

# Software eng - Ausarbeitung 2.

## Prüfung 2018 und weitere Fragen

5. November 2018

### 1 Scala und Methoden

**1.1 (a) Was ist der Unterschied zwischen statischen und dynamischen Typ einer Variable in typisierten objektorientierten Sprachen wie zum Beispiel Java? Definieren sie dynamische Bindung. (b) Was versteht man unter Call-Back-Style of Programming? Wie kommt dieser in objektorientierten Sprachen wie in Java vor?**

#### 1.1.1 (a)

- statischer Typ
  - Typ einer Variable, der bei der Variablendeklaration angegeben wird. Dieser ist schon beim Compilieren bekannt.
  - Bsp. Counter c; //c ist statischer Typ Counter
- dynamischer Typ
  - Typ einer Variable, der erst zur Laufzeit bekannt ist und kann vom statischen Typen abweichen.
  - Nur bei Vererbung möglich
  - Bsp. (LtDCouter -> Unterklasse von Counter). Counter c; c = new UnderClassCounter(23); //dynamischer Typ von c ist LTD Counter
- dynamische Bindung
  - Beim ausführen von Methoden wird während der Laufzeit automatisch die Methode vom dynamischen Typ genommen und nicht vom statischen

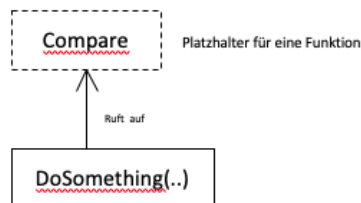
#### 1.1.2 (b)

- Funktionen und Procedures als Parameter

- wenn dynamisches laden und linken im Laufzeitsystem von Programmteilen unterstützt wird, können Prozeduren und Funktionen in einem ausgeführten Softwaresystem hinzugefügt und verwendet werden -> problematisch in nicht-objektorientierten Sprachen

### Beispiel in C:

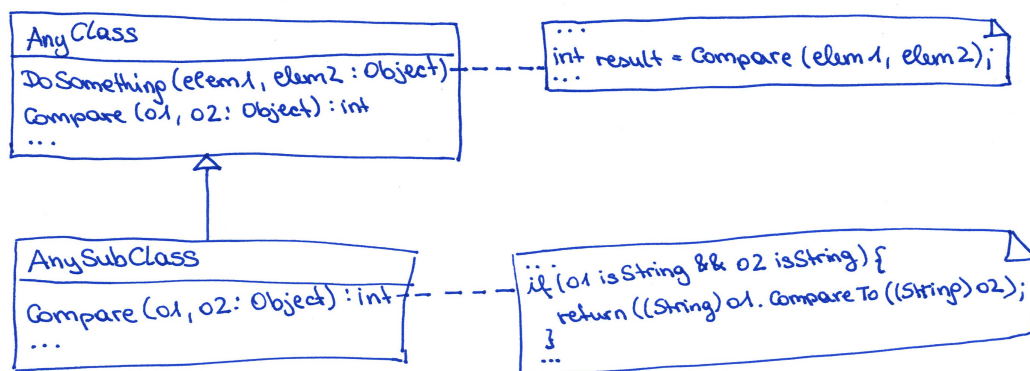
`void DoSomething(int(*Compare)(void*, void*), void* elem1, void* elem2)`



`DoSomething()` hat als ersten formalen Parameter einen Funktionsparameter (`Compare()`). Vergleichsobjekte werden wegen Verallgemeinerung nicht festgelegt, der generische Typ `void` wird verwendet. Wenn `StringCompare` als Parameter an `DoSomething()` übergeben werden soll, muss die Typüberprüfung durch einen Typcast auf `(*char)` umgangen werden.

Objektorientierte Programmiersprachen ermöglichen mit Vererbung den "Call-Back-Style of programming" syntaktisch eleganter und unter Beibehaltung der Typprüfung zu realisieren.

**Siehe UML:**



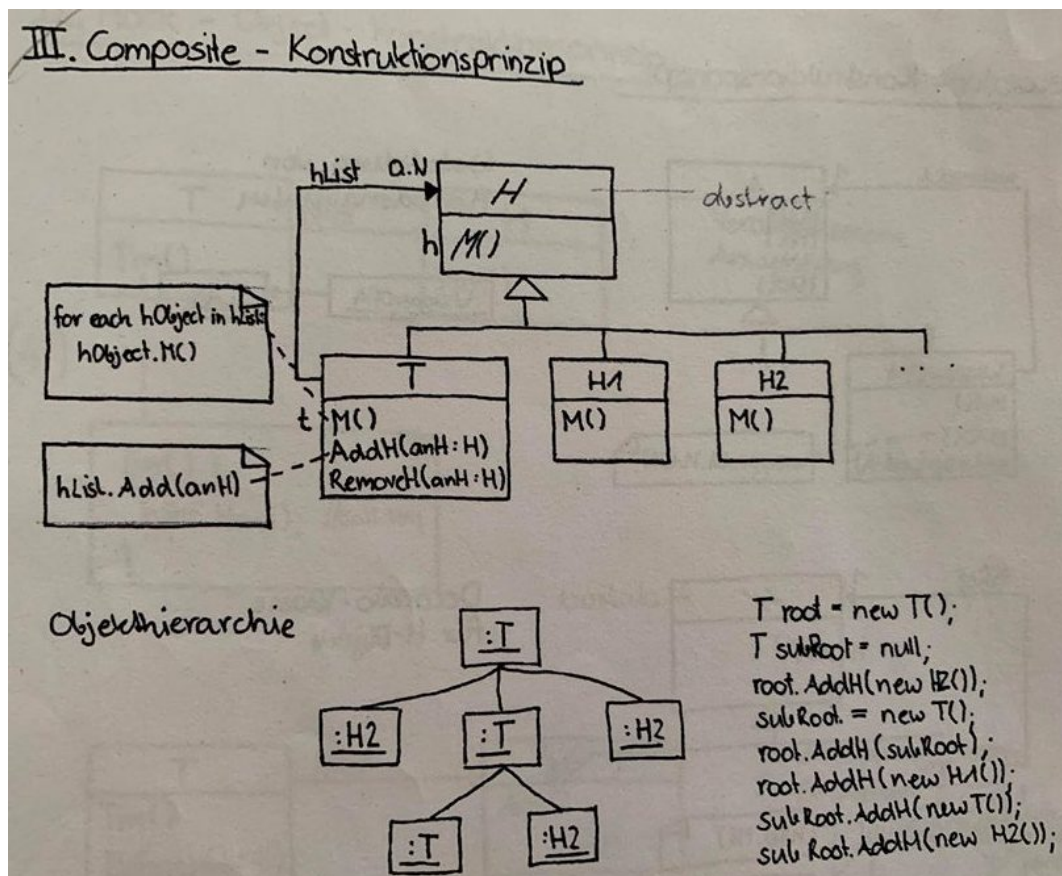
Durch Überschreibung der Methode `Compare()` in einer Unterklasse von `Anyclass` wird quasi eine spezifische Funktionsimplementierung als Parameter von `DoSomething()` und an andere die `Compare` aufrufen übergeben.

**1.2 Erläutern sie im Detail das Decorator Konstruktionsprinzip (Composite Pattern) anhand eines UML-Diagramms und beschreiben Sie dessen Vor- und Nachteile. Zeigen Sie, wie eine Objekthierarchie damit definiert wird. Beschreiben Sie die Struktur und die Eigenschaften der Template- und Hook-Methoden. Ist die Template-Methode beim Composite Konstruktionsprinzip eine rekursive Methode? Begründen Sie die Antworten genau**

- Methoden zur Verwaltung der H-Objekte (Add/Remove)
- Die Implementierung einer Methode M() ist in der Template Klasse dadurch definiert, dass über die vom T-Objekt referenzierten H-Objekte iteriert wird und für jedes H-Objekt Methode M() aufgerufen wird
- Konstruktionsprinzip:
  - Template Methode: M() die über H-Objekte iteriert
  - Hook Methode: M() jedes einzelnen Objekts (wird beim iterieren aufgerufen)
  - Damit können Objekthierarchien komponiert werden - hierarchische Beziehungen ausdrücken
- Objekthierarchie
  - T root = new T();
  - T subRoot = null;
  - root.AddH(new H2());
  - subRoot = new T()
  - root.AddH(subRoot)
  - root.AddH(new H1())
  - subRoot.AddH(new T())
  - subRoot.AddH(new H2())
- Aufgrund der Struktur von Template Methode M() (iteriert und ruft M() auf) kann die Hierarchie wie ein einziges Objekt behandelt werden -> Vorteil der automatischen Iteration
- Dadurch scheint Template-Methode rekursiv, ist sie aber NICHT, nicht M() selbst, sondern die zum H-Objekt gehörende Methode M(). Jedes Objekt hat Methode bei sich (die gleiche, aber nicht die selbe)
- Datenstruktur ist aber rekursiv
- Vorteile
  - auf einfache Weise anpassbar und flexibel verwendbare Objekthierarchien können gebildet werden

- es können problemlos Unterklassen der Hook-Klasse definiert werden und für die Bearbeitung von Objekthierarchien herangezogen werden, ohne Template-Klasse ändern zu müssen
- Objekthierarchien kommen häufig vor: zB gruppierte GUI Elemente
- Nachteile
  - liegt in der Komplexität der Interaktionen der Objekte in Hierarchie, um automatische Iteration über Baumhierarchie durchzuführen. Iteration von Template Methode realisieren.

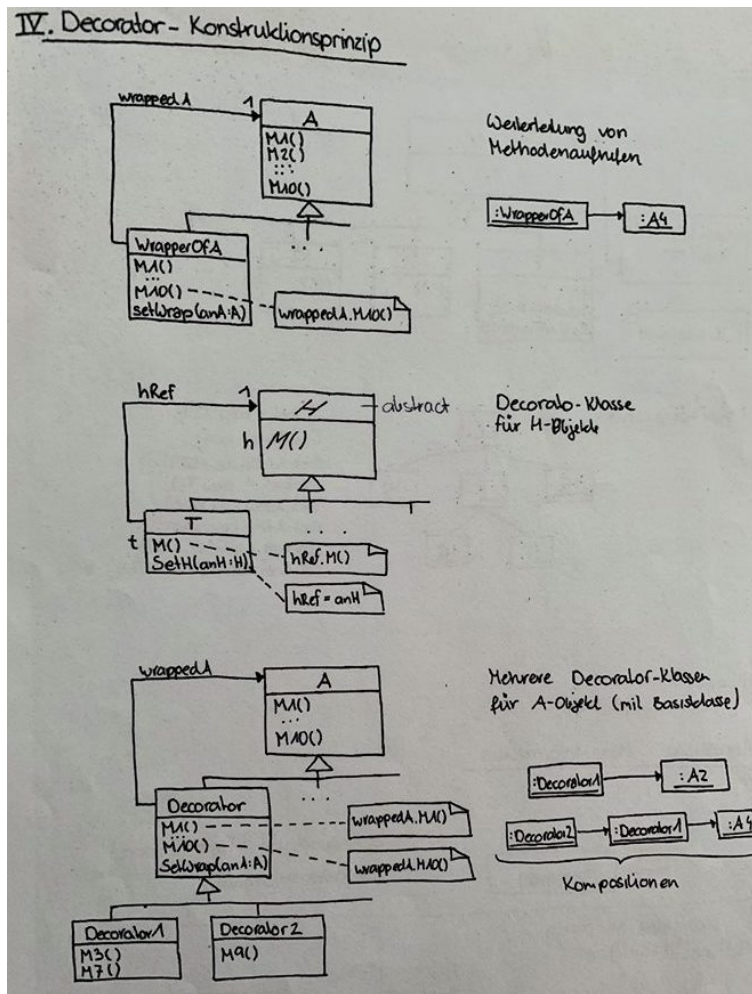
Siehe UML:



**1.3 Erläutern Sie im Detail das Decorator Konstruktionsprinzip (=Decorator Pattern) und dessen Vor- und Nachteile. Zeigen Sie, welche Datenstruktur damit typischerweise definiert wird. Was ist der wesentliche Unterschied zum Composite Konstruktionsprinzip, wenn man ausschließlich die Beziehung zwischen der Template- und Hook-Klasse betrachtet? Erläutern Sie im Detail, wozu das Decorator Konstruktionsprinzip typischerweise verwendet wird und welche Voraussetzung dabei erfüllt sein muss?**

- Decorator-KP löst auf elegante Weise durch Komposition statt Vererbung
- In Klasse WrapperOfA werden alle Methoden von A überschrieben, indem Methodenaufruf an ein über die Instanzvariable wrappedA referenziertes Objekt delegiert wird. Zuweisung über SetWrappedA()
- Da Unterklasse: überall wo statisch A gefordert wird, ist WrapperOfA möglich da Instanzvariable statisch kann sie auf jedes Objekt einer Unterklasse von A verweisen
- Indirektion bei Methodenaufruf wird eingeführt (Aufruf weitergeleitet)
- WrapperOfA wird Decorator von A, wenn in weitergeleiteten Methodenaufruf zusätzlich Verhalten verändert wird (wie Filter)
- Instanz von T zusammen mit Unterklasse von H kann wie H-Objekt verwendet werden
- Da Decorator selbst Unterklasse von H ist, können beliebig viele Filter vor H-Objekt gestellt werden
- Für mehrere Decorator-Klassen ist Basisklasse für Weiterleitung sinnvoll, Unterklassen können dann beliebig Methoden überschreiben

**Siehe UML:**



- Vor- und Nachteile

- Anpassung des Verhaltens eines Objekts durch Komposition mit einem oder mehreren Decorator-Objekten
- Nützlich in Situationen, wenn in Wurzelklasse eines umfangreichen Teilbaums einer Klassenhierarchie die Funktionalität von Methoden geändert werden soll.
- Klassenhierarchie mit Klasse A (Wurzel eines Teilbaumes) stellt Methoden M1()-M10() bereit und Implementierung von M3()-M7() soll geändert werden.
- Quelltext der Klasse A ändern nicht ideal- ungewollte Nebeneffekte
- Wenn kein Quelltext kann angestrebtes Verhalten über Vererbungskonzept realisiert werden.
- neue Unterklassen bilden mit Überschreiben von Methoden, sehr aufwendig nicht ideal (auch nicht über MixIn-Klassen bei Mehrfachvererbung)

1.4 Vergleichen Sie die vier Konstruktionsprinzipien (alle mit Ausnahme von Chain-of-Responsibility) objektorientierter Plug-In-Architekturen tabellarisch, anhand folgender Kriterien: Charakteristika der Template- und Hook-METHODEN: (Positionierung, Namensgebung, Vererbungsbeziehung), Anzahl der beteiligten Objekte und die Art der Anpassung?

		Konstruktionsprinzipien				
		Hook-Method	Hook-Object	Composite	Decorator	COR
Charakteristika von Template- und Hook-Methoden	Positionierung	T() und H() in einer Klasse	T() und H() in getrennten Klassen			T() = H()
	Namensgebung	verschiedene Namen		Namen von T() und H() sind gleich		
	Vererbungsbeziehung	n.a.		H erbt von T		T = H
Anzahl der beteiligten Objekte		1	1(T) + 1(H) oder 1(T) + N(H)	N Objekte, die wie ein Objekt verwendet werden		
Anpassung		durch Vererbung und Instanziierung der entsprechenden Klasse	durch Komposition (bei Bedarf zur Laufzeit)			

## 1.5 Scala Traits versus Java Interfaces (a) Beschreiben Sie Unterschiede zwischen den beiden Sprachkonstrukten. Handelt es sich dabei um mehrfache Vererbung? Begründen Sie Ihre Antwort. (b) Was ist der Unterschied zwischen einem Thin Interface und einem Rich Interface? (c) Wie umgeht Scala Probleme der Mehrfachvererbung durch linearization, verwende Pfeile des UML Diagramms?

Code zu (b) -> Erläutern Sie das an folgenden Scala Beispiel Code:

```
trait Ordered[T] {  
  def compare(that: T): Int  
  def <(that: T) : Boolean = (this compare that) < 0  
  def >(that: T) : Boolean = (this compare that) > 0  
  def <=(that: T) : Boolean = (this compare that) <= 0  
  def >=(that: T) : Boolean = (this compare that) >= 0  
}  
  
class Rational(n: Int, d: Int) extends Ordered[Rational]{  
  //...  
  def compare(that: Rational) = (this.number * this.denom) -  
    ↪ (that.number * this.denom)  
}
```

Code zu (c) um Linearisierung zu Zeichnen

```
class Animal  
trait Furry extends Animal  
trait HasLegs extends Animal  
traits FoulLegged extends HasLegs  
class Cat extends Animal with Furry with FourLegged
```

**Antworten:**

### 1.5.1 (a)

- Scala Traits und Interfaces dürfen abstracte Methoden und Default-Methoden haben -> Unterschied in Auflösung, welche Methode verwendet wird, wenn eine von 2 Interfaces/Traits implementiert wird
- Bei Interfaces wird dies durch eine extra Syntax angegeben, bei Traits ist dies durch die Auflösungsreihenfolge vorgegeben.
- ES GIBT NICHTS VERGLEICHBAR GUTES ZU SCALA TRAITS!!!!!!
- "Diamond Problem" entsteht durch die strikten Vererbungsregeln nicht
- Traits müssen nicht in einer extra Klasse implementiert werden, können direkt in der Deklaration angegeben werden



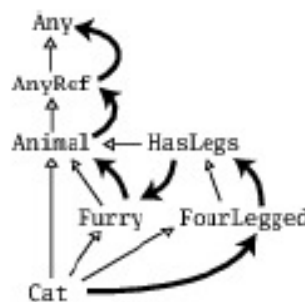
- Bsp: `val variable = new Class with Trait` -> In Scala handelt es sich nicht um Mehrfachvererbung, da immer klar ist welche Methode gerade verwendet wird.
- seit Java 8 gibt es Mehrfachvererbung mit Default-Methoden

### 1.5.2 (b)

- Rich Interface
  - viele Methoden, macht es für den Caller leicht
- Thin Interface
  - weniger Methoden, ist also einfacher zu implementieren
- Erklärung zu Code von b
  - Beim Rich Interface müssen alle 4 Methoden (`<`, `>`, `>=`, `=<`) implementiert werden
  - Beim Thin Interface muss nur eines implementiert werden (`<`). Es ist aber nicht möglich andere aufzurufen (`>`, `>=`, `=<`).
  - Beide Fälle sind nicht perfekt
  - Der Obige Code für (b) ist die Erklärung und Lösung um dieses Problem loszuwerden, weiters wird auch noch ein Rationales (Bruch) Objekt im Code implementiert welches verschiedene Ausgaben darstellt
  - Bsp - `val half = new Rational(1,2) = 1/2`, gleiches für `third = 1/3`
  - mittels Compare Methode kann man die beiden dann vergleichen `half < third` -> `Boolean = false`

### 1.5.3 (c)

Linearization ist die spezielle Auflösungsreihenfolge von Traits, dies zeigt uns der oben angegebene Code zu (c) im Detail. Eine Zeichnung zu der Linearization sehen wir hier:



Linearization ist mit dicken Schwarzen Pfeilen gekennzeichnet, die Vererbungsbeziehung ist wie gehabt mit UML-Pfeilen gekennzeichnet.

Erklärung wie Linearization umgesetzt wird:

1. Nur die erste Klasse wird stehen gelassen, alle anderen werden umgedreht
  - Cat -> FourLegged -> Furry -> Animal

2. Dann von links nach rechts -> alle Traits durch ihre "Linearization" jeden Trait dazuschreiben den der Trait extended, und wenn dieser weider eine Trait extended den auch dazuschreiben usw.
  - Cat -> FourLegged -> HasLegs -> Furry -> Animal -> AnyRef -> Any

## 2 Machinelearning

### 2.1 Was ist der Unterschied zwischen Supervised und Unsupervised Learning? Welche Kategorie ist k-means zugeordnet? Erklären sie zumindest drei typische Parameter wenn sie k-means aus der scien-kit-learn-Phyton-Bibliothek anwenden.

- supervised machine learning: Bei supervised machine learning geht es darum von einem input zu einem output zu kommen wobei der output bekannt ist. Es wird also eine Funktion gesucht um von einem bestimmten input zu einem bestimmten (bekannten) output zu kommen. Das Ziel ist eine möglichst gute Funktion für dieses mapping zu bekommen, sodass auch Daten mit unbekannten output möglichst gut evaluiert werden können.
- unsupervised machine learning:
- unsupervised vs. supervised learning: Bei unsupervised machine learning gibt es keinen bekannten output. Hier ist das Ziel die zugrunde liegende Struktur oder Verteilung in den Daten zu modellieren, um mehr über die Daten zu erfahren.
  - k-means, clustering und PCA (Principal component analysis)
  - Reinforcement learning (supervised learning) -> MDPS oDA SO KA
- K-means ist unsupervised learning zugeordnet

## 2.2 Erläutern sie folgenden Python-Code im Detail -> (ist das Classification Algorithmus?)

```
0. import pandas
1. import numpy
2.
3. data = pandas.read_csv('data.csv')
4.
5. x = numpy.array(data[['x1', 'x2']])
6. y = numpy.array(data['y'])
7.
8. from sklearn.linear_model import LogisticRegression
9. from sklearn.ensemble import GradientBoostingClassifier
10. from sklearn.svm import SVC
11.
12. classifier = LogisticRegression()
13. classifier.fit(X,y)
14.
15. classifier = GradientBoostingClassifier()
16. classifier.fit(X,y)
17.
18. classifier = SVC()
19. classifier(X,y)
```

wobei data.csv unter anderem folgende Daten enthält:

```
x1,x2,y
0.78051,-0.063669,0
0.28774,0.29139,0
0.40714,0.17878,0
0.2923,0.4217,0
...
0.15254,0.2168,1
0.45558,0.43769,1
0.28488,0.52142,1
...
```

### Code Beschreibung:





- In den Zeilen 0. und 1. werden diverse Pakete importiert
- In der 3. Zeile werden die Daten aus dem CSV-File 'data.csv' in dem Data Frame 'data' gespeichert.
- In den Zeile 5. und 6. ... (folgt)
- In den Zeilen 8-10 werden diverse imports von Paketen durchgeführt
- In Zeile 12. wird eine Instanz von dem Modell angelegt.  
Logistische Regression ist ein Machine Learning classification Algo, der verwendet

wird, um die Wahrscheinlichkeit einer kategorial abhängigen Variablen vorherzusagen.

- In Zeile 13. wird das Training durch den Aufruf der Methode `fit()` durchgeführt. Dieser Methode werden die Trainingsdaten, also die Eingabedatensätze `X` und die zugehörige Sollausgabe `y` übergeben.
- In Zeile 15 wird die Funktion `GradientBoostingClassifier()` angewendet. GB baut ein additives Modell auf eine stufenweise vorwärts gerichtete Art auf; Es ermöglicht die Optimierung beliebiger differenzierbarer Verlustfunktionen. In jeder Stufe sind  $n_{classes\_Regressionsbäume}$  an den negativen Gradienten der Binomial- oder Multinomialabweichungsverlustfunktion angepasst. Die binäre Klassifizierung ist ein Sonderfall, bei dem nur ein einzelner Regressionsbaum induziert wird.
- In Zeile 16. wird das Training durch den Aufruf der Methode `fit()` durchgeführt. Dieser Methode werden die Trainingsdaten, also die Eingabedatensätze `X` und die zugehörige Sollausgabe `y` übergeben.
- In der Zeile 18. wird auf classifier die Funktion `SVC()` (Support-Vektorklassifizierung) angewendet. Die Implementierung basiert auf `libsvm`. Die Fit-Time-Komplexität ist mit der Anzahl der Samples mehr als quadratisch, was die Skalierung auf Datensätze mit mehr als einigen 10000 Samples erschwert. Die Multiclass-Unterstützung wird nach einem Eins-gegen-Eins-Schema behandelt.

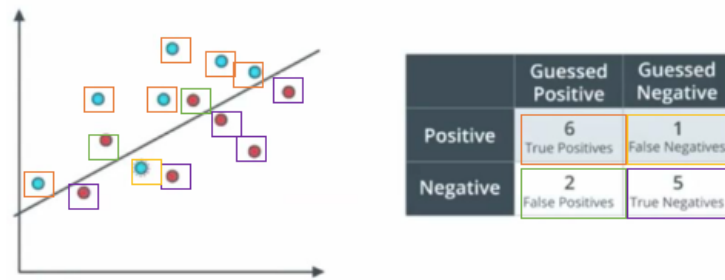
## 2.3 Was versteht man unter "Confusion Matrix"? Erläutern Sie diesen Begriff am Beispiel einer medizinischen Diagnose.

- Confusion Matrix:
  - Eine Confusion Matrix ist eine Tabelle, die häufig verwendet wird, um die Leistung eines Klassifizierungsmodells für eine Gruppe von Testdaten zu beschreiben, für die die wahren Werte bekannt sind. Sie ist eine Metriken, die zur Ermittlung der Korrektheit und Genauigkeit eines Modells verwendet wird. Es wird für Klassifizierungsprobleme verwendet, bei denen die Ausgabe aus zwei oder mehr Arten von Klassen bestehen kann.
- Bsp. Medizinische Diagnose:
  - Angenommen wir möchten aus einer Anzahl von Personen herausfinden, wie viele Personen krank sind.
  - Es kann folgendes Modell dafür erstellt werden:

	Diagnosed Sick	Diagnosed Healthy
Sick	 True Positive	 False Negative
Healthy	 False Positive	 True Negative

- Das Modell bedeutet folgendes
  - Die Personen im orangenen Feld (True Positive) sind krank und wurden auch als krank diagnostiziert.
  - Die Personen im gelben Feld (False Negative) sind krank und wurden aber als gesund diagnostiziert.
  - Die Personen im grünen Feld (False Positive) sind gesund, wurden aber als krank diagnostiziert.
  - Die Personen im lilanen Feld (True Negative) sind gesund und wurden auch als gesund diagnostiziert.
- Die Personen im grünen Feld (False Positive) zählen auch zu den kranken Personen.

Ein Beispiel der Klassifizierung sieht wie folgt aus:

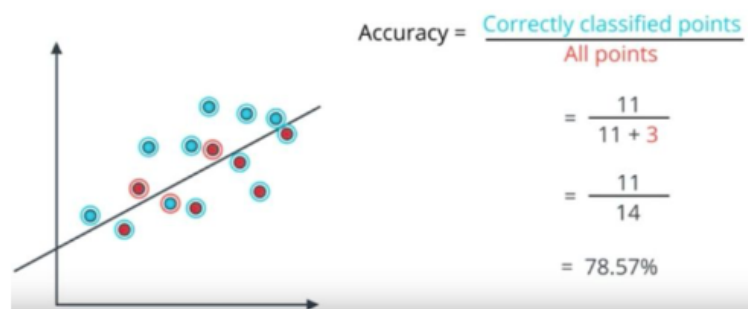


Um herauszufinden, wie viele Patienten wir richtig klassifiziert haben, gibt es eine Möglichkeit dies zu berechnen:

	Diagnosed sick	Diagnosed healthy
Sick	1,000	200
Healthy	800	8,000

$$\text{Accuracy} = \frac{1,000 + 8,000}{10,000} = 90\%$$

Angewendet auf unser Diagramm würde die Berechnung folgendermaßen aussehen:

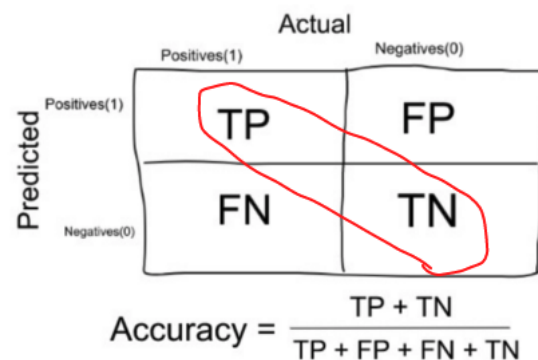


2.4 (a) Definieren Sie die folgenden Metriken: Accuracy, Precision, Recall, F1 Score. (b) Nennen Sie ein Beispiel, bei dem ein guter Recall-Wert zu bevorzugen ist. Nennen Sie ein Beispiel, bei dem ein guter Precisions-Wert zu bevorzugen ist.

a)

Accuracy:

Die Genauigkeit (Accuracy) bei Klassifizierungsproblemen ist die Anzahl der korrekten Vorhersagen, die das Modell über alle Arten von Vorhersagen gemacht hat.



Im Zähler sind unsere korrekten Vorhersagen (True Positives und True Negatives) und im Nenner die Art aller Vorhersagen, die der Algorithmus macht.

Wann ist Accuracy zu verwenden:

Die Genauigkeit (Accuracy) ist ein gutes Maß, wenn die Zielvariablenklassen in den Daten nahezu ausgeglichen sind.

Precision:

Präzision ist ein Maß, das uns sagt, welcher Anteil der Patienten, bei denen wir eine Krankheit diagnostiziert haben, tatsächlich krank sind. Die vorhergesagten positiven Ergebnisse (Personen, die als krank eingestuft werden, sind TP und FP) und die tatsächlich erkrankten Personen sind TP.

$\text{Precision} = TP / (TP + FP)$

Bsp.:

		DIAGNOSIS		
		Diagnosed Sick	Diagnosed Healthy	
PATIENTS	Sick	1000	200	<p>PRECISION: OUT OF THE PATIENTS WE DIAGNOSED WITH AN ILLNESS, HOW MANY DID WE CLASSIFY CORRECTLY?</p> $\text{PRECISION} = \frac{1,000}{1,000 + 800} = 55.6\%$
	Healthy	800	9000	

Recall:

Rückruf ist ein Maß, das uns sagt, welcher Anteil der Patienten, die tatsächlich krank



waren, durch den Algorithmus als krank diagnostiziert wurden. Die tatsächlichen positiven Ergebnisse (Personen die krank waren sind TP und FN) und die Personen, bei denen das Modell eine Krankheit diagnostiziert hat, sind TP. (Hinweis: FN ist enthalten, da die Person tatsächlich krank war, obwohl das Modell anders vorhersagte).

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

Bsp.:

		DIAGNOSIS	
		Diagnosed Sick	Diagnosed Healthy
PATIENTS	Sick	1000	200 ⊗
	Healthy	800	8000

OUT OF THE SICK PATIENTS,  
HOW MANY DID WE CORRECTLY  
DIAGNOSE AS SICK?

$$\text{RECALL} = \frac{1,000}{1,000 + 200} = 83.3\%$$

### F1 Score:

Wir möchten nicht unbedingt jedes Mal, wenn wir ein Modell zur Lösung eines Klassifizierungsproblems erstellen, sowohl Präzision als auch Rückruf einzeln erhalten. Daher ist es am besten, wenn wir eine einzige Punktzahl erhalten, die sowohl für Precision (P) als auch für Recall (R) steht. Es wird dafür das harmonische Mittel verwendet. Das harmonische Mittel ist eine Art Durchschnitt, wenn x und y gleich sind. Wenn x und y jedoch unterschiedlich sind, ist es näher an der kleineren Zahl als an der größeren Zahl. Für unser vorheriges Beispiel: F1 Score = harmonischer Mittelwert (Präzision, Rückruf)

$$\text{F1 Score} = 2 * \text{Präzision} * \text{Rückruf} / (\text{Präzision} + \text{Rückruf}) = 2 * 55,6 * 83,3 / 138,9 = 66,68\%$$

b)

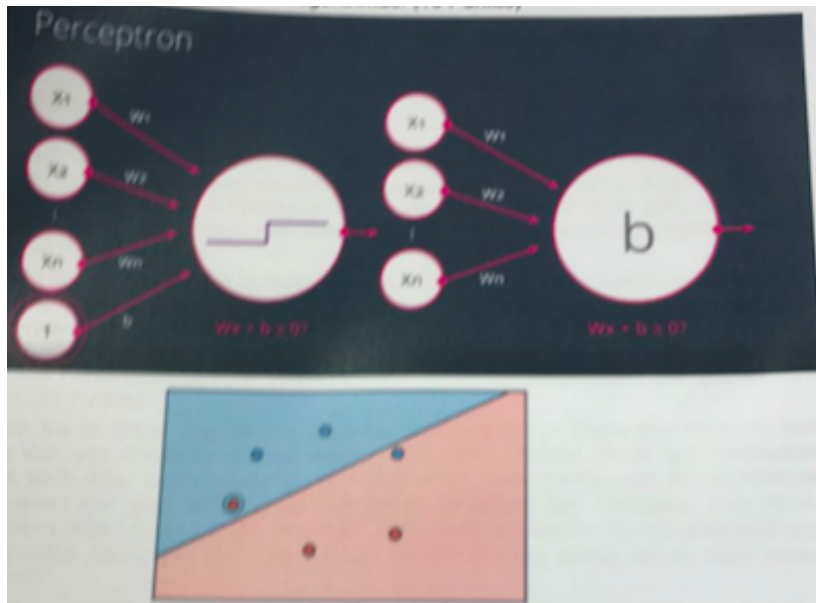
Beispiel, bei dem ein guter Recall-Wert zu bevorzugen ist:

Bei der Modellierung seltener Krebsdaten ist alles, was nicht falsch-negativ ist, eine Straftat. Recall ist ein besseres Maß als Precision.

Beispiel, bei dem ein guter Precisions-Wert zu bevorzugen ist:

Bei YouTube-Empfehlungen sind Falsch-Negative weniger bedenklich. Präzision ist hier besser.

## 2.5 Künstliche Neuronale Netzwerke: Skizzieren Sie den Perceptron-Algorithmus sowie die Grundidee des Backpropagation-Algo



## 3 Weiter mögliche Fragen (Stoff)

### 3.1 Erklärung Python-Basics (zB Spalten aus 2D Array)

Angenommen wir haben folgendes 2D-Array:

	A	B	C	D
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

Wenn wir Spalte A extrahieren möchten, machen wir Folgendes:

```
>> df['A']
```

```
0    1
```

```
1    5
```

```
2    9
```

Bsp. Array Indexierung:

```
>>> a[0,3:5]
array([3,4])
```

```
>>> a[4:,4:]
array([[44, 45],
       [54, 55]])
```

```
>>> a[:,2]
array([2,12,22,32,42,52])
```

```
>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

## 3.2 Erklärung „Classification-Code“ (classifier.fit)

Klassifizierung:

Klassifizierung ist der Prozess der Vorhersage der Klasse gegebener Datenpunkte. Klassen werden manchmal als Ziele/Labels oder Kategorien bezeichnet. Klassifizierungsprädiktive Modellierung ist die Aufgabe, eine Abbildungsfunktion ( $f$ ) von Eingangsvariablen ( $X$ ) an diskrete Ausgangsvariablen ( $y$ ) anzunähern.

Beispielsweise kann die Spamererkennung bei E-Mail-Diensteanbietern als ein Klassifizierungsproblem identifiziert werden. Dies ist eine binäre Klassifizierung, da es nur 2 Klassen als Spam und nicht als Spam gibt. Ein Klassifizierer verwendet einige Trainingsdaten, um zu verstehen, wie bestimmte Eingabevariablen sich auf die Klasse beziehen. In diesem Fall müssen bekannte Spam- und Nicht-Spam-E-Mails als Trainingsdaten verwendet werden. Wenn der Klassifizierer genau trainiert wurde, kann er zur Erkennung einer unbekannten E-Mail verwendet werden. Die Klassifizierung gehört zu der Kategorie des überwachten Lernens, bei der die Ziele auch mit den Eingangsdaten versehen wurden. Es gibt viele Anwendungen in der Klassifizierung in vielen Bereichen, z. B. bei der Kreditgenehmigung, der medizinischen Diagnose, dem Target Marketing usw.

.fit():

Beim Verwenden der `.fit()` -Methode wird ein Modell an die Trainingsdaten angepasst. Es werden die Koeffizienten für die angegebene Gleichung ermittelt.

Dieser Befehl `fit()` bewirkt, dass eine Reihe modellabhängiger interner Berechnungen stattfindet. Die Ergebnisse dieser Berechnungen werden in modellspezifischen Attributen

ten gespeichert, die der Benutzer untersuchen kann. In Scikit-Learn haben alle Modellparameter, die während des `fit()`-Verfahrens erlernt wurden, nachträglich Unterstriche (Bsp.: `model.coef_`)

### 3.3 Evaluation metrics

Basis für evaluation Metriken ist die Confusion Matrix

- accuracy metric
- precision
- recall
- F1 Score
- F-Beta score
- Roc Curve
- Regression Metrics
- R2 Score
- Detecting errors

### 3.4 Cross-validation

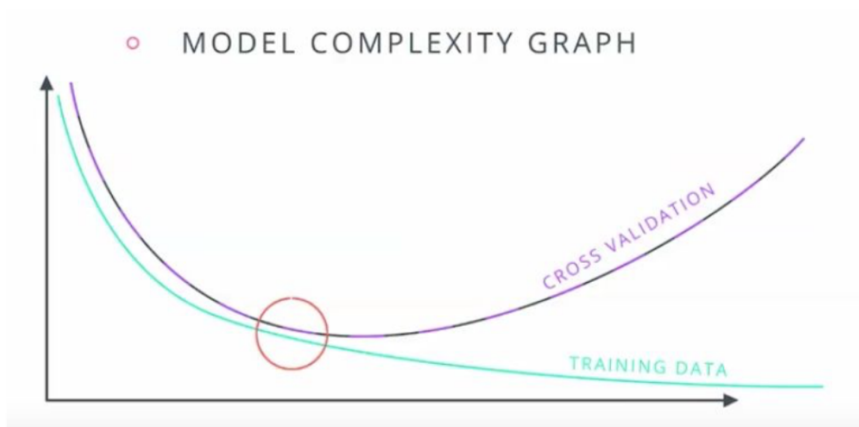
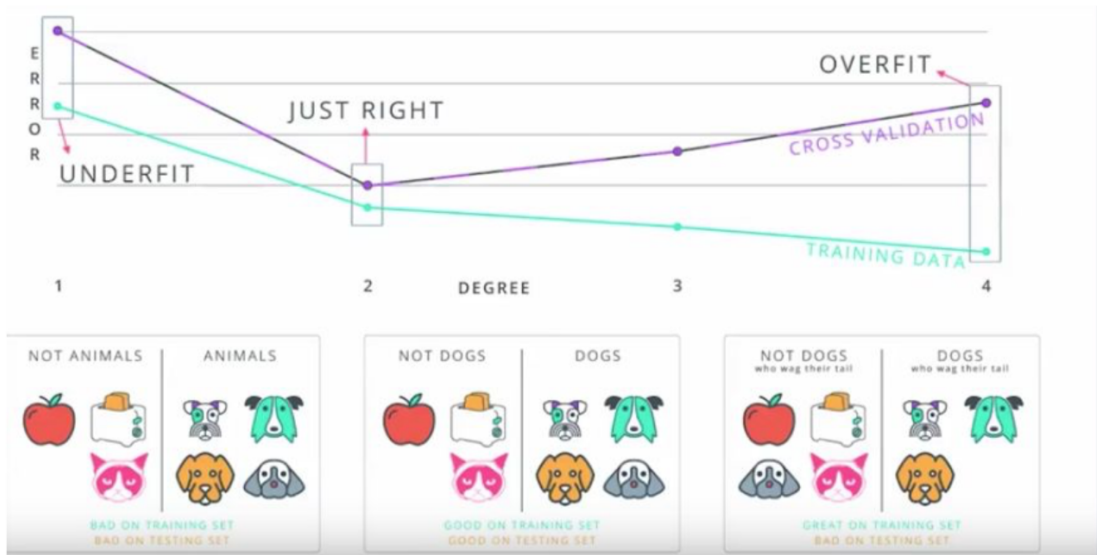
Bei einem Vorhersageproblem erhält ein Modell normalerweise einen Datensatz mit bekannten Daten, zu denen ein Training ausgeführt wird (Trainingsdatensatz), und einen Datensatz mit unbekannten Daten, mit denen das Modell getestet wird (Validierungs-Dataset oder Test-Set genannt). Das Ziel der Kreuzvalidierung besteht darin, einen Datensatz zu definieren, um das Modell in der Trainingsphase (dh den Validierungsdatensatz) zu "testen", um Probleme wie Überanpassung zu begrenzen und einen Einblick in die Verallgemeinerung des Modells zu geben zu einem unabhängigen Datensatz (dh einem unbekannten Datensatz, zum Beispiel von einem echten Problem) usw.

#### Overfitting

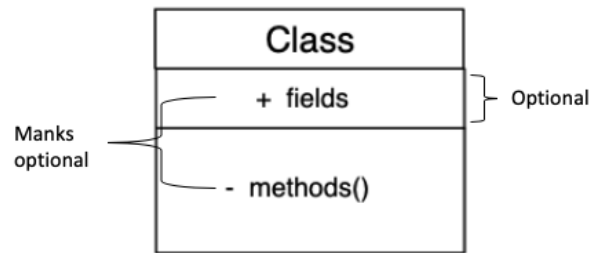
Overfitting tritt auf, wenn ein Modell die Details und das Rauschen in den Trainingsdaten in dem Maße lernt, dass es die Leistung des Modells bei neuen Daten negativ beeinflusst. Dies bedeutet, dass das Rauschen oder zufällige Schwankungen in den Trainingsdaten vom Modell als Konzepte erfasst und gelernt werden. Das Problem ist, dass diese Konzepte nicht für neue Daten gelten und die Generalisierbarkeit von Modellen negativ beeinflussen.

#### Underfitting:

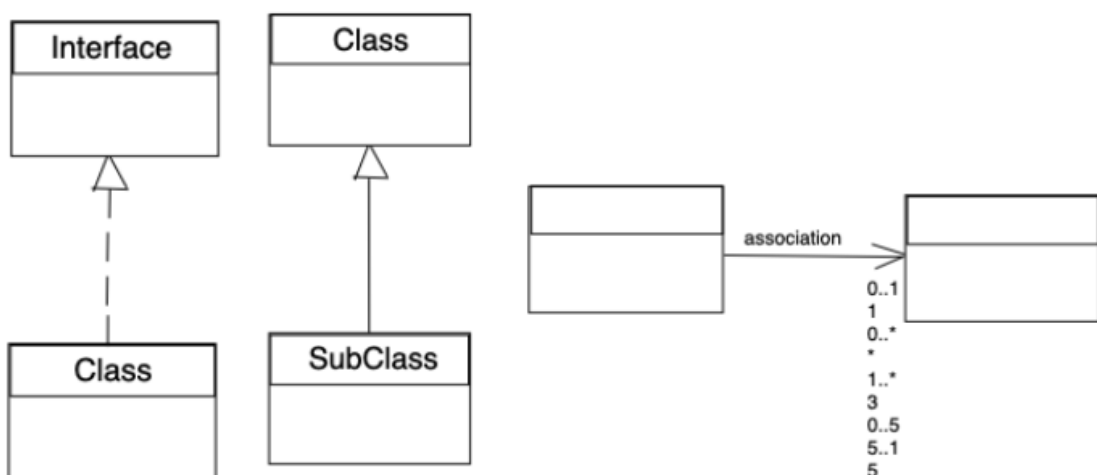
Unter Underfitting wird ein Modell verstanden, das die Trainingsdaten weder modellieren noch auf neue Daten generalisieren kann. Ein maschinelles Underfit-Lernmodell ist kein geeignetes Modell und wird offensichtlich sein, da die Trainingsdaten schlecht abschneiden.



### 3.5 Zusammenfassung des UML's



- for abstract class -> Class or Class «abstract»
- for interface -> Class «Interface»
- fields:
  - name : attributeType
  - x : int
- methodes -> methodenName(attribute : attributeType)
- Manks:
  - + public
  - - private
  - # protected
  - ~package
- Arrows:



### 3.6 Stackable Modification anhand des Codes erklären

```
abstract class IntQueue {
  def get() : Int
  def put(x: Int) }
import scala.collection.mutable.ArrayBuffer

class BasicQueue extends IntQueue{
  private val buf = new ArrayBuffer[Int]
  def get() = buf.remove(0)
  def put(x: Int) {buf += x} }

trait Doubling extends IntQueue{
  abstract override def put(x: Int) {
    super.put(2*x) }}

trait Incrementing extends IntQueue {
  abstract override def put(x: Int) {
    super.put(x+1) }}

trait Filtering extends IntQueue {
  abstract override def put(x: Int) {
    if(x >= 0)
      super.put(x) }}}
```

Hier ist nur der Supercall und die extra Syntax für das überschreiben der abstracten Methode in den Traits interessant.

So wird es verwendet:

```
scala> val queue = (new BasicIntQueue with Increment and Filtering)
//queue mit Incrementierung mit Filterung
scala> queue.put(-1); queue.put(0); queue.put(1)
scala> queue.get() = Int 1
scala> queue.get() = Int 2
```

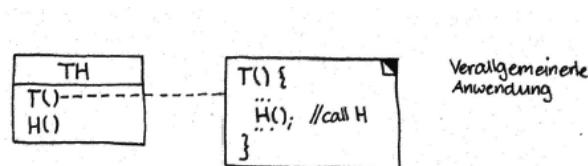
Zuerst wird Filterung angewendet, dann Incrementing daher auch die Ausgabe mit 1 und 2. Wenn man das Vertauscht, also zuerst Increment, dann filterung sieht es so aus:

```
scala> val queue = (new BasicIntQueue with Filtering and Incrementing)
//queue mit Filterung mit Incrementierung
scala> queue.put(-1); queue.put(0); queue.put(1)
scala> queue.get() = Int 0
scala> queue.get() = Int 1
scala> queue.get() = Int 2
```

### 3.7 Hook-Methoden-Konstruktionsprinzip

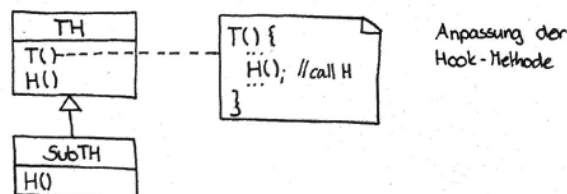
Hier ist Template-Methode und Hook-Methode in EINER Klasse!

- ermöglicht Anpassung von Software durch Vererbung und Überschreibung von Methoden
- Im Vergleich zu anderen Konstruktionsprinzipien hat es den geringsten Grad an Flexibilität für Anpassung
- verallgemeinerte Anwendung Hook-Methode- Konstruktionsprinzip:

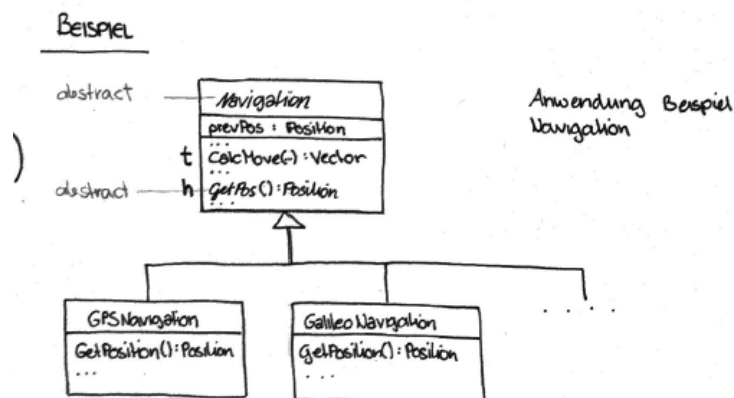


- Anpassung
  - Verhalten der Template-Methode durch überschreiben der Hook-Methode in einer Unterklasse anpassen
  - Wenn es kein typisches Verhalten für Hook-Methode gibt, soll sie als abstrakte Methode definiert werden (Überschreibungszwang)
  - Anpassung erfordert, dass eine Instanz der Unterklasse erzeugt wird. Üblicherweise mit Neustart der Anwendung verbunden (Unterklasse implementieren, im Programmcode Instanz auf Unterklasse ändern)

Siehe Anpassung:



Navigationsbsp:



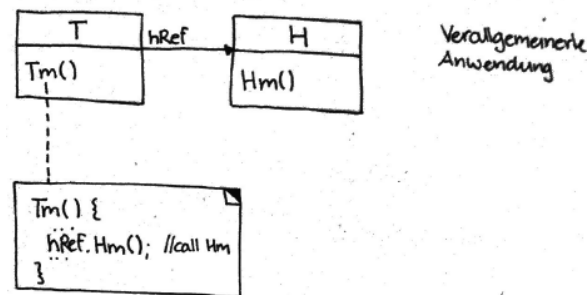


- Vorteile
  - Einfachheit -> es muss nur eine Hook-Methode für das Verhalten, dass anpassbar sein soll, vorgesehen werden
- Nachteile
  - Anpassung erfordert Unterklassenbildung und Überschreiben von Hook-Methode

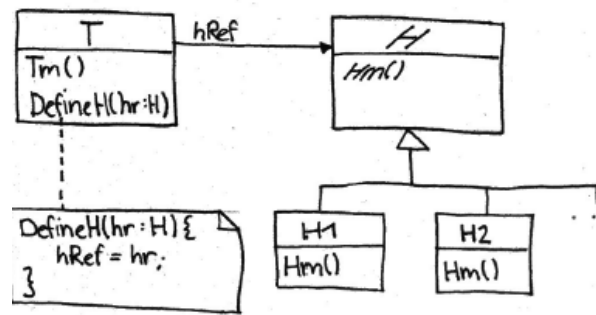
### 3.8 Hook-Objekt-Konstruktionsprinzip

Hier sind Template-Methode und Hook-Methode in Verschiedenen Klassen!

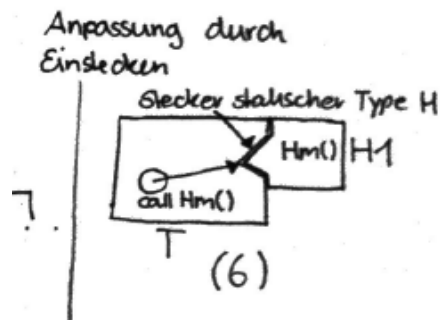
- ermöglicht Anpassung von Software durch Komposition von Objekten anstatt durch Vererbung
- gestiegene Anpassungsflexibilität im Vergleich zu Hook-Methoden-KP
- Anpassung ist zur Laufzeit möglich
- verallgemeinerte Anwendung Hook-Objekt- Konstruktionsprinzip:



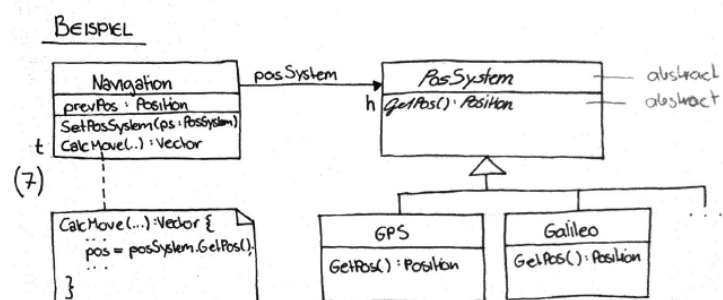
- Klassen sind gekoppelt - Arten der Koppelung:
  - Meist durch definieren einer Instanzvariable mit statischem Typ (H)
  - Template Methode erhält Referenz auf H-Objekt als Parameter (nur für Ausführung der Template-Methode verbunden)
  - über globale Variable, die ein H-Objekt referenziert, gekoppelt (globale Variablen schlecht - Probleme!!)
- Anpassung:
  - Verhalten der Template-Methode durch Einstecken es H-Objekts (mit passender Hook-Methode) in das Objekt, dass diese Template-Methode enthält (hier: T-Objekt)
  - Einstecken bedeutet (bei Koppelung über Referenzvariable), das H-Objekt wird der Referenzvariable des T-Objekts zugewiesen (z.B. Methode für "Einstecken")
  - Da Zuweisung während der Laufzeit möglich ist, kann Verhalten der Template-Methode zur Laufzeit geändert werden
- Methode DefineH() zur Anpassung durch Einstecken von H-Objekten zur Laufzeit:



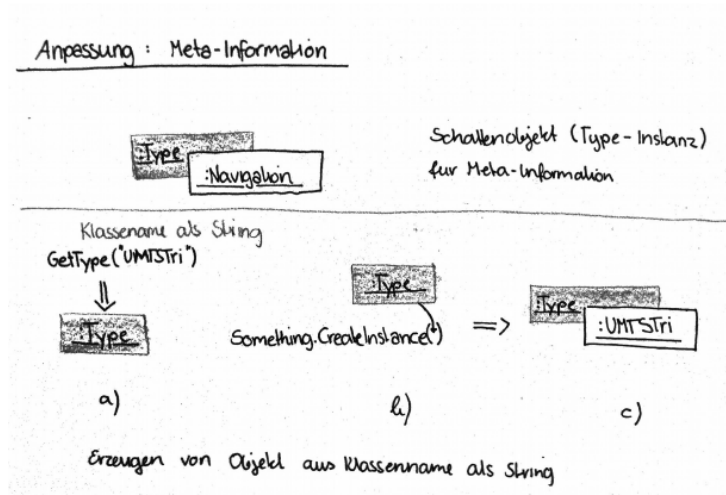
- Unterklassen einer abstrakten H-Klasse für gewünschtes Verhalten einstecken (T-Objekt konfigurieren)
  - \* T sampleT = new T();
  - \* sampleT.DefineH(new H1());
- Semantische Darstellung der Anpassung durch Komposition T-Objekt mit H1-Objekt:



- Anwendung des Hook-Objekt-KP beim Navigationssystem



- Semantische Darstellung



- Erweiterung zur Laufzeit
  - Konfiguration durch Einstecken funktioniert gut, wenn passende Unterklasse bereits vorhanden
  - auch wenn Unterklasse erst implementiert werden muss, ist Konfiguration zur Laufzeit möglich! -> Über Meta-Info
  - soll eine Instanz eingesteckt werden, während die Anwendung in Betrieb ist, muss das Laufzeitsystem das dynamische Laden und Linken unterstützen. (nötige Mechanismen über Bibliotheken)
  - Da bei Objekterzeugung mit 'new' der Name der Klasse fix im Quellcode angegeben werden muss und die Verwendung von Variablen nicht möglich ist, muss man auf Meta-Informationen zurückgreifen
- Meta-Informationen:
  - Informationen zur Laufzeit über Objekte und Klassen (zB zur Laufzeit Name der Klasse eines Objektes erfahren, feststellen, ob Objekt direkt/indirekt zu einer Klasse gehört)
  - Konzeptionell existiert zu jedem Objekt ein SchattenObjekt (Instanz der Klasse Typ)
  - Tatsächlich existiert nur ein Schattenobjekt pro Klasse (das ist allen Instanzen zugeordnet)
  - Klasse Type hat Methoden um Infos über Objekt zu bekommen (zB welche Methoden/Instanzvariablen gibt es?)
- Erzeugung eines Objekts über Klassennamen als Zeichenkette -> zu Bild Semantische Darstellung!
  - a) entsprechendes Typ-Objekt erzeugt (über statische Methode GetType("name")) -> Rückgabe: Referenz auf Instanz der Klasse Typ(Schattenobjekt)
  - b) diese Instanz für CreateInstance("Type") verwenden - Objekt erzeugt
  - c) Rückgabe: Referenz auf erzeugtes Objekt (satischer Type Objekt)
- Hook-Objekt-KP bildet Grundlage für so hohen Grad an Flexibilität, Implementierungsaufwand steigt mit Flexibilität:

- Hook-Methode: 1 Klasse
- Hook-Objekt; mind. 2 Klassen (abstrakt gekoppelt: 3 Klassen)
- Möglichkeit Komponenten dynamisch zu erweitern - Komplexität steigt (dynamisches Instanzieren, Konfigurieren (graphische Schnittstelle...))
- Softwaresystem nicht notwendigerweise besser, wenn mehr Flexibilität
- Ziel: richtiges Maß an Flexibilität (im Gleichgewicht mit Komplexität)
- Vorteile
  - Einfachheit der Anpassung -> Durch Komposition mit Hook-Objekten
- Nachteile
  - Komplexität des Entwurfs und der Implementierung ist höher als Hook-Methoden