

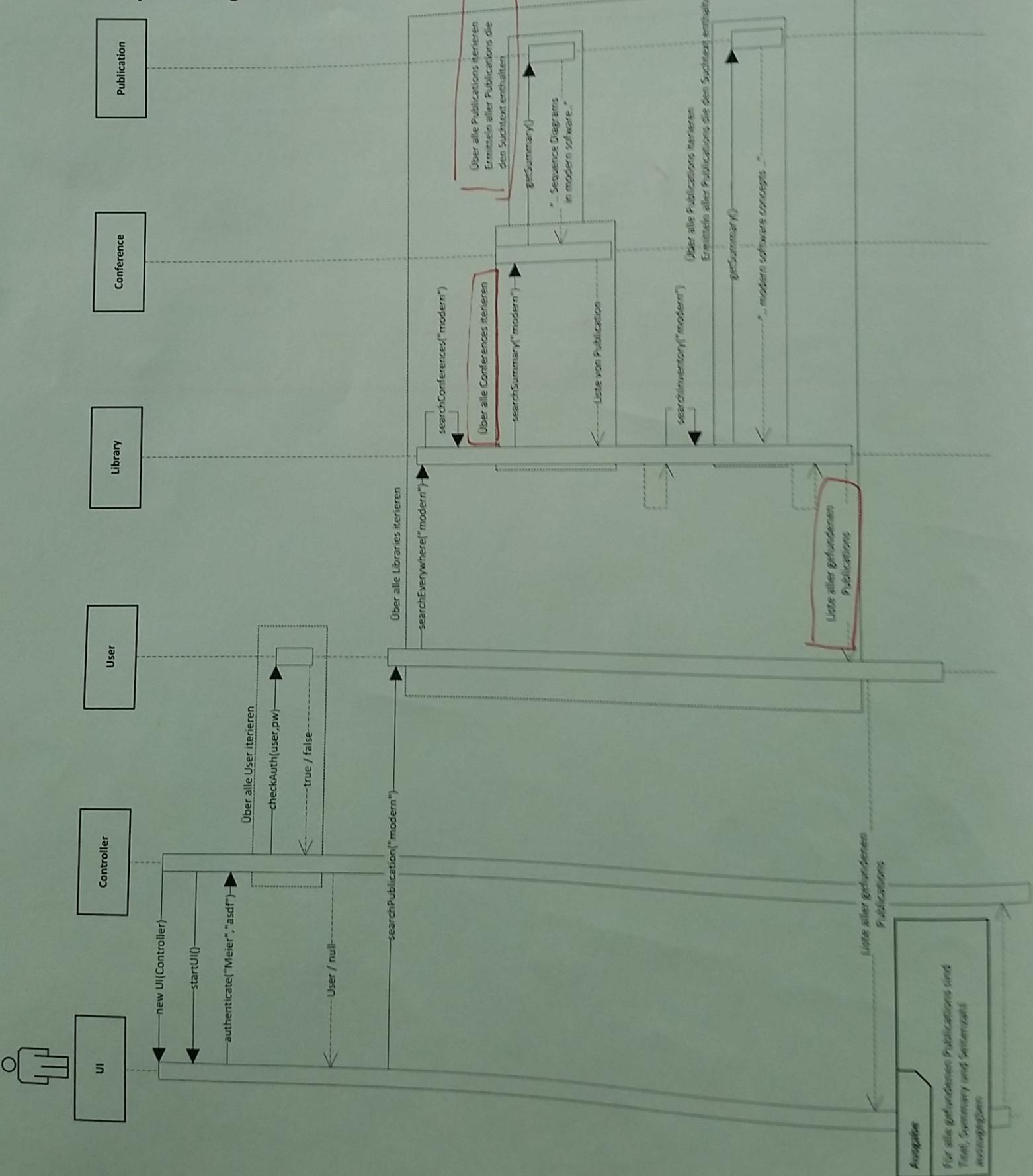
Beispiel 1 (55 Punkte)

Implementieren Sie die rot markierten Klassen und Interfaces (Methoden und Attribute) des Klassendiagramms entsprechend dem Sequenzdiagramm und dem angegebenen Code der Klasse "Controller".

Das Sequenzdiagramm beschreibt den Vorgang einer Bibliotheks-Suche bei einem User Interface. Das Objekt der Klasse `UserInterface` (=UI) kapselt die Schnittstelle zum Anwender. Sie brauchen jedoch keine interaktive Benutzereingabe in der Klasse UI zu implementieren.

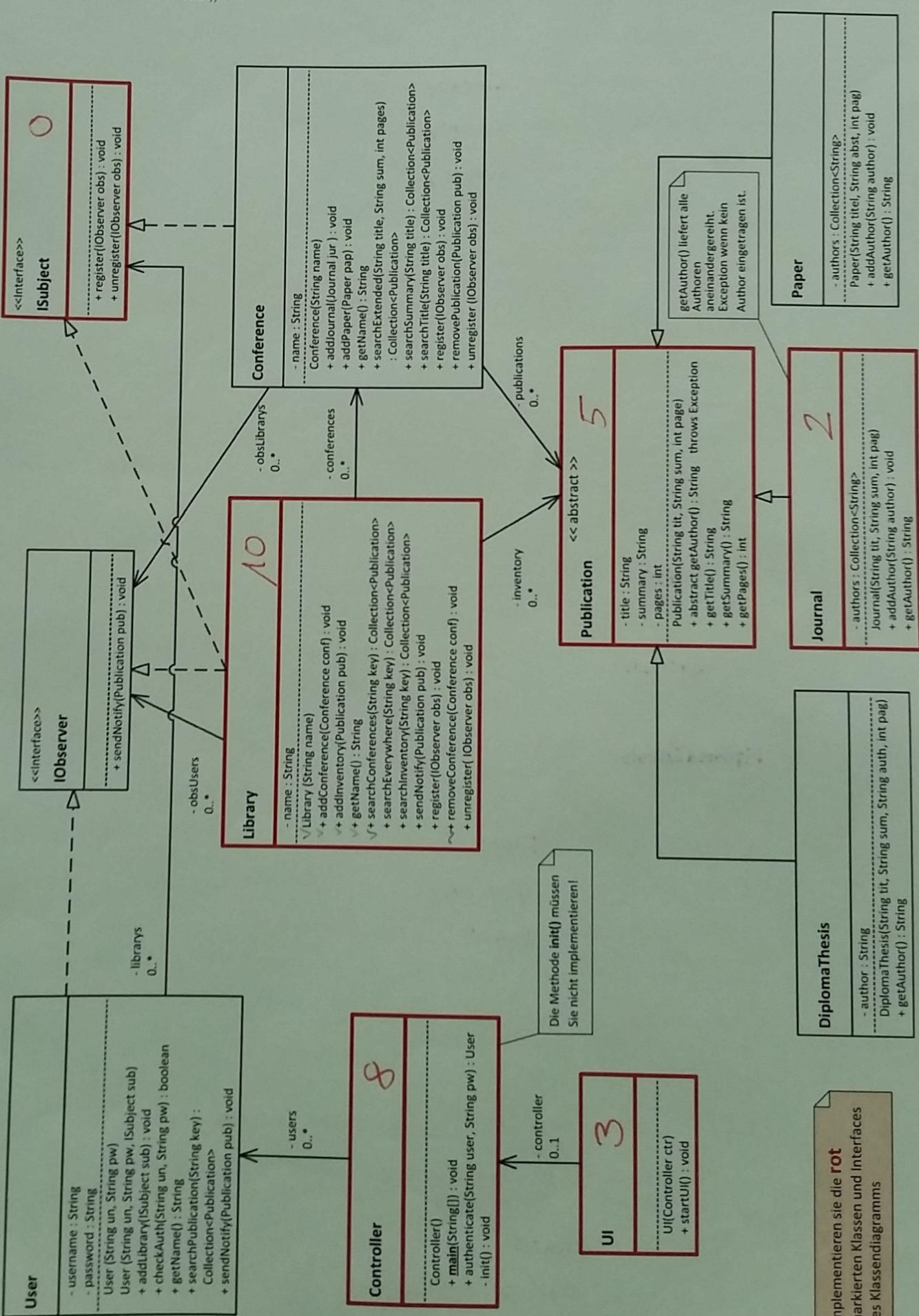
Hinweis: Verwenden Sie die Methode `boolean java.lang.String.contains(String s)` zur Suche nach enthaltenen Text (Returns true if and only if this string contains the specified sequence of String s).

Sequenzdiagramm:



Klassendiagramm

Assoziationen mit einem Pfeil (gerichtete Assoziationen) im Klassendiagramm bedeuten, dass jene Klasse beim Pfeilursprung eine Referenz (gespeichert in einer Instanzvariable) zur Klasse am Pfeilende besitzt. Diese Referenzen werden entsprechend der Kardinalitäten auf der Relation entweder als Objekt oder als Liste von Objekten gespeichert. Bitte beachten Sie, dass Publication eine abstrakte Klasse ist und unterstrichene Methoden statisch (d.h., static) sind. Variablen und Methoden sind entweder private, gekennzeichnet durch ein „-“ im Diagramm, oder public, gekennzeichnet durch ein „+“.



Klassen

```
import java.util.Collection;
import java.util.Vector;

public class Controller { // Ergänzen Sie eventuell fehlenden Teile
    public Collection<User> users = new Vector<User>();

    private void init() {
        Library bibl1 = new Library ("Bib_TU_Wien");
        Library bibl2 = new Library ("Bib_UNI_Wien");
        Library bibl3 = new Library ("Bib_WU_Wien");

        User user1 = new User("Meier", "asdf", bibl1);
        user1.addLibrary(bibl2);

        User user2 = new User("Mueller", "geheim", bibl2);
        User user3 = new User("Berger", "secret", bibl3);

        users.add(user1);
        users.add(user2);
        users.add(user3);

        Conference conf1 = new Conference("Conf1");
        Conference conf2 = new Conference("Conf2");

        bibl1.addConference(conf1);
        bibl1.addConference(conf2);

        bibl3.addConference(conf1);

        Paper paper = new Paper("Diagram usage",
            "... Sequence Diagrams in modern software ..", 5);
        paper.addAuthor("Meier");

        Journal journal = new Journal("JournalTitle",
            "JournalSummary", 15);
        journal.addAuthor("Müller");

        DiplomaThesis publ3 = new DiplomaThesis("Berger",
            "Using Sequence Diagrams",
            "... modern software concepts ..", 120);
        conf1.addPaper(paper);
        conf1.addJournal(journal);
        conf2.addPaper(paper);
        bibl1.addInventory(publ3);
    }

    public static void main(String[] args) {
        Controller contrl = new Controller();
        contrl.init();
    }
}
```

2P

```
... // IMPLEMENTIEREN SIE AB HIER ENTSPRECHEND DEM KLASSEN- UND
    // SEQUENZDIAGRAMM!
    // ... vergessen Sie nicht auf die Methode ...
    // public User authenticate(String username, String password) {...}
```

Main 2P

Authentif. 4P

public abstract class Publication

5P

private String title = null;

private String summary = null;

private int pages = 0; ✓

public Publication(String tit, String sum, int page) //Konstruktor

this.title = tit;

this.summary = sum;

this.pages = page; ✓

✓ public abstract String getAuthor() throws Exception;

✓ public String getTitle()

{
 return this.title;

}

✓ public String getSummary()

{
 return this.summary; ✓

✓ public int getPages()

{
 return this.pages; ✓

}

Name: _____

Import java.util.Collection; Implementieren Sie hier die notwendigen Klassen:

2P

import java.util.Vector;
public class Journal extends Publication

{
 private Collection<String> authors = null;

 public Journal(String tit, String sum, int pag)

 {
 super(tit, sum, pag);

 this.authors = new Vector<String>();

}

 public void addAuthor(String author)

 {
 this.authors.add(author);

}

throws ...

 public String getAuthor() // liefert den ersten Author zurück (mit Aufsicht geklärt!)

{
 catch(NullPointerException) ~~throw~~ ?
 if(this.authors.isEmpty()) // Collection ist leer?
 {
 return null;
 }

try

{
 return this.authors.firstElement();

Methode firstElement gibt

wirft wenn kein Element vorhanden
aber eine Exception NoSuchElementException!

}

// Methode main wird weiter geführt
UI ui = new UI(controller);
ui.startUI();
} // Ende der Main-Methode

2P

public User authenticate(String username, String password)
{
 if(this.users.isEmpty())
 return null; ✓
~~User user = null;~~
 for(User u : this.users)
 {
 if(u.checkAuth(username, password))
 return u; ✓
 }
 return null; ✓
}

4P

public Controller() // Konstruktör der Klasse Controller
{
 this.users = new Vector<User>();
}
} // Ende der Klasse Controller

Name: _____

Implementieren Sie hier die notwendigen Klassen:

* //Klasse Library

```
public Collection<Publication> searchConferences(String key)
{
    Collection<Publication> pub = null;
    for(Conference conf: conferences)
    {
        pub = conf.searchSummary(key);
        if(conf.name.contains(key)) if(pub.isEmpty()) //Publication ist leer
            continue;
        else return pub; → Nur in einer Konferenz
    } //Ende
} //return null;
```

~~public Collection<Publications> search Everywhere(String key)~~

~~Collection<Publication> pub1 = searchConferences(key);~~

~~Collection<Publication> pub2 = searchInventory(key);~~

//Collection aneinander hängen

~~: return pub1~~

Implementieren Sie hier die notwendigen Klassen:

✗ public Collection<Publication> searchInventory (String key)

{ collection = Publications; pub = null;

for (Publication p : this.inventory)

{

if (p.getSummary().contains(key))

return p; *nur eine Publication und keine Collection*

}

return null;

}

✓ public void sendNotify (Publication prb)

{

: *Keine*

}

✗ public void register (Observer obs)

{

: *Keine* *Observer Pattern*

}

✗ public void unregister (Observer obs)

{

: *Keine* *Observer Pattern*

}

3) Ende der Klasse Library

Implementieren Sie hier die notwendigen Klassen:

```
public interface ISubject  
{  
    private Vector<User> librarys = null;  
  
    public void register(IObserver obs);  
    public void unregister(IObserver obs);  
}
```

OP

```
public class UI  
{  
    private Controller controller=null;  
  
    public UI(Controller ctr)  
    {  
        this.controller=ctr;  
    }  
  
    public void startUI()  
    {  
        User user=null;  
        user=controller.authenticate("Meier","dsdf");   
        if(user==null)   
            return; //Error, falscher Name/Passwort  
        Collection<Publication> pub=controller.searchPublication("modern");  
        if(pub.isEmpty())  
        {  
            System.out.println("Keine Publication vorhanden");  
            return;  
        }  
  
        for(Publication p: pub)  
        {  
            System.out.println("Titel: " + p.getTitle() + ", Summary: " + p.getSummary() + ", Pages: " + p.getPages());  
        }  
    }  
}
```

3P

import java.util.Vector; Implementieren Sie hier die notwendigen Klassen!
import java.util.Collection;

10P

```
public class Library implements IObserver, ISubject  
{  
    private String name=null;  
    private Collection<IObserver> obsUsers=null;  
    private Collection<Conference> conferences=null;  
    private Collection<Publication> inventory=null;  
  
    public Library(String name)  
    {  
        this.name = name;  
        this.obsUsers= new Vector<IObserver>();  
        this.conferences = new Vector<Conference>();  
        this.inventory = new Vector<Publication>();  
    }  
  
    public void addConference(Conference conf)  
    {  
        this.conferences.add(conf);  
    }  
  
    public void addInventory(Publication pub)  
    {  
        this.inventory.add(pub);  
    }  
  
    public String getName()  
    {  
        return this.name;  
    }  
  
    public void removeConference(Conference conf)  
    {  
        this.conferences.remove(conf);  
    }  
}
```

Beispiel 4 (20 Punkte) 14P

Beantworten Sie die nachfolgenden Fragen kurz aber prägnant:

5P

- a) Erklären Sie das Konzept der Datenkapselung (*Data encapsulation*) allgemein und anhand eines Beispiels.

Datenkapselung dient dazu, um Daten vor Missbrauch zu schützen.

Nur Objekte der Klasse wissen über ihren Status Bescheid.

Objekte definieren Methoden um auf Variablen zugreifen zu können.

Bsp: public class Fahrrad

{ private int gear = 0;

 | public void setGear(int gear)

 | { this.gear = gear;

 }

 | public int getGear()

 | { return this.gear;

 }

Auf die Variable gear kann nicht direkt zu greifen werden.

Nur durch def. Methoden.

2P

- b) Was ist Polymorphismus? Was ist der Unterschied zwischen Overloading und Overriding?

Polymorphismus ist ~~✓ (-3)~~

Beim Overloading wird die Methode innerhalb der selben Klasse überladen.

Die Methoden ändern sich durch die Anzahl und den Typ der Parameter.

Overriding:

Die Methode der Superklasse wird überschrieben.

5P

- c) Welche Arten von Tests kennen Sie?

Stress-test, Leistungstest, Whitebox-Test, Blackbox-Test

Modultest

Komponententest

2P

- d) Was versteht man unter „Patterns“? Erklären Sie das Singleton-Pattern.

Patterns sind strukturierte Beschreibungen für Lösungsansätze wiederkehrender Problemstellungen.

Singleton-Pattern dient dazu, dass von einer Klasse immer nur eine Instance gleichzeitig bestehen kann.

Bsp: Windows-Manager

✗ Detektiv z. Singleton. (-3)