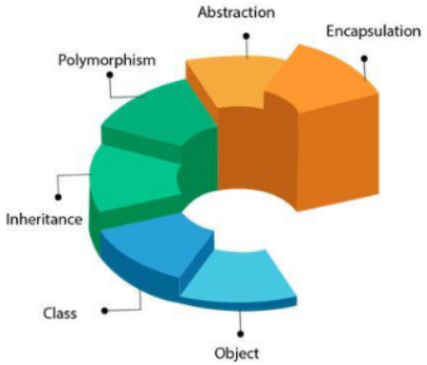
Objektorientierte Softwareentwicklung

Übersicht und Einführung in objektorientierten Grundkonzepte

Die Inhalte der Vorlesung wurden primär auf Basis der angegebenen Literatur erstellt. Darüber hinaus sind ausgewählte Aspekte aus der Vorlesung von Prof. Dr.-Ing. Faustmann (ebenfalls HWR Berlin) übernommen worden. Für die Bereitstellung dieses Vorlesungsmaterials möchte ich mich an dieser Stelle noch einmal recht herzlich bedanken.



Übersicht zur Vorlesung

Quelle der oberen Abbildung: https://www.nerd.vision/post/polymorphism-encapsulation-data-abstraction-and-inheritance-in-object-oriented-programming, letzter Zugriff: April 2020

Quelle der rechten unteren Abbildung: https://www.oracle.com/java/technologies , letzter Zugriff: April 2020

Quelle der rechten unteren Abbildung: https://www.uml.org - 20 Jahre UML Version 1.1, letzter Zugriff: April 2020

Übersicht zur Vorlesung I (Objektorientierte Programmierung)

- Motivation zur objektorientierten Softwareentwicklung
  - Entwicklung des objektorientierten Programmierparadigma
  - Erwartete Vorteile der objektorientierten Softwareentwicklung
  - Kernkonzepte der objektorientierten Programmierung
- Klassen, Objekte und Methoden
  - Datenabstraktion und –kapselung
  - Instanzbildung zur Verwendung von Objekten
  - Aufgaben des Konstruktors und Destruktors
  - Zugriffsmethoden und statische Klassenelemente

Übersicht zur Vorlesung II (Objektorientierte Programmierung)

- Vererbung & Polymorphie
  - Abgeleitete Klassen
  - Elementzugriffe und Einschränkungen
  - Polymorphe Verwendung von Objekten
  - Statische und dynamische Bindung (late binding)
  - Abstrakte Klassen und Methoden
- Schnittstellen-Konzept (interface)
  - Grundkonzept (Nachbildung der Mehrfachvererbung)
  - Definition (Spezifikation) und Implementierung
  - Beispiele für die Anwendung von Interfaces
  - Aktuelle Entwicklungstendenzen (Java RMI und EJB-Konzept)

Übersicht zur Vorlesung III (Objektorientierte Programmierung)

- Persistente und transiente Objekte
  - Grundlagen persistenter Objekte in Java
  - Objekte schreiben und lesen mit Hilfe der Schnittstelle *Serializable*
  - Ausgewählte Probleme der Objektserialisierung
  - Weiterführende Möglichkeiten zur Serialisierung von Objekten in Java
- Weiterführende Konzepte
  - Ausnahmebehandlung
  - Aspekte der generischen Programmierung
  - Traditionelle Java Collections (z.B. Vector, Stack, HashTable)
  - Java Collections Framework (JCF) ab der JDK-Version 1.2

Übersicht zur Vorlesung IV (Objektorientierte Analyse/Design)

- Zielstellungen von Analyse und Design
  - Übergang von der Projektierung zur objektorientierten Analyse
  - Ergebnisse der Analyse und Design-Phase
  - OMG UML-Standard als Basis der Modellierung (Notation & Prozess)
  - Entwicklung der UML-Notation (Booch, Rumbaugh, Jacobson)
- Anforderungsspezifikation und -modellierung
  - Pflichtenheft und Lastenheft vs. Product-, Release- und Sprint-Backlog
  - Geschäftsprozess – Geschäftsprozessfunktion - UseCase
  - Modellierung von Anwendungsfällen (UseCase)
  - Vergleich von Use Cases und User Stories

Übersicht zur Vorlesung V (Objektorientierte Analyse/Design)

- Grundkonzepte der UML
  - Übersicht zu den Diagrammen der UML
  - Statische und dynamische Notationselemente der UML
  - Erweiterungen im Kontext der UML Version 2.0
  - Rational Unified Process
  - Werkzeuge zur Modellierung mit UML (OMONDO – EclipseUML)
- Objektorientierte Analyse
  - Verwendete Notationselemente der UML
  - Prototypische Implementierung einer GUI
  - Verwendung von Analyse-Mustern

Übersicht zur Vorlesung VI (Objektorientierte Analyse/Design)

- Objektorientiertes Design
  - Erweiterung des Analysemodells
  - Identifikation von Subsystemen
  - Verwendung von Design-Pattern
  - Schnittstellenspezifikationen (verteilte Entwicklung)
- Weiterführende Aspekte
  - Object Constraint Language - OCL
  - Überfügung des Design in die Implementierung (Java-Abbildung)
  - Qualitätssicherung innerhalb der Analyse und Design
- Ausführen eines Beispielprojektes

Literatur & Organisation

Literatur - Programmierung

- [Krüger 2014] Krüger, G.; Hansen, H.: Java-Programmierung - Das Handbuch zu Java 8, Addison-Wesley, München, 2014 (ISBN 978-3-95561-514-7)
- [Brügge 2004] Brügge, B.; Dutoit, A. H.: Objektorientierte Softwaretechnik mit UML, Entwurfsmustern und Java, Pearson Studium, München 2004 (ISBN 3-8273-7261-5)
- [Inden 2012] Inden, M.: Der Weg zum Java-Profi – Konzepte und Techniken für die professionelle Java-Entwicklung, dpunkt.verlag, Heidelberg 2012 (ISBN 978-3-86490-005-1)
- [Inden 2015] Inden, M.: Java 8 – Die Neuerungen (Lambdas, Streams, Date and Time API und JavaFX 8), dpunkt.verlag, Heidelberg 2012 (ISBN 978-3-86490-290-1)

Literatur – Analyse/Design

- [Oestereich 2014] Oestereich, B.; Scheithauer, A.: Die UML-Kurzreferenz 2.5 für die Praxis, de Gruyter Oldenbourg-Verlag, 2014 (ISBN 3-486-74909-9)
- [Goll 2012] Goll, J.: Methoden des Software Engineerings, Funktions-, daten-, objekt- und aspektorientiert entwicklen, Springer Vieweg, Wiesbaden, 2012 (ISBN 978-3-8348-2433-2)
- [Pieper 2017] Peper, F. U.; Roock, S.: Agile Verträge – Vertragsgestaltung bei agiler Entwicklung für Projektverantwortliche, dpunkt.verlag, 2017 (ISBN 978-3-86490-400-4)
- [Versteegen 2000] Versteegen, G.: Projektmanagement mit dem Rational Unified Process, Springer-Verlag, 2000 (ISBN 3-540-66755-5)

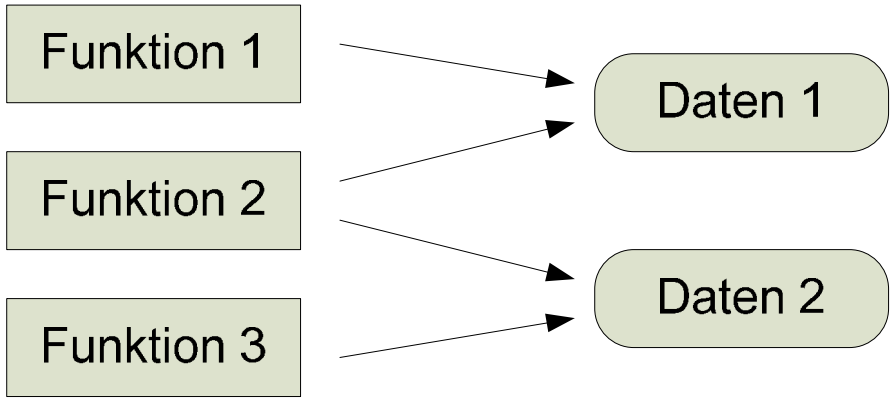
Organisation

- Unterlagen werden in Moodle zur Verfügung gestellt. Bitte berücksichtigen Sie das eine geringfügige Anpassung im laufenden Semester erfolgt!
- Bei Fragen stehe ich Ihnen im Rahmen meiner Sprechzeit (jeweils Dienstags 13:00 bis 14:00 Uhr) und über meine Email-Adresse an der HWR Berlin andreas.schmietendorf@hwr-berlin.de zur Verfügung.
- Webseite des Dozenten: https://blog.hwr-berlin.de/schmietendorf/
- Abgeschlossen wird das Fach durch 2 Kurzvorträge (Schwerpunkt Java, Schwerpunkt UML) und eine Projektarbeit (Review und Beleg), dabei sind alle Phasen der objektorientierten Entwicklung zu berücksichtigen.

Motivation zur objektorientierten Softwareentwicklung

Prozedurale Programmierung

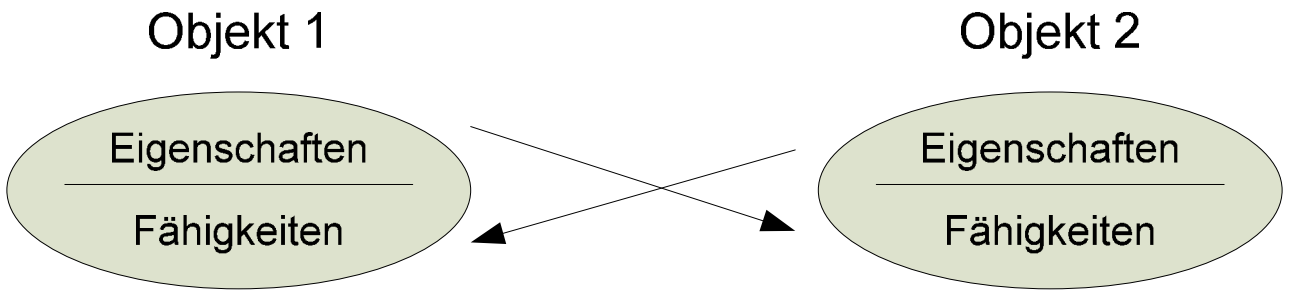
- Trennung von Daten und Funktionen
- Konsequenzen:
  - Programmierer muss vor Verwendung der Daten für eine Initialisierung mit Anfangswerten sorgen und Gewährleisten das korrekte Werte übergeben werden.
  - Änderungen der Daten (z.B. Erweiterung eines Datensatzes) hat eine Anpassung der Funktion zur Folge



Folge: Fehleranfälligkeit in der Erstellung und Wartung

Objektorientierte Programmierung

- Im Mittelpunkt der OOP stehen Objekte, die eine Einheit aus Daten (vgl. Eigenschaften) und Funktionen (vgl. Fähigkeiten) bilden.
- Die Kommunikation zwischen Objekten erfolgt durch Nachrichten, die die Fähigkeiten von Objekten aktivieren.



- Mit einer Klasse wird ein Objekttyp definiert, der sowohl die Eigenschaften als auch die Fähigkeiten von Objekten dieses Typs festlegt.

Vorteile der objektorientierten Programmierung I

- Potentielle Unterstützung der Wiederverwendbarkeit
  - Eigenständigkeit eines Objekte
  - Objekte zeigen Grundzüge von Komponenten auf
- Vorteile bei der Strukturierung komplexer Softwaresysteme
  - Unterteilung der Software in Schichten
  - Verschiedene Arten von Klassen (Controller, Proxies, …)
- Verbesserungen im Kontext der Wartung & Pflege von Software
  - Änderungen wirken sich nur auf keinen kleinen Teil aus
  - Entkopplung der internen Datendarstellung von den Anwendungen

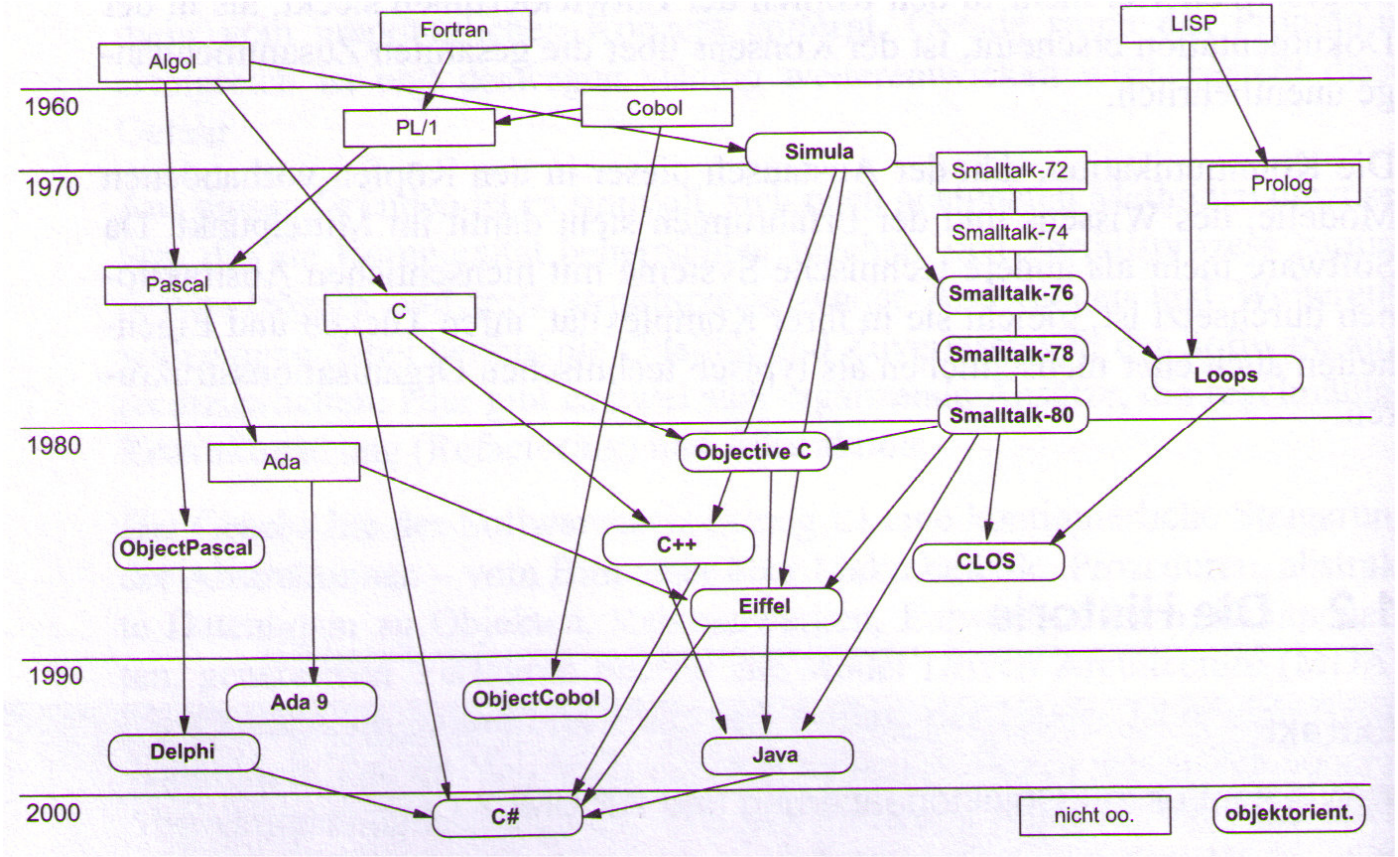
Vorteile der objektorientierten Programmierung II

- Geringere Fehleranfälligkeit
  - Ein Objekt kontrolliert den Zugriff auf seine Daten selbt
  - Fehlerhafte Zustände können gezielt abgewehrt werden
- Verbesserung der Kommunikation mit den Kunden
  - Objektorientierte Analyse- und Designmethoden setzen kein technisches Wissen über die Programmierung voraus
  - Vielfalt angebotener Modellelemente (Diagrammarten)
- Viele Probleme der realen Welt lassen sich objektorientiert einfacher modellieren als mit strukturierten Analyse- und Design-Methoden.

Instrumente der OO-Programmierung

- Datenabstraktion
  - Bildung von Klassen zur Beschreibung von Objekten
  - Objekt als Abstraktion
- Datenkapselung
  - Kontrollierter Zugriff auf Daten eines Objektes
  - Vermeidung von Seiteneffekten
- Vererbung
  - Bildung abgeleiteter Klassen (Einfach- oder Mehrfachvererbung)
  - Abgeleitete Klassen erben Eigenschaften darüber liegender Klassen
- Polymorphie (griech. Vielgestaltigkeit)
  - Implementierung von Anweisungen die zur Laufzeit verschiedene Wirkungen haben können

Entwicklung objektorientierter Programmiersprachen



Quelle: Oestereich, B.: Objektorientierte Softwareentwicklung – Analyse und Design mit UML 2.0, Oldenbourg-Verlag, 2005 (ISBN 3-486-27266-7)

OO-Programmierung mit Java I

- *einfach:* hinsichtlich seiner Erlernbarkeit, u.a. keine Zeiger, einer ausschließlich einfachen Vererbungsmöglichkeit, der Vermeidung der GOTO's, des C-Präprozessors und der Überladungstechniken,
- *objektorientiert:* mit der Ausnahme der einfachen Datentypen, wie int, char und boolean, sind alle anderen Verarbeitungskomponenten Objekte,
- *verteilt:* Java-Klassenbibliotheken wurden von Anfang an so ausgelegt, dass sie eine Realisierung verteilter Systeme über Netzwerke unterstützen,
- *interpretiert:* durch eine virtuelle Maschine (Java Virtual Machine (JVM)) wird der Java-Code als so genannter Bytecode einheitlich interpretiert,
- *robust:* durch die Einfachheit werden eine Reihe typischer Programmier-fehler bereits vermieden bzw. sind durch Testhilfen analysierbar,

OO-Programmierung mit Java II

- *sicher:* durch die Bereitstellung unterschiedlichster Sicherheitsfunktionen, wie beispielsweise durch für die sogenannte Applet-Technik (s. u.),
- *architekturneutral:* durch die Berücksichtigung der meisten, gängigen Plattformen und deren Spezifika,
- *portabel:* durch die strenge Kapselung plattformabhängiger Merkmale,
- *performant:* Java-Code läuft zwar ca. 20-mal langsamer als übersetzter C-Code, jedoch sind dafür die Entwicklungszeiten wesentlich kürzer,
- *dynamisch:* Java lädt genau die Klassen (übers Netz), die die Anwendung benötigt, und passt sich leicht neuen Umgebungen an.

Übung

Lösen Sie zum Wiedereinstieg in die Programmiersprache Java und die Nutzung von Eclipse folgende kleine Programmieraufgaben:

Aufgabe 1:

Schreiben Sie eine Methode `long getPotenz(int basis, int exponent)`, die `basis` hoch `exponent` berechnet. Schreiben Sie auch eine geeignete `main`-Funktion, die zwei Werte einliest und `getPotenz()` aufruft. Benutzen Sie **nicht** die interne Funktion `pow`!

Aufgabe 2:

Schreiben Sie eine Methode, welche vom Nutzer eine beliebige Anzahl von Zahlen einliest und als Ergebnis den Mittelwert ausgibt.

Schreiben Sie eine Methode, welche ein Array von Integerwerten übergeben bekommet und als Ergebnis den arithmetischen Mittelwert (Rückgabewert) zurückliefert.

Bem.: Versuchen Sie bei den zu programmierenden Methoden auf statische Eigenschaften zu verzichten! → Ausnahme `main()`!