

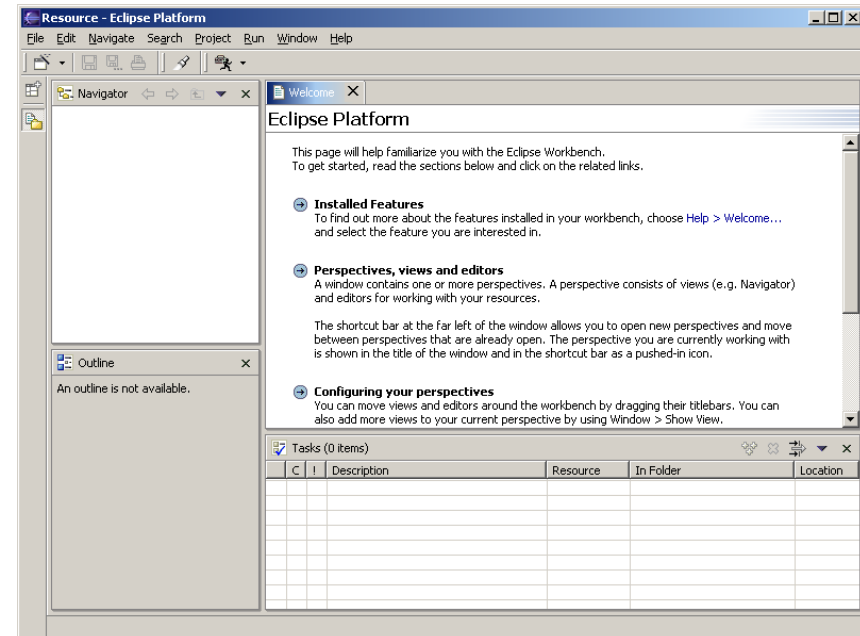
## **JAVA** **- Eclipse**

## Integrierte Entwicklungsumgebungen

- Integrierte Entwicklungsumgebung (IDE):
  - ◆ vereint Editor, Compiler, Debugger und andere Entwicklungswerkzeuge unter einer grafischen Benutzungsschnittstelle
- Open-Source:
  - ◆ *Eclipse* („IBM“, alle Plattformen, kostenlos unter [www.eclipse.org](http://www.eclipse.org))
  - ◆ *Netbeans* („Sun“, alle Plattformen, kostenlos unter [www.netbeans.org](http://www.netbeans.org) )
- Kommerzielle:
  - ◆ *IntelliJ IDEA* (ca. 300 €), <http://www.jetbrains.com>
  - ◆ *JBuilder* (ca. 1500 € ), Borland, alle Java-Plattformen, Personal-Edition kostenlos von [www.borland.com](http://www.borland.com)
  - ◆ *JDeveloper*, Oracle, nur Windows, Download von [technet.oracle.com](http://technet.oracle.com)
  - ◆ *Sun Java Studio*, Sun, basiert auf Netbeans, Name ändert immer wieder, <http://java.sun.com>
  - ◆ *WebSphere Studio Application Developer* (WSAD, ca. 2000 €), IBM, basiert auf Eclipse, Name ändert auch immer wieder einmal
- Das zeitliche gesegnet haben:
  - ◆ VisualAge for Java (IBM, jetzt Websphere Studio Application Developer, auf Basis Eclipse)
  - ◆ Visual Cafe (Symantec, dann BEA)

## Integrierte Entwicklungsumgebung Eclipse

- Alle Übungen werden mit der frei verfügbaren IDE Eclipse durchgeführt.
- Start Rechnerpool (ADS):
  - Start -> Programmieren -> Eclipse -> Eclipse 3
  - Aktuelle Version 3.1
- Folgende Fenster erscheinen:



- Installation (z.B. eigenen PC)
  - JDK benötigt, Download (mind. Version 1.4)  
<http://java.sun.com/j2se/1.5.0/download.jsp>
  - Eclipse SDK Download:  
<http://www.eclipse.org/downloads/index.php>
  - Beides jeweils für gewünschte Plattform (Windows, Linux, ...)
  - Erst JDK installieren, dann Eclipse
  - Eclipse Installation: Archiv extrahieren.
  - Start: eclipse.exe starten (Windows), ggf. Link zu eclipse.exe erstellen und auf Desktop verschieben

### Get Eclipse.

Welcome to the Eclipse downloads section. If you're new to Eclipse, find the useful tools and plugins that you need. If you have problems try posting a question to the [newsgroup](#). All downloads are provided unless otherwise specified.

Download now: [Eclipse SDK 3.0.2](#), Windows.

**JDK 5.0 Update 2** includes the JVM technology

The J2SE Development Kit (JDK) supports creating J2SE applications. [More info...](#)

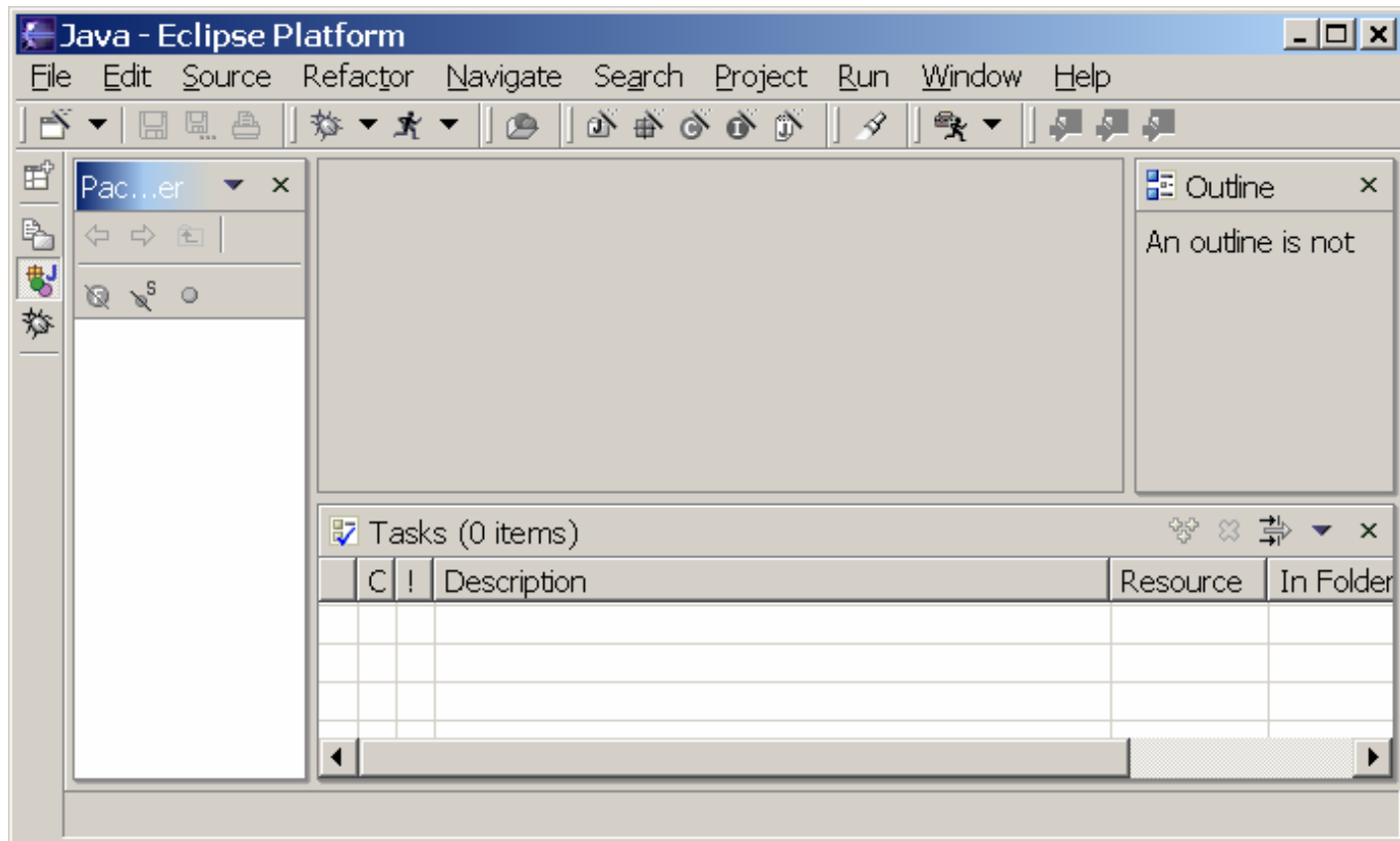
[Download JDK 5.0 Update 2](#)

[Installation Instructions](#) [ReadMe](#) [ReleaseNotes](#)  
[Sun License](#) [Third Party Licenses](#)

**JRE 5.0 Update 2** includes the JVM technology

- Arbeiten mit Eclipse
  - ◆ Java Projekt erstellen
  - ◆ Klassen erstellen und programmieren
  - ◆ Syntaxfehler entfernen
  - ◆ Compiler wird implizit bei Speichern einer (syntaxfehlerfreien) Klasse im Hintergrund gestartet
  - ◆ Programm testen, ggf. logische Fehler mit Quelltextdebugger suchen

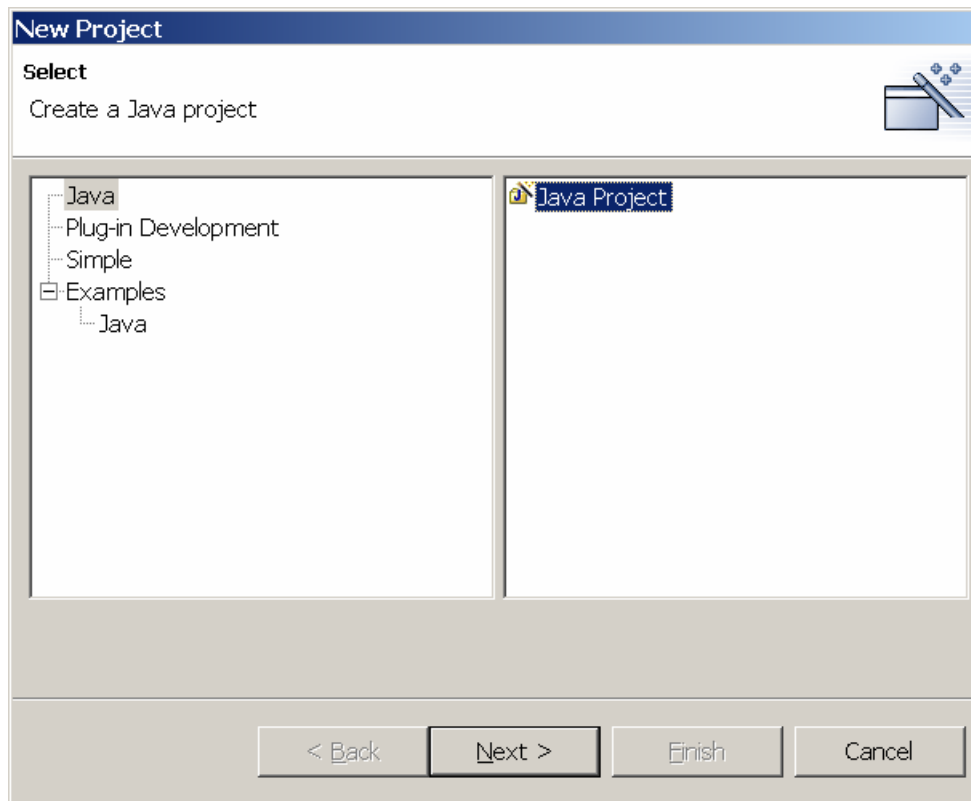
- Nach dem Start erscheint ein leeres Fenster:



# JAVA

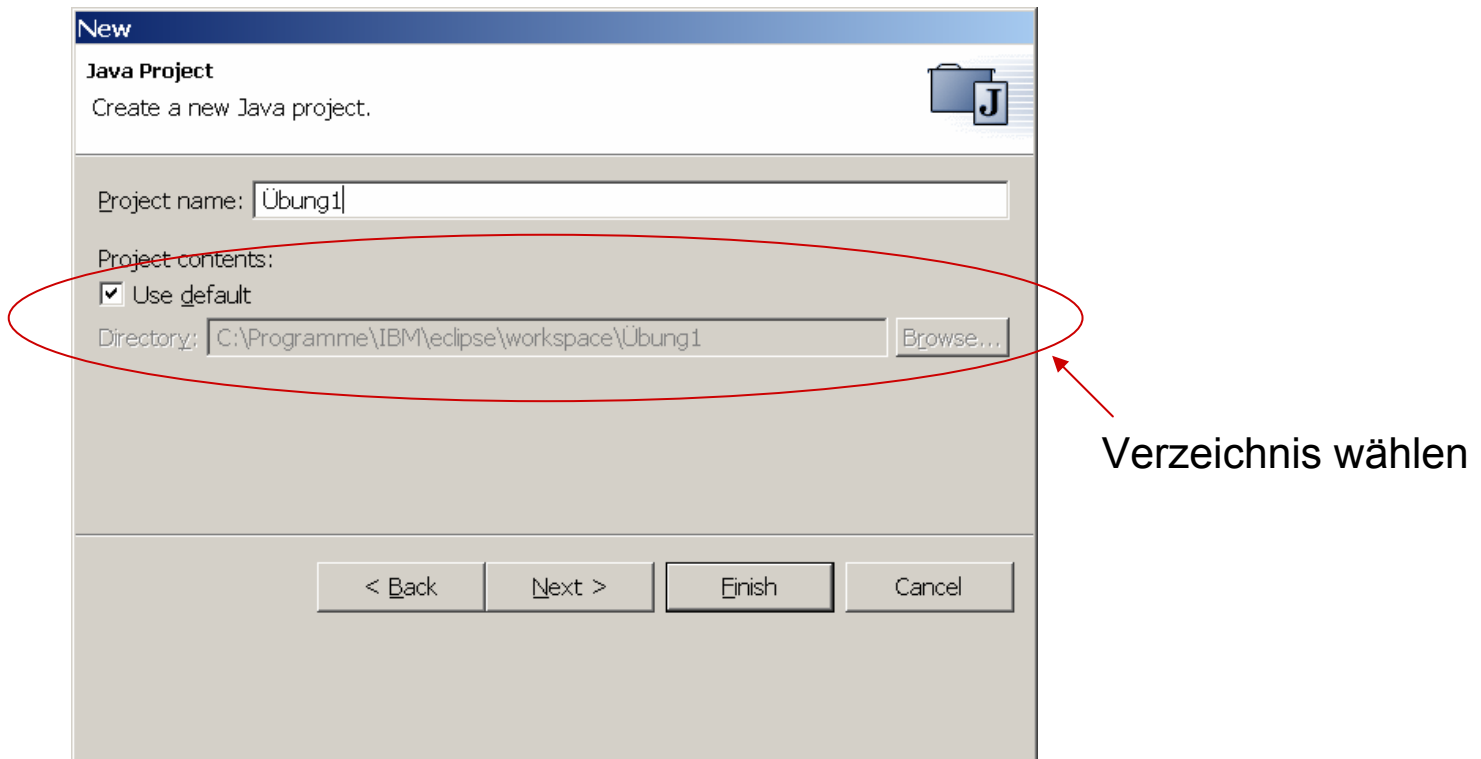
## Eclipse

- Projekttyp „Java Project“ auswählen:



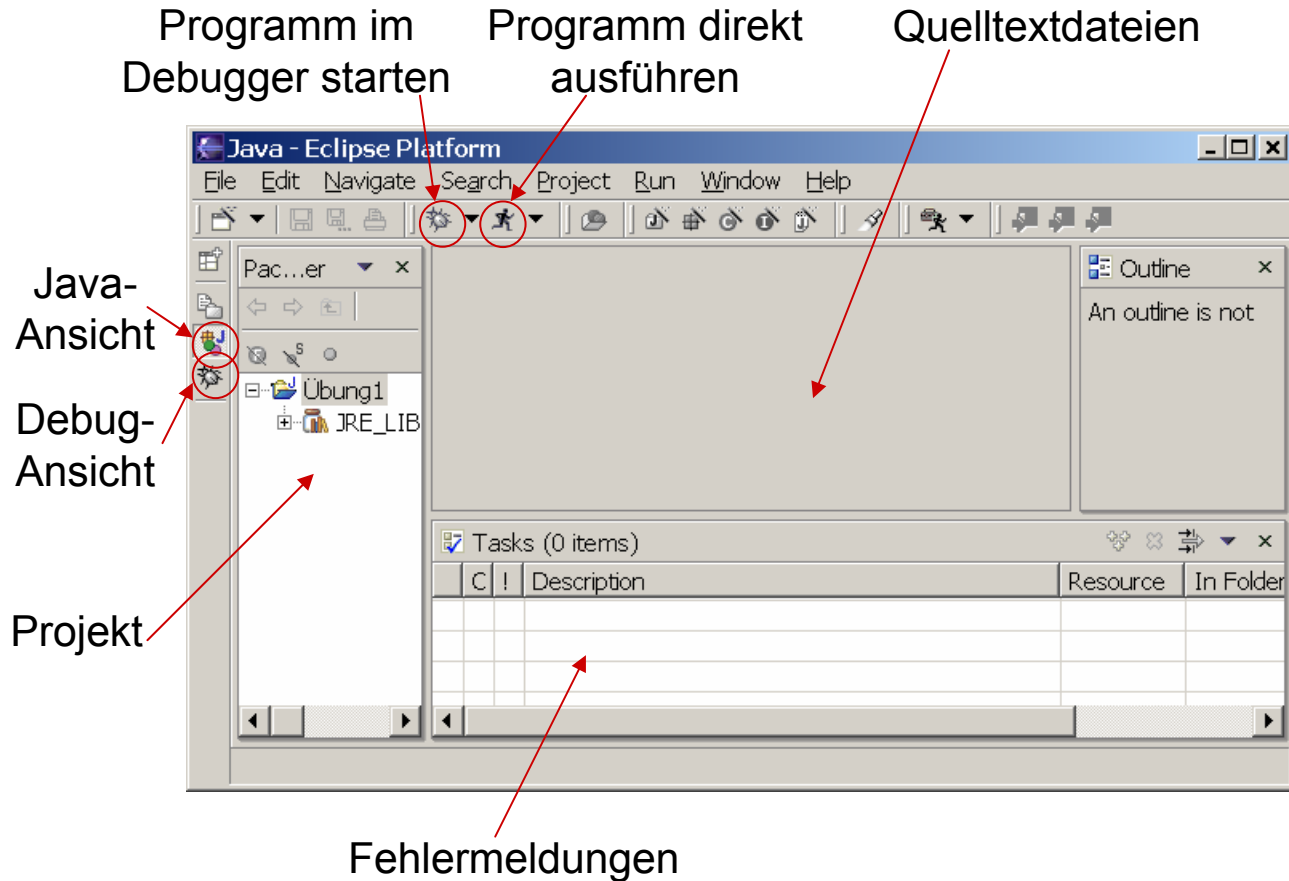
- „Next >“ wählen.

- Projektnamen und eventuell das Verzeichnis des Projektes auswählen:



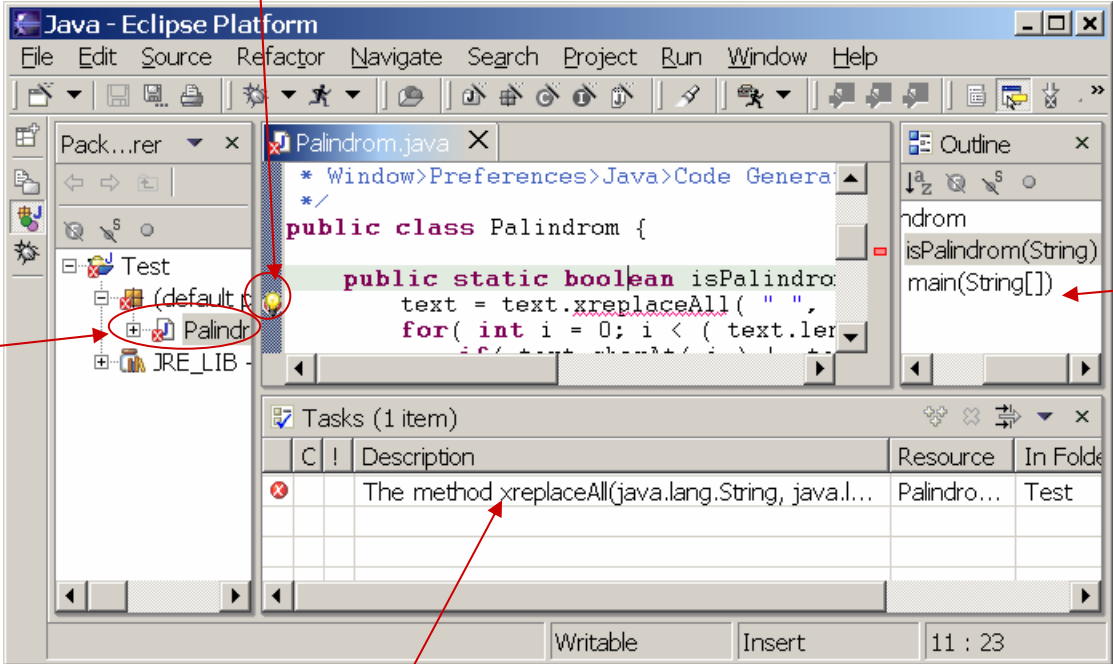
- „Finish“ auswählen.

- Wichtige Elemente:



- Beispielprojekt:

### Fehlerstelle (Syntaxfehler)



Java-Dateien

Methoden der aktuellen Klasse

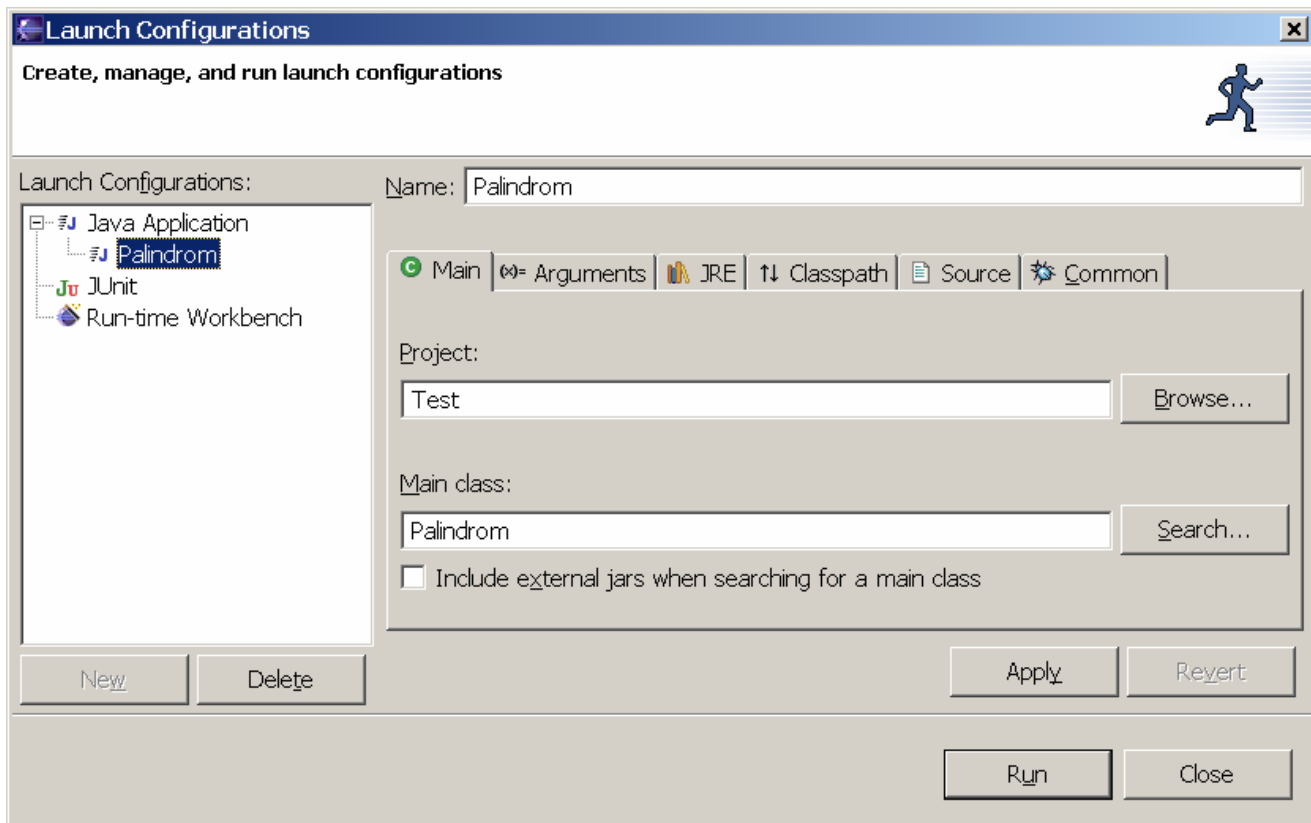
```
public class Palindrom {  
    public static boolean isPalindrom(  
        text = text.xreplaceAll(" ",  
        for( int i = 0; i < ( text.ler  
        if( text.charAt(i) != text
```

C	!	Description	Resource	In Folder
✗		The method xreplaceAll(java.lang.String, java.l...	Palindro...	Test

### Fehlerstelle (Syntaxfehler)

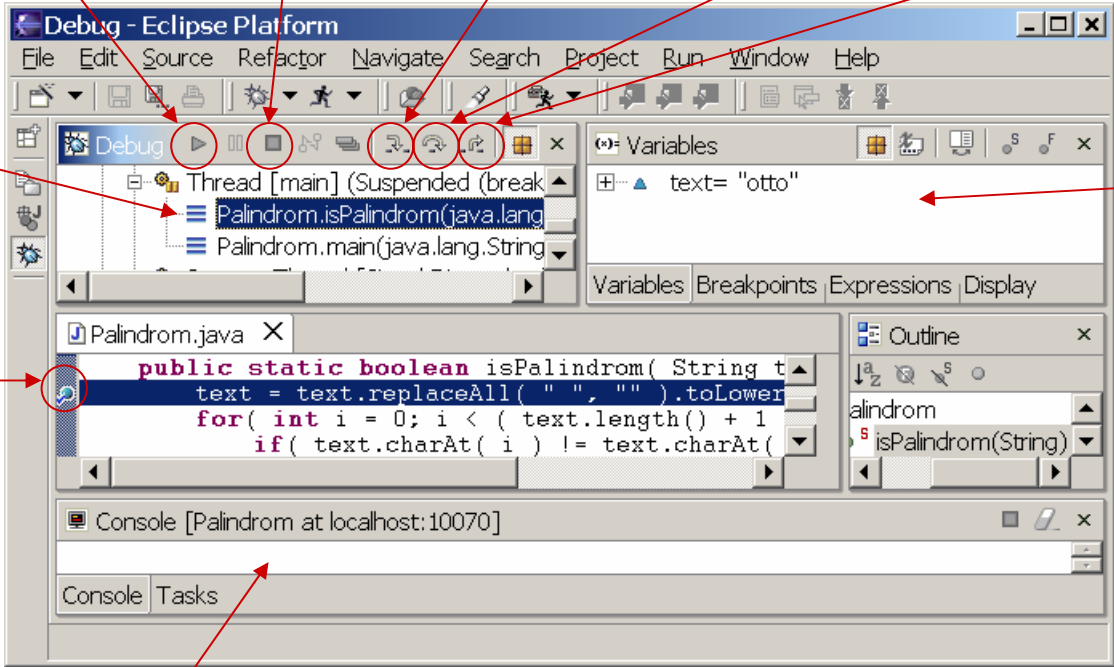
Bei Doppelklick wird direkt in Quelltext gesprungen

- Beim ersten Start oder durch Auswahl des Menüs Run → Run... muss die Klasse ausgewählt werden, deren Start-Methode (public static void main(String argv[])) aufgerufen werden soll:



- Arbeiten mit dem Debugger;

Ausführung fortsetzen    Ausführung beenden    Methode debuggen    Methode ausführen    Methode verlassen



The screenshot shows the Eclipse IDE's Debug console. The top toolbar contains icons for: Resume (play), Stop (square), Step Into (bug with arrow), Step Over (bug with arrow), Step Return (bug with arrow), and Close (X). The Call Stack (Aufruf-hierarchie) shows the current thread and method calls. The Variables window shows the current method's variables, such as 'text = "otto"'. The Source Editor shows the code with a breakpoint (Haltepunkt) set on the first line of the 'isPalindrome' method. The Console window shows the program's output (Ausgaben des Programms). The Outline window shows the project structure.

Aufruf-hierarchie

Haltepunkt (Breakpoint)

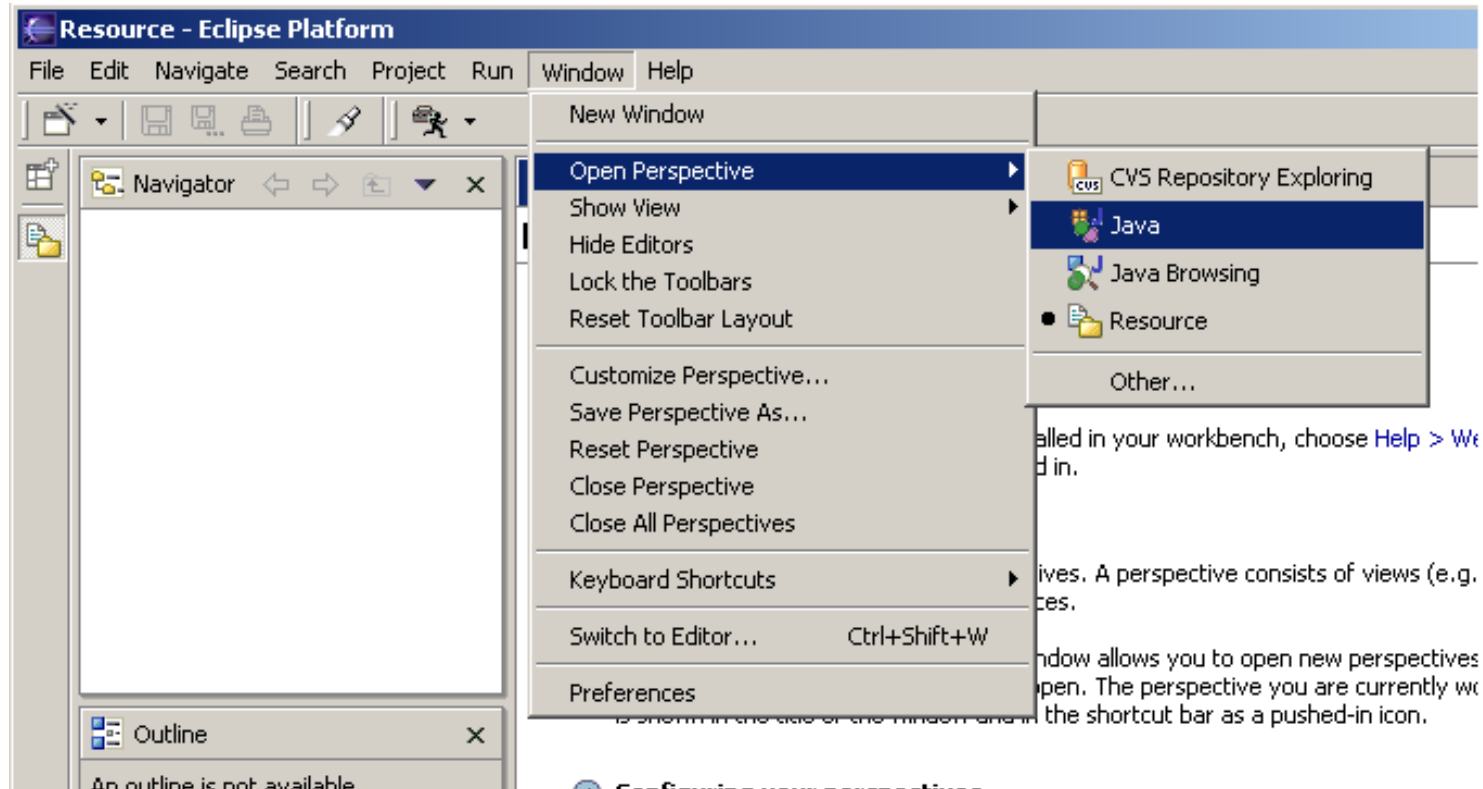
Ausgaben des Programms

Variablen der Methode

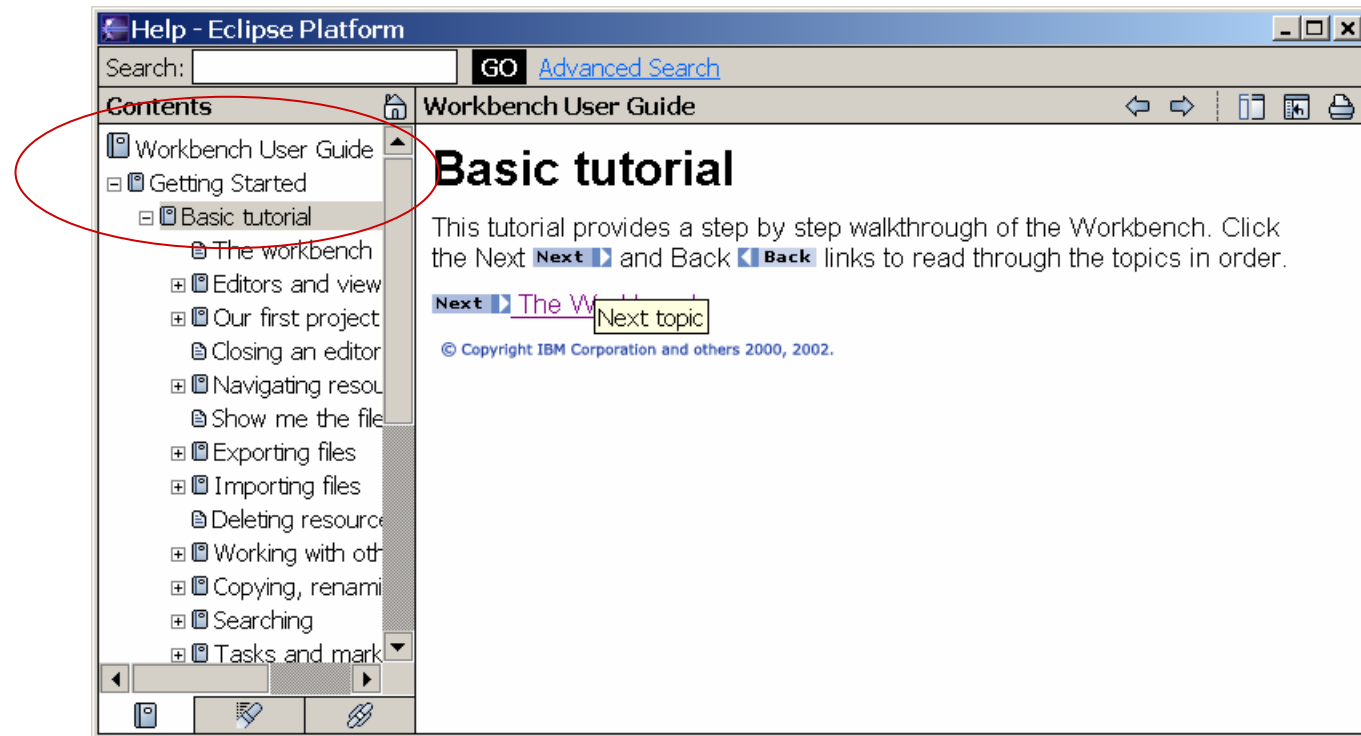
# JAVA

## Eclipse

- Perspektive auswählen:



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



## Grundlegender Aufbau einer Java Klasse (EBNF)

*Klasse* = { *Modifier* } “**class**” Bezeichner “{“ *KlassenRumpf* “}”  
*Modifier* = “**public**” | “**protected**” | “**private**” | ...

- *Bezeichner* ist eine beliebige Folge von Unicodezeichen (Unicode siehe Vorlesung)

MeineKlasse, Smørebrød, Αλφα

- Konvention für Klassennamen nur A-Z, a-z, 0-9 verwenden. Erste Buchstabe jedes Teilworts gross, Rest klein

- *modifier*: Bestimmt die Sichtbarkeit der Klasse nach aussen
  - public**: Klasse kann von jeder anderen Klasse verwendet werden
  - protected**: später
  - private**: Klasse von aussen nicht verwendbar (inklusive Interpreter java),

mind. eine Klasse in einer Datei sollte deswegen public sein.

## Grundlegender Aufbau einer Java Klasse (EBNF)

← nur „sinnvolle“ Modifier erlaubt

*Klasse* = { *Modifier* } **“class”** Bezeichner **“{“** *KlassenRumpf* **“}”**

*Modifier* = **“public”** | **“protected”** | **“private”** | ...

- „Sinnvolle“ *Modifier* für Klasse: obige drei sowie **final**, **abstract**, **static**, keine doppelten erlaubt
- Neuere Compiler Versionen mahnen bestimmte Reihenfolge an
- Beliebige Leerzeichen, Zeilenende oder Tabulatoren zwischen Sprachelementen möglich (Compiler ignoriert diese)
- Zur Formatierung Quelltext für Menschen verwenden

```
public class Konto {  
    public static void main(String[] args) {  
        double guthaben = 0.0;  
        guthaben = guthaben + 100.0;  
        System.out.println ("Guthaben    = " + guthaben);  
    }  
}
```

## Grundlegender Aufbau einer Java Klasse (EBNF)

$KlassenRumpf = \{ \text{AttributDeklarationen} \mid \text{KonstruktorDeklarationen} \mid \text{MethodenDeklarationen} \mid \text{KlassenDeklarationen} \}$   
 $MethodenDeklarationen = \{ \text{Modifier} \} \text{ Typ Bezeichner}$   
 $\quad \text{"(" ParameterDeklaration ")"} \text{"{" MethodenRumpf "}"}$   
 $\text{Typ} = \text{Bezeichner} \mid \text{"void"} \mid \text{elementarerDatentyp} \mid \dots$

- „Sinnvolle“ Modifier für *MethodenDeklaration* wie bei Klassen plus **synchronized, native**
- Konvention Methodennamen: Verb verwenden, erste Buchstabe klein, erste Buchstabe Teilwörter groß, nur A-Z, a-z, 0-9.

```
public class Konto {  
    public static void macheWas() {  
    }  
}
```

## Grundlegender Aufbau einer Java Klasse (EBNF)

*MethodenDeklarationen* = { *Modifier* } *Typ* *Bezeichner*

“(“ *ParameterDeklaration* “)” “{“ *MethodenRumpf* “}”

*ParameterDeklaration* = *Typ* { “[“ “]” } *Bezeichner* { “[“ “]” }  
[ “,” *ParameterDeklaration* ]

- Parameter werden bei Aufruf einer Methode als Platzhalter für Werte verwendet (später)
- Hier: main-Methode muss String [ ] Parameter besitzen, um vom Interpreter aufgerufen werden zu können
- Parameter args enthält Folge Kommandozeilen Parameter
- javac Konto wert1 4711
- args[0] enthält Zeichenkette „wert1“, args[1] enthält Zeichenkette „4711“

```
public class Konto {  
    public static void main(String[] args) {  
    }  
}
```

## Grundlegender Aufbau einer Java Klasse (EBNF)

*MethodenDeklarationen* = { *Modifier* } *Typ* *Bezeichner*

“(“ *ParameterDeklaration* “)” “{“ *MethodenRumpf* “}”

*MethodenRumpf* = { *VariablenDeklaration* | *Anweisung* | ... }

*VariablenDeklaration* = *Typ* *Bezeichner* [ “=“ *Literal* ] “;”

*Anweisung* = ( *Zuweisung* | *Kontrollanweisung* | *Methodenaufruf* | ... ) “;”

- (lokale) Variablen sind Platzhalter für Werte (double guthaben = 17.1;)
- Eindeutige Variablennamen pro Methode
- Literale sind konstante Werte (5, “zeichenkette“, 17.1, true, false, ...)
- Anweisungen manipulieren Werte
  - ◆ Neue Werte aus alten berechnen (guthaben + 100.0)
  - ◆ Variablen neuen Werte zuweisen (guthaben = guthaben + 100.0)
  - ◆ Programmfluss steuern (Verzweigungen, Wiederholungen)

Literale sind Werte eines primitiven Datentyps, eines Strings oder **null**.

Literale dienen dazu, in einem Javaprogramm Konstanten aufzuschreiben. (Sie sind nicht zu verwechseln mit Konstantendeklarationen).

Java kennt folgende Typen von Literalen:

<b>Ganze Zahlen:</b>	<code>0; 123; 0xDadaCafe</code>
<b>Float-Zahlen:</b>	<code>3.2f; -3e-22f</code>
<b>Double-Zahlen:</b>	<code>3.2; -3e-22</code>
<b>Logische Literale:</b>	<code>true; false</code>
<b>Referenz-Literale:</b>	<code>null; this; super</code>
<b>Character-Literale:</b>	<code>'A',</code>
<b>Zeichenketten-Literale:</b>	<code>""; "\\\""; "Hello World"</code>

- ☺ **Fließkomma-Literal: Gleitkommawert**
- |                       |                      |
|-----------------------|----------------------|
| <b>Float-Zahlen:</b>  | <b>3.2f; -3e-22f</b> |
| <b>Double-Zahlen:</b> | <b>3.2; -3e-22</b>   |

Konstanter Fließkomma-Wert, der nach der IEEE-Norm 754 abgelegt wird.

- Es werden einfache Genauigkeit mit 32 Bit (**float**) und doppelte Genauigkeit mit 64 Bit (**double**) unterstützt.
- Verschiedene Zahlenformate sind erlaubt, Beispiele:
  - **0.31415**: entspricht der Zahl **0,31415**
  - **3e-2**: entspricht der Zahl  $3 \cdot 10^{-2} = 0,03$
  - **-3e2**: entspricht der Zahl  $-3 \cdot 10^2 = -300$
- Standardtyp ist **double** => Ohne extra Suffix handelt es sich immer um doppelte Genauigkeit. Um die Genauigkeit zu steuern, kann ein sog. Suffix angehängt werden:
  - **f, F**: Einfache Genauigkeit (**float**), Beispiel: **3e2f**
  - **d, D**: Doppelte Genauigkeit (**double**, redundante Angabe), Beispiel: **3e2d, 3e2**

### Dokumentationskommentare

- Für Klassen, Interfaces, Methoden und deren Parameter.
- Die Dokumentationskommentare werden durch `/**` eingeleitet:

```
/**  
 * The example class provides...  
 */  
public class Example {  
  
    /**  
     * comment  
     */  
    public void test (int times) {  
    }  
}
```

- Die Kommentare sollten zusätzliche Tags erhalten, die durch das Programm **javadoc** ausgewertet werden können.
- Damit kann beispielsweise eine komplette HTML-Dokumentation aller öffentlichen Methoden generiert werden.
- Die Java-API-Dokumentation wird so erstellt.

- Dokumentationstags:

Klassen und Interfaces		Konstruktoren und Methoden	
<code>@author</code>	Name	<code>@param</code>	Name description
<code>@version</code>	1.0, 20 Sep. 2001	<code>@return</code>	description
<code>@since</code>	JDK 1.0	<code>@exception</code>	classname descr.
<code>@see</code>	classname	<code>@see</code>	classname
<code>@deprecated</code>		<code>@deprecated</code>	

- Beispiel:

```
/**  
 * Sorts an array of integer values.  
 * @param values Unsorted array of values.  
 * @return Sorted array of values.  
 * @exception NullPointerException Thrown, if values is null.  
 * @see java.util.Arrays.sort (int[] a)  
 */  
int[] sort (int[] values) throws NullPointerException
```

# Javadoc erzeugen

- Aufruf unter: Project -> Generate Javadoc
- Bei ersten mal muss javadoc.exe eventuell angegeben werden
- Ort: verwendetes bin Verzeichnis des JDKs

