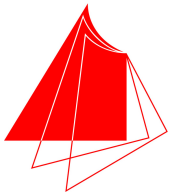
Softwarelabor
1. Einführung

Prof. Dr.-Ing. Holger Vogelsang
holger.vogelsang@fh-karlsruhe.de

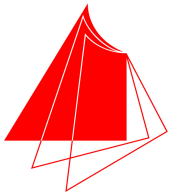


Inhalt

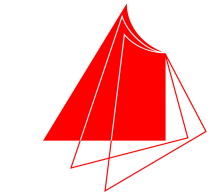
1. Einführung

- Warum Java?
- Marktanforderungen
- Geschichte von Java
- Eigenschaften von Java
- Einführung in Eclipse
- Erste Fingerübungen

1. Einführung
Warum Java?



- Java ist derzeit die Programmiersprache für das Internet.
- Java ist ein Programmierkonzept der Zukunft.
- Java-Entwickler haben gute Marktchancen (siehe z.B. www.gulp.de).
- Objekt-orientierte Programmierung ist für komplexe und umfangreiche Aufgaben unerlässlich.
- Plattformunabhängigkeit und Netzwerkfähigkeit sind Voraussetzungen für eine weltweite Verfügbarkeit einer Anwendung.
- Java verzichtet auf prozedurale Elemente (gute Lehrsprache für die objekt-orientierte Programmierung).
- Sehr umfangreiche und gute Klassenbibliothek als Standard vorhanden.
- Java macht Spaß.



Java ist ...

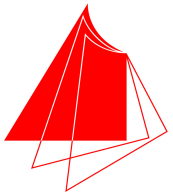
- **Insel Indonesiens**

(126 650 km 2 , 65 Mill. Einw.)

- **am. sl. für Kaffee**

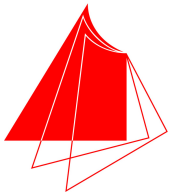
  („Betriebsstoff mancher Java-Programmierer“)
- Software-Technologie für objekt-orientierte Programmierung und verteilte Anwendungen (und damit Thema der folgenden Folien)

1. Einführung
Java OOP und das Internet



- Java ist neben einer Programmiersprache auch eine Softwaretechnologie und umfasst:
  - objektorientierte Programmierung
  - eine leistungsfähige Klassenbibliothek (API)
  - „Internet-Fähigkeit“
  - Module und Schnittstellen für die schnelle Erstellung verteilter Anwendungen
- Java wird von SUN Microsystems entwickelt und als frei verfügbare Entwicklungsumgebung (JDK) verteilt.
- Java nutzt das Internet als Infrastrukturbasis
- Java-Programme (Bytecode) sind plattformunabhängig
- Typen von Programmen:
  - Applets: In einem Browser ausführbar.
  - Applikationen: eigenständig ausführbares Java-Programm
  - Weitere (nicht Bestandteil der Vorlesung)

1. Einführung
Geschichte von Java



- 1991: Erstes Projekt bei SUN (James Goslin), Ziele:
  - einfache Programmiersprache für Steuerungselektronik
  - grafische Benutzeroberfläche für Unterhaltungs- und Haushaltselektronik
  - klein, einfach portierbar
- 1992: Oak (Vorgänger von Java)

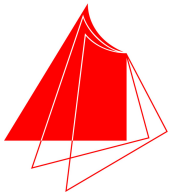
  Computer zur Steuerung von Haushaltsgeräten

  „Duke who helped users through the easy-to-use, image rich, graphical user interface“

- 1993: Hotjava

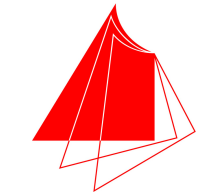
  WWW-Browser mit Java-Interpreter, vollständig in Java geschrieben.
- 1995: SUN kündigt Java auf der SunWorld’95 an, Netscape wird Java-fähig
- jetzt: JDK1.4.2

  Es werden sehr viele Java-Entwickler in der Industrie benötigt.

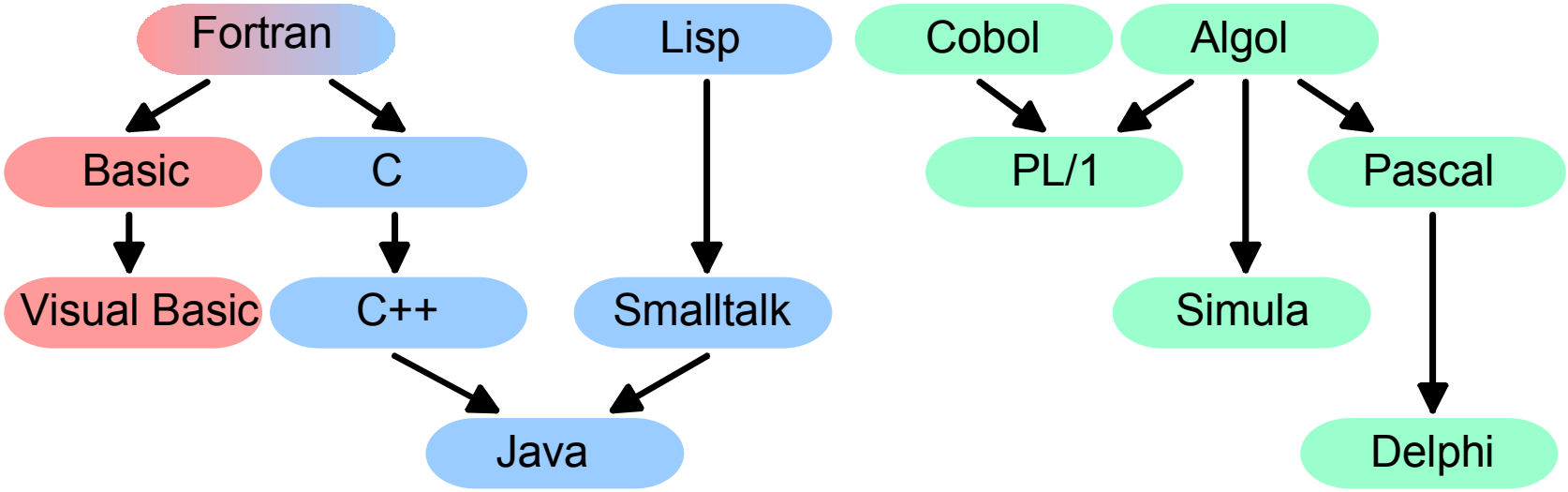


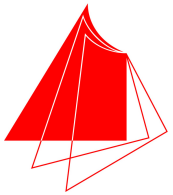
James Goslin

- Vize-Präsident von
  Sun Microsystems, Inc.
- Chef-Ingenieur und maßgebender
  Architekt der Java-Technik
- Seit 1984 befasste er sich mit
  verteilten Computersystemen
- Konstruierte ...
  - Mehrprozessorversion für UNIX
  - Das Andrew Window-System
  - Einige Compiler und mail-Systeme
  - Emacs-Texteditor für UNIX.



Genealogie einiger wichtiger Programmiersprachen

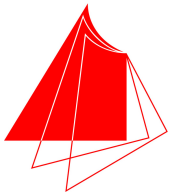




Java White Paper:

> *„... **simple**, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded and dynamic language“*

- Unkompliziert
  - einfach zu lernen (viel einfacher als C++)
  - Java ist eine Weiterentwicklung von C++
    - Beschränkung auf das Notwendigste (keine Zeiger, keine Header-Dateien, keine Unions)
    - Verzicht auf Trickkisten
    - Kein Precompiler (keine Makros)
- Klein: Läuft auf Rechnern mit kleiner Ausstattung. Das Grundsystem (Interpreter, Standardklasse, Thread-Unterstützung) benötigt ca. 215 KB Hauptspeicher.



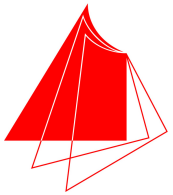
Java White Paper:

> *„... simple, **object-oriented**, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded and dynamic language“*

- Vollständig objekt-orientiert:
  - Ein Objekt besteht aus Daten und Zugriffsmethoden.
  - Im Idealfall erzeugt man wiederverwendbare Objekte.
  - Im Laufe der Arbeit kann auf eine Bibliothek bereits erzeugter Objekte (Baukasten) zugegriffen werden (Wiederverwendung).
  - Ein Satz solcher Bibliotheken wird mit der Sprache mitgeliefert.
  - Die wesentlichen Konzepte der objekt-orientierten Programmierung wurden umgesetzt:
    - Kapselung (Daten und Funktionen)
    - Vererbung
    - Polymorphie

1. Einführung

Java als Programmiersprache (distributed)



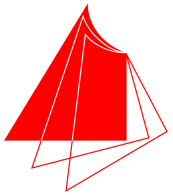
Java White Paper:

> *„... simple, object-oriented, **distributed**, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded and dynamic language“*

- Verteilt = internet-fähig:
  - Durch Java-Applets:
    - Die können über das Internet verteilt und lokal in einem Browser ausgeführt werden.
  - Durch eine geeignete Klassenbibliothek:
    - TCP/IP-Kommunikation (Sockets)
    - HTTP/FTP-Anwendungen
    - RMI für einen sehr einfachen entfernten Methodenaufruf
- Ziele:
  - Software muss nicht mehr auf dem Client installiert werden.
  - Durchbrechen des Microsoft-Monopols der Betriebssysteme.

1. Einführung

Java als Programmiersprache (interpreted, high-performance)



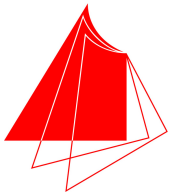
Java White Paper:

> *„... simple, object-oriented, distributed, **interpreted**, robust, secure, architecture neutral, portable, **high-performance**, multithreaded and dynamic language“*

- interpretiert = schnell?
  - Java-Code wird nicht direkt durch den Prozessor ausgeführt, sondern durch einen zwischengeschalteten Interpreter.
  - Vorteile:
    - Kein Linker erforderlich
    - Plattformunabhängiger Code
  - Nachteile:
    - Geringere Ausführungsgeschwindigkeit
  - Lösung:
    - Bytecode als maschinennaher Code
    - JIT- und Hotspot-Compiler, Java-Prozessoren

1. Einführung

Java als Programmiersprache (Compiler und Interpreter)

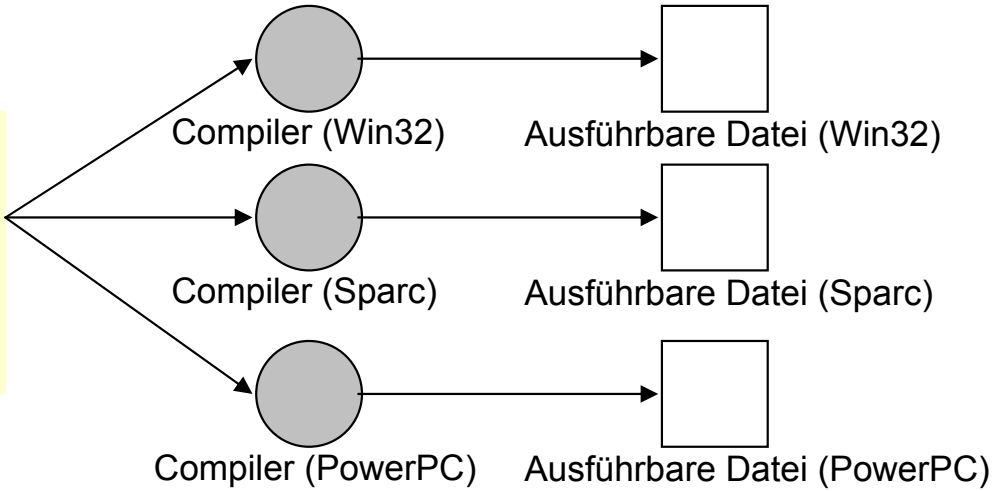


- Compiler

Code

```
import java.text.*;

class Test
{
  void print(){}
}
```

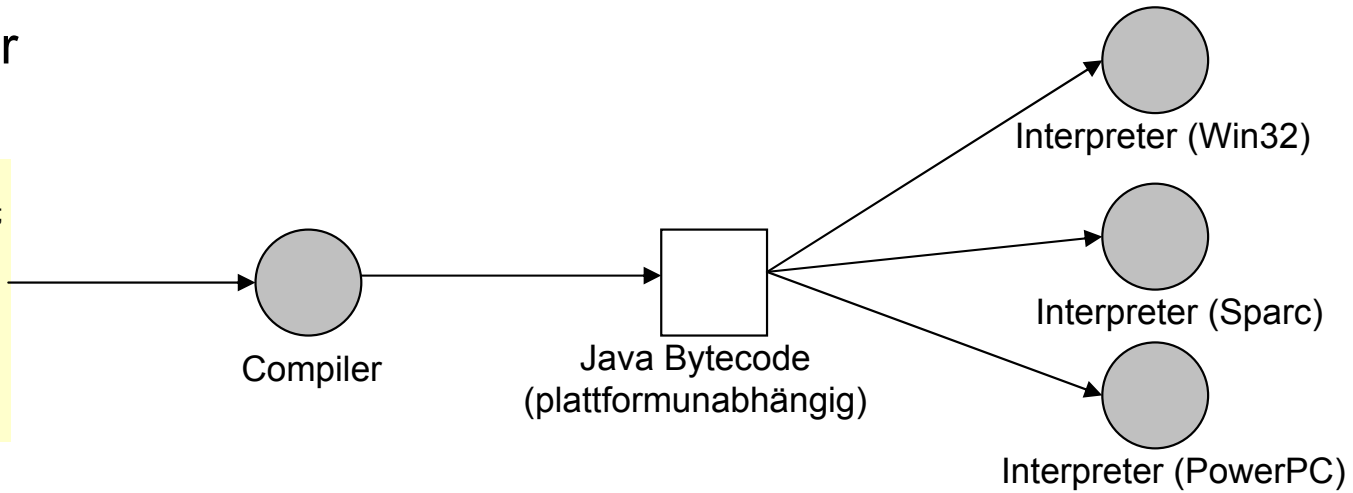


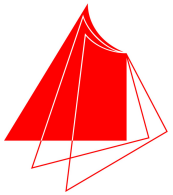
- Interpreter

Code

```
import java.text.*;

class Test
{
  void print(){}
}
```

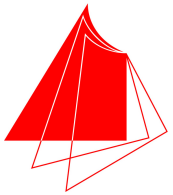




Java White Paper:

> *„... simple, object-oriented, distributed, interpreted, **robust**, **secure**, architecture neutral, portable, high-performance, multithreaded and dynamic language“*

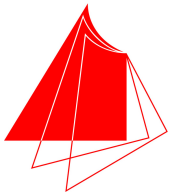
- robust und sicher: Vorteile durch die Verwendung von Compiler und Interpreter:
  - Der Compiler kann Typfehler und andere logische Programmierfehler finden.
  - Der Interpreter stellt sicher, dass kein Zugriff auf fremde Programme oder Daten erfolgen kann.
  - Der Interpreter kann Laufzeitfehler mittels sog. Exceptions abfangen, auf die das Programm gezielt reagieren kann (keine Rechnerabstürze durch Programmierfehler).



Java White Paper:

> *„... simple, object-oriented, distributed, interpreted, robust, secure,* ***architecture neutral****,* ***portable****, high-performance, multithreaded and dynamic language“*

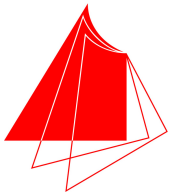
- plattformunabhängig und portierbar
  - Der erzeugte Bytecode läuft auf allen Java-Plattformen unabhängig vom Prozessor.
  - Keine ungenauen Spezifikationen wie in C++ (die Größe des Integer-Datentyps `int` ist nicht festgelegt).



Java White Paper:

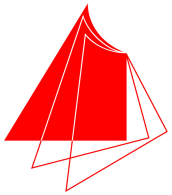
> *„... simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance,* ***multithreaded*** *and* ***dynamic language****“*

- Multithreading:
  - In einem modernen Programm laufen viele Aktionen „gleichzeitig“ (besser: unabhängig und möglicherweise parallel) ab.
  - Java bietet eine Unterstützung für die Parallelität, die dem Entwickler viel Arbeit abnimmt.
- Dynamik:
  - In Java können Klassen zur Laufzeit nachgeladen werden (z.B. auch über das Netzwerk). Dieses funktioniert nur durch die Verwendung eines Interpreters.



Java-Werkzeuge

- Der Java-Compiler erzeugt Java-Bytecode.
- Der Java-Bytecode ist
  - plattformunabhängig,
  - im Internet verteilbar und
  - auf jedem Rechner mit einer Java Virtual Machine (JVM) lauffähig.
- Die Java Virtual Machine besteht aus:
  - Bytecode-Lader
  - Bytecode-Verifizierer (überprüft die Sicherheitsregeln)
  - Bytecode-Interpreter (führt das Programm aus)
  - eventuell Hotspot-Compiler oder JIT: Übersetzung des Bytecodes in Maschinencode zur Laufzeit!

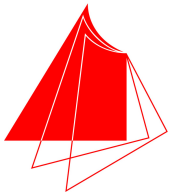


Java-Einsatzgebiete

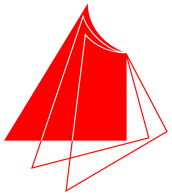
- Geeignet für prinzipiell alle programmierbaren Anwendungen:
  - Benutzeroberflächen
  - Grafik und Animation
  - verteilte Anwendungen
  - Information, Animation, Kunst, Unterhaltung im WWW
- Beispiele:
  - Intelligente Information mit Multimedia und Interaktion in Wissenschaft und Ausbildung.
  - Benutzeroberflächen für Datenbankzugriffe
  - Kommunikations- und Büroanwendungen

Java ist ungeeignet für

- Hardware-nahe Anwendungen (Treiber), da das Sicherheitskonzept einen Hardwarezugriff verbietet. Ausweg: Java Native Interface (Kombination aus Java und C/C++/Assembler).



- Zur Übersetzung und zum Ausführen von Programmen wird eine Entwicklungsumgebung benötigt.
- Hier wird das frei verfügbare Java Development Kit (JDK) von SUN in der Version 1.4.2 verwendet.
- Unterstützte Plattformen (Auszug):
  - SUN Solaris ab Version 2.6
  - LinuX (auf Intel, Alpha, Sparc)
  - Windows (95/98/ME/NT/2000/XP, aktuell nur noch auf Intel)
  - OS/2 (Fremdanbieter)
  - AIX
  - MacOS X
  - ...



Im JDK verfügbare Tools (Auszug)

- **javac**: Java-Compiler

  Erzeugt Java-Bytecode aus Java-Quellcode
- **java**: Java-Interpreter für Bytecode

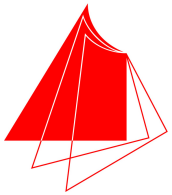
  Zum Ausführen von Java-Programmen

  (enthält ab JDK 1.1.7 einen Just-In-Time-Compiler und ab JDK 1.2.2 einen Hotspot-Compiler)
- **appletviewer**: Java Applet-Viewer

  Einfacher Browser zum Testen von Java-Applets
- **jdb**: Java Debugger

  Notwendig, um Fehler in Java-Programmen zu finden
- **javap**: Class-File Disassembler

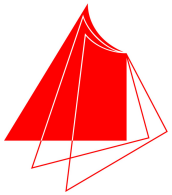
  Disassembliert compilierte Dateien



- **javadoc**: Java Documentation Generator

  Parst Deklarationen und Dokumentationskommentare in einer Menge von Quelldateien und erzeugt eine HTML-Dokumentation
- **jar**: Java Archive Tool

→ Viele der Werkzeuge sind in einer IDE eingebettet.

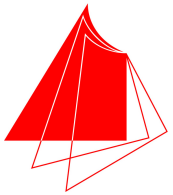


- Doppelklick auf das ausführbare JDK-Archiv.

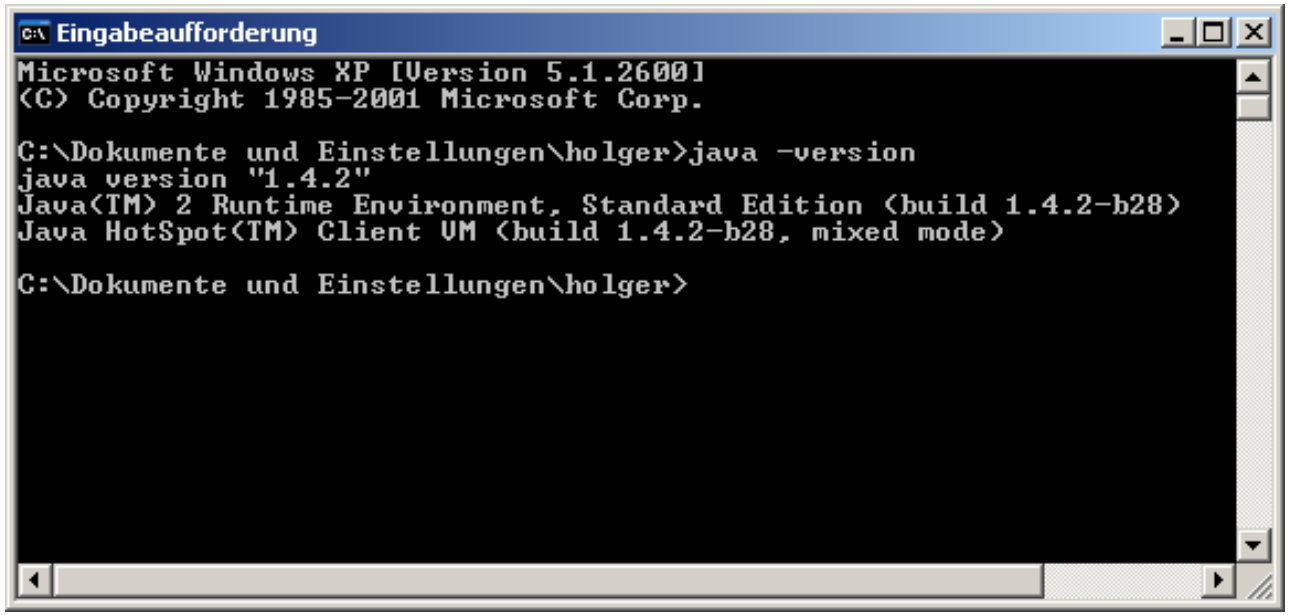
  (Zielverzeichnis z. B. C:\ Programme\Sun\JDK1.4.2)
- Installation der JDK-Dokumentation

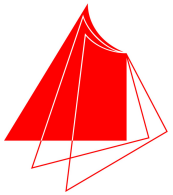
  Benötigt wird ein „Entpackungs-Programm“ (z. B. WinZip). Die Dokumentation sollte in das Verzeichnis der JDK-Software entpackt werden (also z. B. C:\ Programme\Sun\JDK1.4.2), damit die HTML-Links der JDK-Dokumentation und JDK-Demodateien korrekt funktionieren.
- Verzeichnisstruktur:

  C:\Programme\Sun\JDK1.4.2\bin
  \demo
  \docs
  \include
  \lib
  \src



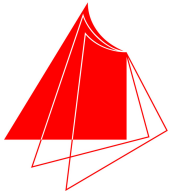
- Wenn das Verzeichnis der ausführbaren Programme (hier: C:\Programme\Sun\JDK1.4.2\bin) nicht im Suchpfad liegt, sollte das Verzeichnis in die PATH-Variable aufgenommen werden.
- Die Programme können dann von der Kommandozeile aus aufgerufen werden, z.B. die Abfrage der JDK-Versionsnummer mit **`java -version`**:



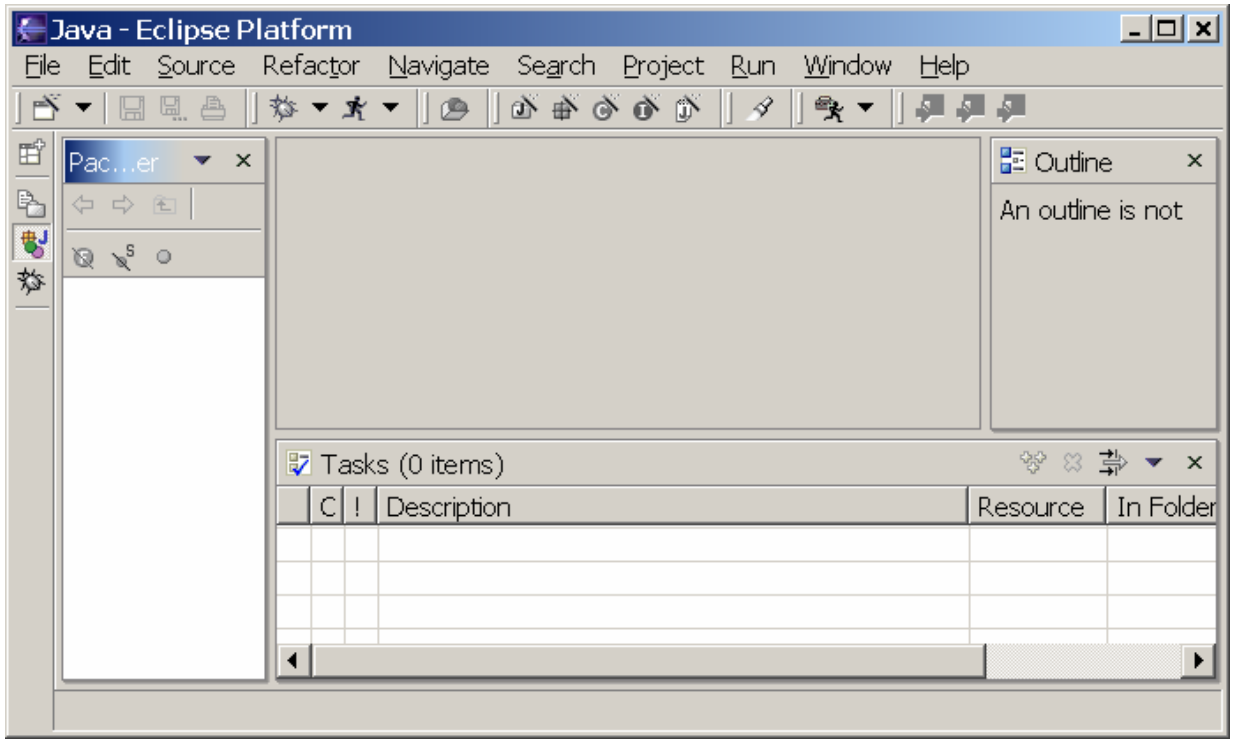


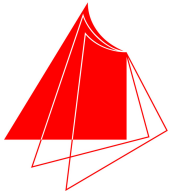
- Eclipse (IBM, alle Plattformen, kostenlos unter www.eclipse.org)
- Forté4Java (SUN, alle Java-Plattformen, Entry-Edition kostenlos von developer.java.sun.com)
- VisualAge for Java (IBM, nur Windows, Entry-Edition kostenlos von www.ibm.com) → wird eingestellt.
- JBuilder (Borland, alle Java-Plattformen, Personal-Edition kostenlos von www.borland.com)
- JDeveloper (Oracle, nur Windows, Download von technet.oracle.com)

1. Einführung
Eclipse

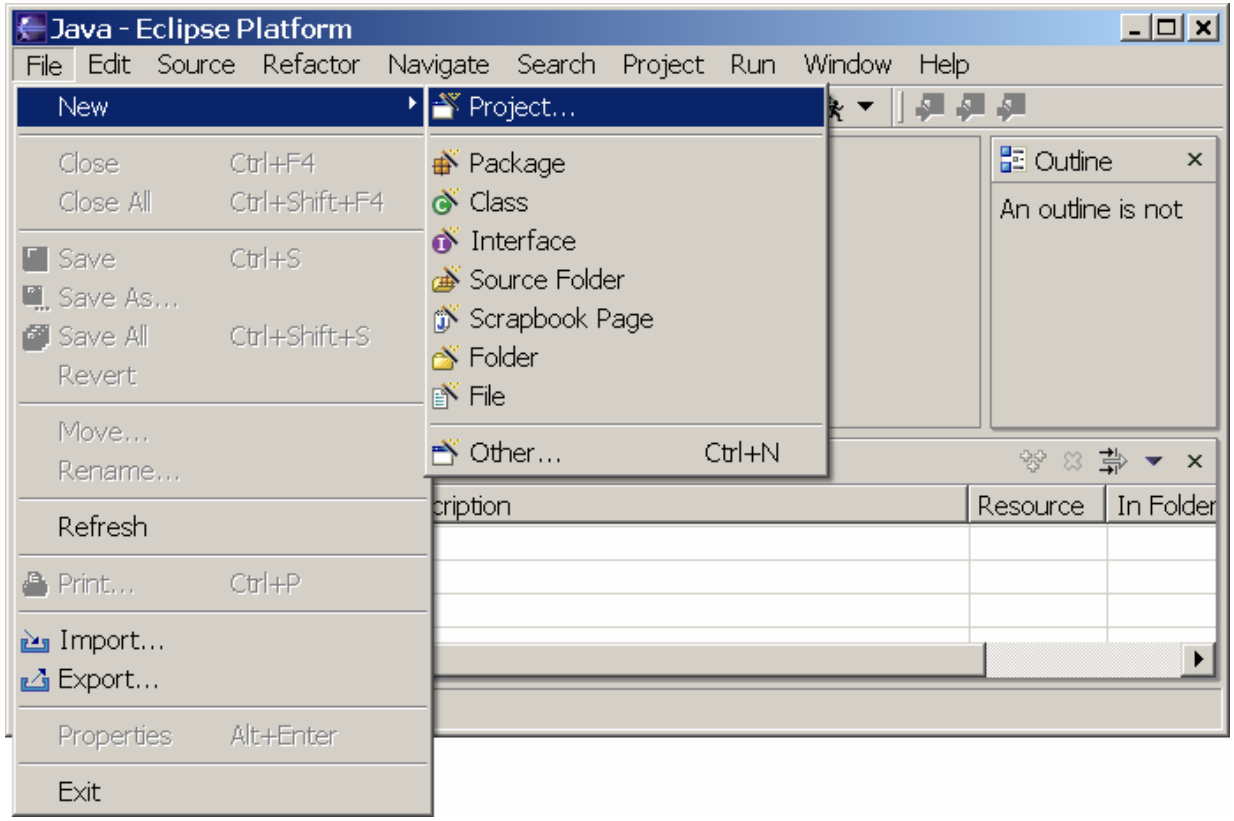


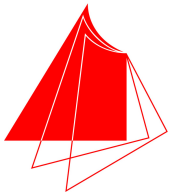
- Alle Übungen werden mit der frei verfügbaren IDE Eclipse durchgeführt.
- Installation: Archiv extrahieren.
- Start: Die ausführbare Datei im Eclipse-Verzeichnis aufrufen.
- Nach dem Start erscheint ein leeres Fenster:



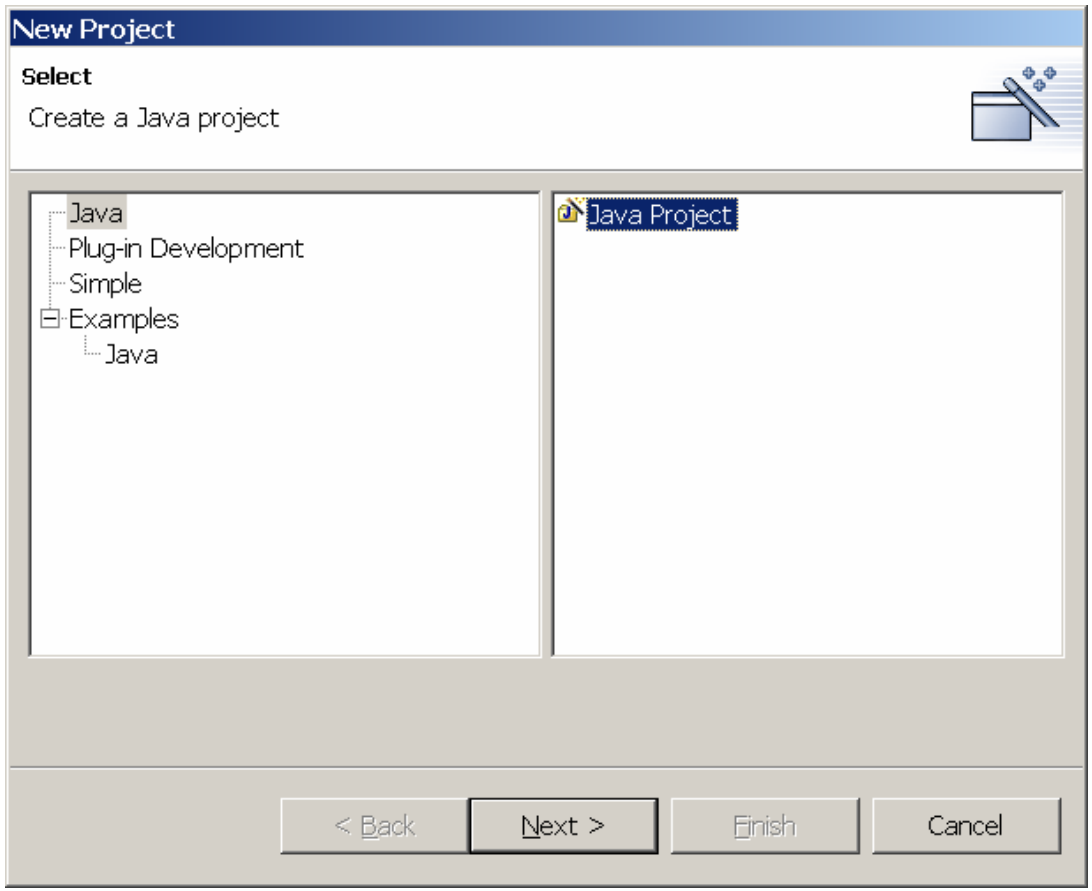


- Erstellung eines neuen Projektes:

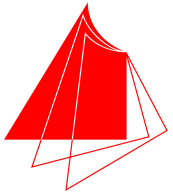




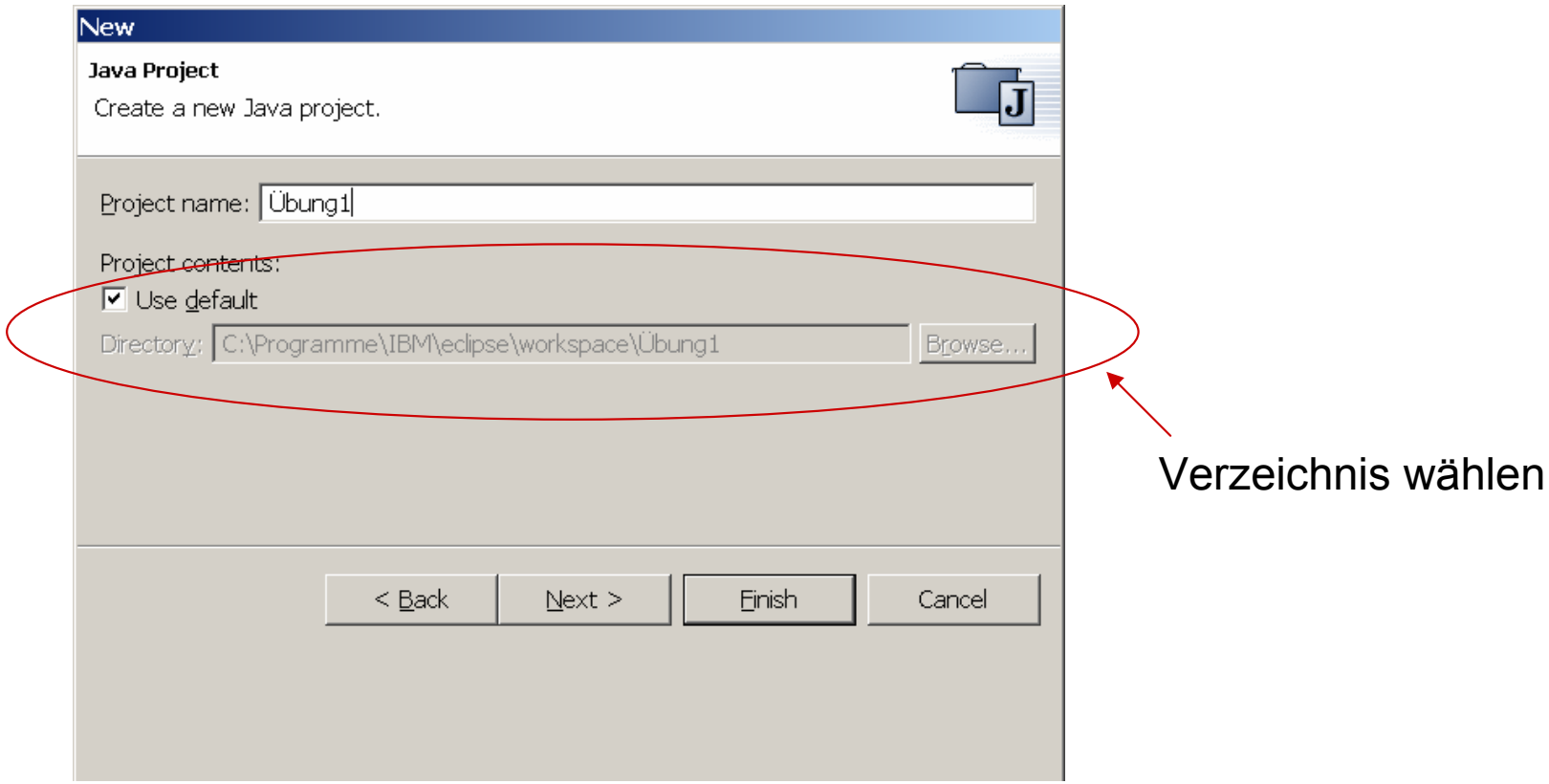
- Projekttyp „Java Project“ auswählen:



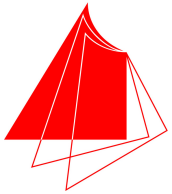
- „Next >“ wählen.



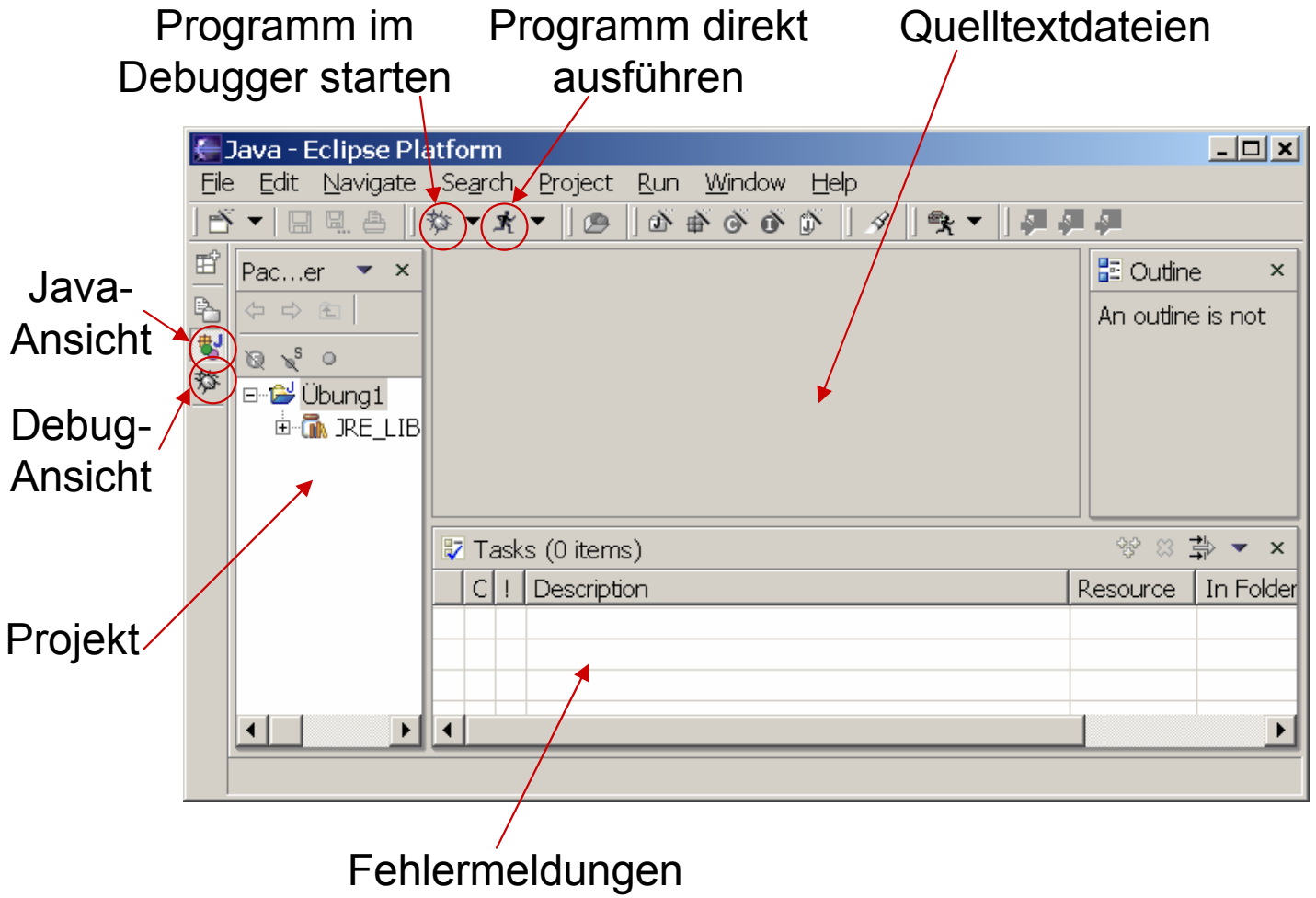
- Projektnamen und eventuell das Verzeichnis des Projektes auswählen:

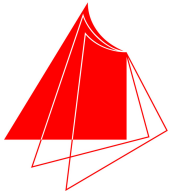


- „Finish“ auswählen.



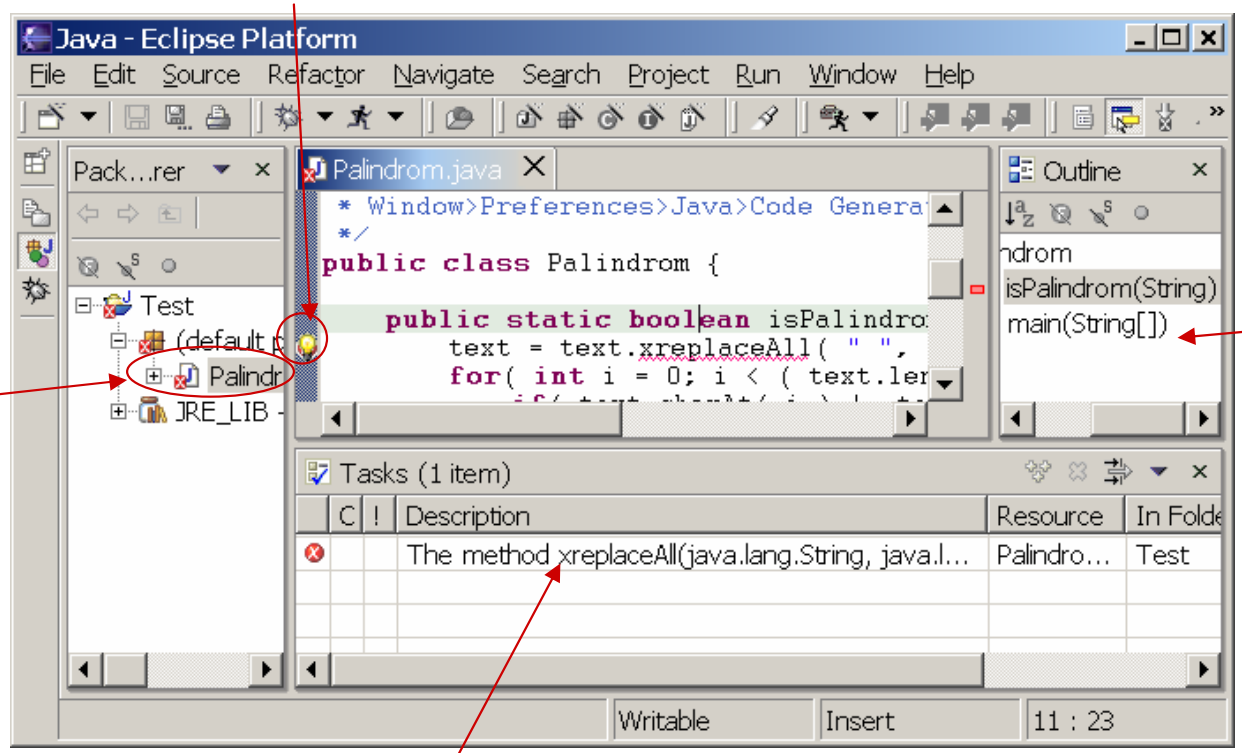
- Wichtige Elemente:





- Beispielprojekt:

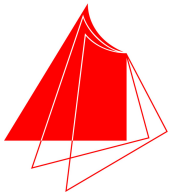
Fehlerstelle



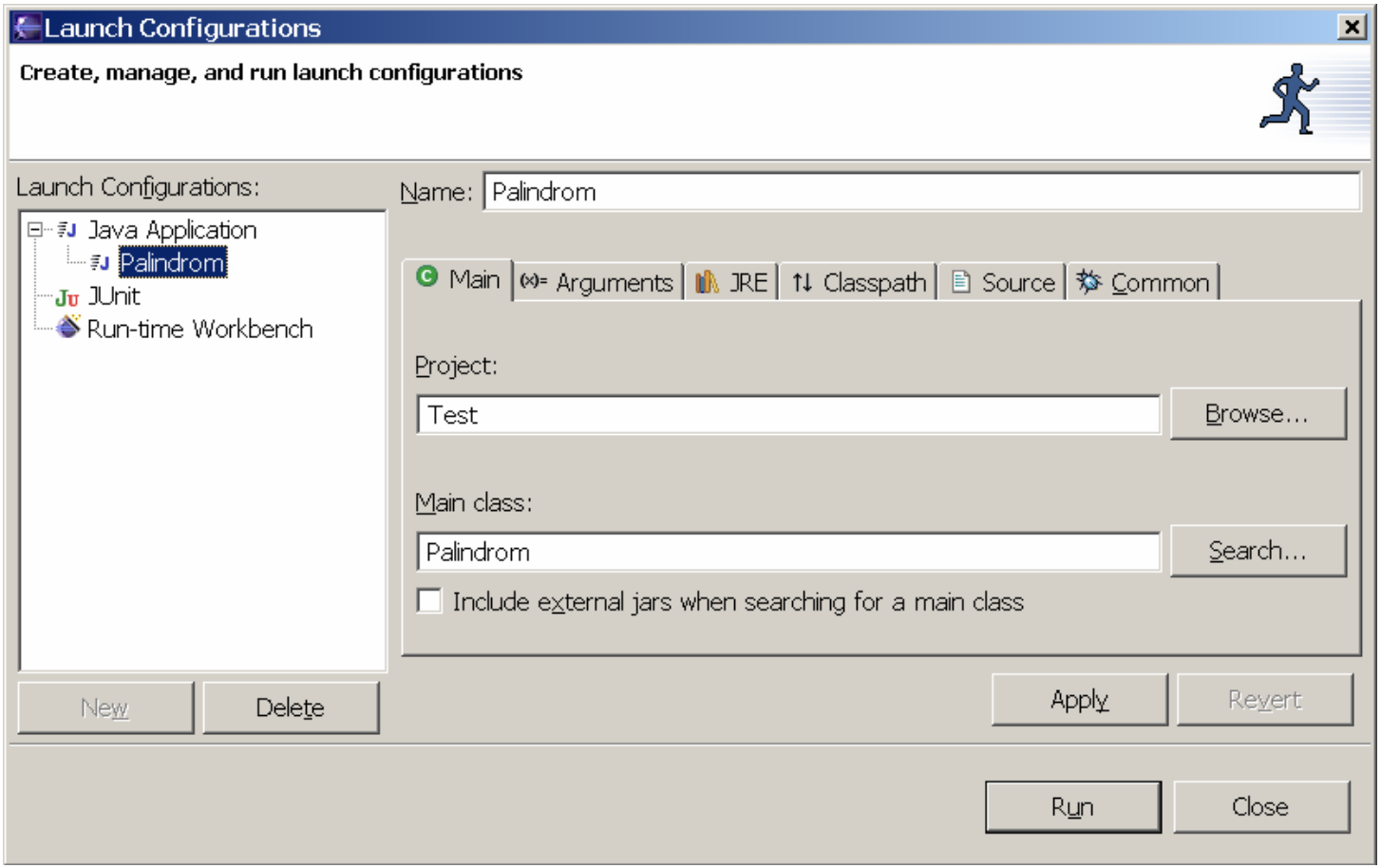
Java-Dateien

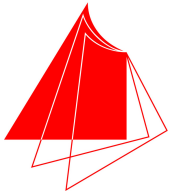
Methoden der aktuellen Klasse

Übersetzungsfehler (Doppelklick springt in den Quelltext)

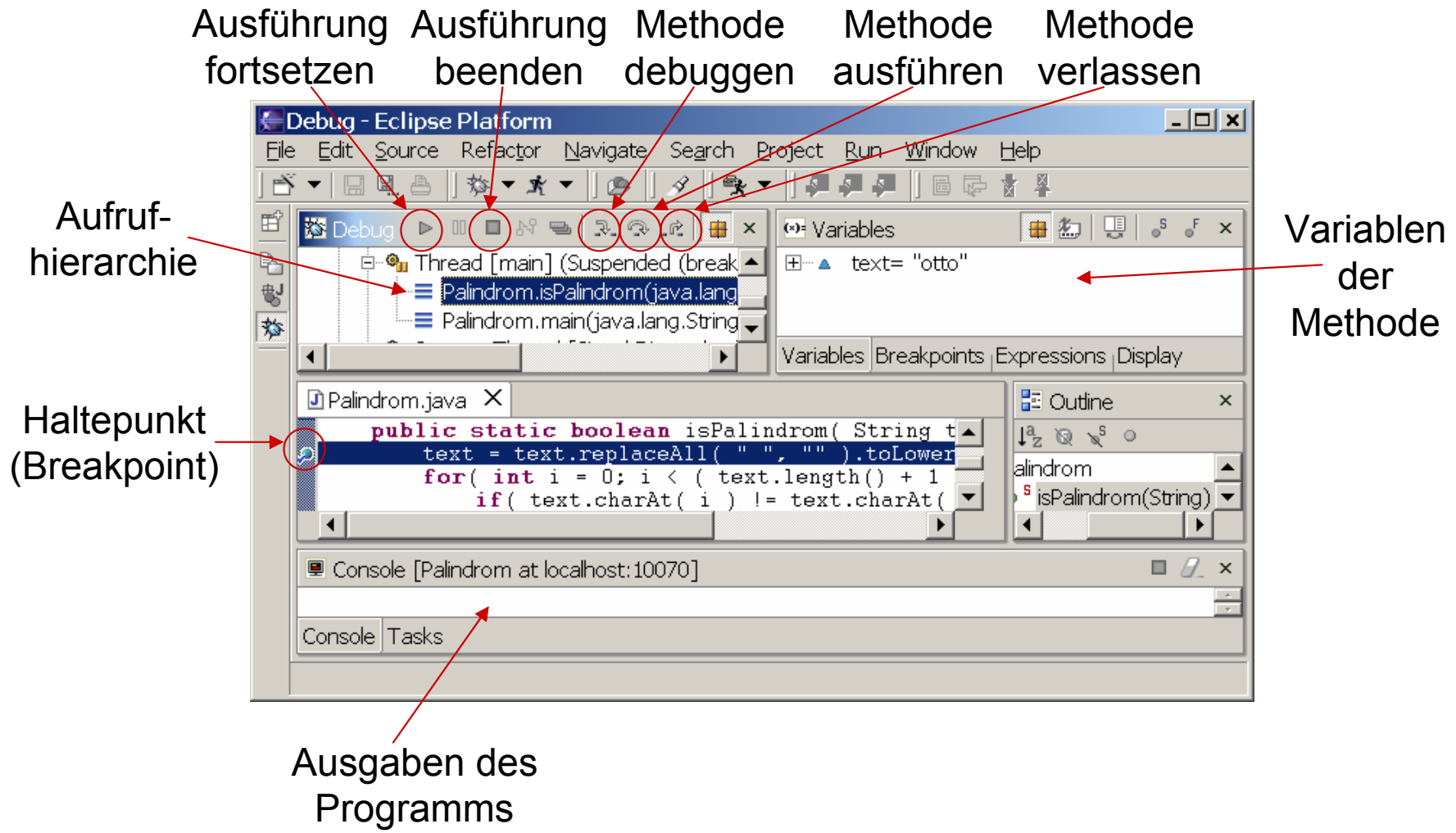


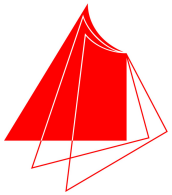
- Beim ersten Start oder durch Auswahl des Menüs Run → Run… muss die Klasse ausgewählt werden, deren Start-Methode aufgerufen werden soll:



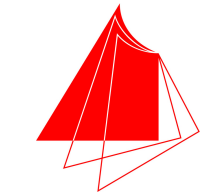


- Arbeiten mit dem Debugger;

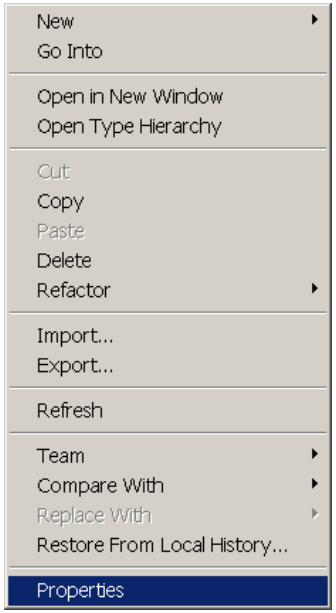


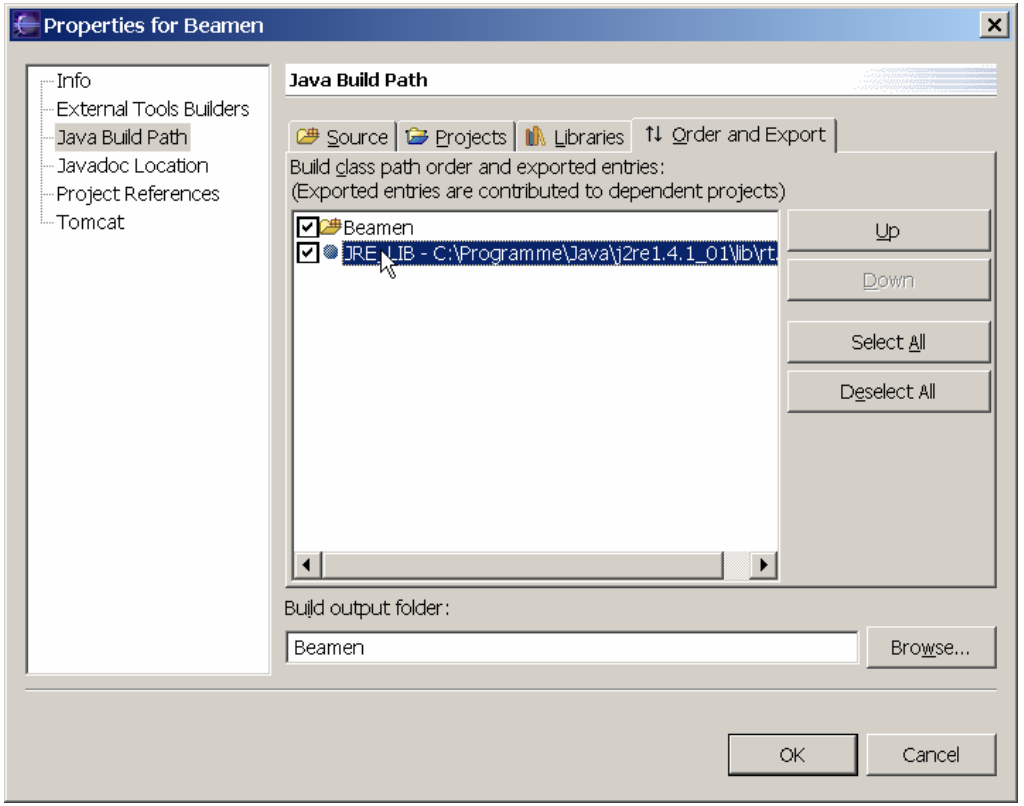


- Änderungen am Code während des Programmablaufs (Hot Code Replacement):
  - Bedingung: Das eigene Programm wird an einem Haltepunkt angehalten.
  - Jetzt direkt Code in der aufgerufenen Methode verändert werden.
  - Nach dem Abspeichern wird der Code neu kompiliert und der Programmzähler automatisch an eine Stelle vor der Änderung zurückgesetzt.
  - Damit das funktioniert, muss in der Pfad auf das JRE in den „Java Build Path“ aufgenommen werden:
    - Click mit der rechten Maustaste auf das Projekt.
    - Auswahl des Menüpunktes „Properties“

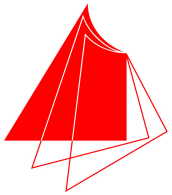


- Im „Java Build Path“ unter „Order and Export“ das JRE auswählen:

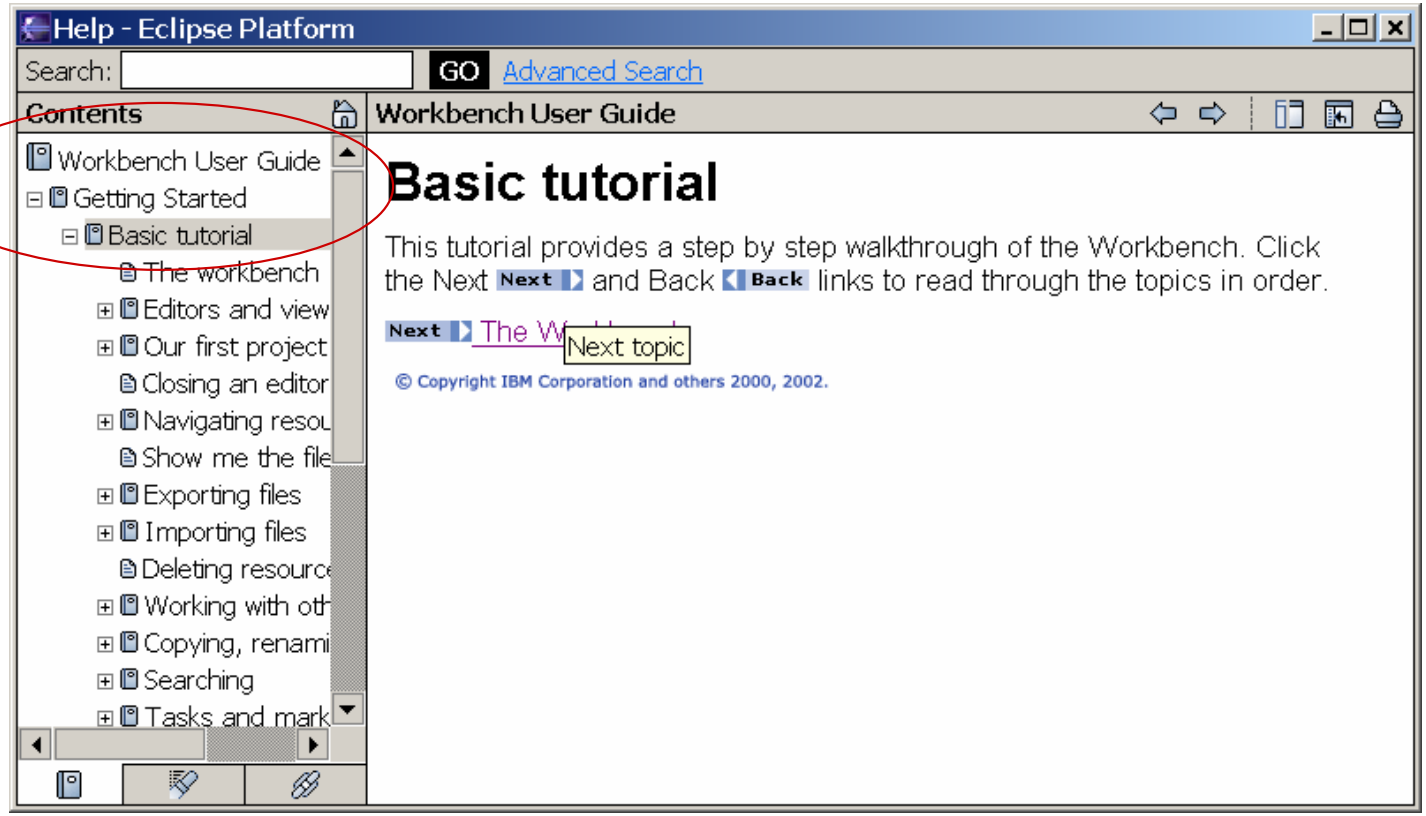


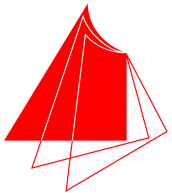


1. Einführung
Eclipse



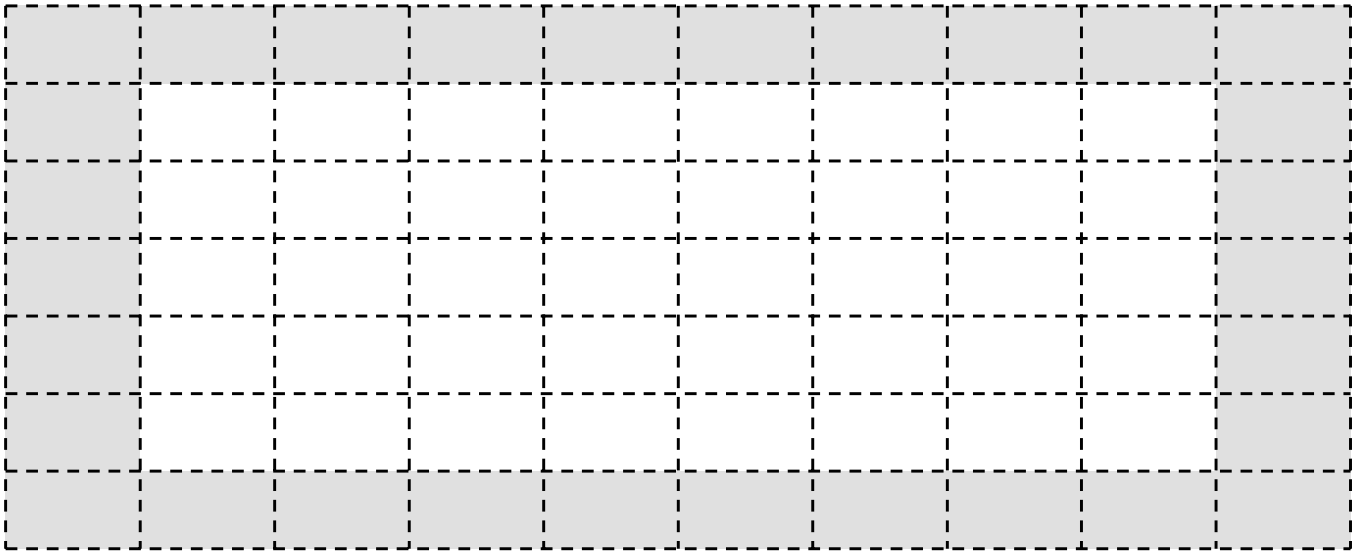
- Tiefergehende Einführung in die Workbench von Eclipse im Menü Help → Help Contents:

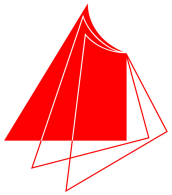




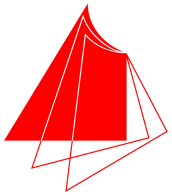
Eine kleine Simulation mit Arrays

- Ein Array stellt ein kleines Spielfeld zur Verfügung, über das ein Ball bewegt wird.
- Das Spielfeld ist von einer Mauer umgeben.
- Beispiel für ein Feld:

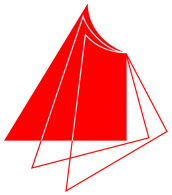




- Ein Element des Arrays ist entweder frei oder durch den Rand belegt.
- Die Größe des Spielfeldes soll beliebig veränderbar sein.
- Schreiben Sie eine Methode **`init`**, die zwei Integer-Parameter übergeben bekommt:
  - **`width`**: Breite des Feldes
  - **`height`**: Höhe des Feldes
- Die Methode erzeugt aus diesen Daten ein Spielfeld und trägt es als Attribut der Klasse ein.
- Die Koordinate (0,0) liegt in der linken oberen Ecke.



- Jetzt wird der Ball an die linke, obere Ecke gesetzt und diagonal in Richtung rechte, untere Ecke bewegt.
- Stößt der Ball auf den Rand, so wird er gemäß Einfallswinkel = Ausfallswinkel bewegt.
- Schreiben Sie eine Methode **`move`**, die den Ball bewegt:
  - Fügen Sie zunächst die folgenden Attribute zur Klasse hinzu:
    - **`ballX`**: X-Position des Balls
    - **`ballY`**: Y-Position des Balls
    - **`direction`**: Richtung des Balls. Überlegen Sie sich, wie Sie die Richtung ausdrücken wollen und welche Möglichkeiten es hier gibt.
  - Die Methode **`move`** verwendet die Attribute, um die neue Ballposition zu bestimmen.
  - Anschließend trägt sie die neuen Werte in die drei Attribute ein.



- Gehen sie dabei von der folgenden Vereinfachung aus:
  - In der Ecke des Spielfeldes wird der Ball immer direkt zurückbewegt.
- Schreiben Sie eine weitere Methode **`print`**, die das Spielfeld in einer einfachen grafischen Darstellung auf dem Bildschirm ausgibt:
  - Rand: #
  - Ball: *
  - Freies Feld: (Leerzeichen)
- Schreiben Sie eine **`main`**-Methode, die ein Spielfeld erzeugt und den Ball 20 mal bewegt. Nach jeder Bewegung geben Sie das Spielfeld aus.
- Kommentieren Sie Ihre Lösung!