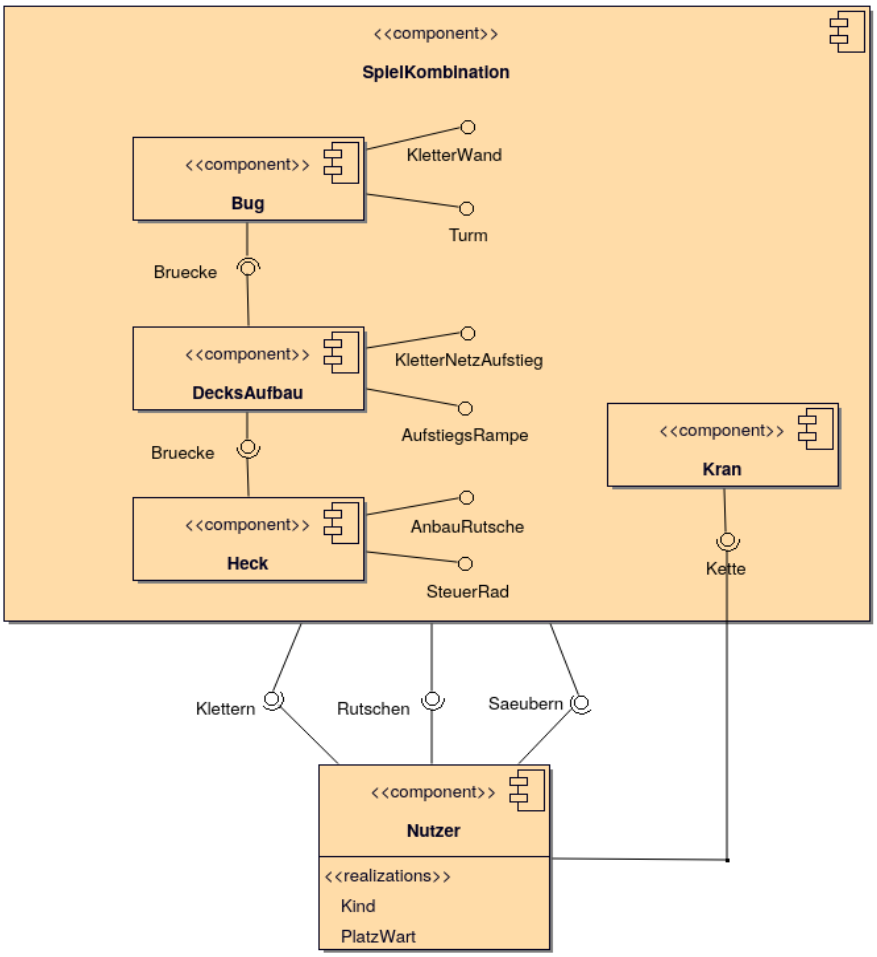
l) Verbinden Sie die Komponente Nutzer mit allen drei durch die Komponente SpielKombination angebotenen Schnittstellen! [2]

m) Verdeutlichen Sie über eine Schnittstelle, dass die Komponente Nutzer den Kran via Kette nutzen kann! [2]

n) Tragen Sie „Kind“ und „PlatzWart“ als „Realizing Classes“ in der Komponente „Nutzer“ ein! [2]

o) Konfigurieren Sie die Diagrammeinstellungen so, dass „realizing classes“ angezeigt werden! [2]

Lösung:



Aufgabe 3: Generative Programmierung „3-Tier“ [20]

Zweck: Generieren von Quelltext sowie einer Dokumentation.

a) Erstellen Sie ein Verteilungsdiagramm, um Quelltext automatisch generieren zu können! [2]

b) Vervollkommnen Sie das Diagramm durch eine physische Architektur bestehend aus mindestens den folgenden drei Knotentypen: Client, ApplicationServer, DBMS! [2]

c) Ordnen Sie dem ApplicationServer einen Artefakt, welcher eine Quelltextdatei repräsentiert, über eine lose Ankerbeziehung zu! [2]

d) Betten Sie einen WebServer-Prozess in den ApplicationServer ein! [2]

e) Bezeichnen Sie die Netzwerkverbindungen mit passenden Begriffen Ihrer Wahl! [2]

f) Erstellen Sie per separatem Klassendiagramm zwei Klassen, die durch Vererbung miteinander verbunden sind! Geben Sie einer der Klassen ein Attribut und eine Methode! [2]

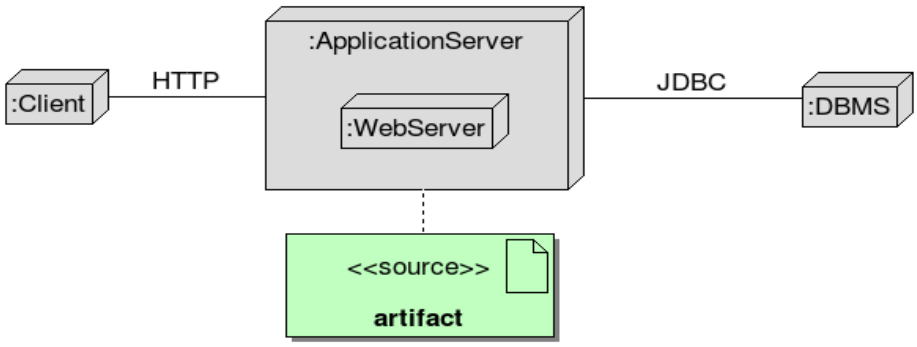
g) Ordnen Sie dem im Verteilungsdiagramm dargestellten Artefakt beide Klassen zu! [2]

h) Generieren Sie den zugehörigen Java-Quelltext in ein Unterverzeichnis namens „src/“! [2]

i) Hinterlegen Sie für das Diagramm eine Beschreibung mit einem kurzen Text. [2]

j) Generieren Sie eine HTML-Dokumentation in ein Unterverzeichnis namens „doc/“! [2]

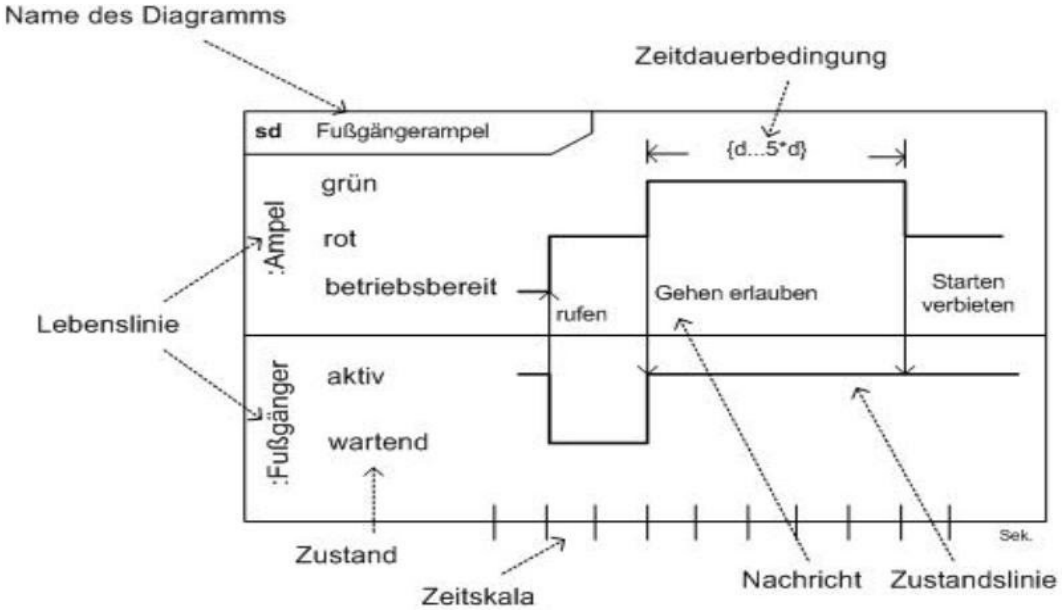
Lösung:



Aufgabe 4: Zustandsmaschinendiagramm „Fußgängerampel“ [20]

Zweck: Modellieren der Situation von Fußgänger und Ampel.

Dazu sei ein Zeitverlaufsdiagramm gegeben, dessen Objekte und Zustände in einem Zustandsmaschinendiagramm abgebildet werden sollen.



a) Erstellen Sie ein Zustandsmaschinendiagramm für eine Zustandsmaschine namens „Fuszgaengerampel“ und stellen Sie sie dar! [2]

b) Stellen Sie die beiden im Zeitverlaufsdiagramm gezeigten Objekte mit gleichem Namen als Unterzustandsmaschinen von „Fuszgaengerampel“ im Diagramm dar! [2]

c) Fügen Sie die Zustände von Ampel und Fußgänger auf passende Weise ein! [2]

d) Ergänzen Sie die Zustandsübergänge für die Ampel! [2]

e) Ergänzen Sie die Zustandsübergänge für den Fußgänger! [2]

f) Geben Sie beiden Unterzustandsmaschinen einen Pseudostartzustand! Vom Zustand „aktiv“ des „Fuszgaengers“ soll außerdem eine Transition auf einen Pseudoendzustand führen! [2]

g) Passen Sie die Diagrammeinstellungen so an, dass Definitionen von Transitionen angezeigt werden! [2]

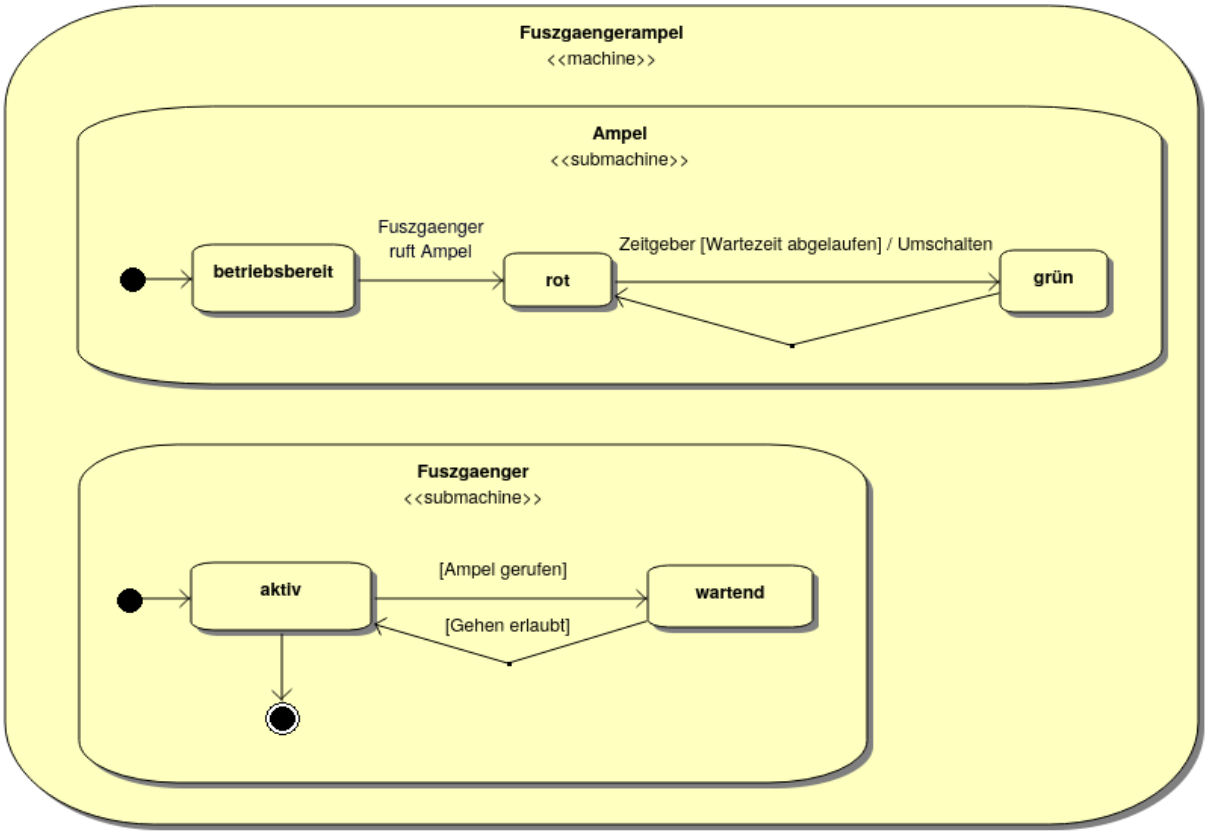
h) Schreiben Sie als auslösendes Ereignis „Fuszgaenger ruft Ampel“ an die Transition zwischen „betriebsbereit“ und „rot“! [2]

Hinweis: Textfeld „trigger“

i) Ergänzen Sie die Transition zwischen „rot“ und „grün“ durch das Ereignis „Zeitgeber“, die Bedingung „Wartezeit abgelaufen“ und die Aktion „Umschalten“! [2]

j) Ergänzen Sie schließlich die Transitionen zwischen „aktiv“ und „wartend“ durch die Bedingungen „Ampel gerufen“ und „Gehen erlaubt“! [2]

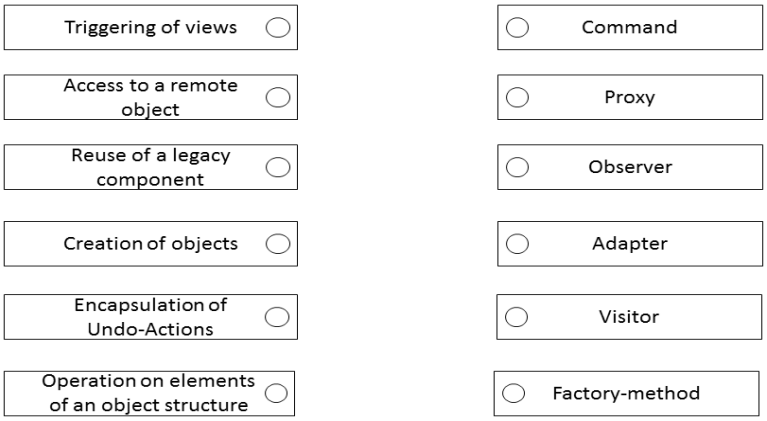
Lösung:


Fuszgaengerampel
<<machine>>
Ampel
<<submachine>>
betriebsbereit
Fuszgaenger
ruft Ampel
rot
Zeitgeber [Wartezeit abgelaufen] / Umschalten
grün
Fuszgaenger
<<submachine>>
aktiv
[Ampel gerufen]
[Gehen erlaubt]
wartend

Aufgabe 5: Software-Muster „Zuordnung“ [10 + 2]

Zweck: Zuordnen von Entwurfsmuster und Anwendungszweck.

Die folgende Grafik zeigt Einsatzmöglichkeiten linker- und entsprechende Muster rechterhand. Ordnen Sie je eine Anwendungsmöglichkeit je einem Muster zu! Geben Sie Ihre Lösung in einer einfachen Textdatei an!



a) Triggering of views [2]

b) Access to a remote object [2]

c) Reuse of a legacy component [2]

d) Creation of objects [2]

e) Encapsulation of Undo-Actions [2]

f) Zusatzaufgabe: Operation on elements of an object structure [2]

Lösung:

a) Triggering of views --> Observer

b) Access to a remote object --> Proxy

c) Reuse of a legacy component --> Adapter

d) Creation of objects --> Factory-method

e) Encapsulation of Undo-Actions --> Command

f) Operation on elements of an object structure --> Visitor

Viel Erfolg!