

## Beispiel 1 - Lagerverwaltung (55 Punkte)

Ihre Aufgabe besteht darin Teile eines Lagerverwaltungssystems zu implementieren. In dem Lager können beliebige Lagerobjekte in Lagerbereichen (*StorageCompartments*) aufbewahrt werden. Ein Lagerbereich kann dabei mehrere Lagerobjekte beinhalten. Ein Lagerbereich hat eine Breite (width) und ein maximal zulässiges Gesamtgewicht (weight). Diese dürfen durch die darin aufbewahrten Lagerobjekte in Summe nicht überschritten werden.

Es gibt unterschiedliche Lagerbereiche, gekühlte und ungekühlte. Jeder gekühlte Lagerbereich hat eine festgelegte Kühltemperatur. Entsprechend dürfen nur Lagerobjekte darin gelagert werden, wenn die Kühltemperatur des Lagerbereichs niedriger ist als die maximale Lagertemperatur des Lagerobjekts (siehe Abbildung 1).

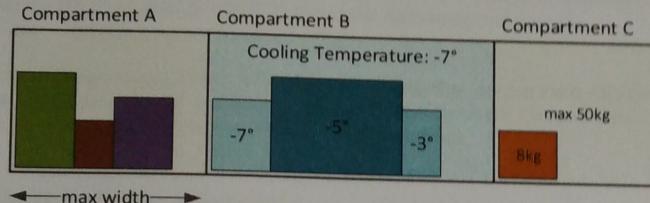


Abbildung 1 - Lagerverwaltung

Jedes Lagerobjekt hat eine Breite und ein Gewicht. Zusätzlich wird zwischen gekühlt und ungekühlt zu lagernden Lagerobjekten unterschieden. Ist eine Kühlung erforderlich, so hat ein solches Lagerobjekt zusätzlich eine maximale Lagertemperatur.

Über die Lagerverwaltung können neue Lagerbereiche angelegt und darin Lagerobjekte gelagert werden. Die Einordnung von Lagerobjekten in Lagerbereiche erfolgt automatisiert. Zusätzlich kann anhand einer Bewertungsfunktion der Lagerbereich bestimmt werden, dessen Ressourcen derzeit am schlechtesten genutzt sind.

$$\text{Bewertungsfunktion} = 2 \times (\text{Maximalgewicht} - \sum \text{Objektgewicht}) + (\text{Maximalbreite} - \sum \text{Objektbreite})$$

Je höher der Wert dieser Bewertungsfunktion, desto schlechter ist die Ressourcenausnutzung.

### Implementierung



Die im Klassendiagramm **rot** markierten Klassen sind Klassen, von denen Sie Teile implementieren müssen. Die dazugehörigen Beschreibungen der Methoden finden Sie im Abschnitt *Methoden-Beschreibungen*. Weiters existieren zu diesen Methoden-Beschreibungen Sequenzdiagramme. Bitte beachten Sie, dass die Sequenzdiagramme nur Ausschnitte aus der gesamten Funktionalität der Methode darstellen. Die vollständige Funktionalität ist aus der Beschreibung der Methode zu entnehmen.

### Methoden-Beschreibungen

Implementieren Sie

- die Klassen **CooledCompartment** und **NotStorableException** vollständig.
- den **Header**, inklusive aller Attribute und deren Initialisierung sowie der Konstruktoren, der Klasse **StorageManager**. Weiters sind in dieser Klasse die folgenden **rot** markierten Methoden zu implementieren.

#### StorageManager-Klasse:

Die Klasse hat einen Typ-Parameter T mit der oberen Schranke *StorageCompartment*.

##### **storeObject (...)**

Sucht nach dem ersten Lagerbereich, in dem noch ausreichend Ressourcen vorhanden sind um das übergebene Lagerobjekt einzulagern. Wird ein solcher Lagerbereich gefunden, so wird das Lagerobjekt eingelagert und TRUE zurückgegeben. In allen anderen Fällen wird FALSE zurückgegeben.

##### **findCompartmentByMetric (...)**

Bestimmt anhand einer Bewertungsfunktion den Lagerbereich, in dem die Ressourcen am schlechtesten genutzt werden. Dabei werden alle Lagerbereiche, in denen noch kein Lagerobjekt eingelagert ist, nicht berücksichtigt. Die Bewertung erfolgt anhand der oben beschriebenen Bewertungsfunktion. Der Lagerbereich der die schlechteste Bewertung hat, wird in einem *Optional* zurückgegeben. Wird kein Lagerbereich gefunden, wird eine leere Optional-Instanz zurückgegeben.

### StorageCompartment-Klasse:

`calculateSumOfWeights (...)`

Errechnet für einen Lagerbereich das Gesamtgewicht der darin enthaltenen Lagerobjekte.

`calculateSumOfWidth (...)`

Errechnet für einen Lagerbereich die Gesamtbreite der darin enthaltenen Lagerobjekte.

`checkWeight (...)`

Überprüft für einen Lagerbereich, ob das übergebene Lagerobjekt eingefügt werden kann. Das Gesamtgewicht ergibt sich aus den bereits eingelagerten und dem übergebenen Lagerobjekt. Wird das maximale Gesamtgewicht nicht überschritten, wird TRUE zurückgegeben. In allen anderen Fällen wird FALSE zurückgegeben.

`checkWidth (...)`

Überprüft für einen Lagerbereich ob das übergebene Lagerobjekt eingefügt werden kann. Die Gesamtbreite ergibt sich aus den bereits eingelagerten und dem übergebenen Lagerobjekt. Wird die maximale Gesamtbreite nicht überschritten, wird TRUE zurückgegeben. In allen anderen Fällen wird FALSE zurückgegeben.

`getMaxWeight (...)`

Gibt das maximal zulässige Lagerungsgewicht zurück.

`getMaxWidth (...)`

Gibt die Breite des Lagerbereichs zurück.

`Collection<Storable> retrieveStoredObjects (...)`

Gibt die in dem Lagerbereich enthaltenen Lagerobjekte zurück.

`storeObject (Collection<...>...)`

Lagert die übergebenen Lagerobjekte im Lagerbereich ein. Es werden dabei keinerlei Überprüfungen auf Gewicht oder Breite durchgeführt. Tritt ein Fehlerfall auf, wird eine `NotStorableException` geworfen.

`storeObject (Storable ...)`

Lagert das übergebene Lagerobjekt im Lagerbereich ein. Es werden dabei keinerlei Überprüfungen auf Gewicht oder Breite durchgeführt. Tritt ein Fehlerfall auf, wird eine `NotStorableException` geworfen.

### CooledCompartment-Klasse:

`storeObject (...)`

Lagert das übergebene Lagerobjekt in den Lagerbereich ein. Dabei wird überprüft, ob das zulässige Gesamtgewicht bzw. die zulässige Breite nicht überschritten werden. Zusätzlich können nur Lagerobjekte eingelagert werden, die vom StorageTyp COOLED sind und deren maximale Lagertemperatur nicht geringer ist als die Kühltemperatur des Lagerbereichs. Wird eine dieser Anforderungen verletzt, wird eine `NotStorableException` geworfen.

### NotStorableException-Klasse:

`getMessage (...)`

Gibt eine aussagekräftige Fehlerbeschreibung zurück. In dieser Fehlerbeschreibung sind Informationen über das einzulagernde Lagerobjekt und den Lagerbereich enthalten.

### Optional-Klasse:

Ein Optional ist ein Container, in dem ein Objekt enthalten sein kann. Ist ein Objekt enthalten ist liefert die Methode `isPresent()` TRUE und `get()` liefert das enthaltene Objekt.

`empty (...)`

Liefert eine leere Optional-Instanz zurück. Das bedeutet, dass kein Objekt enthalten ist.

`get (...)`

Ist ein Objekt enthalten liefert die Methode das Objekt zurück, andernfalls wird eine `NoSuchElementException` geworfen.

`isPresent (...)`

Liefert TRUE wenn ein Objekt enthalten ist andernfalls FALSE.

`of (...)`

Gibt ein Optional zurück in dem das übergebene Objekt enthalten ist. Ist der übergebene Parameter NULL, so wird eine `NullPointerException` (Laufzeit-Exception!) geworfen.

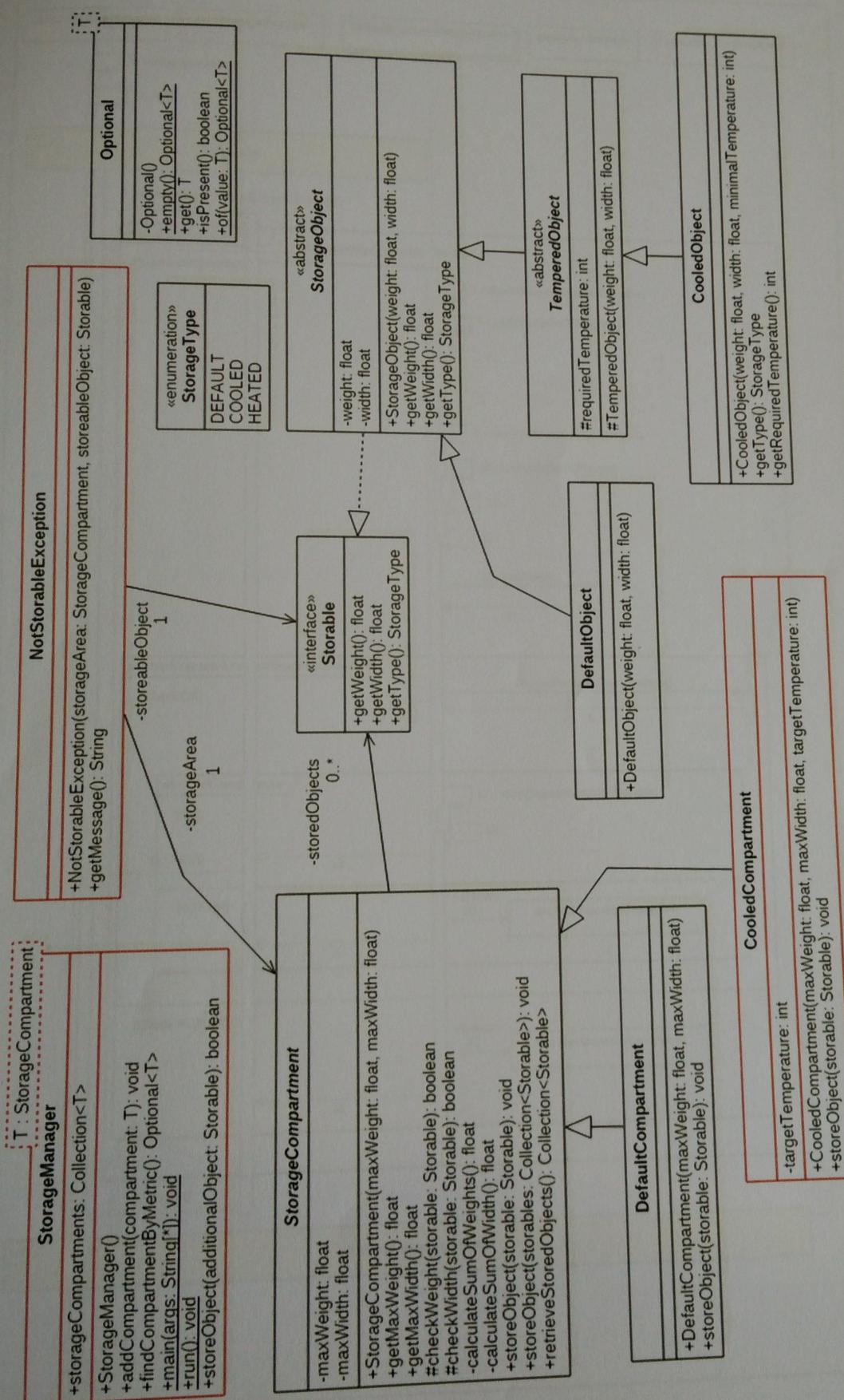


Abbildung 2 - Lagerverwaltungssystem Klassendiagramm

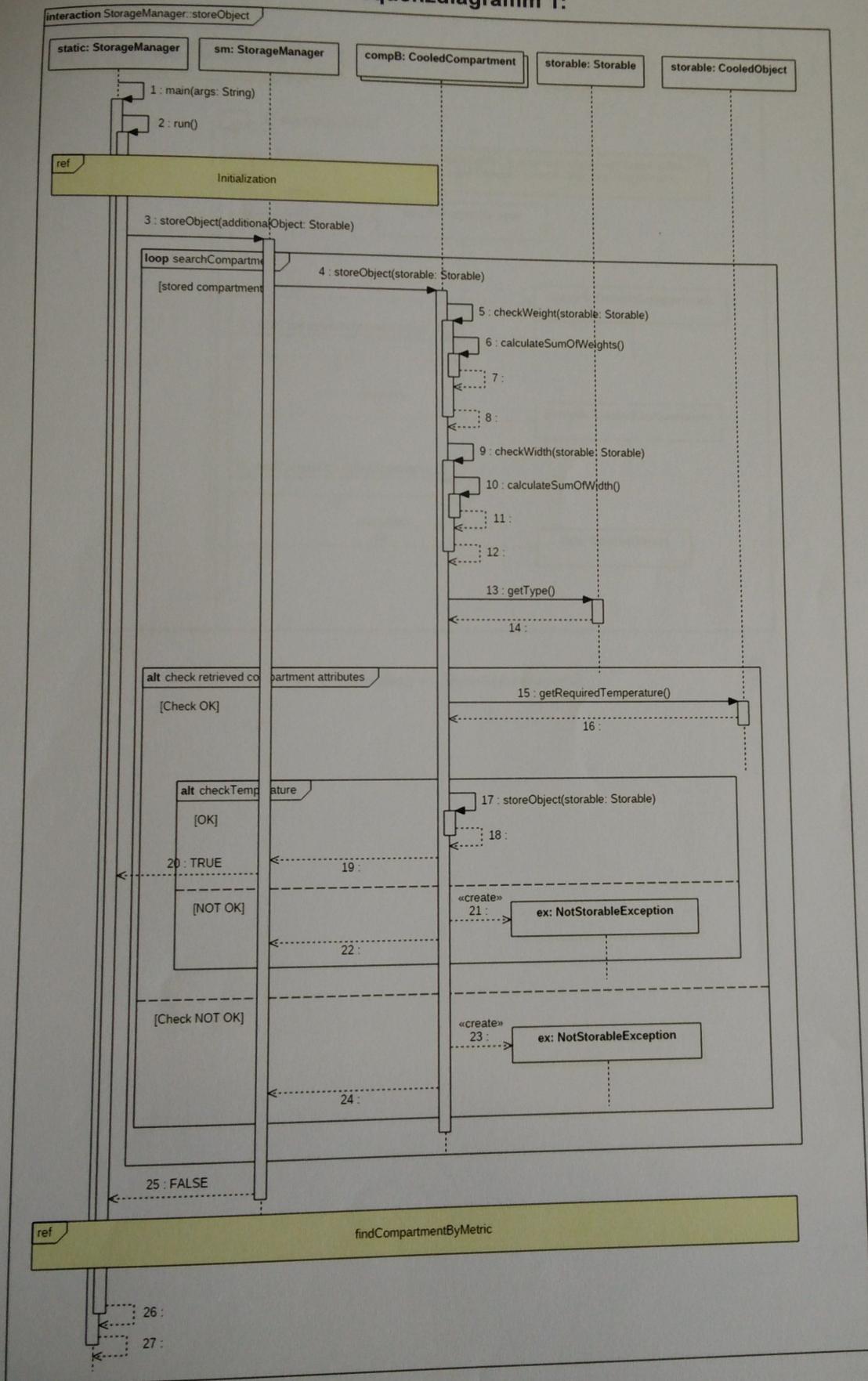


Abbildung 3 – Sequenzdiagramm 1

Angabe Beispiel 1 – Seite 4 von 6

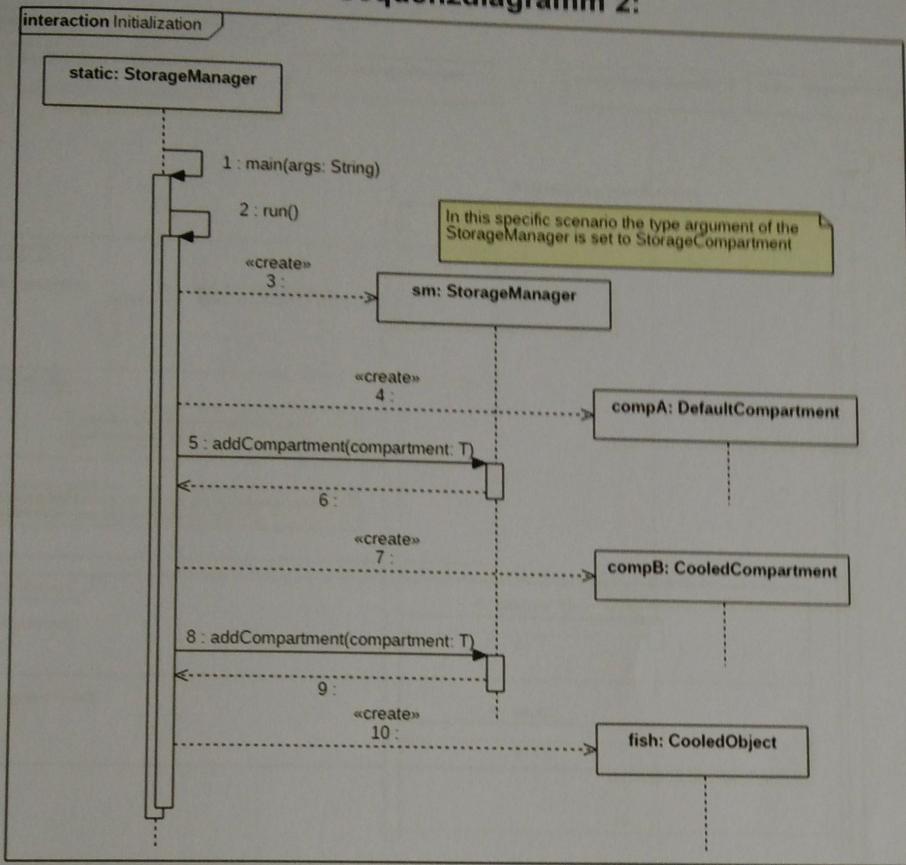


Abbildung 4 - Sequenzdiagramm 2

Beispiel 1

Sequenzdiagramm 3:

24.10.2018

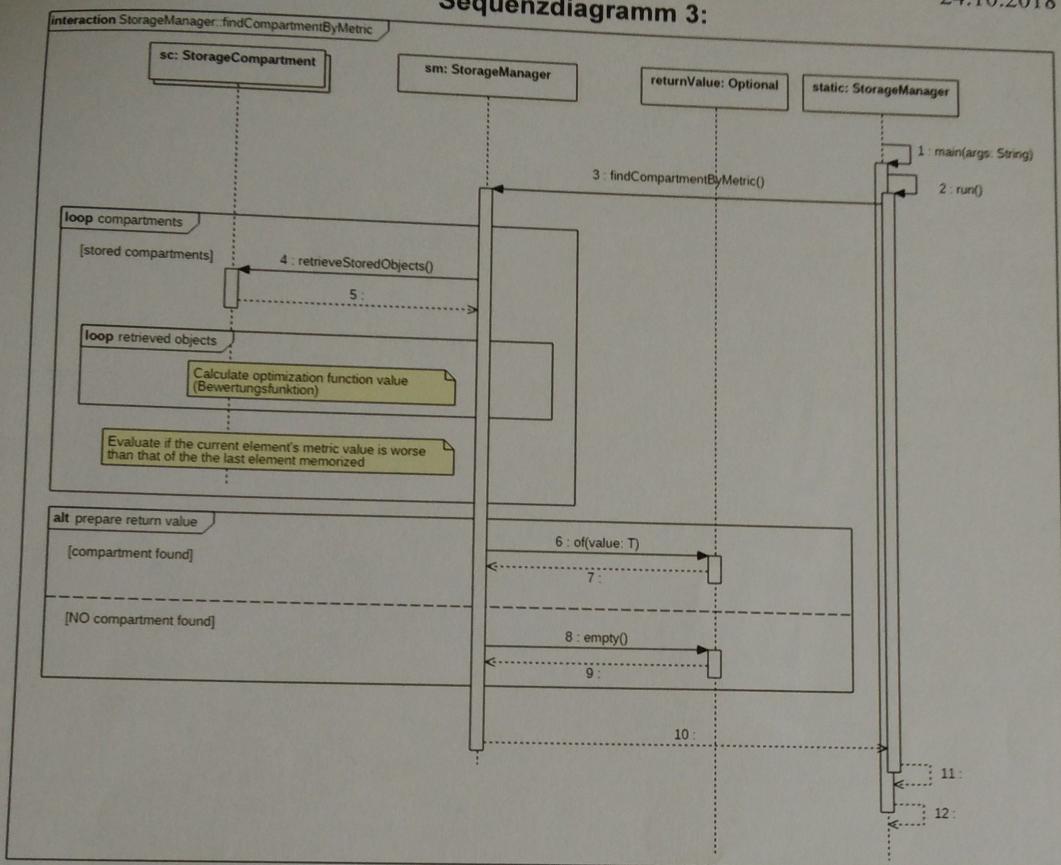


Abbildung 5 –Sequenzdiagramm 3

```

1 // Abdulhamid Baghdadi
2 // 01329815
3
4 public class StorageManager {
5     public Collection<StorageCompartment> storageCompartments; ✗ 0
6
7     public StorageManager() {
8         ✗
9
10    public void addCompartment(StorageCompartment compartment) {
11        ✗
12
13
14    → Klassendiagramm 8
15    public Optional<StorageCompartment> findCompartmentByMetric() {
16        float bewertungsfunktion;
17        float iteratorBewertungsfunktion = 0;
18        StorageCompartment bestStorageCompartment = new StorageCompartment(); ✗
19
20        for (StorageCompartment storCom : this.storageCompartments) ✗
21
22            float sumofWeights = 0;
23            float sumOfWidths = 0;
24            for (Storable storableObject : storCom.retrieveStoredObjects()) {
25                sumofWeights += storableObject.getWeight();
26            }
27            for (Storable storableObject : storCom.retrieveStoredObjects()) {
28                sumOfWidths += storableObject.getWidth();
29            }
30
31            if (sumofWeights != 0) { → Deal Empty
32                bewertungsfunktion = ( 2*(storCom.getMaxWeight() - sumOfWeights)
33                                + storCom.getMaxWidth() - sumOfWidths);
34
35                if (bewertungsfunktion > iteratorBewertungsfunktion) {
36                    bestStorageCompartment = storCom; ✓
37                } → Bewertung
38            }
39
40        }
41
42        if (bestStorageCompartment == null) {
43            return Optional.empty<StorageCompartment>(); ✓
44        } else {
45            return Optional.of<StorageCompartment>(bestStorageCompartment); ✓
46        }
47    }
48
49
50    public static void main(String args[]) {
51
52    }
53
54    public static void run() {
55
56    }
57
58
59    public boolean storeObject(Storable additionalObject) {
60        for (StorageCompartment storCom: this.storageCompartments) { ✓
61
62            try {
63                storCom.storeObject(additionalObject); ✓
64            } catch (NotStorableException e) {
65                continue; ✓
66            }
67
68            return true; ✓
69        }
70    }

```

```

1 // Abdulhamid Baghdadi
2 // 01329815
3
4 public class ColledCompartment extends StorageCompartment {
5
6     private int targetTemperature;
7
8     public ColledCompartment(float maxWeight, float maxWidth, int
9         targetTemperature) {
10    }
11
12     public void storeObject(Storable storables) throws {
13
14         boolean acceptedWeightStorable = super.checkWeight(storables);
15         boolean acceptedWidthStorable = super.checkWidth(storables);
16
17         float sumOfWeights = 0;
18         float sumOfWidths = 0;
19         for(Storable storablesObject : this.retrieveStoredObjects()){
20             sumOfWeights += storablesObject.getWeight();
21         }
22         for(Storable storablesObject : this.retrieveStoredObjects()){
23             sumOfWidths += storablesObject.getWidth();
24         }
25
26         String typeOfStorable = getType(storables); # check
27         int requiredTemp = ((CooledObject) storables).getRequiredTemperature();
28
29         NotStorableException e = new NotStorableException (this, storables);
30
31
32         if(acceptedWeightStorable == false){
33             e.getMessage("Weight of Storable is above the maximum allowed weight");
34             throw e; Paranormal?
35         }
36
37         if(acceptedWidthStorable == false){
38             e.getMessage("Width of Storable is above the maximum allowed width");
39             throw e; ?
40         }
41
42         if(this.getMaxWidth() < (sumOfWidths + storables.getWidth())){
43             e.getMessage("Sum of the widths is above the maximum allowed width");
44             throw e;
45         }
46
47         if(this.getMaxWeight() < (sumOfWeights + storables.getWeight())){
48             e.getMessage("Sum of the weights is above the maximum allowed width");
49             throw e;
50         }
51
52         if(typeOfStorable != "CooledObject"){
53
54             throw e;
55         }
56         if (typeOfStorable != "CooledObject") {
57
58             if(typeOfStorable != "CooledObject"){
59                 e.getMessage("Object is not a cooled object");
60                 throw new NotStorableException(this, storables);
61             }
62
63             if (requiredTemp > this.targetTemperature){ ✓
64                 e.getMessage("Compartment temperature ist not suitable for this
65                 object");
66                 throw new NotStorableException(this, storables); ✓
67             }
68
69         }

```

(3)

3

Separation!

Separation!

```
70     super.storeObject(storable);  
71 }  
72  
73  
74  
75  
76 }  
77
```

1

```
1 // Abdulhamid Baghdadi
2 // 01329815
3
4 public class NotStorableException {
5     private StorageCompartment storageArea;
6     private Storable storeableObject;
7
8     public NotStorableException(StorageCompartment storageArea, Storable
9         storeableObject) {
10        this.storageArea = storageArea;
11        this.storeableObject = storeableObject;
12    }
13
14     public String getMessage(String message) {
15         System.out.println("Error in storing " + this.storeableObject.toString() +
16             " in Area " + this.storageArea.toString());
17         System.out.println(message);
18     }
19 }
```

~~return~~

Beispiel3.txt

// Abdulhamid Baghdadi  
// 01329815

A) 1,5

Patterns: verallgemeinerte Lösungen von wiederkehrenden Problemen.  
Sie bestehen aus Beschreibungen, die Texte, Module, Hypermedia usw.. verwenden.

?

Singleton Pattern. X  
sorgt dafür, dass nur eine Instanz von dieser Klasse erstellt wird. ~  
besteht aus:  
privater Constructor  
Referenz zu der Instanz  
static Methode, die eine Instanz zurückgibt (z.B. getInstance())

```
public class Singleton{  
    private Singleton reference;  
    private Singleton(){  
        Das ist kein Singleton  
    }  
}
```

```
public static Singleton getInstance(){  
    if (reference == null){  
        reference = new Singleton();  
        Zugriff fehlerhaft nicht  
        return this.reference;  
    } else {  
        return this.reference;  
    }  
}  
}
```

*> Singleton Pattern so NICHT*

Observer Pattern

wird verwendet, wo Änderungen stattfinden, die für eine andere Klasse relevant sind.  
Diese werden dann bekanntgegeben.

Wie

Besteht auf einem Subject, der Observers registriert und ihnen bekanntgegeben, wenn  
eine Änderung auftritt.

```
public class Subject(){  
    Vector<Observer> observers; Init...  
    Regelmäßig? von wem, bei wen?  
    public void registerObserver(Observer observer){  
    }  
  
    public void unregisterObserver(Observer observer){  
    }  
  
    public class Observer(){  
        public void update(Subject subject){  
        }  
    }  
}
```

B) 2,5

Datenkapselung: Der direkte Zugriff auf die Klassenvariablen verbieten.  
Nur durch eigene Klassen-Methoden erlauben.

*Instanz - nicht Klassenvariable!*

Instanz.

Verletzt, wenn eine Variable public definiert wird, die als private definiert sein sollte, und umgekehrt.

Die Umkehr der Aussage bedeutet das nicht!

mit Klassenhierarchie: Die Subklasse vererbt die privaten Variablen, aber hat keinen direkten Zugriff auf sie (außer mit definierten Methoden).

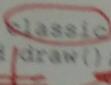
C) 0

? Vier ?

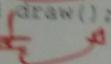
Polymorphismus: Vier Gestaltigkeiten.  
Erklärung durch ein Beispiel: Zeichnen eines graphischen Objekts ist polymorphist,  
dass das Zeichnen von einem Kreis sehr unterschiedlich ist vom Zeichnen eines Rechtecks.

public class GraphicObject {  
 abstract void draw();

absolutin Draw privat?

}  public class Circle extends GraphicObject {  
 void draw();

> Schre classie => class

}  public class Rectangular extends GraphicObject {  
 void draw();

Das sind absolute Werte

D) 3,5

Typ: hat verschiedene Definitionen in Bezug auf die Verwendung:  
z.B. ein Systemingenieur definiert ihn als die Art, wie die Daten im Speicherbereich interpretiert werden sollten.

In OOP definiert ein Typ eine Sammlung von Attributen und Methoden, die die Operationen auf diesen Attributen definieren.

Vererbung/Ersetzbarkeit: eine Subklasse sollte immer dort einsetzbar sein, wo eine Superklasse zu erwarten ist.

Warum genau?

Die Ersetzbarkeit wird geprüft mit verschiedenen Kriterien (Kontravarianz von den Eingangsvariablen, Kovarianz von den Ausgangsvariablen, Invarianz von den Instanzen)

Velen  
ahgungs  
werte

Spezialisierung: definiert eine Beziehung zwischen zweier Klassen, die eine ist generalisiert, die andere ist spezifischer. Diese Beziehung bedingt aber keine Ersetzbarkeit.

Z.B. ein Pensionist ist eine Spezialisierung von einer Person.

Ein Pensionist kann keine Person ersetzen (Unterschied in Altergruppe) usw..

Ja, kann auf den Bedingen an und ob diese der Wörter des Ersetzbarkeit prüfen

wichtig