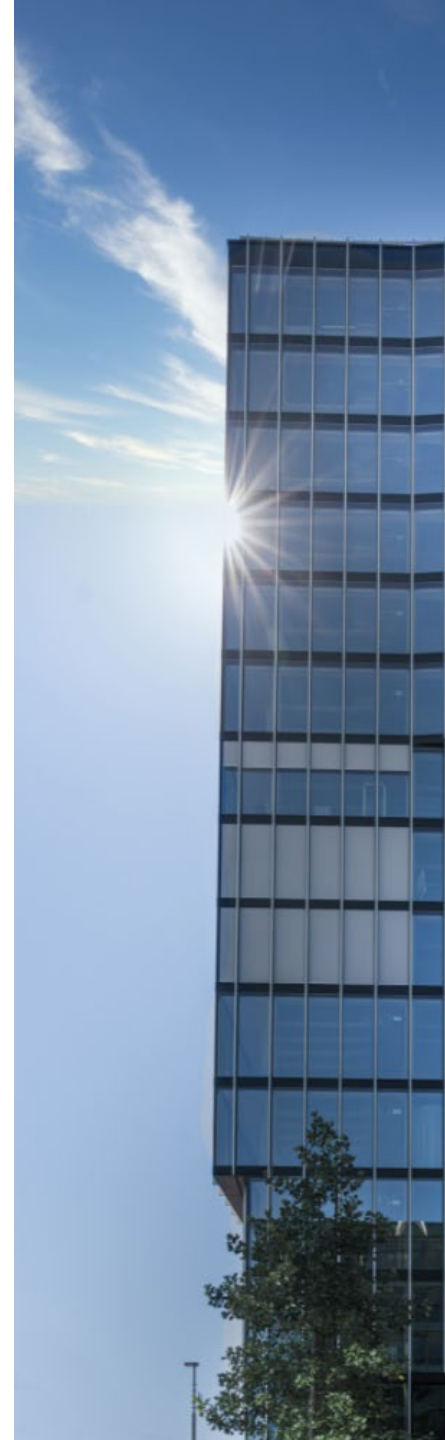


Objektorientierte Programmierung

# Entwicklungsumgebungen

**IDE – Integrated Development Environment**

Roland Gisler



# Inhalt

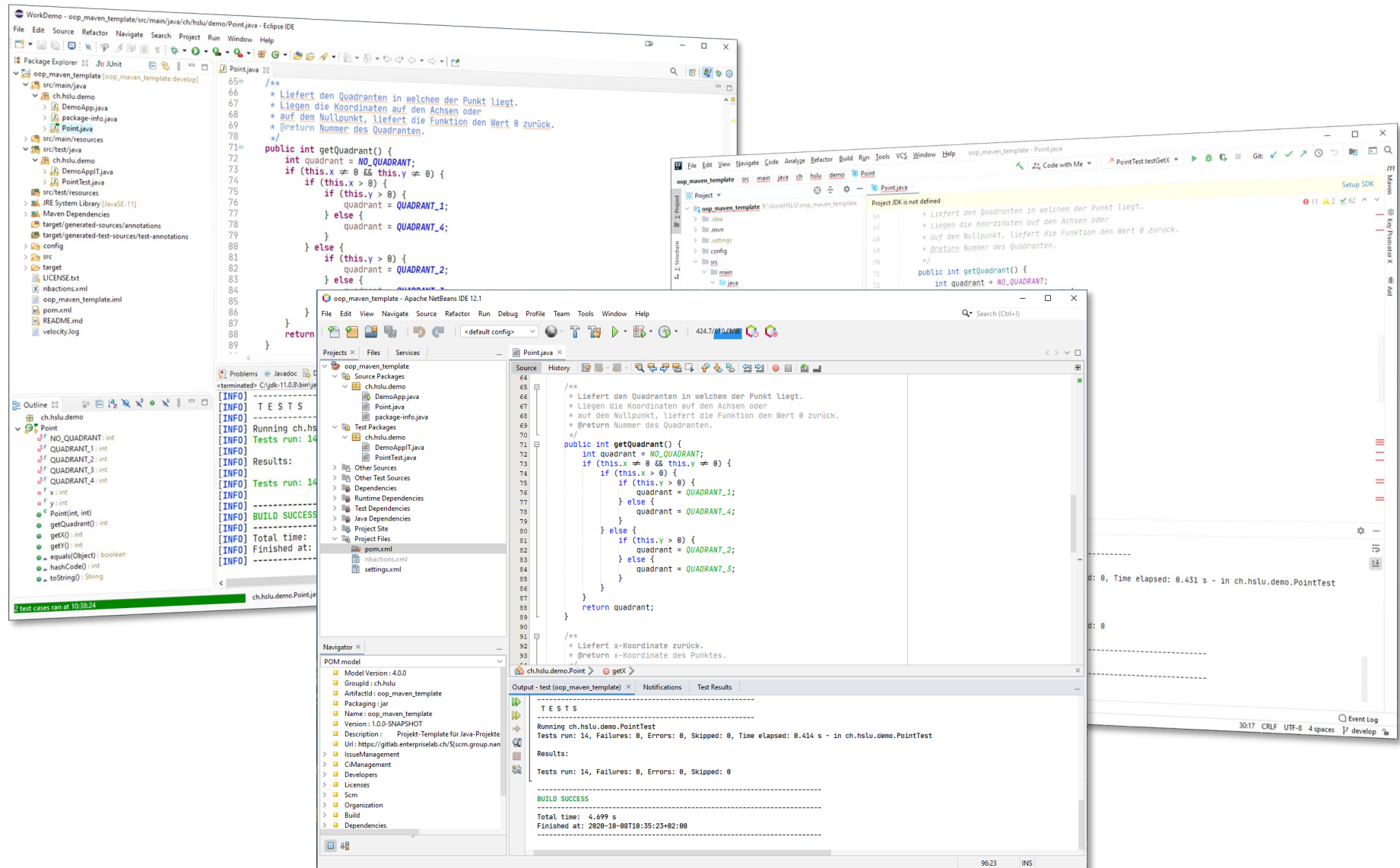
- Was ist eine IDE und was sind ihre Aufgaben?
- Wie sind IDEs eigentlich entstanden?
- Aktuelle IDE's für Java
- Einsatz und Nutzung der IDE in OOP
- IDE versus BlueJ
- Empfehlungen zu IDE's
- Empfehlungen zum Einsatz in OOP

# Lernziele

- Sie wissen, was eine IDE ist und welchem Zweck sie dient.
- Sie kennen die wichtigsten Aufgaben einer IDE.
- Sie können mit einer IDE grundlegende Programmiertask erledigen.
- Sie kennen eine für Java-Projekte adäquate Projekt- bzw. Verzeichnisstruktur.

# **Was ist eine IDE?**

# Was ist eigentlich eine IDE\*?



\*IDE = Integrated Development Environment

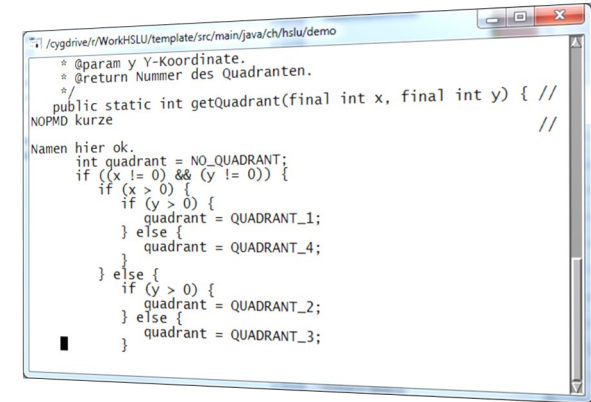
# Die wichtigsten Aufgaben einer IDE

- Visualisierung und Navigation von/im Quellcode.
- Leistungsfähige Quellcode-Editoren.
- Werkzeuge für Refactoring.
- Organisation von Quellcode in Projekten.
- Organisation der Konfiguration von Projekten.
- Quellcode kompilieren und ausführen, Debugging.
- Anbindung an Versionskontrollsysteme.
- Erzeugung, Import und Export von verschiedenen Artefakten.
- Und vieles, vieles mehr!

# **Wie sind IDEs entstanden?**

# Wie haben sich die IDEs entwickelt?

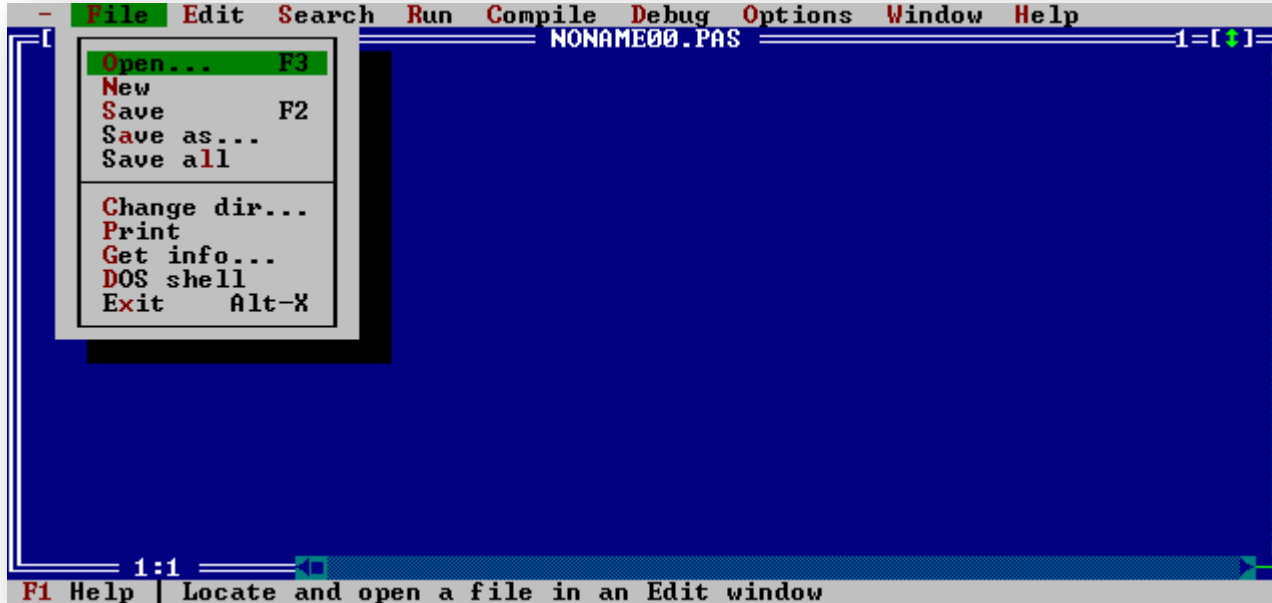
- Ursprünglich wurde Code mit normalen Editoren editiert...
  - z.B. unter Unix/Linux: **vi**, **vim**, **emacs**
- mit einem Compiler-Programm kompiliert...
  - z.B. für Java: **javac <java-Datei>**
- und zu einer Library verlinkt / verpackt.
  - z.B. für Java: **jar <jar-Datei> <class-Dateien...>**
- Ein fehleranfälliger Vorgang mit grossem Overhead!
  - Editieren, Kompilieren, Testen – Editieren, Kompilieren...
- Editoren (z.B. **emacs**) konnten funktional erweitert werden, so dass z.B. die Kompilation innerhalb des Editors gestartet werden konnte.
  - **Integration** der Entwicklungs-Tools ➔ die Geburt der **IDE**!





# Populäre frühe IDEs

- z.B. Borland Turbo Pascal und Turbo C/C++



- DOS/konsolenbasierendes Programm – Ende der 1980er-Jahre
- Editor mit integriertem Compiler und Debugger  
➔ Geburt der IDE im heutigen Sinne!

# **Aktuelle IDEs für Java**

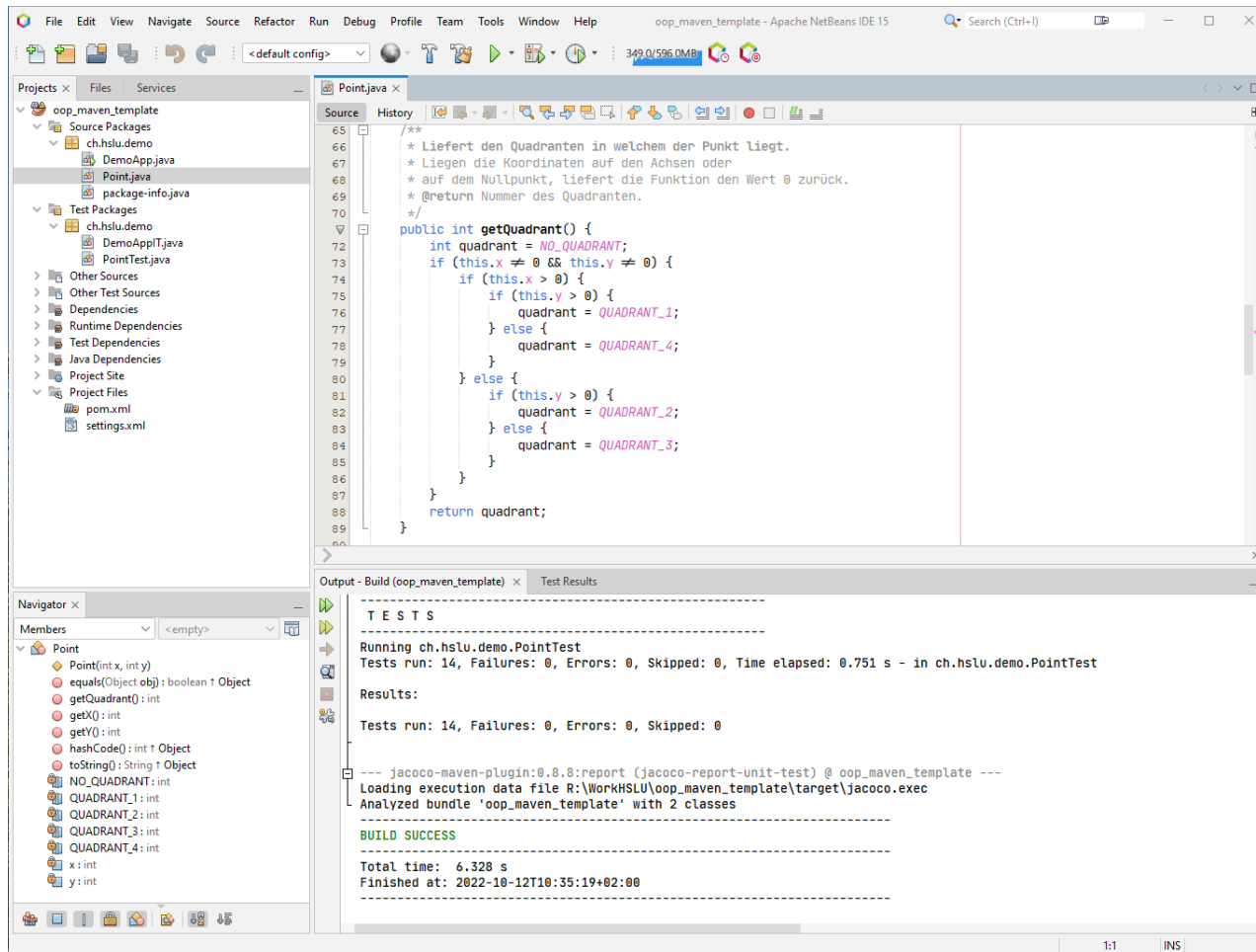
# Aktuelle IDEs / Beispiele

- IDEs haben sich immer mehr und weiter zu «integrierenden» Entwicklungsumgebungen verändert.
- Flexible Plugin-Konzepte erlauben:
  - Eine nahezu beliebige Erweiterung der Funktionalität.
  - Integration oder Verknüpfung mit anderen Systemen.
    - z.B. SCM, Datenbanken, Buildserver, Issue-Tracking etc.
- Beispiele von populären (primär Java-)IDEs:
  - **Apache NetBeans**: Open Source, eine der ältesten Java-IDE's.
  - **Eclipse JDT**: Open Source, ursprünglich nur Java, heute auch für C/C++, PHP und viele andere Sprachen.
  - **IntelliJ IDEA**: Kommerzielle IDE von JetBrains, Community Edition gratis, leistungsfähig, für diverse Sprachen.
  - **Visual Studio Code**: Kann auch für Java genutzt werden.

# Apache NetBeans IDE



- Die Altbewährte - <http://netbeans.apache.org/>
- Aktuelle Version: **21**



# Apache NetBeans IDE - Hinweise



- Perfekte Integration der auf Apache Maven basierten Projekte.
- «Klassische» Entwicklungsumgebung, einfaches GUI, relativ übersichtlich und einfach in der Bedienung.
- IDE merkt sich globale Einstellungen zentral für alle Projekte.
- Projekte können einfach «geöffnet» und «geschlossen» werden.
- Es ist problemlos möglich, auch mehrere Projekte parallel geöffnet zu halten.
- **Empfehlung:** Studierende, welche noch nie mit einer Entwicklungsumgebung gearbeitet haben, und/oder sich neu mit Programmierung beschäftigen: **NetBeans ist Ihre erste Wahl!**



# Eclipse JDT – «Eclipse IDE for Java Developers»

- Populär und leistungsfähig – <https://www.eclipseide.org/>
- Aktuelle Version: **2024-03**

```
64  
65  /**  
66   * Liefert den Quadranten in welchem der Punkt liegt.  
67   * Liegen die Koordinaten auf den Achsen oder  
68   * auf dem Nullpunkt, liefert die Funktion den Wert 0 zurück.  
69   * @return Nummer des Quadranten.  
70   */  
71  public int getQuadrant() {  
72      int quadrant = NO_QUADRANT;  
73      if (this.x != 0 && this.y != 0) {  
74          if (this.x > 0) {  
75              if (this.y > 0) {  
76                  quadrant = QUADRANT_1;  
77              } else {  
78                  quadrant = QUADRANT_4;  
79              }  
80          } else {  
81              if (this.y > 0) {  
82                  quadrant = QUADRANT_2;  
83              } else {  
84                  quadrant = QUADRANT_3;  
85              }  
86          }  
87      }  
88      return quadrant;  
89  }  
90  
91  /**
```

ch.hslu.demo  
Point  
NO\_QUADRANT: int  
QUADRANT\_1: int  
QUADRANT\_2: int  
QUADRANT\_3: int  
QUADRANT\_4: int  
x: int  
y: int  
Point(int, int)  
getQuadrant(): int  
getX(): int  
getY(): int  
equals(Object): boolean  
hashCode(): int  
toString(): String

2 test cases ran at 10:42:02

ch.hslu.demo.PointTest - oop\_maven\_template/src/main/java

Tests run: 14, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.648 s - in ch.hslu.demo.PointTest

Results:

Tests run: 14, Failures: 0, Errors: 0, Skipped: 0

--- jacoco-maven-plugin:0.8.8:report (jacoco-report-unit-test) @ oop\_maven\_template ---

Loading execution data file R:\WorkHSLU\oop\_maven\_template\target\jacoco.exec

Analyzed bundle 'oop\_maven\_template' with 2 classes

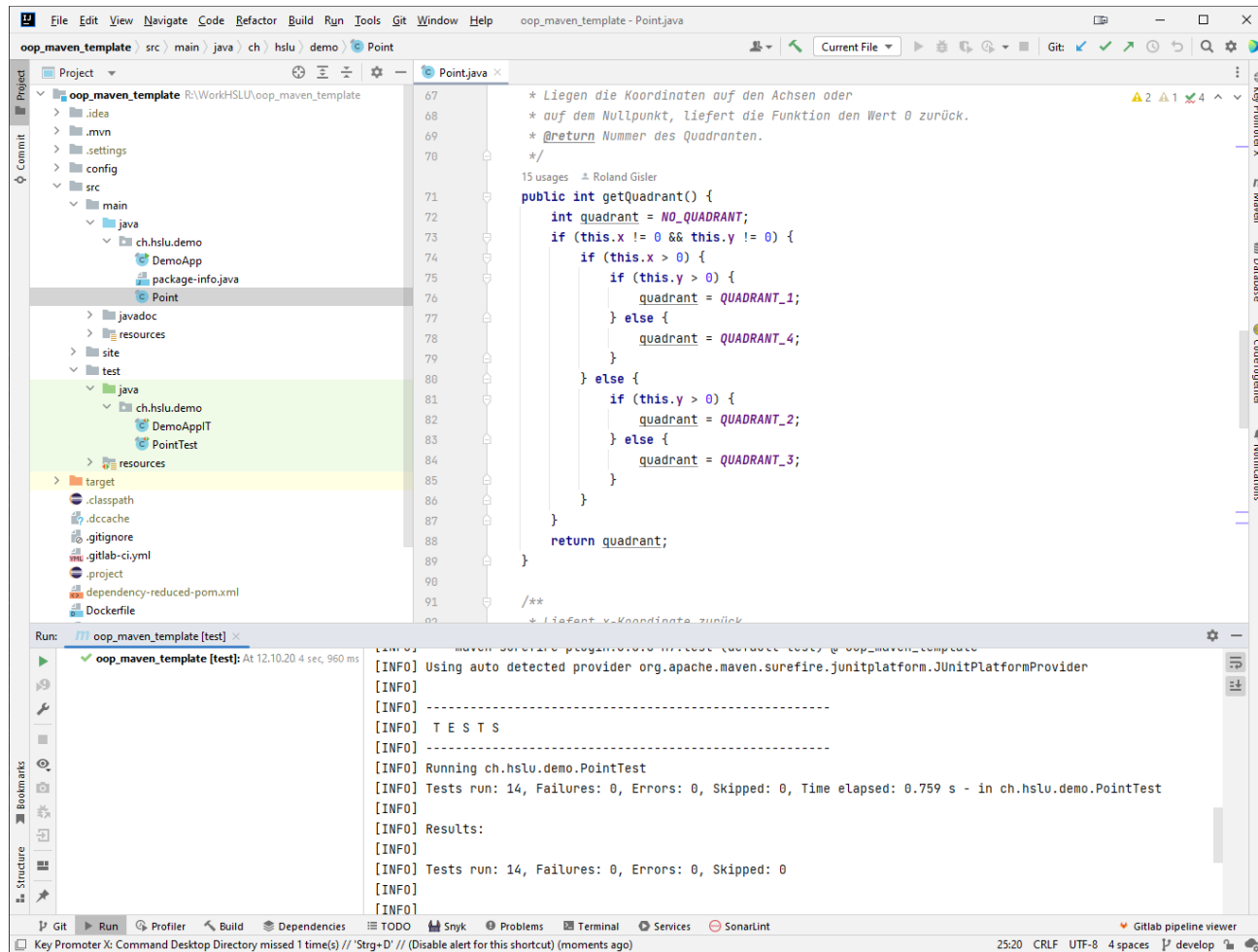


- Gute Integration der auf Apache Maven basierten Projekte.
- Bedienung ebenfalls über ein klassisches Menü möglich.
- Konzept der «Workspaces»: Einstellungen werden pro Basisverzeichnis (kann mehrere Projekte enthalten) gespeichert.
- Mehrere Projekte können gleichzeitig in der IDE erfasst sein, und zusätzlich noch einzeln geöffnet/geschlossen (aktiv/inaktiv) sein.
  - Einfache Gruppierung von Projekten möglich (Project-Sets)
- Flexibles Plugin-Konzept, zusätzlich auch über «Marketplace» und Account über mehrere Installationen synchronisierbar.
- **Empfehlung:** Wer Eclipse schon kennt, bleibt dabei! Es ist eine gute Alternative zu NetBeans, sehr ähnlich in der Bedienung.

# IntelliJ IDEA – «Community Edition»



- Unter Kenner\*innen beliebt – <http://jetbrains.com/idea/>
- Aktuelle Version: **2023.3.5** (bald 2024.1)





# IntelliJ IDEA – Hinweise

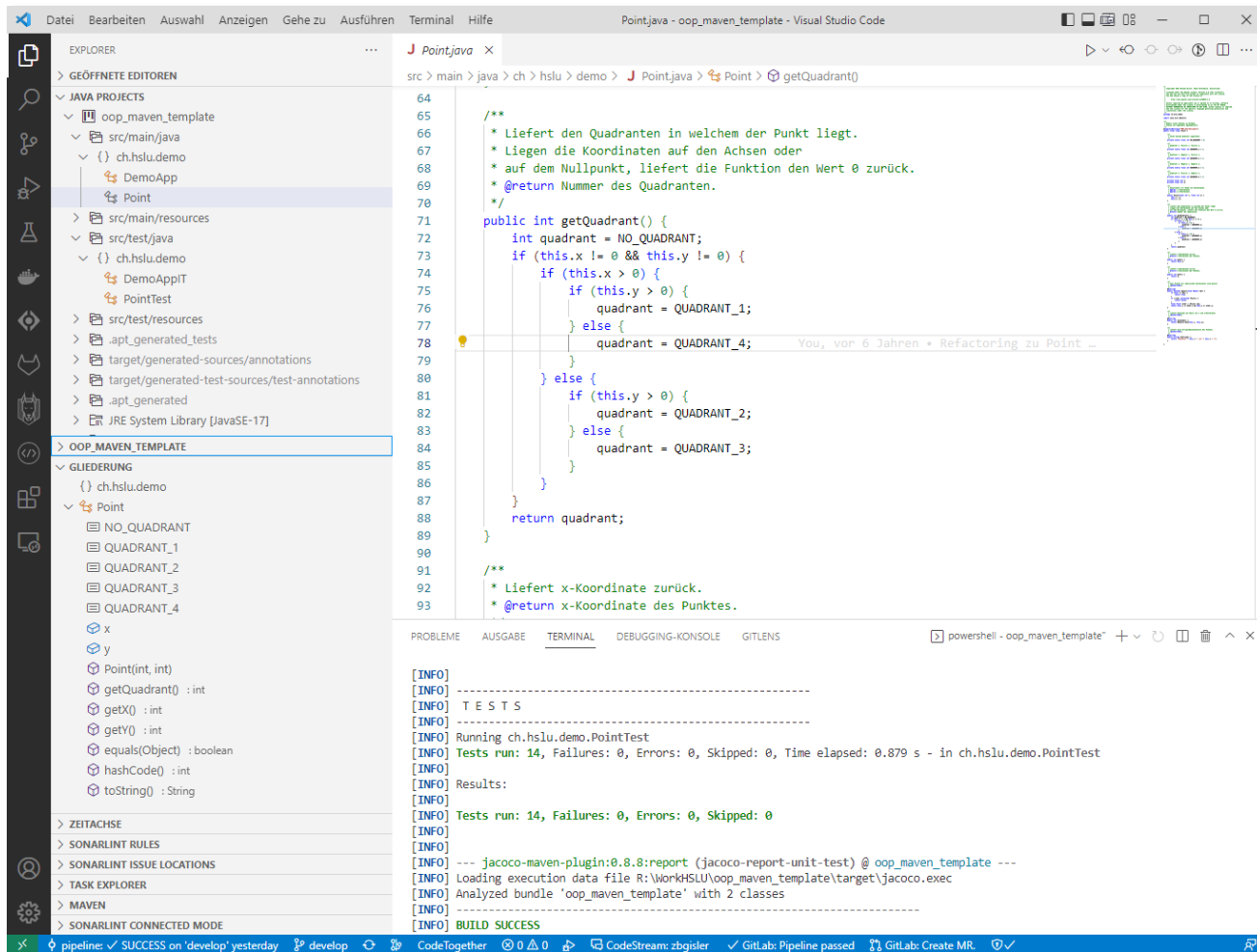


- Gute Integration der auf Apache Maven basierten Projekte.
- Konfiguration pro einzeltem Projekt. Mehrere Projekte werden in mehreren IDE-Instanzen geöffnet.
- IDE ist stark auf wenig Ablenkung und Effizienz getrimmt, Bedienung ist stark auf Shortcuts ausgelegt.
  - Dazu muss man diese aber kennen, und wissen was man will!
- Zielgruppe liegt deutlich bei bereits erfahrenen, professionellen Entwickler\*innen, die genau wissen, was sie wollen und brauchen.
- **Empfehlung:** Bitte nur verwenden, wenn Sie schon über aktive Entwicklungserfahrung verfügen und/oder IntelliJ bereits kennen. IntelliJ ist **keine** IDE für «Anfänger\*innen».

# Visual Studio Code



- Starke Konkurrentin – <https://code.visualstudio.com/>
- Aktuelle Version: **1.87.2**



# Visual Studio Code – Hinweise



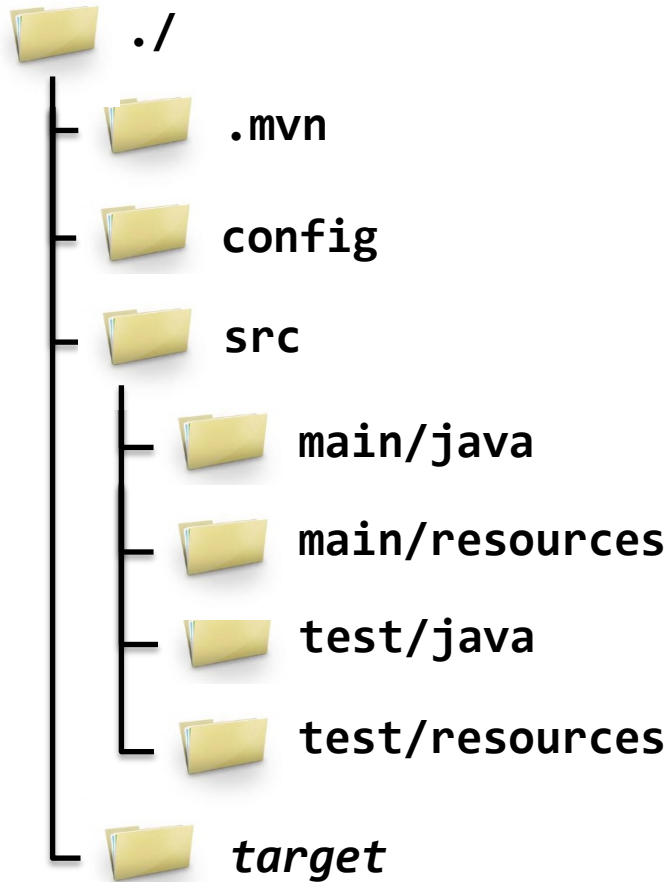
- Ursprünglich keine Java IDE, sondern für C# ausgelegt.
- Kann mittels Plugins mit sehr vielen verschiedenen Programmiersprachen umgehen. Müssen aber installiert werden.
  - Basiert bei Java z.B. auf Eclipse-Technologien.
- Als Microsoft-Software natürlich perfekt in Windows eingebettet, sehr grosse Verbreitung.
- Im Moment das «lebendigste» Ökosystem was die Plugin-Entwicklung betrifft.
- **Empfehlung:** Wenn Sie Visual Studio Code schon beherrschen (nicht nur kennen) ok, aber warum nicht mal eine «echte» Java-Entwicklungsumgebung verwenden und kennenlernen? 😊

# **Wie arbeiten wir mit der IDE?**

# Modul OOP - Arbeitsweise mit IDE

- Wir stellen Ihnen für OOP ein vorkonfiguriertes «Template»-Projekt zur Verfügung.
  - Vereinfacht den Einstieg in die professionelle Entwicklung.
- Dieses Template-Projekt definiert unter anderem eine **Standard-Verzeichnisstruktur** für die Ablage aller Quell-Artefakte.
- Es beinhaltet einen automatisierten, auf → [Apache Maven](#) basierenden und somit **IDE-unabhängigen Buildprozess**.
  - Kompilierung, Generierung der Javadoc, Ausführen von Testfällen, Messung Codeabdeckung, statische Codeanalyse etc.
- Sie können ganz «normal» in Ihrer IDE arbeiten, die enthaltene Funktionalität ist für Sie noch weitgehend eine «Blackbox».
  - Wird auch in Folgemodulen (AD, VSK, SWDA etc.) verwendet und im Modul VSK detaillierter erklärt.


# Verzeichnisstruktur – Template (Quellen)



- Basisverzeichnis des Projektes
- Konfiguration von Apache Maven
- Diverse Konfigurationsdateien
- Basisverz. für produktiven Java Quellcode.
- Konfigurationen und Ressourcen, Produktiv
- Basisverzeichnis für Testcode (JUnit Tests)
- Konfigurationen und Ressourcen, Tests
- Zielverzeichnis für alle Buildartefakte, wird generiert, kann jederzeit gelöscht werden.

# **Demo – IDEs**

# Anwendung des Templates - Kurzfassung

- ZIP-Datei im gewünschten Verzeichnis (Workspace) entpacken.
  - Verzeichnis nach Wunsch umbenennen.
- Projekt in der IDE öffnen bzw. importieren.
  - Projekt ggf. umbenennen (passend zum Verzeichnis).
- Sobald Sie eigene Klassen implementiert haben, können Sie die vorhandenen (Demo-)Klassen und Testfälle löschen.
- **Achtung:** Wir entwickeln plattformübergreifend! 
  - ➔ File-Encoding in Editoren auf **UTF-8** setzen!
    - Damit der Austausch auch über verschiedene Plattformen (Windows, Mac, Linux etc.) funktioniert.
- Mehr und detailliertere Infos finden Sie in der Anleitung auf ILIAS:  
**OOP\_JavaDevelopmentManual\_jdk21.pdf.**



# **IDE vs. BlueJ**

# Vergleich: Arbeit in BlueJ vs. Entwicklungsumgebung

- **Lernumgebung** - BlueJ:
    - Einfacher Einstieg in die objektorientierte Programmierung.
    - Typisch für sehr kleine, überschaubare **Lern**projekte.
    - Quellen und Artefakte liegen in einem einzigen Verzeichnis.
  - **Entwicklungsumgebung** - NetBeans, Eclipse JDT, IntelliJ, VSC
    - Für die professionelle SW-Entwicklung ausgelegt.
    - Es ist deutlich mehr automatisiert und «intelligenter».
    - Wir arbeiten mit einer wohldefinierten Verzeichnisstruktur.
    - Wir arbeiten mit Packages.
    - Wir arbeiten in einem einzigen Projekt (weniger overhead).
    - Wir → testen (ab SW06) wesentlich effizienter.
- Keine Konkurrenz – da **nicht** wirklich **vergleichbar**!

# **Empfehlungen zu IDE's**

# Empfehlungen – sich mit IDE auseinandersetzen



- Setzen Sie sich bewusst mit der ausgewählten IDE auseinander und lernen Sie sie wirklich kennen!
  - Erlernen Sie ganz bewusst die Bedienung der IDE (→Shortcuts).
- Eine IDE können Sie auch wechseln oder ggf. parallel benutzen!
  - Machen Sie sich nicht zu stark von einer IDE abhängig!
  - Verschiedene Projekt(-arten) mit verschiedenen IDEs bearbeiten.
- Eine IDE muss Sie in Ihrer Arbeit(-smethodik) unterstützen.
  - Soll Ihnen keine «Umwege» aufzwingen und nicht behindern.
- Plugins gehören dazu und machen auch Spass, aber:
  - Die Suche und Auswahl von Plugins ist zeitaufwändig.
  - Die Abhängigkeit wird dadurch wieder grösser / komplexer!
- **Jede** IDE unterstützt ein «dark theme» - **kein** Auswahlkriterium!

# Empfehlungen – bleiben Sie offen!



- Mit der Zeit (und Erfahrung) wird sich Ihnen eine "Lieblings-IDE" herauskristallisieren.
  - Dennoch (oder gerade deshalb) kann es auch sein, dass Sie in Zukunft parallel mit mehreren IDE's arbeiten werden.
- **Die perfekte IDE gibt es nicht.** Sie werden immer Nachteile und Schwächen finden.
  - Die «beste» IDE ist eine ziemliche Glaubensfrage...
- Wichtig: Wählen Sie die IDE **nicht** nur aufgrund der Optik aus, oder weil sie als «cool» gilt!
- **Alle** IDE versuchen Ihnen mit zahlreichen Tipps zu helfen. Diese Tipps sind aber für erfahrene Entwickler\*innen.
  - Für Unerfahrene führen die Tipps manchmal in die Sackgasse.

# **Empfehlungen für die Übungen**



# Empfehlungen - für die Übungen

- Wir empfehlen Ihnen für die Übungen des Modules OOP ein **einziges** Projekt zu erstellen.
  - Beispiel: `oop_exercises`
- Die einzelnen Übungen können Sie darin über **Packages** gliedern.
  - Pro Semesterwoche nummeriert, z.B. `ch.hs.lu.oop.sw05`
- Enthält eine Übung mehrere Teile, können Sie die Namensgebung individuell weiter verfeinern, z.B. `ch.hs.lu.oop.sw07.part1`
- **Vorteile** dieses Vorgehens:
  - Sie müssen nicht ständig neue Projekte einrichten.
  - Sie können den Code schnell mit älteren Übungen vergleichen.
  - Sie haben alles beieinander und es ist relativ einfach.
  - Sie können sich auf das Code schreiben konzentrieren.

# Zusammenfassung

- Entwicklungsumgebungen sind hochspezialisierte SW-Produkte!
  - Darauf ausgelegt, EntwicklerInnen die Arbeit zu erleichtern.
  - Man muss die Bedienung aber wirklich erlernen!
- Es lassen sich alle Funktionen interaktiv über Menüs und Wizards ausführen, aber: Für die wirklich effiziente Arbeit nutzt man sehr oft Shortcuts!
  - siehe z.B. <https://www.shortcutworld.com/>
- Damit sie effizient arbeiten können, müssen Sie die Bedienung Ihrer IDE wirklich bewusst erlernen.
- Machen Sie sich nie zu stark von einer bestimmten IDE abhängig!
- **Empfehlung:** Alle Übungen, die **nicht** aus dem Buch kommen, ab sofort in der IDE durchführen.





**Fragen?**

Fragen bitte im  
**ILIAS-Forum**

