OOP - Theorie

1. **Was ist ein Computerprogramm**
   Ist eine Kombination von Computer Anweisungen und Datendefinitionen welche eine Computer Hardware in die Lage versetzen Rechen- oder Kontrollfunktionen wahrzunehmen.

   Bzw: Ein Programm ist die Implementation eines Verfahren um aus gewissen Eingabegrößen durch einen Computer gewisse Ausgabegrößen zu ermitteln
   Bzw.: ein Algorithmus der unter den Regeln einer Programmiersprache auf einem Computer implementiert wird
2. **Was ist Datenkapselung ?**
   Schutz von Daten innerhalb eines Objektes gegen direkten Zugriff von außen. Nur das Objekt kennt seinen Zustand. Um den Status (Variablen, etc) des Objekts zu verändern & abzufragen werden im Objekt eigene Methoden dafür entworfen. Der Aufrufer kennt weiters die Implementierungen des Objekts nicht und kommt nur mit den Bestandteilen des Programms in Berührung mit denen er es auch wirklich muss.
3. **Erkläre das Observer Pattern.**
   Hierbei geht es um Zustandswechsel oder Events, die für andere Objekte relevant sind. Problem: D.h. eine Datenänderung soll allen Interessenten umgehend bekannt gemacht werden. Lösung: Ein Publisher regisriert dynamisch einen oder mehrere Subscriber, die an einem Ereignis interessiert sind, und benachrichtigt sie, wenn ein Ereignis auftritt. Bekannte Applikationen: Synchronisation

   Bzw: Das Observer Pattern wird als Interface implementiert und arbeitet mit dem Subjekt Interface zusammen. Idee : Definiere eine 1:n Abhängigkeit zwischen Objekten, so dass die Änderung des Zustands eines Objekts dazu führt, dass alle Objekte benachrichtig und automatisch aktualisiert werden. Die Änderungen eines Objekts können sich auf andere Objekte auswirken, ohne dass das geänderte Objekt die anderen genauer kennen muss. Das Subjekt trägt sich im Observer ein, um von ihm bei Änderungen benachrichtigt zu werden (zB registerInterest()). Das Subjekt kann sich auch wieder austragen (unregisterInterrest()).
4. **Was ist eine Klasse in OOP ?**
   Eine Klasse ist ein „Prägestempel“ oder Prototyp, der die Variablen und die Methoden gemeinsam für alle Objekte einer bestimmten Art definiert.
5. **Was ist ein Konstruktor?**
   Ein Konstruktor ist eine spezielle Methode, welche zum Initialisieren neuer Objekte, die von einer Klasse kreiert (instanziert) wurden verwendet wird. Hat denselben Namen wie die Klasse. Initiale Werte werden als Argument übergeben. Es kann mehrere Konstruktoren in einer Klasse geben um unterschiedlich viele Initialwerte zu setzen(→ Konstruktoren können überladen werden). Welcher Konstruktor der richtige ist wird an der Anzahl der Argumente erkannt. Besitzt eine Klasse keinen Konstruktor wird ein default-Konstruktor definiert.
6. **Was ist Vererbung?**
   Mechanismus zur gemeinsamen Datenhaltung bzw. um Eigenschaften einer Klasse auf eine andere Klasse zu übertragen. In Java gibt es nur Einfachvererbung dh. Eine Klasse kann nur eine Superklasse haben und nicht(!!!) Mehrere. Etwas Ähnliches wie Mehrfachvererbung (Eine Klasse kann mehrere Superklassen haben) kann mit Hilfe von Interfaces realisiert werden. Eine Subklasse kann Variablen und Methoden

hinzufügen sowie das Verhalten spezialisieren indem bestimmte Methoden durch „spezialisiertere“ überschrieben werden.

7. **Was ist Polymorphismus?**
   Polymorphismus bedeutet direkt übersetzt etwa "Vielgestaltigkeit" und bedeutet, dass eine Objektvariable des Typs X auch Objekte aus allen von X abgeleiteten Klassen aufnehmen kann. (*zB in einen Vector von Zweirad kann man auch Fahrrad, Motorrad – welche von Zweirad abgeleitet wurden - hineinschreiben*). Dies kann elegant dazu genutzt werden, automatische typbasierte Fallunterscheidungen vorzunehmen.
   (*Angenommen, ein Unternehmen verfügt über einen breit gefächerten Fuhrpark von Transportmitteln aus allen Teilen des Ableitungsbaums. Als Unternehmer interessieren dabei natürlich die Kosten jedes Transportmittels pro Monat, und wir würden dazu eine Methode getMonatsKosten in der Basisklasse Transportmittel definieren. Ganz offensichtlich lässt sich diese dort aber nicht implementieren, denn beispielsweise die Berechnung der monatlichen Kosten unseres Fährschiffes gestaltet sich ungleich schwieriger als die der drei Fahrräder, die auch im Fahrzeugfundus sind.*
   Anstatt nun in aufwendigen Fallunterscheidungen für jedes Objekt zu prüfen, von welchem Typ es ist, muss lediglich diese Methode in jeder abgeleiteten Klasse implementiert werden. Besitzt das Programm etwa ein Array von Transportmittel-Objekten, kann dieses einfach durchlaufen und für jedes Element getMonatsKosten aufgerufen werden. Das Laufzeitsystem kennt den jeweiligen konkreten Typ und kann die korrekte Methode aufrufen (und das ist die aus der eigenen Klasse, nicht die in Transportmittel definierte))
8. **Was ist ein Interface?**
   Ein Interface ist eine benannte Sammlung von Methodendefinitionen (ohne Implementierung).
   Durch das bloße Definieren eines Interfaces wird die gewünschte Funktionalität aber noch nicht zur Verfügung gestellt, sondern lediglich beschrieben. Soll diese von einer Klasse tatsächlich realisiert werden, muss sie das Interface implementieren . Dazu erweitert sie die class-Anweisung um eine implements-Klausel, hinter der der Name des zu implementierenden Interfaces angegeben wird. Der Compiler sorgt dafür, dass alle im Interface geforderten Methoden definitionsgemäß in der jeweiligen Klasse implementiert werden.
9. **Was ist ein Objekt in OOP?**
   Objekte nehmen Platz im Speicher ein und haben eine zugeordnete Adresse wie ein Datensatz in Pascal oder eine Struktur in C. Dieser Speicher beinhaltet den Zustand des Objekts zu jedem gegebenen Zeitpunkt, und mit einem Objekt ist eine Menge von Prozeduren/Funktionen verbunden, welche die sinnvollen Operationen an diesem Objekt definieren.
   Bzw.: Ein Objekt ist eine Definitionsmenge von Daten, welche einen bestimmten Speicherbereich belegen, inklusive aller Operatoren und Funktionen die auf dieser Menge definiert sind.
10. **Zweck des Testens ?**
    1.) Um Fehler zu finden
    2.) Für die Verifikation des spezifizierten Verhaltens
    3.) Um die Qualität zu bewerten
11. **Was man für Testing braucht ?**
    1.) Testfälle - Mengen von Test-Inputs, Ausführungsbedingungen und erwarteten Resultaten
    2.) Testkriterien - Kriterien, die erfüllt werden müssen, um einen vorgegebenen Test zu bestehen

3.) Testdokumentation - Spezifikation des Testfalles, Testplan, Testbericht
4.) Tester – Person die diese Aufgabe bewältigt (oft ein Team)☺

12. Was ist der Unterschied zwischen White & Black-Box-Tests ?

Beim White B. Test ist die Struktur des Programms bekannt und somit kann man alle Programmteile sorgfältig prüfen. Dies ist beim Black B. Testen nicht möglich, da man die innere Struktur nicht kennt & nur vom externen Verhalten des Programms Rückschlüsse auf die Fehlerfreiheit machen kann.

13. White box testing ?

Bei dem White Box Testverfahren, auch Struktur-Verfahren bezeichnet, kann das Programm, da die innere Struktur bekannt ist, mit Programmspezifischen Testdaten durchlaufen werden. Das Ergebnis wird mit dem erwarteten Ergebnis verglichen, um so Fehler zu lokalisieren. Es gibt weiters verschiedene Abdeckungsgrade (ob z.B. alle Methoden jeder klasse geprüft wurden..)

14. Black Box testing?

Beim Black Box Testverfahren ist die Programmstruktur nicht bekannt. Die ganze Beurteilung des Testes beruht auf dem externen Verhalten des Programms. Das erwartete Ergebnis kann in diesem Fall nur aus den Spezifikationen (für was ist das Programm geschrieben / zuständig,…) des Programms eruiert werden Das Ergebnis wird wieder mit dem erwarteten Ergebnis verglichen.

15. Arten des Testens ?

- Testen eines Einzelnen Moduls/Einheit/Komponente
  - Stand alone
  - Testumgebung erforderlich
  - White-Box- und/oder Black-Box-Testen
- Integrationstesten
- System / Akzeptanztesten
  - Testen eines vollständig integrierten Systems
  - Evaluierung der
    - Übereinstimmung mit spezifizierten Anforderungen
    - Erfüllung von Abnahmebedingungen
    - Man verwendet Black-Box Testen
- Leistungstesten
- Stresstesten
- Sicherheitstesten
- Regressionstesten

16. Was ist besonders beim Testen in Verbindung mit Objektorierntierung

- Testen einer Klasse als „Einheit“
- Systemtesten, basierend auf Use Cases / Szenarien
- Modellbasiertes Testen unter Verwendung von UML
- Spezialthemen über
  i. Vererbung
  ii. Datenkapselung
  iii. Polymorphismus

17. Was bedeutet Static ?

Eine statische Variable wird auch Klassenvariable genannt .Variablen und Methoden mit dem Attribut `static` sind nicht an die Existenz eines konkreten Objekts gebunden, sondern existieren vom Laden der Klasse bis zum Beenden des Programms. Das `static`-Attribut beeinflußt bei Membervariablen ihre Lebensdauer und erlaubt bei Methoden den Aufruf, ohne daß der Aufrufer ein Objekt der Klasse besitzt, in der die

Methode definiert wurde. Wird das Attribut `static` nicht verwendet, so sind Variablen innerhalb einer Klasse immer an eine konkrete Instanz gebunden. Ihre Lebensdauer beginnt mit dem Anlegen des Objekts und dem Aufruf eines Konstruktors und endet mit der Freigabe des Objekts durch den Garbage Collector.

18. **Was bedeutet synchronized**
    Synchronized wird zB dann auf eine Methode angewendet wenn damit zu rechnen ist, dass von unterschiedlichen Stellen aus gleichzeitig darauf zugegriffen wird (zB von gleichzeitig laufenden threads). Durch synchronized wird das unterbunden.
19. **Erkläre das Singleton Pattern.**
    Der Zweck des S. Patterns ist, nur eine Instanz einer Klasse zu erlauben. Die Zusstandsspeicherung erfolgt in einer Klassen-variable (eine Variable mit dem Klassennamen) um die Klasse nicht mit einer Instanz zu assoziieren. Der Konstruktor dieser Klasse wird als private deklariert, um zu verhindern, dass die Klasse mehrmals intstanziiert wird. In einer statischen Methode (meist getInstance()) wird die einzige Instanz dieser Klasse kreiert. Die Statische Methode kann von jeder anderen Klasse über „klassenname.statischeMethode“ aufgerufen werden.

    Bzw.:

Ein *Singleton* ist eine Klasse, von der nur ein einziges Objekt erzeugt werden darf. Es stellt eine globale Zugriffsmöglichkeit auf dieses Objekt zur Verfügung und instanziert es beim ersten Zugriff automatisch. Es gibt viele Beispiele für Singletons. So ist etwa der Spooler in einem Drucksystem ein Singleton oder der Fenstermanager unter Windows, der Firmenstamm in einem Abrechnungssystem oder die Übersetzungstabelle in einem Parser.

Wichtige Designmerkmale einer Singleton-Klasse sind:

- Sie besitzt eine statische Membervariable ihres eigenen Typs, in dem die einzige Instanz gespeichert wird.
- Sie besitzt eine statische Methode `getInstance`, mit der auf die Instanz zugegriffen werden kann.
- Sie besitzt einen privaten parameterlosen Konstruktor, um zu verhindern, daß andere Klassen durch Anwendung des `new`-Operators eine Instanz erzeugen (er verhindert allerdings auch das Ableiten anderer Klassen).

Eine beispielhafte Implementierung könnte so aussehen:

```
001 public class Singleton
002 {
003   private static Singleton instance = null;
004
005   public static synchronized Singleton getInstance()
006   {
007     if (instance == null) {
008       instance = new Singleton();
009     }
010     return instance;
011   }
012
013   private Singleton()
014   {
015   }
016 }
```

20. Unterschied Klasse – Objekt (1. wichtige Eigenschaft von OO-Programmiersprachen)

Diese Unterscheidung zwischen Objekten und Klassen kann als Abstraktion angesehen werden. Sie bildet die erste wichtige Eigenschaft objektorientierter Sprachen. Abstraktion hilft, Details zu ignorieren, und reduziert damit die Komplexität des Problems. Die Fähigkeit zur Abstraktion ist eine der wichtigsten Voraussetzungen zur Beherrschung komplexer Apparate und Techniken und kann in seiner Bedeutung nicht hoch genug eingeschätzt werden.

21. Kapselung (2. wichtige Eigenschaft von OO-Programmiersprachen)

Diese Zusammenfassung von Methoden und Variablen zu Klassen bezeichnet man als Kapselung. Sie stellt die zweite wichtige Eigenschaft objektorientierter Programmiersprachen dar. Kapselung hilft vor allem, die Komplexität der Bedienung eines Objekts zu reduzieren. Um eine Lampe anzuschalten, muß man nicht viel über den inneren Aufbau des Lichtschalters wissen. Sie vermindert aber auch die Komplexität der Implementierung, denn undefinierte Interaktionen mit anderen Bestandteilen des Programms werden verhindert oder reduziert

22. Wiederverwendung (3. wichtige Eigenschaft von OO-Programmiersprachen)

Durch die Abstraktion und Kapselung wird die Wiederverwendung von Programmelementen gefördert, die dritte wichtige Eigenschaft objektorientierter Programmiersprachen. Ein einfaches Beispiel dafür sind Collections, also Objekte, die Sammlungen anderer Objekte aufnehmen und auf eine bestimmte Art und Weise verarbeiten können. Collections sind oft sehr kompliziert aufgebaut (typischerweise zur Geschwindigkeitssteigerung oder Reduzierung des Speicherbedarfs), besitzen aber in aller Regel eine einfache Schnittstelle. Werden sie als Klasse implementiert und werden durch die Kapselung der Code- und Datenstrukturen die komplexen Details "wegabstrahiert", können sie sehr einfach wiederverwendet werden. Immer, wenn im Programm eine entsprechende Collection benötigt wird, muß lediglich ein Objekt der passenden Klasse instanziert werden, und das Programm kann über die einfach zu bedienende Schnittstelle darauf zugreifen. Wiederverwendung ist ein wichtiger Schlüssel zur Erhöhung der Effizienz und Fehlerfreiheit beim Programmieren.

23. Was ist generell ein Pattern

Als Design-Patterns bezeichnet man (wohlüberlegte) Designvorschläge für den Entwurf objektorientierter Softwaresysteme. Ein Design-Pattern deckt dabei ein ganz bestimmtes Entwurfsproblem ab und beschreibt in rezeptartiger Weise das Zusammenwirken von Klassen, Objekten und Methoden. Meist sind daran mehrere Algorithmen und/oder Datenstrukturen beteiligt. Design-Patterns stellen wie Datenstrukturen oder Algorithmen vordefinierte Lösungen für konkrete und immer wiederkehrende Programmierprobleme dar, allerdings auf einer höheren Abstraktionsebene.

24. **Wichtige Methoden:**

readLine():

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
bzw.: import java.io.*;

public String readLine()
{
	BufferedReader keyb = new BufferedReader(new InputStreamReader(System.in));
	String helpString=null;
	try
	{
		helpString = keyb.readLine();
		return helpString;
	}
	catch(Exception e)
	{
		System.out.println("Fehler beim Einlesen");
		return null;
	}
}
```

Einlesen einer Zahl:

```
int auswahl = 0;
try
{
	auswahl  = Integer.parseInt(readLine()); //parseInt=statische Methode in Kl. Integer
}
catch(NumberFormatException e)
{
	System.out.println("Fehler bei der Eingabe - Zahl erwartet");
}
```

Vector anlegen und durchlaufen und für jedes Vektorelement eine Methode aufrufen:

```
Vector<Meeting>meetings = new Vector<Meeting>();

Iterator<Meeting> i = meetings.iterator();
while(i.hasNext())
{
	Meeting m = i.next();
	m.methode();
}
```

Einlesen eines Datums

```
try
  {
	GregorianCalendar begin = new GregorianCalendar();
	DateFormat formatter = new SimpleDateFormat ("dd.MM.yyyy HH:mm");

	Date datebegin = (Date)formatter.parse(ui.getBegin());

	begin.setTime(datebegin);
	m.setBegin(begin);
	runVar = false;
  }
  catch (ParseException e)
  {
	System.out.println("FEHLER bei der Eingabe! - Nocheinmal eingeben");
	runVar = true;
  }
```

GregorianToString

```
public String GregorianCalendartoString(GregorianCalendar  c)
{
        String s;
        DateFormat formatter = new SimpleDateFormat ("dd.MM.yyyy HH:mm");
        Date date;
        date=c.getTime();
        s=formatter.format(date);

      return s;

}
```