# Programmieren Zusammenfassung

**Extreme Programming:**
Hierbei werden zuerst die Testmethoden geschrieben, dann die Klassenimplementation

**Ausnahme (Exception):**
Eine Ausnahme ist ein alternativer Weg eine Methode im Fehlerfall zu verlassen. Die Methode gibt keinen Wert zurück , aber wirft ein Objekt, welches die Fehlerinformation beinhaltet.

**Throwable:**
Alle Exceptions und Errors werden von Throwable abgeleitet. Unchecked Exceptions die von RuntimeException abgeleitet sind müssen nicht geheilt werden, da sie vom Compiler nicht einfach erkannt werden können. Errors sind Fehler, welche nicht abgefangen werden können und zum Programmabbruch führen. Alle anderen checked Exceptions, die nicht von Runtime abgleitet sind müssen entweder mit try und catch Blöcken abgefangen werden oder an die rufende Methode weitergereicht werden. Es können mehrere Ausnahmen hintereinander gefangen werden, aber zur Bearbeitung wird immer die erste herangezogen.

**Signatur:**
Der Name und die Parameter in der vorgegebenen Reihenfolge bilden die Signatur der Methode. Dazu kommt nun auch die throws-Anweisung und die mit ihr assoziierten Ausnahme.

**Wrapping:**
Dieses geschachtelte Verpacken erlaubt Ausnahmen ,,höherer'' Ordnung zu werfen, ohne die Information über die zugrunde liegende Ursache zu verlieren

**Finally:**
Hierbei handelt es sich um einen Block der trotz eines catch Blocks auf jeden Fall noch ausgeführt wird. Ein try mit Ressourcen gibt die allokierte Ressource nach dem try Block wieder frei und so könnte der finally Block gespart werden

**Singleton-Entwurfsmuster:**
- Erzeugungsmuster

Das Singleton Entwurfsmuster dient dazu sicherzustellen, dass in einer Anwendung in einer Klasse nur eine Instanz auftritt. Der Private Konstruktor verhindert, dass von außen eine Instanz der Klasse erzeugt wird. Die Klassenmethode *getInstanz* gibt eine Referenz auf die existierende Instanz.

**Dekorator-Entwurfsmuster:**
- Strukturmuster
- Wird in Java intensiv in der Ein-/Ausgabebibliothek benutzt

Stattet ein Objekt dynamisch mit zusätzlichen Eigenschaften aus ohne andere Objekte zu beeinflussen und erlaubt die selben Eigenschaften mehrmals hinzuzufügen. Ausserdem vermeidet er mit Eigenschaften überladene Klassen. Besser als Subklassenbildung, da dies statisch vorgenommen wird.
Aber ein Decorator und seine Component sind nicht identisch und für nicht eingeweihte Programmierer schwer zu verstehen.

**Files und Paths und Fabrikmethode-Entwurfsmuster:**
in java.nio.file stehen die Klassen Files und Paths zur Verfügung. Die Klasse FileSystem stellt die Verbindung zum zugrunde liegenden Betriebssystem her.

**Fabrikmethode-Entwurfsmuster:**
- Erzeugungsmuster

Diese erlaubt die Konstruktion einer konkreten Klasse, die jedoch nur über Referenzen auf eine Basisklasse referenziert wird. In dieser Hinsicht hat die Fabrikmethode die Funktionalität eines polymorphen Konstruktors.

**Innere Klassen:**
In Java ist es möglich Klassen innerhalb von Klassen zu implementieren. Diese können auf alle Attribute der Instanzen der beinhaltenden Klassen zugreifen

**Anonyme Klassen:**
sind Spezialfälle von inneren Klassen, welche direkt am Konstruktoraufruf definiert werden. Sie haben keine static Deklarationen, keinen Konstruktor und nur eine Instanz. Besonders häufig tritt der Fall auf, dass nur noch eine abstrakte Methode fehlt und so kann die Syntax der anonymen Klasse durch die der Lambda-Syntax ersetzt werden.

**Strategie-Entwurfsmuster:**
Das Strategiemuster dient dazu eine Familie von Algorithmen vergleichbar und austauschbar zu machen. Dies erlaubt die Algorithmen unabhängig von den sie benutzenden Klienten zu variieren.
Vorteile:
- Hierarchien von Strategieklassen definieren Algorithmen zur Wiederbenutzung durch Kontextklassen
- Alternative zu Subklassen
- Eliminieren konditionale Anweisungen
- Strategien stellen verschiedene Implementationen des selben Verhaltens zur Verfügung

Nachteile:
- Den Klienten müssen die Algorithmen bekannt sein

**Lambda-Ausdrücke:**
Falls eine Methode ein FunctionalInterface als Parameter erwartet, kann eine spezielle Kurzform (Lambda-Ausdruck) verwendet werden. Dies ist eine kompakte Definition einer Implementation dieses Interfaces als Instanz einer Klasse. Die Parameter der Methode werden in Klammern angegeben, ein Methodenname ist nicht nötig. *(argument) -> (body)*

**Referenzen auf Methoden:**
Der Doppelpunkt-Operator erlaubt eine Referenz auf eine Methode einer Klasse an eine Methode als FunctionalInterface zu übergeben

**Die Stream Pipeline:**
Ein Datenstrom fließt entlang der intermediate-Operationen der Stream Pipeline bis zu einer terminalen Operation. Die Bewertung der einzelnen Abschnitte beginnt erst mit dem Aufruf der terminalen-Operation. Die Daten der Quelle durchfließen die Pipeline nur einmal: Der Aufruf der terminalen Operation konsumiert den Datenstrom. Sollen die Daten neu aufgerufen werden, so muss auch die Pipeline neu initialisiert werden.

**intermediate Operationen:**
Filter erwarten ein FunctionalInterface vom Typ Predicate() als Argument. Ergibt die Anwendung auf das Stream-Element true, dann wird dieses weitergeleitet, sonst nicht.

sorted erwartet einen Comparator als Argument. Der Stream wird entsprechend umsortiert.

Eine map bildet Elemente auf neue Elemente ab

**Befehl-Entwurfsmuster:**
- Verhaltensmuster

Es dient dazu, dass Objekte Anforderungen an andere Objekte senden, ohne Näheres über diese Anforderungen zu wissen.

**Iterator-Entwurfsmuster:**
- Erzeugungsmuster

Oft enthält eine Klasse mehrere Elemente desselben Typs (eine Aggregation). Ein Iterator erlaubt es auf diese Elemente sequentiell zuzugreifen, ohne Kenntnis darüber zu haben, wie die Elemente genau in der Klasse gespeichert sind. Es ist ein Spezialfall der Fabrikmethode.

**Das Collection-Framework:**
Eine Collection ist ein Objekt, welches eine Gruppe von Objekten repräsentiert. Das Collection-Framework ist eine einheitliche Software-Architektur zur Darstellung von Collections. Dabei sind die Manipulationsmethoden unabhängig vom inneren Aufbau einer Collection.

## Teil 2 Algorithmen und Datenstrukturen

**Verkettete Liste:**
Menge von Objekten, bei dem jedes Objekt in einem Node ist. Jeder Node hat eine Referenz auf einen Node. Vom Anfang der Liste muss Node für Node traversiert werden. Daher hat der Zugriff auf ein einzelnes Element denselben Aufwand, wie der sequentielle Zugriff auf alle Elemente.
Also:

- Zugriff O(N)
- Einfügen O(N)
- Anfügen O(1)

Bei einer **doppelt verketteten Liste** kennt ein Node nicht nur seinen Vorgänger, sondern auch seinen Nachfolger.
Enthält die letzte Referenz auf einen Nachfolger anstatt der null eine Referenz auf das erste Element, so ist die Liste **zirkulär**.

Und wenn das die letzte Referenz auf ihren Nachfolger statt der null eine Referenz auf das erste Element und das erste Element statt des null Vorgängers eine Referenz auf das letzte Element, so ist es eine **doppelt zirkuläre Liste.**

**Hashing:**
Objekte werden in einem Feld mit den Indices von 0 – N -1 gespeichert. Die einzelnen Speicherpositionen werden Buckets genannt. Eine Hashfunktion bestimmt für ein Element die Position im Feld, bei der ein mit dem Schlüssel assoziiertes Objekt gespeichert werden soll.

**Behandlung von Kollisionen bei Hashing:**

- Verkettung
  - Überläufer werden in jedem Bucket in einer verketteten Liste untergebracht
- lineares Sondieren
  - Die Buckets werden der Reihe nach überprüft, das Element wird im ersten gefundenen freien Bucket gespeichert
- Quadratisches Sondieren
  - Wie lineares Sondieren, nur dass die Konzentration der Elemente in der Nähe weniger Buckets verhindert wird
- Rehashing
  - Hashtabelle muss vergrößert werden. Es muss für jeden Eintrag ein neuer Bucket, entsprechend der neuen Hashfunktion bestimmt werden

**Hashfunktion und Gleichheit:**
Falls die Hashfunktion für zwei Objekte denselben Wert ergibt, heisst dies nicht, dass die Objekte gleich sein
➔ aber wenn die Hashfunktion für beide nicht denselben Wert ergibt folgt, dass die Objekte nicht gleich sein können

**Traversierung:**

- inorder
  - erst rekursiv der linke Baum, dann den Node bearbeiten, dann der rechte
- Preorder
  - erst den Node bearbeiten, dann rekursiv den linken und den rechten Baum
- Postorder
  - erst jeweils rekursiv die linken und rechten Subbäume, dann den Node bearbeiten
- Levelorder
  - Erst alle Nodes eines Levels bearbeiten, bevor auf einen im tieferen Level übergegangen wird. (Benötigt extra Speicher in Form einer Warteschlange)

**AVL-Bäume:**
Ziel ist es, dass sich die Tiefe der Teilbäume nicht mehr unterscheiden dürfen als um den Wert 1. Wenn dies doch der Fall sein sollte, dann muss sich der Baum einfach rotieren ( falls im linken Subbaum des linken Kindes oder im rechten Subbaum des rechten Kindes eingefügt wurde) oder doppelt rotieren ( Falls im linken Subbaum des rechten Kindes oder im rechten Subbaum des linken Kindes eingefügt wurde )

**Spiele**

Falls es sich um ein Nullsummen-spiel handelt ( der Gewinn des einen ist der Verlust des anderen ), kann man den Minimax durch den Negamax vereinfachen. Dazu muss die statische Bewertungsfunktion an den Spieler angepasst werden.
Bemerkung (Statische Bewertungsfunktion Minimax):

- Spieler gewinnt: Wert ist höchst möglicher
- Spieler verliert: Wert ist niedrigst möglicher

Da der Negamax mehr Konfigurationen als nötig untersucht, gibt es die α-β-Beschneidung, welche dem Algorithmus erlaubt Teile des Baums zu ignorieren. Dabei ist der Intervall [α,β] das Suchfenster. Nur darin werden die Züge betrachtet, alle anderen werden beschnitten. Zu Beginn ist das Suchfenster [-∞,∞] .
Während des Durchlaufs wird das Fenster enger.
Ziel: Fenster so eng wie möglich zu machen.

**Graph-Algorithmen:**

**Graph-Definition:**
Es besteht aus Vertices und Kanten, welche jeweils 2 Vertices verbinden. Sowohl Vertices, als auch Kanten können Attribute besitzen.

**Kantenzug, Weg:**
Eine Folge von Kanten heisst Kantenzug, falls es eine Folge von Vertices des gibt. Ein Weg ist ein Kantenzug, indem alle Kanten verschieden sind. Ein Graph ist zusammenhängend, falls es für je zwei beliebige Vertices des Graphen einen Weg gibt, der die beiden Vertices verbindet. Ein nicht zusammenhängender Graph besteht aus zusammenhängenden Komponenten.

**Graphendarstellung: Adjadenzmatrix:**
Die vertices werden über einen Index identifiziert. Die Kanten eines Graphen mit n Vertices werden in n x n Adjadenzmatrix A notiert.
aij = 1 , falls die Vertices vi und vj mit einer Kante verbunden sind.
aij = 0, falls dies nicht so ist.

**Arten von Graphen:**

- Zyklisch, Kanten mit Richtung
  - gerichteter Graph
- Zyklisch, Kanten ohne Richtung
  - allgemeiner Graph
- Keine Zyklen, Kanten mit Richtung
  - gerichteter azyklischer Graph
  - directed acyclic graph ( DAG )
- Keine Zyklen, Kanten ohne Richtung
  - Azyklischer Graph

**Bipartite Grapen:**
Ein Graph ist bipartit, falls seine Vertices auf zwei disjunkte Untermengen augeteilt werden können, so dass nicht beide Vertices der Kante zur selben Untermenge gehören

**Baum:**
Ein Baum ist ein einfacher Graph, in dem man von jedem Vertex einen anderen Vertex auf genau einem Weg erreichen kann.

**Wald:**
Ein Wald ist ein nicht zusammenhängender graph, dessen Komponenten Bäume sind

**Aufspannende Bäume:**
Es sei G ein zusammenhängender Graph mit geschlossenen Wegen. Dann kann man aus den geschlossenen Wegen Kanten so entfernen, dass der Graph zusammenhängend bleibt und trotzdem noch alle Vertices erreicht werden können. Wiederholt man dies bis keine Schleifen mehr übrig sind, dann ist der entstandene Graph ein Baum der alle Vertices des Graphen enthält.

**Minimaler aufspannender Baum:**
Der aufspannende Baum mit der geringsten Summe der Kantengewichte ist ein minimaler aufspannender Baum.

**Algorithmus von Kruskal:**
Dieser sagt aus, dass der minimale aufspannende Baum eines gewichteten zusammenhängenden Graphen gefunden werden soll. (Mehr auf Seite 105)

**Beobachter-Entwurfsmuster:**
Definiert eine Abhängigkeit zwischen einem Objekt und mehreren Anderen. Sobald dieses eine Objekt verändert wird, werden alle anderen benachrichtigt. Man möchte eine zu Enge Kopplung der Objekte vermeiden, da dies die Wiederverwendbarkeit reduziert. Der Vorteil ist, dass die Empfänger automatisch die Nachrichten empfangen (broadcast Kommunikation).

**Model-View-Controller Entwurfsmuster:**
Model: Datenspeicherung
View: Darstellung der Daten für den Benutzer
Controller: Entgegennahmen und Verarbeitung von Benutzereingaben

Das Modell im MVC enthält die der Darstellung im View zugrunde liegenden Daten. Änderungen an den daten werden dem Modell über den Controller mittels des Befehl-Muster mitgeteilt.

**Klausurvorbereitung:**

**SS17**

**Nr. 2a**
**Weshalb sind Implemtierungen allgemeiner Algorithmen mittels generischer Klassen vorzuziehen?**

**Generische Klassen erlauben Typsicherheit. Probleme werden während der Übersetzung und nicht erst zur Laufzeit erkannt.**

**Nr. 2b**
**Erläutern sie die Anwendung von beschränkten Platzhaltern in generischen Klassen an Hand von Vererbungshierarchien:**

**B ist eine erbende Klasse von A. Klasse<B> ist aber keine erbende Klasse von Klasse<A>. Um Instanzen von Klasse<B> und Klasse<A> zu übergeben, muss man mit beschränkten Platzhaltern arbeiten.**

**Nr. 3c**
**Erläutern sie den Zusammenhang zwischen Wrapperklassen und generischen Klassen**

**Wrapperklassen dienen dazu Typumwandlungen von generischen Klassen in atomare Datentypen vorzunehmen, um generische Klassen auf atomare Datentypen anwenden zu können. Zu jedem Datentyp gibt es eine entsprechende Wrapperklasse.**

**Nr. 4**
**Erläutern sie das Singelton Entwurfsmuster, warum ist die Implementierung mittels Enums vorzuziehen**

**Das Singelton-Muster dient dazu, dass es in einer Anwendung in einer Klasse nur eine Instanz gibt. Da Enums in Java Klassen sind, eignen sie sich ideal zur Implementierung von Singeltons, da hierbei die Einzigartigkeit garantiert wird.**

**Nr. 5a**
**Wozu dient das iterator-Muster? Geben sie eins an und erklären sie die Methoden.**

**Ein Iterator arbeitet alle in einer Klasse gespeicherten Elemente sequentiell ab.**

```
public interface Iterator<T> {
        boolean hasNext();
        T next();
}
```

**HasNext fragt, ob weitere unbearbeitete Elemente folgen. Wenn ja, dann liefert next eine Referenz auf das nächste zu bearbeitende Element.**

**Nr. 5b**
**Geben sie ein Java-Beispiel für die Anwendung eines iterators in einer while-Schleife**

```
List<A> list = new LinkedList<A>();
Iterator<A> i = list.iterator();
while(i.hasNext()) {
        A a = i.next();
}
```

**Probeklausur 2015**

**Was bewirkt final, wenn es vor eine Klassendeklaration geschrieben wird?**

**Die Klasse kann nicht als Basisklasse einer anderen Klasse dienen.**

**Was bewirkt Das Schlüsselwort final, wenn es vor eine Klassenmethode geschrieben wird.**

**Die Klassenmethode kann in einer Unterklasse nicht überladen werden. Also darf eine Subklasse keine Implementation dieser Methode enthalten.**

**Was ist die Signatur einer Methode. Welche Rolle spielt der Rückgabetyp?**

**Die Signatur besteht aus dem Methodennamen, sowie den Parametertypen in der vorgegebenen Reihenfolge. Der Rückgabetyp spielt in der Signatur keine Rolle.**

**Weslhab spielt finalize keine große Rolle mehr?**

**Da finalize nur unmittelbar vor der Freigabe der Speicherressourcen durch den Garbage-Collector aufgerufen wird, ist der Aufrufzeitpunkt vom Programmierer nicht vorhersehbar.**

**Was bewirkt das Schlüsselwort abstract, wenn es vor eine Klassenmethode geschrieben wird?**

**Es kann keine Instanz dieser Klasse erzeugt werden. Nur Instanzen nicht abstrakter erbender Klassen können erzeugt werden. Man kann aber eine Referenz auf eine abstrakte Klasse unterhalten.**

**Erläutern sie die Intention des Dekorator Entwurfsmuster.**

**Das Dekorator dient dazu Objekte dynamisch mit neuen Eigenschaften auszustatten und bildet eine alternative zur Subklassenbildung.**

**Geben sie ein Beispiel für die Anwendung des Dekorator in der java-Bibliothek.**

**Die Java-Stream-Bibliothek ist als Dekorator implementiert. Die Streams können dynamisch mit neuen Eigenschaften versehen werden.**

**Erläutern sie die Intention des Fabrikmethode Entwurfsmuster.**

**Eine Fabrikmethode dient der Erzeugung von Klassen, die jedoch nur über Referenzen auf andere Klassen referenziert werden. Dadurch kann die Erzeugung von Objekten polymorph beschrieben werden.**

**Implementieren sie ein Beispiel für die Anwendung des Strategie Entwurfsmuster.**

```
Public class P { public int x, y;}

Collection.sort(list, new Comparator<Punkt>(){
        public int compare (P a, P b) {
                int ca = a.x * a.x + a.y * a.y;
                int cb = b.x * b.x + b.y * b.y;
                return ca – cb ;
        }
}
```

**Erläutern sie die Unterschiede, insbesondere auf das Laufzeitverhalten zwischen einer verketteten Liste und einem Array.**

**Ein Array hat random access Zugriffszeiten, das heisst der zugriff ist O(1) . Das Einfügen und Löschen ist O(N). Bei einer verketteten Liste ist das Verhalten genau umgekehrt. Einfügen und Löschen beträgt O(1) und das Auffinden beträgt O(N).**

**Wie unterscheidet sich eine Deque von einer Priorityqueue?**

**Beides sind Warteschlangen. Eine double ended queue ist eine doppelt verkettete Liste. Es können an beiden Seiten Elemente hinzugefügt oder auch gelsöcht werden. Bei einer Priorityqueue handelt sich sich um eine einfach verkettete Liste. Das heisst Elemente können nur an einer Seite hinzugefügt werden, mit dem Unterschied zur Queue, dass sie nach ihrer Wichtigkeit geordnet werden.**

**Probeklausur 2018**

**Wie lautet die Syntax einer anonymen Klasse?**

```
public Interface Schnittstelle {
        public void methode(){
}
```

```
public class KlasseA{
        public void a(Schnittstelle schnitt){
                (…) ;
        }
}
public class KlasseB {
        public void methode(KlasseA irgendwas) {
                Schnittstelle schnitt = new Schnittstelle(){

                public void methode{
                        (…);
                }
                }
                irgendwas.a(schnitt);
        }
}
```