

Aufgabe 1

(4 Punkte)

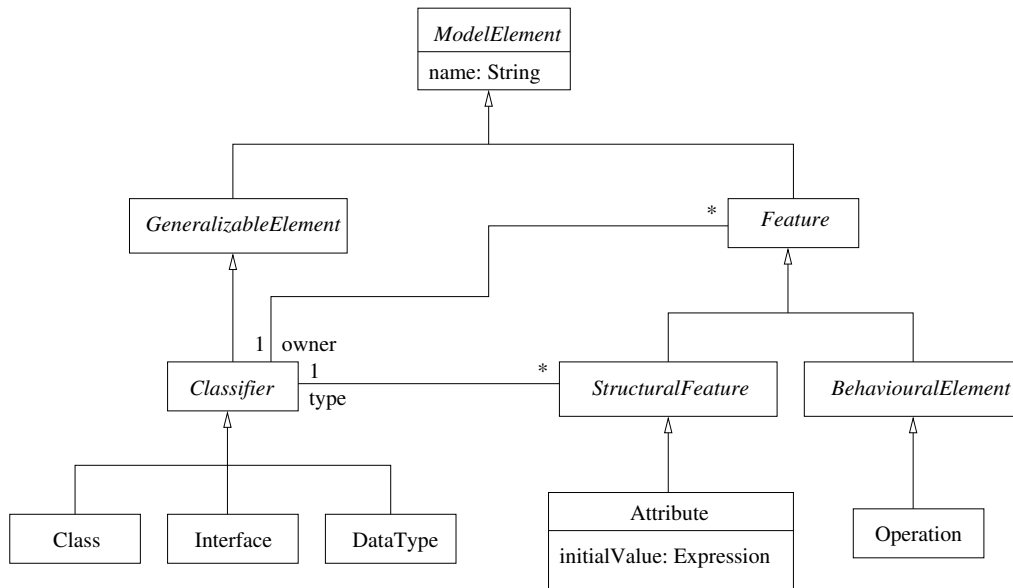
- (a) Beschreiben Sie das Prozessmodell der objektorientierten Analyse durch ein Aktivitätsdiagramm. (2 Punkte)

- (b) Nennen Sie die vier Säulen des Konfigurationsmanagements. (2 Punkte)

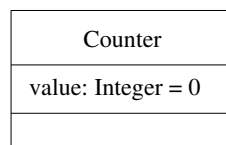
Aufgabe 2

(4 Punkte)

Das folgende Klassendiagramm zeigt einen vereinfachten Ausschnitt des UML-Metamodells.



Wie kann die folgende Klasse durch ein Instanzendiagramm des Metamodells dargestellt werden?

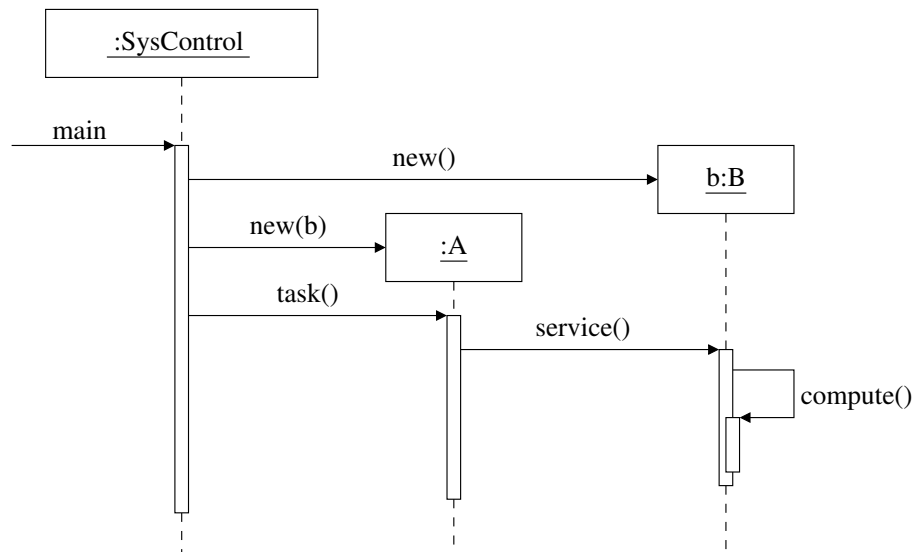


Aufgabe 3

(12 Punkte)

Gegeben sind die beiden auf der Rückseite dieses Blatts angegebenen Subsysteme und eine Klasse mit der main-Methode zur Systemkontrolle. Die beiden folgenden Anforderungen werden an das System gestellt:

1. Zur Laufzeit sollen die folgenden Interaktionen stattfinden:



2. Zur Programmierzeit soll die Verbindung zwischen den beiden Subsystemen über ein Interface hergestellt werden, so dass keine Abhängigkeiten zwischen den Klassen *A* und *B* bestehen.
- (a) Zeichnen Sie in das Diagramm auf der Rückseite dieses Blatts alle Modellelemente ein (Interface, Operationen, Konstruktoren, Assoziationen, Realisierungsbeziehungen), die benötigt werden um die oben beschriebenen Anforderungen zu erfüllen. Die Operationen und Konstruktoren sollen mit den notwendigen Sichtbarkeitsmarkierungen versehen sein und ggf. mit Parametern und Parametertypen. Für die main-Methode und für die öffentlichen Operationen und benutzerdefinierten Konstruktoren der Klassen *A* und *B* ist der Code in Form von Notizen hinzuzufügen. Assoziationen sind ggf. auszurichten und mit Rollennamen und Multiplizitäten zu versehen. (10 Punkte)
 - (b) Geben Sie unten den Verzeichnisbaum mit allen Unterverzeichnissen und Sourcecode-Dateien an, der bei einer Java-Implementierung des Systementwurfs entsteht. Als Wurzel kann dabei ein Verzeichnis mit Namen *home* verwendet werden. (2 Punkte)

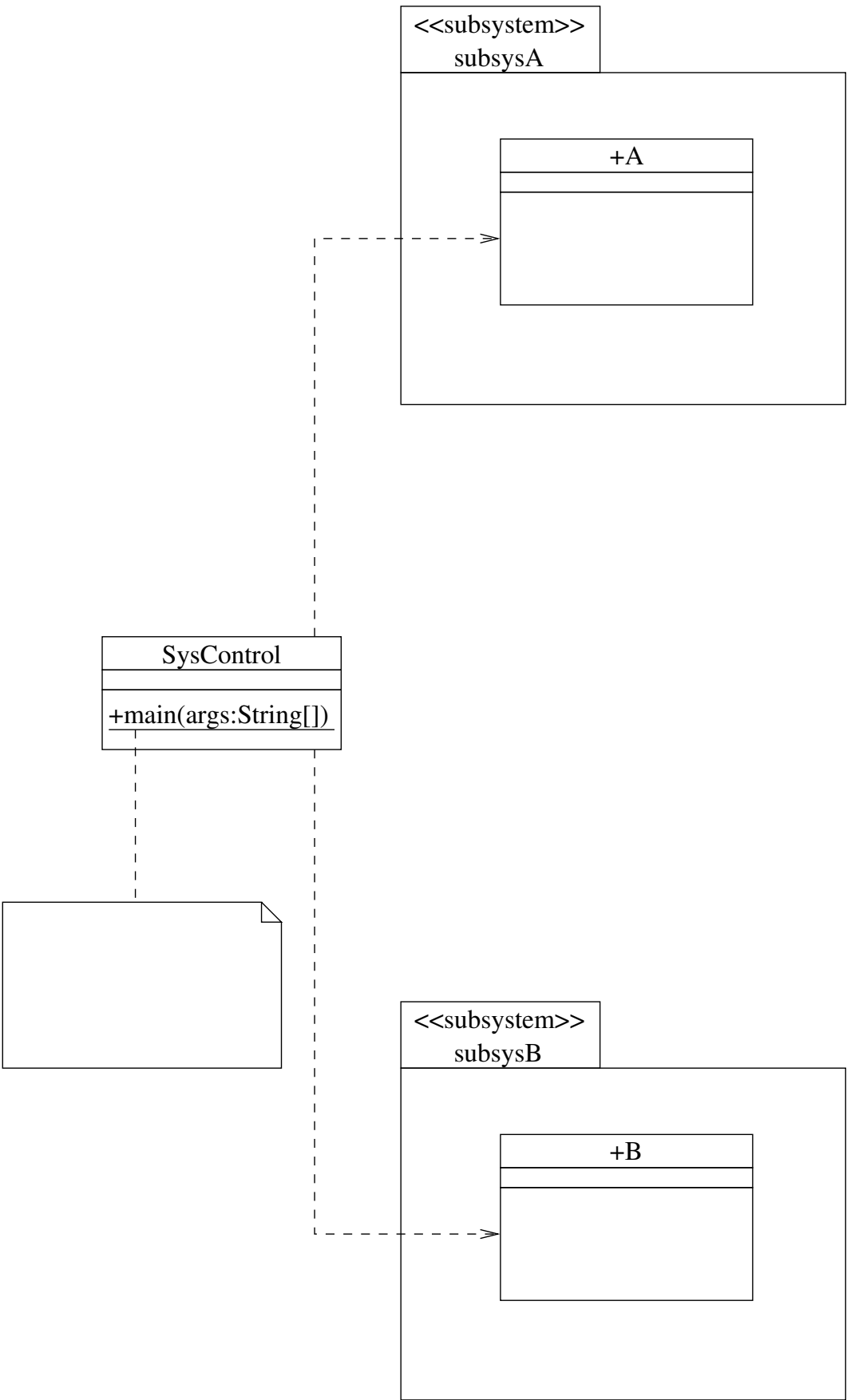
Aufgabe 4

(10 Punkte)

Gegeben sind die folgenden Klassen-, Interface- und Operationsnamen, die in dem in der Vorlesung angegebenen statischen Modell eines verteilten Objektsystems mit Broker und Proxy-Objekten verwendet werden:

ServerClass, ClientSideProxy, Broker, IServerClass, ClientClass, ServerSideProxy, task, service, packData, unpackData, forwardRequest, findServer, findClient

- (a) Geben Sie das statische Modell des Broker-Systems mit Proxy-Objekten an (einschließlich der benötigten Assoziationen und Realisierungsbeziehungen). Die Operationen sind dabei den passenden Klassen bzw. Interfaces zuzuordnen. Beachten Sie, dass manche Operationsnamen in mehreren Klassen bzw. Interfaces vorkommen können. (5 Punkte)
- (b) Geben Sie auf der Rückseite dieses Blatts ein Sequenzdiagramm an, das die in der Vorlesung beschriebenen Interaktionen bei der Verarbeitung eines entfernten (synchronen) Operationsaufrufs zeigt. (5 Punkte)



Aufgabe 5

(10 Punkte)

Das folgende Java-Programm implementiert ein Zustandsdiagramm, das das Verhalten von Objekten einer Klasse K beschreibt. Bei der Implementierung wurde die Methode der Realisierung durch Zustandsobjekte verwendet. Geben Sie auf der Rückseite dieses Blatts an, welches Zustandsdiagramm implementiert wurde. Verwenden Sie dabei stabile Zustände und Aktivitätszustände und heben Sie die Unterscheidung deutlich hervor. Anfangs- und ggf. Endzustand sind ebenfalls anzugeben. Als Ereignisse sollen nur Call-Events und (ggf. bewachte) Completion-Events vorkommen.

```
class K {
    private State state;
    public K() {
        hello(); state = new Paul();
    }
    public void back() {state = state.back(this);}
    public void hello() {...}
    public void help() {state = state.help(this);}
    public void obladi() {...}
    public void please() {state = state.please(this);}
    public void run() {...}
    public int yesterday() {...}
}

class State {
    public State back(K k) {return this;}
    public State help(K k) {return this;}
    public State please(K k) {return this;}
}

class George extends State {
    public State help(K k) {
        k.obladi(); return new Paul();
    }
}

class John extends State {
    public State please(K k) {
        k.run(); return new Paul();
    }
}

class Paul extends State {
    public State back(K k) {
        System.exit(0); return this;
    }
    public State please(K k) {
        k.run(); return new Ringo();
    }
}

class Ringo extends State {
    public State help(K k) {
        k.obladi(); return new Paul();
    }
    public State please(K k) {
        int check = k.yesterday();
        if (check == 1) return new John();
        else return new George();
    }
}
```