

Patterns

(für VO-OOP ausreichend!)

- listener: events are messages that are sent from one object to another; the component sending the event (aka firing the event) is the producer, the component receiving the event (aka handling the event) is the consumer.

The producer of an event should have the ability to add & delete listeners for the events produced by itself

factory:

This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object; here, we can create objects without exposing the creation logic to the client & refer to newly created object using a common interface.

is based on encapsulation object oriented concept; factory method is used to create different object ~~from~~ from factory often referred as Item & it encapsulate the creation code → so, instead of having object creation code on client side, we encapsulate inside Factory method in Java

singleton :

A singleton in Java is a class for which only one instance can be created provides a global point of access the instance.

The singleton pattern describe how this can be achieved.

Singletons are useful to provide a unique source of data or functionality to other Java objects; for example you may use a singleton to access your data model from within your application or to define logger which the rest of the application can use.

observer :

defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified & updated automatically; the object which is being watched is called the subject; the objects which are watching the state changes, observers or listeners are called

for example, you can define a listener for a button in an user interface; if the button is selected, the listener is notified & performs a certain action

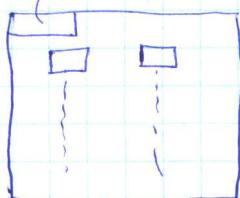
but the observer pattern is not limited to single user interface components

! Sequenzdiagramm:

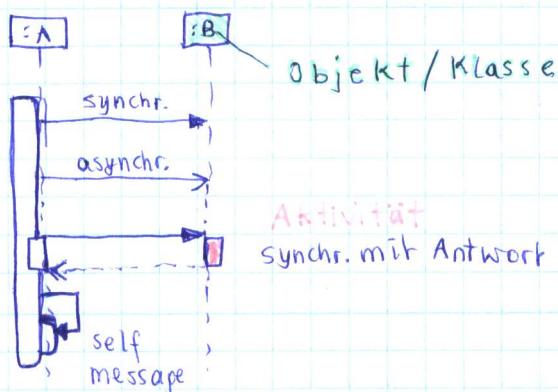
- stellt Interaktionen dar
- beschreibt Austausch von Nachrichten zw. Ausprägungen mittels Lifelines
- zeitl. Ordnung d. Ereignisse (welches nach welchem kommt)
- stellt einen Weg dr. einen Entscheidungsbaum innerhalb eines Systemablaufs dar
- sollten Übersichten über alle Entscheidungsmöglichkeiten entwickelt werden, so müsste für jeden Ablauf ein eigenständiges Sequenzdiagramm modelliert werden

im Kopfbereich steht Schlüsselwort :

- interaction
- sd



von jedem Kommunikationspartner geht eine Lebenslinie aus (gestrichelt)
synchrone Operationsaufruf (ausgefüllte Pfeilspitze)



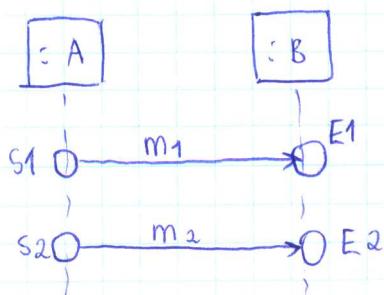
asynchrone Nachrichten mit offener Pfeilspitze

bei synchronen Nachrichten wird die ausgehende Lebenslinie für weitere Nachrichten „blockiert“, bis sie eine Antwort erhalten hat
→ bei asynchronen nicht der Fall

Aktivitätsbalken: zeigt Focus of Control, jenen Bereich, in dem ein Objekt über den Kontrollfluss verfügt & aktiv an Interaktionen beteiligt ist.

Zeitachse hier läuft von oben nach unten, aber nicht als absolute Zeit zu verstehen.
Empfangs (rechts) Ereignis nach sende-Nachricht-Ereignis

bei asynchronen Kommunikation kann E1 nicht nur vor, sondern auch erst nach S2 od. E2 vorkommen



OOP - Theorie

(1) Unterschied zw. (method) overriding & overloading erklären!

- Methode der Superklasse wird überschrieben
- is used to provide the specific implementation of the method that is already provided by its super class
- parameter must be the same
- is the example of runtime polymorphism

- is the example of compile time polymorphism
- parameter must be different
- is used to increase the readability of the program
- Methode innerhalb derselben Klasse überladen
→ Methoden ändern sich dr. Anzahl & Typ der Parameter

(2) Was ist Polymorphismus?

= Vielgestaltigkeit

In Java spricht man davon, wenn 2 Klassen denselben Methodennamen verwenden, aber die Implementierung der Methoden sich unterscheidet;
... ist Eigenschaft, dass der formal gleiche Aufruf einer Methode unterschiedl. Reaktionen (je nach adressiertem Objekt) auslösen kann;
eine Referenzvariable kann zur Laufzeit auf Objekte verschiedenster Typs zeigen → Typ der Ref. var. legt nicht fest, auf welchen Typ von Objekt sie zur Laufzeit zeigt

(3) Welche Arten von Tests kennen Sie?

- Sicherheitstesten
- Integrationstesten
- Blackbox-Test (→ System-, Akzeptanztesten)
- Stressstesten
- Whiteboxtest (→ Testen einer Einheit / Modultest / Komponententest)
- Regressionstesten
- Leistungstesten

(4) Was versteht man unter "Patterns"? Erklären Sie das "Singleton-Pattern"!

- sind verallgemeinerte Lösungsansätze für wiederkehrende Probleme
- stellen wiederkehrende Probleme mit Lösungen in Beziehung, je nach Kontext
- strukturierte Beschreibung mittels Text, Modelle, ...
- verwendet für Gebäudearchitektur, Softwaredesign, Geschäftsprozesse, UI
- eigene Pattern-Sprachen

- Singleton-Pattern:
 - Kontext = Kreieren von Instanzen
 - Problem = zu gewährleisten, dass es nur eine Instanz einer Klasse gibt
 - angewendet bei zB: Window Manager
 - man verwendet Klassenvariable für Zustandsspeicherung, damit bemerkt wird, ob schon Instanz gibt
 - man kreiert nur dann, wenn noch keine Instanz kreiert wurde
 - Instanz innerhalb einer statischen Methode kreiert
("in der Klasse")
 - Deklaration der Constructors als private, um zu verhindern, dass Klasse von außen instanziert wird

5) Erklären Sie 2 Patterns genauer!

- Observer Pattern:
 - Kontext = Zustandswechsel od. Events, die für andere Objekte relevant sind
 - Problem = Datenänderung soll bekannt gemacht werden
 - Lösung = Publisher registriert dynamisch einen od. mehrere Subscriber, die an einem Ereignis interessiert sind & benachrichtigt sie, wenn ein Ereignis auftritt
 - bekannte
Applikation = verwendet für Synchronisation

• Factory-Pattern:

... in this pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common 'interface'

→ instead of having object creation code on client side, we encapsulate inside Factory method 'in Java'

→ wenn man mehrere Konstruktoren hat, ist es oft günstiger, statt den Konstruktoraufruf mit "new" die Objekte über ein od. mehrere Factory-Methoden zu machen

⇒ Factory-Methoden kapseln die Erzeugung von Objekten
man macht dann zB. nur mehr einen Konstruktor, der private ist,
und ruft aus den Methoden diesen auf → von außen ist Konstruktor
nicht mehr aufrufbar

⑥ Geben Sie ein Bsp. in Java für Polymorphismus an!

```
import java.io.*;  
  
abstract class Stift  
{  
    String farbe;  
    public abstract void gibSchreibfarbeAus();  
}  
  
dass Kugelschreiber extends Stift  
{  
    public Kugelschreiber (String e)  
    {  
        farbe = e;  
    }  
  
    public void gibSchreibfarbeAus()  
    {  
        System.out.println ("Schreibfarbe vom Kulli ist " + farbe + ",\n");  
    }  
}
```

⑦ Erklären Sie das Konzept der Daten Kapselung (Data Encapsulation) allg. & anhand eines Bsp's. Wie können Sie das Konzept verletzen? In welchem Zsh. steht es zu Klassenhierarchie & Vererbung?

Datenkapselung = Schutz von Daten innerhalb eines Objekts gegen direkten Zugriff von außen
→ Zugriff nur dr. Methoden (von Objekten)
→ man verwendet das Keyword private, um zB: Attribute, Konstruktoren od. Methoden nur für die eigene Klasse zu nutzen

Bsp: public class Fahrrad
{
 private int gear=0;

 public void setGear (int gear)
 {
 this.gear=gear;
 }

 public int getGear()
 {
 return this.gear;
 }
}

Zsh. zu Klassenhierarchie & Vererbung: auf Attribute, Methoden & Konstruktoren, die mit "protected" gekennzeichnet sind können die eigene & anderen Klassen innerhalb denselben Pakets zugreifen;

derartige Attribute, Methoden werden an alle Subklassen, die auch anderen Paketen angehören ^{können}, weitervererbt & sind dort zugänglich

⑧ Was ist ein Objekt im Sinne von objektorientiertem Programmieren?

- Objekte sind Sammlungen von Operationen (=Methoden), die einen Zustand gemeinsam haben, der von der Außenwelt verborgen ist
- Objekte nehmen Platz im Speicher ein & haben eine zugeordnete Adresse wie ein Datensatz in Pascal od. eine Struktur in C.
Dieser Speicher beinhaltet den Zustand des Objekts zu jedem geg. Zeitpunkt & mit einem Objekt ist eine Menge von Prozeduren/Fkt. verbunden, welche die sinnvollen Operationen an diesem Objekt definieren

⑨ Was ist ein Interface? Klasse vs. Interface!



= Sammlung von Methodendefinitionen (ohne Implementierungen!)
△ einem vereinbarten Verhalten

Klasse

- kann Methoden implementieren
- ist Teil der Klassenhierarchie
- kann nur eine Klasse erweitern

- Klassen können 1 od. mehrere Interfaces mittels "implements" implementieren
→ jede nicht abstrakte Klasse, die ein Interface implementiert, muss alle abstrakten Methoden davon implementieren (→ Implementierung kann auch von einer Superklasse geerbt werden)

- kann nur eine Basisklasse haben, aber beliebig viele Interfaces implementieren

Interface

- kann das nicht
- ist es nicht
- kann beliebig viele Schnittstellen erweitern

- können von mehreren anderen Interfaces abgeleitet werden
- Interface erbt alle Methoden seiner Superinterfaces

⑩ Was ist eine Klasse in OOP?

... ist ein "Prägestempel" od. Prototyp, der die Variablen & Methoden gemeinsam für alle Objekte einer bestimmten Art definiert

11) Was ist ein Konstruktor?

- ... Unterprogramm zum Initialisieren neuer Objekte, die von einer Klasse erweitert wurden
- ... hat denselben Namen wie die Klasse
- ... initiale Werte werden dem Konstruktor vom Aufrufer als Argument übergeben

12) Was ist Vererbung?

- Mechanismus für Halten gemeinsamer Informationen
- Klasse erbt Zustand und Verhalten der Superklasse
- Subklassen können Variablen & Methoden hinzufügen
- Verhalten kann spezialisiert werden

13) Was ist ein Programm?

Computer Program = "a combination of computer instructions & data definitions that enable a computer hardware to perform computational or control functions."

14) Was ist ein Typ? Nenne Typen in Java! Vgl. zu Subklasse & Subtyp!

Antwort davon abhängig, welche Rolle von größtem Interesse ist, die ein Typ spielt.

- Sicht d. Systemprogrammierers: Filter für Interpretation von Rohdaten
- --- d. Implementierenden: Speicher-Abbr. für Werte
- --- d. Typenkontrolle: Kompatibilität v. Operator u. Operand
- OO-Sichtweise: Verhaltensspezifikationen

... in Java: 2 Kategorien

primitiv:

byte
short
int
long
float
double
char
boolean

Referenz:

Arrays
Klassen
Schnittstellen

→ Vgl. Subtyp & Subklasse: jeder Subtyp kann eine Subklasse sein, aber nicht umgekehrt!

weil er sonst die Bedingungen der Ersetzbarkeit erfüllen müsste

→ Vgl. Subtyp & Spezialisierung: Subtyp unterliegt Ersetzbarkeitsbedingungen, Spezial. kann immer sein

(15) Was bedeutet "static"?

... damit werden Felder u. Methoden als "class fields" ausgewiesen



werden von allen Instanzen einer Klasse geteilt

... class fields sind immer verfügbar, auch wenn kein Objekt angelegt wurde

... Aufruf mittels <Klassename>.<classfield>

zB: [public static int a=2;
Rechnen.a]

↳ in der Klasse "Rechnen"

... static-Methoden können somit aufgerufen werden, ohne, dass ein Objekt der Klasse schon besteht

(16) Was bedeutet "synchronized"?

... wird zB: dann auf eine Methode angewendet, wenn von unterschiedl. Stellen des Programms gleichzeitig darauf zugegriffen wird (zB: von gleichzeitig laufenden threads) → dr. synchronized wird das unterbinden

(17) Wozu testet man? Was braucht man fürs Testen?



- um Fehler zu finden
- für Verifikation des spezifizierten Verhaltens
- um Qualität zu bewerten



- Testfälle: Mengen von Testinputs, Aufführungsbedingungen & erwarteten Resultaten
- Testkriterien: Kriterien, die erfüllt werden müssen, um einen vorgegebenen Test zu bestehen
- Testdokumentation: Spezifikation des Testfalles, Testplan, Testbericht
- Tester: Person, die die Arbeit ausführt (oft im Team)

(18) Was ist besonders beim Testen in Verbindung mit objektorientierung?

- Testen einer Klasse als "Einheit"
- Systemtesten, basierend auf Use Cases/Szenarien
- Modellbasiertes Testen unter Verwendung von UML
- Spezialthemen über → Vererbung
→ Datenkapselung
→ Polymorphismus

(19) Worin liegt der Unterschied zw. White-Box & Blackbox-Testen? Erklären Sie jeweils die Testart!

- beruht auf die innere Struktur (man kennt sie)
 - verschiedene Grade der Abdeckung
(ob jede Methode jeder Klasse geprüft wird)

- Programmstruktur unbekannt
 - beruht auf externem Verhalten
 - Evaluierung der Übereinstimmung mit Spezifikation

(für was ist das Programm geschrieben/zuständig?)

Ergebnis des Tests wird mit dem erwarteten Ergebnis verglichen

(20) Was sind "Generics"?

- generische Programmierung
 - generics $\hat{=}$ parametrisierte Typen
 - Typ-Variablen repräsentieren zum Zeitpunkt der Implementierung unbekannte Typen
 - erst bei Verwendung der Klassen, Schnittstellen & Methoden werden die Typ-Variablen dr. konkrete Typen ersetzt
 - ... sind ein allg. Konzept, um Typsicherheit zu gewährleisten

(21) Was bedeutet das Keyword "abstract"? In Verbindung mit einer Klasse?
Unterschied zw. Konstruktor & abstrakter Instanztyp?

- eine Klasse, die "abstract" ist, kann nicht instanziiert, aber spezialisiert werden
 - eine Methode, die abstract ist, besteht nur aus der Methodensignatur
 - nur Deklaration, keine Implementierung
 - nur in abstract Klassen möglich
 - Interface-Methoden sind "per default" abstract

Zum Vgl.: Konstruktor := Unterprogramm zum Initialisieren neuer Objekte, die von einer Klasse kreiert wurden

(22) Deklarieren vs. Instanziieren!

// Deklaration
Fahrzeug fz;
Klassenname Instanzname

// Instanzierung

fz = new Fahrzeug ("Volvo");

↓
Instanzierungsoperator

↓
Konstruktor

↓
Stringkonstante

(23) Was sind exceptions?

man kann die Fehlerbehandlung dr. "exceptions" machen

- auslösen einer Ausnahme ("throws"), sobald ein Fehler eintreift
- Abfangen ("catch") einer Ausnahme in einer übergeordneten Methode

Java-Methoden lösen eine Ausnahme aus, wenn sie aus irgendeinem Grund versagen → Programmsteuerung gibt unmittelbar an den entsprechenden "exception handler" weiter

- dr. throws "Exceptionklasse" geben Methoden an, welche Ausnahmen sie auslösen könnten, od. selbst nicht abfangen werden
- diese Ausnahmen muss man irgendwo im Code abfangen, sobald Methode verwendet wird

(24) Unterschied zw. Klasse & Objekt nennen!

Unterschied = einer Abstraktion

→ Abstraktion hilft, Details zu ignorieren und reduziert Komplexität des Problems

dr. Abstraktion & Kapselung kann man Wiederverwendung erreichen, zB: mit "Collections"
(= Objekte, die Sammlungen anderer Objekte aufnehmen & auf eine bestimmte Art verarbeiten können)

→ dr. Wiederverwendung erreicht man höhere Effizienz & Fehlerfreiheit beim Programmieren

(25) Was sind Sub-/ Superklassen?

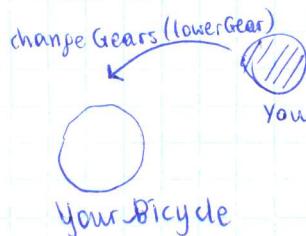
... ist eine Klasse, die eine andere Klasse "erweitert"

ist eine Klasse, von der eine bestimmte Klasse abgeleitet wird, auch indirekt

(26) Was ist eine "message"?

Softwareobjekte interagieren & kommunizieren miteinander, indem sie sich Messages senden

man braucht dazu (erklärt an einem Bsp):



Your Bicycle.change Gears (lower Gear);

empfangene Objekt

Parameter

Name d. Methode

27) Was versteht man unter "Ersetzbarkeit"? Wie lauten die Bedingungen? Was sind die direkt prüfbaren Bedingungen?

- jede Instanz eines Subtyps kann verwendet werden, wo auch immer eine Instanz des Supertyps erwartet wird
- wichtig für Wiederverwendung
- ist Anforderung an Vererbbarkeit bezogen auf Typen

Bedingungen:

- kompatible Schnittstellen
- "kompatible" Resultate (was Zusicherungen angeht)

• vor Methodenauftrag:
was gibts für Bedingungen an
Methoden

- Pre-conditions von Methoden
- Post-conditions von Methoden
- Invariants

• nach Aufruf: was sichert sie mir zu, was ich damit machen kann

Übergabe parameter (umgekehrt zu Kovarianz)

direkte prüfbare Bedingungen:

- Kontravarianz (man geht entgegen der Richtung) ↗↑
- Kovarianz (man geht in "Richtung der Vererbung") ↘↓
- Invarianz (gilt für alles andere, zB: auf Attribute, die gleich bleiben in Sub- & Supertypen)
für Rückgabewerte (nach unten hin weiter, nach oben spezieller)

28) Was macht/ist ein Sequenzdiagramm / Klassendiagramm?

- stellt Interaktionen dar
- zeitl. Ordnung d. Ereignisse
- stellt eine Möglichkeit dr. einen Entscheidungsbaum innerhalb eines Systemablaufs dar
- von jedem Kommunikationspartner (Objekt/Klasse) geht eine Lebenslinie aus
- Zeitachse von "oben nach unten"
- bei synchronen Nachrichten wird ausgehende Lebenslinie für weitere Nachrichten blockiert, bis sie eine Antwort erhalten hat (im Vgl. zu asynchronen)

→ Klassendiagramm := ist ein Strukturdigramm der UML zur graphischen Darstellung (Modellierung) von Klassen, Schnittstellen sowie deren Beziehungen

29) Was sind packages?

... enthält eine od. mehrere Klassen (Klassennamen müssen nur innerhalb eines packages eindeutig sein → vermeidet mögliche Namenskonflikte zw. Klassen)

mit einem "*" kann man bei "import java.util.*" alle Klassennamen des packages bzw. alle Variablen & Methoden einer Klasse importieren

(30) Was ist "casting"?

damit macht man eine Typumwandlung

→ implizit, wenn man von einem niedrig- zu einem höherwertigen Typ wandelt, wie zB: int → long ⇒ keinen eigenen Operator, es wird einfach ausgeführt im Hintergrund

→ explizit, wenn man einen cast-Operator verwendet (hier kann auch von einem höher- zu niedrigwertigen Operator gewandelt werden)

der Zieltyp steht in den Klammern davor!

(31) Was sind "Enumerations"?

ein Aufzählungstyp (= enumeration type) kann als eine bestimmte Art einer normalen Klasse angesehen werden & wird mittels "enum"-Schlüsselwort deklariert

↳ kann Enum-Konstanten enthalten, dafür gilt: (property modifier)

↳ sie definiert eine Instanz eines Aufzählungstyps

(32) Erklären Sie die Testarten außer Black- & Whiteboxtesten!

- Unit-Test:

Testen einer Einheit;

zB: JUnit-Test in Java;

→ 1 Testfall pro Methode zB

- Regressionstest:

immer wieder ausgeführt, wenn Änderungen gemacht werden;
hängt auch von anderen Testfällen ab

- Stress-testen:

Programm ausreizen

→ viele Daten hinschicken & auf Zeit achten

- Leistungstesten:

ähnlich zu Stress-testen, aber Umfang statt Zeit spielt eine Rolle!

- Usability-Test:

UI wird getestet mit Testpersonen

- Abnahmetest:

dem Kunden wird nach einer Zeit das bisherige Projekt gezeigt, um zu sehen, ob es den anfangs abgemachten Anforderungen entspricht

- Sicherheitstesten:

Sicherheitslücken vom Code herausfinden