**Zusammenfassung Theoretische Informatik**

**1. Unified Modeling Language**

**UML:** Die Unified Modeling Language ist eine Sprache und Notation zur Spezifikation, Konstruktion, Visualisierung und Dokumentation von Modellen für Softwaresysteme.

**Klassendiagramm:** Klassendiagramme dienen dazu eine oder mehrere Klassen unabhängig von der Sprache abzubilden. Hierbei sollen alle Eigenschaften, Methoden und Beziehungen zu anderen Klassen dargestellt werden. Bsp.:

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung

**Objektdiagramm:** Ein Objektdiagramm ist ähnlich wie ein Klassendiagramm aufgebaut, bildet allerdings einzelne Objekte mit ihren Attributen ab. Bsp.:

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

**4. Abstrakte Datentypen**

**ADT:** Ein abstrakter Datentyp ist eine Datenstruktur zusammen mit darauf definierten Operationen. In Java ist eine Implementation abstrakter Datentypen durch Klassen und Interfaces möglich. Interfaces geben dabei vor, welche Methoden notwendig sind. In Klassen werden die Methoden dann implementiert.

**5. Liste**

**Def.:** Eine Liste ist eine Folge von Elementen zusammen mit einem sogenannten (ggf. undefinierten) aktuellen Element.

**Methoden:**

* Empty: Liefert true, falls Liste leer ist
* Endpos: Liefert true, wenn das aktuelle Element das letzte ist
* Reset: Das erste Listenelement wird zum aktuellen
* Advance: Der Nachfolger des aktuellen Elements wird zum aktuellen
* Elem: Liefert das aktuelle Element
* Insert: Fügt vor das aktuelle Element ein Element ein und macht das neue Element zum aktuellen
* Delete: Löscht das aktuelle Element und macht den Nachfolger zum aktuellen

**Konzept:**

**Ein Bild, das Diagramm, Entwurf, technische Zeichnung, Reihe enthält.

Automatisch generierte Beschreibung**

**Arten von Listen:**

* Verweisliste: Jeder Eintrag verweist auf das Objekt nach ihm. Das letzte Objekt verweist auf null. Über die Verweise und das aktuelle Element kann auf alle Elemente zugegriffen werden.
* Arrayliste: Alle Inhalte werden in einem zentralen Array gespeichert. Es gibt Variablen, welche den Index der Position und die Anzahl der Elemente speichern.
* Singly-Linked-List: Es wird an das Ende der Liste ein sogenannter Wächter gehangen, auf welchen von dem letzten Element verwiesen wird und welcher wieder auf das erste Element verweist. Die Position zeigt auf das Element vor dem aktuellen Element.
* Doubly-Linked-List: Es gibt wieder ein Wächterelement. Jedes Element zeigt allerdings nicht nur auf seinen Nachfolger, sondern auch auf das Element, welches vor ihm steht.

**Iterator:** Iterator in Java haben folgende drei wichtigen Methoden:

* Hasnext: Liefert true, wenn der Iterator ein nächstes Element hat
* Next: Liefert das nächste Element
* Remove: Löscht das letzte Element, welches zurückgegeben wurde

**6. Keller (Stapel)**

**Def.:** Ein Keller (engl. Stack) ist eine Folge von Elementen zusammen mit einem so genannten Top-Element. Man kann es sich wie einen echten Stapel vorstellen. Kommt ein neues Element hinzu wird es obendrauf gelegt. Wird ein Element entfernt, ist es das oberste.

**Methoden:**

* Empty: Liefert true, wenn der Keller leer ist
* Push: Legt ein Element auf den Keller
* Top: Liefert oberstes Element
* Pop: Entfernt oberstes Element

**Konzept:**

Ein Bild, das Screenshot, Text, Diagramm, Design enthält.

Automatisch generierte BeschreibungEin Bild, das Diagramm, Reihe, technische Zeichnung, Screenshot enthält.

Automatisch generierte Beschreibung

**Type – Casting:** Um in Java zum Beispiel eine Variable des Typen Object als untergeordneten Typen z.b. Schüler zu verwenden muss dieses Object gecastet werden. Dies macht man mit der Klasse, in welche man casten möchte, in Klammern vor dem Object.

**Generic – Types:** Generic Types sind Variable Objekttypen, welche es erlauben Code für mehrere möglichen Eingangsklassen zu definieren.

**Shunting – Yard – Algorithmus:** Unter dem Shunting Yard Algorithmus versteht man einen Algorithmus, welcher eine menschlich verständliche Formel (Infix) in eine maschinell verarbeitbare Formel (Postfix) umwandelt. Bsp.:

* Infix: ((a + b) \* c) / (e – f)
* Postfix: a b + c \* e f - /

Man geht bei dem Algorithmus von Zeichen zu Zeichen und macht folgendes:

* Öffnende Klammer: Man legt die Klammer auf den Stack
* Schließende Klammer: Jedes Element im Stack ausgeben, bis zu einer öffnenden Klammer und beide Klammern löschen
* + oder -: Alle Elemente vom Stack ausgeben, bis zur öffnenden Klammer und das + oder – auf den Stack
* \* oder /: Falls oben auf dem Stack bereits \* oder / liegt dieses ausgeben und dann das neue auf den Stack
* Operand: Sofort ausgeben
* Ausdruck beendet: Den Rest vom Stack ausgeben

**7. Schlange**

**Def.:** Eine Schlange (engl.: Queue) ist eine Folge von Elementen zusammen mit einem so genannten Front-Element.

**Methoden:**

* Empty: Liefert true, wenn die Schlange leer ist
* Enqueue: Fügt ein Element hinten ein
* Front: Liefert vorderstes Element
* Dequeue: Entfernt vorderstes Element

**Konzept:**

**Ein Bild, das Reihe, Text, Quittung enthält.

Automatisch generierte Beschreibung**

**8. Baum**

**Def.:** Ein binärer Baum (engl. Binary tree) ist entweder leer oder besteht aus einem Knoten (engl. Node), dem ein Element und zwei binäre Bäume zugeordnet sind.

**Methoden:**

* Empty: Liefert true, wenn der Baum leer ist
* Value: Liefert Wurzelelement
* Left: Liefert den linken Teilbaum
* Right: Liefert den rechten Teilbaum

**Konzept:**

Ein Bild, das Diagramm, Reihe, Text, Plan enthält.

Automatisch generierte Beschreibung

**9. Traversierungen**

**Def.:** Eine Traversierung eines binären Baumes besteht aus dem systematischen Besuchen aller Knoten in einer bestimmten Reihenfolge. Hierbei besteht ein Knoten immer aus einem „Vater“ einem linken und einen rechten „Sohn“.

**Inorder-Traversierung:** Bei der Inorder-Traversierung gilt die Reihenfolge LVR, also linker Sohn, Vater, rechter Sohn. Die Infix-Notation ist a + b.

Ein Bild, das Kreis, Diagramm enthält.

Automatisch generierte Beschreibung

**Preorder-Traversierung:** Bei der Preorder-Traversierung gilt die Reihenfolge VLR, also Vater, linker Sohn, rechter Sohn. Die Infix-Notation ist + a b.

Ein Bild, das Kreis, Diagramm enthält.

Automatisch generierte Beschreibung

**Postorder-Traversierung:** Bei der Postfix-Traversierung gilt die Reihenfolge LRV, also linker Sohn, rechter Sohn, Vater. Die Infix-Notation ist a b +.

Ein Bild, das Kreis, Diagramm enthält.

Automatisch generierte Beschreibung

**Tiefensuche:** Bei der Tiefensuche gilt immer links vor rechts. Man fängt oben an und geht systematisch so lange nach links, bis es nicht mehr weitergeht und geht dann eins nach oben und nach rechts. (Ist gleich der Preorder-Traversierung).

Ein Bild, das Kreis, Diagramm, Clipart enthält.

Automatisch generierte Beschreibung

**Breitensuche:** Bei der Breitensuche geht man durch jede Ebene des Baumes von oben beginnend und nimmt jedes Element von links nach rechts.

Ein Bild, das Kreis, Diagramm enthält.

Automatisch generierte Beschreibung

**Höhe eines Baumes:** Die Höhe eines Baumes beschreibt, in der wievielten Ebene das letzte Element liegt. (Die Bäume der Suche haben die Höhe 4).

**10. Suchbaum**

**Def.:** Ein binärer Suchbaum ist ein binärer Baum, bei dem alle Einträge im linken Teilbaum eines Knotens x kleiner sind, als der Eintrag im Knoten x und bei dem alle Einträge im rechten Teilbaum eines Knotens x größer sind als der Eintrag im Knoten x.

**Suche:** Die Suche eines Elementes verläuft recht leicht. Man vergleicht den Knoten mit dem gesuchten Element. Ist das Gesuchte größer geht man nach rechts, ist es kleiner nach links. Das wiederholt man, bis man das Element findet.

**Komplexität:** Im Best Case hat jeder Knoten 2 Söhne, der Baum eine Höhe h und   
 Knoten. Die Anzahl der Weg-Knoten ist . Im Worst Case ist der Baum eine Liste und der Aufwand beträgt . Im Average Case beträgt der Aufwand .

**Einfügen:** Das Einfügen eines Elements ist ähnlich der Suche. Man vergleicht das Element, welches man einfügen will mit den Knoten und geht nach links, wenn das Einfügelement kleiner ist und nach rechts, wenn es größer ist. Man wiederholt es so lang, bis man einen leeren Platz findet. Dort fügt man das Element ein.

**Löschen:** Das löschen eines Elements wird für verschiedene Anzahl an Söhnen unterschiedlich gemacht.

* Ohne Söhne: Man kann das Element einfach rausnehmen und die Verbindung zum Vater trennen.
* Mit einem Sohn: Das Sohnelement übernimmt den Platz des gelöschten Elements.
* Mit zwei Söhnen: Man sucht das rechteste Element des linken Sohnes, oder das linkste Element des rechten Sohnes und ersetzt das gelöschte Element durch dieses.

**11. AVL-Baum**

**Def.:** Ein binärer Suchbaum heißt AVL-Baum, wenn alle seine Knoten ausgeglichen sind. Ein Knoten heißt ausgeglichen, wenn sich die Höhe seiner Söhne maximal um 1 unterscheiden.

Ein Bild, das Entwurf, Diagramm, Zeichnung, Muster enthält.

Automatisch generierte Beschreibung

**Ausbalancieren:** Beim Ausbalancieren eines AVL-Baumes beginnt man bei den Gliedern letzter Ebene und bestimmt für alle von unten nach oben den Balance Wert. Sollte ein Balancewert 2 oder -2 sein geht man wie folgt vor:

* Linker Sohn mit linkem Sohn, oder rechter Sohn mit rechtem Sohn: Man rotiert den Baum, sodass der linke Sohn der Vater wird.

Ein Bild, das Kreis, Screenshot enthält.

Automatisch generierte Beschreibung

* Linker Sohn mit rechtem Sohn, oder rechter Sohn mit linkem Sohn: Man rotiert den Baum, sodass der Sohn zweiter Ebene der Vater wird.

Ein Bild, das Kreis, Screenshot, Diagramm enthält.

Automatisch generierte Beschreibung

**12. Heap**

**Def.:** Ein Heap ist ein binärer Baum mit h Ebenen, in dem die Ebenen 0, 1, …, h - 2 vollständig besetzt sind. Die letzte Ebene h – 1 ist von links beginnend bis zum so genannten letzten Knoten vollständig besetzt. Die Knoten enthalten Schlüssel. Der Schlüssel eines Knotens ist kleiner oder gleich den Schlüsseln seiner Kinder.

**Siften:** Beim Siften sortiert man einen Heap, sodass das kleinste Element wieder am Anfang steht. Dabei beginnt man unten und vergleicht den Vater mit seinen Kindelementen. Ist eines von diesen kleiner als der Vater tauscht man beide miteinander aus, ohne die Struktur zu verändern. Dies wiederholt man für jeden Knoten.

**Einfügen:** Beim Einfügen eines Elements beginnt man unten und fügt das Element an letzter Stelle ein. Danach siftet man den Baum, sodass das Element einsortiert wird.

**Löschen:** Zum Löschen eines Elementes ersetzt man es mit dem letzten Element des Heaps und siftet danach.

**13. Priority Queue**

**Def.:** Eine Priority Queue ist ein abstrakter Datentyp. Diese Vorrangswarteschlange operiert auf einer Struktur von Objekten mit einer Größer-Relation für den Schlüssel (Comparable). Man unterscheidet MinPriorityQueue und MaxPriorityQueue, je nachdem ob das kleinste Objekt den Vorrang erhöht oder das größte Objekt.

**Heap als Queue:** Man kann einen Heap ganz einfach als eine Queue darstellen, indem man es per Breitensuchreihenfolge durchgeht und jedes Element abspeichert.

**14. Hashing**

**Def.:** Zum abspeichern und wiederfinden von Objekten wäre eine Funktion hilfreich: f : Objekte -> N. Dann könnte Objekt x bei Adresse f(x) gespeichert werden. Gilt f(x) = f(y) liegt eine Kollision vor, welche bei offenem und geschlossenem Hashing anders behandelt werden

**Offenes Hashing:** Bei offenem Hashing werden kollidierende Elemente in einer Liste verwaltet. Im Array mit dem Index befindet sich also eine Liste mit drei Einträgen.

Ein Bild, das Diagramm, Reihe, technische Zeichnung enthält.

Automatisch generierte Beschreibung

**Geschlossenes Hashing:** Falls f(x) bereits belegt ist wird ein Alternativplatz mithilfe einer anderen Formel gesucht.

* Lineares Sondieren: y + 1, y + 2, y + 3, …
* Quadratisches Sondieren: y + 1, y + 4, y + 9, …
* Double Hashing: y + g(x), y + 2g(x), …

Ein Bild, das Text, Quittung, Zahl enthält.

Automatisch generierte Beschreibung

**16. Graphen**

**Gerichteter Graph:** Ein gerichteter Graph besteht aus Knotenmenge V und Kantenmenge E. Die Kanten können gewichtet sein und zeigen stehts von einem Knoten auf einen anderen Knoten.

Ein Bild, das Reihe, Kreis, Diagramm enthält.

Automatisch generierte Beschreibung

**Ungerichteter Graph:** Ein ungerichteter Graph besteht ebenfalls aus einer Knotenmenge V und einer Kantenmenge E. Die Kanten können ebenfalls gewichtet sein, zeigen aber nicht von einem Knoten auf einen anderen, sondern verbinden lediglich zwei Knoten.

Ein Bild, das Diagramm, Reihe, Kreis enthält.

Automatisch generierte Beschreibung

**Adjazenzmatrizen:** Man kann einen Graphen als Adjazenzmatrix darstellen. Dabei geht man wie folgt vor:

1. Man weißt jedem Knoten einen eindeutigen Index zu (a = 0, b = 1, usw.)
2. Man erstellt eine Liste mit den Indizes als Spalten- und Reihenüberschriften
3. Bei einer ungewichteten Matrix trägt man für jede existierende Verbindung eine 1 und für jede nicht-existierende Verbindung eine 0 ein. Für eine gewichtete Matrix trägt man für jede existierende Verbindung deren Kosten und für jede nicht-existierende Verbindung ein. Hierbei gibt die Reihe stehts den Startwert der Verbindung an.

Ein Bild, das Reihe, Kreis, Diagramm enthält.

Automatisch generierte Beschreibung Ein Bild, das Zahl, Quadrat, Text enthält.

Automatisch generierte Beschreibung Ein Bild, das Zahl, Screenshot, Quadrat, Text enthält.

Automatisch generierte Beschreibung

**Adjazenzlisten:** Man kann Graphen auch als Adjazenzlisten darstellen. Diese bestehen aus den Indizes der Knoten und einem Pfeil auf eine Verbindung zu einem anderen Knoten. Diese Verbindung beinhaltet die Kosten, den Zielknoten und eventuell eine Verbindung zu einer anderen verbindung, welche auch vom Ursprungsknoten abgeht.

Ein Bild, das Diagramm, Plan, Rechteck, Reihe enthält.

Automatisch generierte Beschreibung

**Floyd’s Algorithmus (All Pairs Shortest Paths):** In den Distanzmatrizen bedeuten die Einträge den kürzesten Weg unter Verwendung nur der Knoten 0, 1, 2, bis k als Zwischenknoten, welche auf dem Weg durchlaufen werden dürfen. Die Wegematrizen werden erstellt, indem man auch wieder alle Knoten von 0, 1, 2, bis k betrachtet, diesmal allerdings die Knoten, welche überlaufen werden, einträgt. Überläuft man keine Knoten wird der Startknoten angegeben.

Ein Bild, das Text, Diagramm, Rechteck, parallel enthält.

Automatisch generierte Beschreibung

**22. Maximaler Fluss**

**Def.:** Gegeben sein ein gerichteter Graph , gewichtet mit einer Funktion . Die Gewichtet werden hier als Kantenkapazität interpretiert und beschreiben die maximale Zahl an Transporteinheiten, die pro Zeiteinheit durch diese Kante fließen können. Zwei Knoten sind als Quelle S und Senke t definiert. Gesucht wird immer der maximale Fluss, der von einer Quelle ausgeht, durch die inneren Knoten fließt und an der Senke ankommt. Dabei muss die Flussmenge, welche in einen Knoten reinfließt, gleich der Flussmenge sein, welche aus ihr rausfließt.

Ein Bild, das Reihe, Diagramm, Kreis, Screenshot enthält.

Automatisch generierte Beschreibung

**Vorgehen:** Um den maximalen Fluss in einem Netzwerk zu finden, sucht man sogenannte erweiternde Wege. Ein erweiternder Weg ist ein Pfad von der Quelle s zur Senke t, bei dem für jedes Knotenpaar (a, b) auf dem Pfad mindestens eine der folgenden Bedingungen gilt:

* Es existiert eine Vorwärtskante von a nach b, deren Restkapazität > 0 ist (also deren aktuelle Flussmenge kleiner ist als die Kapazität).
* Oder es existiert eine Rückwärtskante von b nach a, über die man Fluss zurücknehmen kann (d.h. auf dieser Kante fließt bereits Fluss > 0, der reduziert werden kann).

Man erhöht den Fluss entlang eines Pfades um die minimale Restkapazität auf dem Pfad und wiederholt den Vorgang, bis kein erweiternder Weg mehr existiert.

**Cut:** Bei einem Cut trennt man den Graphen zwischen Quelle und Senke, sodass alle möglichen Flüsse von s nach t notwendigerweise Kanten von s nach t überqueren müssen. Der maximale Fluss ist immer gleich der minimalen Kapazität eines Cuts. (Bsp.: Cut = 14)

