



Einführung

in

Eclipse

The Eclipse Project logo features a stylized globe in the background with horizontal lines passing through it. Overlaid on the globe is the word 'eclipse' in a large, white, lowercase sans-serif font. Below 'eclipse' is the version number '2.1' in a smaller, blue font, and at the bottom is the text 'THE ECLIPSE PROJECT' in a white, uppercase sans-serif font.

eclipse
2.1
THE ECLIPSE PROJECT

Hilke Fasse
Wintersemester 2006/2007

1. Eclipse	3
1.1. Installation unter Windows und Linux.....	3
1.2. Eclipse in den Rechnerpools	3
1.3. Tutorials	3
2. Javaprojekte erstellen	4
2.1. Projekt Hello World	4
2.2. Projekt Konto	4
2.3. Neue Methode hinzufügen:.....	5
2.4. Codevervollständigung (content assist)	6
2.5. Neue Klasse hinzufügen	7
2.6. Refactoring.....	8
2.6.1. Java Elemente umbenennen.....	8
2.6.2. Klassen in andere Packages verschieben.....	8
2.6.3. Neue Methode extrahieren	8
2.7. Programme starten.....	8
2.8. Java Projekt mit JUnit:.....	9
3. UML-Diagramme	11
3.1. Klassendiagramm erzeugen.....	11
3.1.1. Neues Klassendiagramm erzeugen	11
3.1.2. Klassendiagramm für bestehende Pakete.....	12
3.2. Andere UML-Diagramme.....	13
4. Projekte anderen zur Verfügung stellen:	14
4.1. Unter Windows:	14
4.1.1. Exportieren eines Projekts:.....	14
4.1.2. Importieren des Projekts:.....	14
4.2. Unter Linux:	14
4.2.1. Exportieren eines Projekts:.....	14
4.2.2. Importieren des Projekts:.....	14
5. Anmerkungen oder Probleme	14
5.1. Shortcuts.....	14
5.2. Omondo geht nicht.....	14

1. Eclipse

1.1. Installation unter Windows und Linux

Die Beschreibung zur Installation von Eclipse steht auf der SWT-Webseite unter <http://www.technik-emden.de/~swtlab/eclipse.html>

1.2. Eclipse in den Rechnerpools

In den Rechnerräumen E213, E215, E216 und E217 können Sie unter Linux Eclipse benutzen.

Starten Sie Eclipse durch Eingabe von `eclipse` in einer Konsole.

Beim ersten Ausführen von Eclipse müssen Sie noch die Javaversion angeben, die Sie benutzen möchten.

Im Menü `Window - Preferences` wählen

`Java - Installed JREs` auswählen

Auf `Add` klicken

Für Java Version 1.3.1 müssen Sie unter `JRE home directory`

```
/usr/opt/j2sdk1.3.1
```

eintragen und unter `JRE name` z.B. den Namen `JRE_1.3.1` eintragen.

Für Java Version 1.5 müssen Sie unter `JRE home directory`

```
/usr/opt/jdk1.5.0_05
```

eintragen und unter `JRE name` z.B. den Namen `Java5` eintragen.

Mit `<OK>` bestätigen.

Um die Javaversion 5 zu verwenden ist es außerdem nötig, unter

`Window->Preferences->Java->Compiler` dann `Compiler Compliance Level 5.0` auszuwählen.

Sie können auch beide Javaversionen nacheinander eintragen und eine davon dann als Standard auswählen.

1.3. Tutorials

Sie sollten in der Eclipse-Hilfe unbedingt die beiden Tutorien "Workbench User Guide" und "Java Development User Guide" durcharbeiten.

Falls die Hilfe in Eclipse nicht funktionieren sollte (unter Linux gibt es da evtl. Probleme) muss der Pfad des Browsers eingetragen werden.

Dafür im Menü `Window->Preferences->Help` wählen.

Unten bei "Custom Browser Command" entweder den korrekten Browserpfad angeben oder über "Browse" einen auswählen.

Der Eintrag sieht dann z.B. so aus `"usr/local/mozilla/mozilla" %1`

Man kann auch die Online-Dokumentation auf <http://eclipse.org/documentation/main.html> mit einem Browser verwenden oder sich die älteren Versionen von "Workbench User Guide" und "Java Development User Guide" als PDF-Dateien herunterladen.

2. Javaprojekte erstellen

2.1. Projekt Hello World

Im Menü `File -> New -> Project` wählen. `JavaProject` auswählen und `<Next>` klicken. Dem Projekt den Namen `Einfuehrung` geben und mit `<Finish>` bestätigen.

Eine Klasse erstellt man über das Menü `File -> New -> Class` oder über das nebenstehende Symbol in der Toolbar.



Im Dialogfenster muss das Package und der Klassenname angegeben werden. Hier geben Sie bei `Package` `"test"` und bei `Name` `"HelloWorld"` ein. Unten in der Checkbox sollte `public static void main(String[] args)` markiert werden. Dadurch wird automatisch eine Main-Methode erstellt. Nach dem Bestätigen mit `<Finish>` wird die Datei `HelloWorld.java` erstellt und im Editor angezeigt.

Jetzt das bekannte `System.out.println("Hello World");` in die Main-Methode schreiben und die Datei abspeichern. Beim Speichern wird die Klasse automatisch kompiliert und eventuell vorhandene Fehler werden angezeigt.

Zum Ausführen des Programms wählt man im Menü `Run -> Run As -> Java Application`. Die Ausgabe des Programms (`Hello World`) wird im Konsolenfenster angezeigt.

2.2. Projekt Konto

Beispiel aus Dietmar Abts Grundkurs Java.

Erstellen Sie die beiden Dateien für die Klassen `Konto` und `KontoTest` im Package `test`.

Um sich Schreibarbeit zu sparen kann man die `Get-` und `Setmethoden` der Attribute automatisch erstellen lassen. Dafür deklariert man im Quelltext das Attribut und wählt im `Package-Explorer` im Kontextmenu des Attributs `Source->Generate Getters and Setters`.

```
class Konto {
    int kontonummer;
    double saldo;

    Konto () {}
    Konto (int kontonummer) {
        this.kontonummer = kontonummer;
    }
    Konto (int kontonummer, double saldo) {
        this.kontonummer = kontonummer;
        this.saldo = saldo;
    }
    int getKontonummer(){
        return kontonummer;
    }
    void setKontonummer (int nr){
        kontonummer = nr;
    }
    double getSaldo() {
        return saldo;
    }
    void setSaldo(double betrag) {
        saldo = betrag;
    }
}
```

```
public class KontoTest {
    public static void main(String[] args) {
        Konto meinKonto = new Konto();
        meinKonto.setKontonummer(4711);
        meinKonto.setSaldo(500.);
        System.out.println(meinKonto.getSaldo());
    }
}
```

Im Package-Explorer kann man den Inhalt der Pakete sehen: die Javodateien, die enthaltenen Klassen, Methoden und Attribute. Durch ein Doppelklick auf eines dieser Elemente wird die zugehörige Javodatei im Editor geöffnet und das angewählte Element angezeigt. Falls der Package-Explorer nicht angezeigt wird im Menü Window -> Show View -> Package Explorer wählen.

2.3. Neue Methode hinzufügen:

Am Ende der Datei Konto.java den Methodenkopf `void einzahlen (double betrag)` hinzufügen.

Die Methode wird sofort im Package-Explorer angezeigt. Und gleichzeitig erscheint in der rechten oberen Ecke des Editors ein kleines rotes Quadrat, was signalisiert, dass ein Fehler in der Datei ist. Wird jetzt die Datei gespeichert, dann werden im Task Fenster, Fehlermeldungen angezeigt, hier `{}` fehlen. Nach Einfügen der Klammern und erneutem Abspeichern verschwinden die Fehlermeldungen.

Vervollständigen der Klasse mit

```
void einzahlen (double betrag) {
    saldo += betrag;
}

void auszahlen (double betrag) {
    saldo -= betrag;
}
```

und Datei speichern.

2.4. Codevervollständigung (content assist)

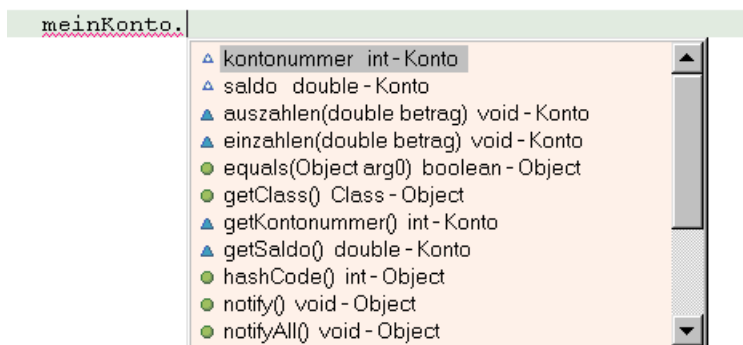
Eclipse kann automatisch den Javacode vervollständigen indem man einen Teil des Codes eingibt und dann <Ctrl+Space> drückt. Der Code wird ergänzt oder wenn er nicht eindeutig zu ergänzen ist, werden Vorschläge dazu gemacht.

Beispiel:

Geben Sie in der Datei KontoTest.java `mei` ein und drücken Sie <Ctrl+Space>. Es wird automatisch `meinKonto` eingefügt.

Löschen Sie das `meinKonto` wieder und geben Sie nur `me` ein. Nach der Eingabe von <Ctrl+Space> erscheint eine Liste zur Auswahl.

Diese Liste erscheint auch nach der Eingabe des Punktes hinter `meinKonto.`



Durch das Eingeben weiterer Buchstaben wird diese Liste eingeschränkt. `e` drücken und dann aus der Liste `einzahlen (double Betrag) ...` wählen und mit Enter bestätigen. Die Methode wird eingefügt.

Jetzt vervollständigen zu

```
meinKonto.setSaldo(500.);
meinKonto.einzahlen(10000.);
System.out.println(meinKonto.getSaldo());
```

und die Datei speichern.

Es ist ebenfalls möglich ganze Kontrollstrukturen wie Schleifen und if-Anweisungen einfügen zu lassen. Nachzulesen im Kapitel Using source code templates im Java Development User Guide.

2.5. Neue Klasse hinzufügen

Es soll eine Klasse Girokonto erzeugt werden, die von Konto abgeleitet wird.

File->New->Class

Name: Girokonto

Superclass: Konto

Fügen Sie folgenden Code hinzu und nutzen Sie dabei auch die Codevervollständigung.

```
class Girokonto extends Konto {
    double limit;

    Girokonto(int kontonummer, double saldo, double limit){
        super (kontonummer, saldo);
        this.limit = limit;
    }

    void setLimit (double limit) {
        this.limit = limit;
    }

    void zahleAus (double betrag) {
        if (betrag <= saldo + limit)
            saldo -= betrag;
    }
}
```

Erweitern Sie zusätzlich noch die Datei KontoTest.java um folgenden Code:

```
Girokonto gk = new Girokonto (1234, 500., 2000.);
System.out.println(gk.getSaldo());
gk.zahleAus(30.);
System.out.println(gk.getSaldo());
gk.zahleAus(3000.);
System.out.println(gk.getSaldo());
```

2.6. Refactoring

2.6.1. Java Elemente umbenennen

Das Element, was umbenannt werden soll im Package Explorer, anwählen und im Menü oder Kontextmenü Refactor > Rename wählen. Sämtliche Vorkommnisse dieses Elements werden umbenannt. Mit Preview kann man sich die Details ansehen, die geändert würden und gegebenenfalls bestimmte Änderungen verwerfen.

Rückgängig machen über das Menü Refactor > Undo ...

Beispiel: Umbenennen der Methoden einzahlen und auszahlen in zahleEin bzw. zahleAus.

2.6.2. Klassen in andere Packages verschieben

Momentan befinden sich im Package test die vier Dateien HelloWorld.java, Konto.java, KontoTest.java und Girokonto.java. Die drei Kontodateien sollen nun in ein eigenes Package mit Namen bank. Dieses wird über das Menü File -> New -> Package erstellt.

Dann die drei Klassen im Package-Explorer markieren und in das neue Package ziehen. Unter Linux funktioniert das mit dem Drag&Drop evtl. nicht. Dann müssen die Klassen markiert werden und über das Kontextmenü Refactor -> Move verschoben werden.

2.6.3. Neue Methode extrahieren

Herausziehen eines Codeabschnitts aus einer Methode, um eine eigene Methode zu bilden.

Zweckfreies Beispiel:

Markieren des Codeabschnitts

```
System.out.println(gk.getSaldo());
gk.zahleAus(30.);
System.out.println(gk.getSaldo());
gk.zahleAus(3000.);
System.out.println(gk.getSaldo());
```

Kontextmenü "Refactor > Extract Method"

Methodenname auszahlenGirokonto eingeben und <OK> klicken

2.7. Programme starten

Zum Starten eines Javaprogramms im Menü "Run -> Run As -> Java Application" anwählen.

Wählt man im Menü "Run" nur "Run" können im aufklappenden Dialogfenster z.B. unter

(x)= Arguments die Argumente für den Start angegeben werden.

2.8. Java Projekt mit JUnit:

Informationen zu JUnit finden sich unter

<http://www.frankwestphal.de/UnitTestingmitJUnit.html>

Um JUnit verwenden zu können, muss die Datei `junit.jar` zum Projekt hinzugefügt werden.

Im Kontextmenü des Projektes `Properties` auswählen. Dort unter `Java Build Path` -> `Libraries` -> `Add External JARs` wählen und ein geeignetes `junit.jar` auswählen. (`/opt/junit3.8.1`)

Mit OK bestätigen und JUnit wird zum Projekt hinzugefügt.

Um die Methoden der Klasse `Girokonto` zu testen, muss eine Testklasse `GirokontoTest.java` erstellt werden.

Der Rumpf einer JUnit-Testklasse sieht folgendermaßen aus:

```
package bank;

import junit.framework.*;

public class GirokontoTest extends TestCase {

    public GirokontoTest(String name) {
        super(name);
    }

    public static void main(String[] args) {
        junit.textui.TestRunner.run(GirokontoTest.class);
    }
}
```

Die Erstellung der Klasse und die Tipparbeit kann man sich mit Eclipse auch sparen, indem man sich die Testklasse automatisch generieren lässt. Dafür muss die Klasse `Girokonto` im Package-Explorer ausgewählt und im Kontextmenü `New` -> `Other` angeklickt werden. Im Dialogfenster wählt man `Java` -> `JUnit` und `TestCase` und bestätigt das mit `<Next>`. Die Einstellungen können übernommen werden, nur unten sollten die beiden Checkboxes "`public static void main ...`" und "`Add testRunner statement ...`" ausgewählt sein.

Die Bestätigung von `<Finish>` erzeugt den Rumpf der Testklasse `GirokontoTest.java`.

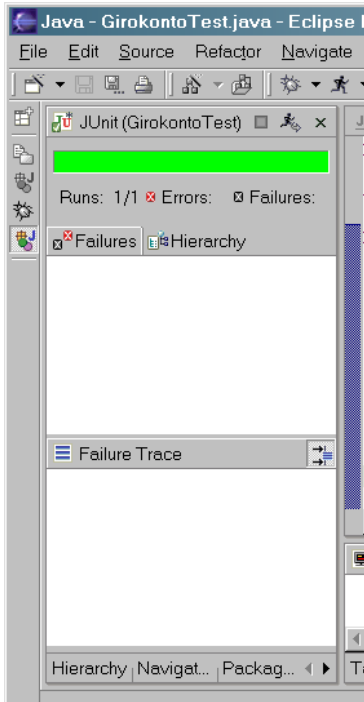
Als nächstes müssen die Testfälle erstellt werden. Beispielsweise soll getestet werden, ob das Limit korrekt gesetzt wird. Dafür wird der Testfall `testSetLimit` erzeugt. Ein Testfall hat immer die Methodensignatur `public void test...()`, da JUnit daran erkennt, dass es sich um einen Testfall handelt.

In der Methode wird als erstes ein Objekt des Typs `Girokonto` erzeugt und dann das Limit auf 2000 gesetzt. Jetzt wird geprüft, ob das Limit tatsächlich 2000 beträgt. Dafür dient die Methode `assertEquals`. Sie vergleicht einen Sollwert mit dem zu holenden Istwert (`gk.limit`). Da beides Doublewerte sind, muss noch ein Schwellwert (hier 0.01) angegeben werden.

Der Testfall sieht dann folgendermaßen aus.

```
public void testSetLimit() {
    Girokonto gk = new Girokonto(22, 1000., 5000.);
    gk.setLimit(2000.);
    assertEquals(2000., gk.limit, 0.01);
}
```

Über das Menü Run -> Run As -> JUnitTest wird der Test ausgeführt und es sollte sich nachfolgende Anzeige einstellen.



Der grüne Balken zeigt an, dass der Test fehlerfrei durchgelaufen ist.

Leider wird in den Rechnerpools kein roter bzw. grüner Balken dargestellt. Aber an der Anzeige Errors: 0 Failures: 0 kann man ebenfalls erkennen, dass der Test durchgelaufen ist.

Als nächstes wird eine weitere Testmethode eingefügt, um die Auszahlung zu testen. Der Kontostand wird mit 1000 vorgegeben, dann 200 ausgezahlt und geprüft, ob das Saldo 700 beträgt.

```
public void testAuszahlung() {
    Girokonto gk = new Girokonto(2222, 1000., 5000.);
    gk.zahleAus(200.);
    assertEquals(700., gk.getSaldo(), 0.01);
}
```

700 ist natürlich ein falscher Wert und JUnit quittiert das mit einem roten Balken bzw. mit der Anzeige Failures : 1.

Nachdem der Wert auf 800 korrigiert wurde ist der Balken wieder grün.

3. UML-Diagramme

Eine Bemerkung vorweg:

In der Free-version von Omondo ist es nicht möglich die erstellten Diagramme anderen zum Bearbeiten zur Verfügung zu stellen. Ändern kann die Diagramme nur der jeweilige Autor.

Falls es notwendig ist andere Gruppenmitglieder an den Diagrammen arbeiten zu lassen, sollte anstatt des Omondo Plugins besser das Programm Jude zum Erstellen der Diagramme genutzt werden.

3.1. Klassendiagramm erzeugen

3.1.1. Neues Klassendiagramm erzeugen

Soll ein ganz neues Diagramm erzeugt und nicht auf schon bestehende Javaklassen zurück gegriffen werden, muss als erstes ein neues Package erstellt werden.

File -> New -> Package

Als Namen "diagramme" eingeben.

Das Projekt diagramme im Package-Explorer wählen und im Kontextmenü UML -> Class diagram editor anklicken. Ein leeres UML-Diagramm wird erstellt.

Um eine neue Klasse zu erstellen, Rechtsklick auf eine leere Fläche im UML-Diagramm und aus dem Kontextmenü New -> Class auswählen.

Die Klasse soll den Namen `Person` bekommen und keine main-Methode enthalten. Eine Bestätigung mit <Finish> erzeugt eine Klasse im UML-Diagramm und gleichzeitig eine Java-Datei mit Namen `Person.java`.

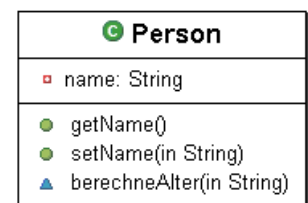
Ein Attribut wird über das Kontextmenü der Klasse hinzugefügt.

Rechtsklick auf die Klasse `Person` New -> Attribute. Hier als Namen "name" angeben und `private` auswählen.

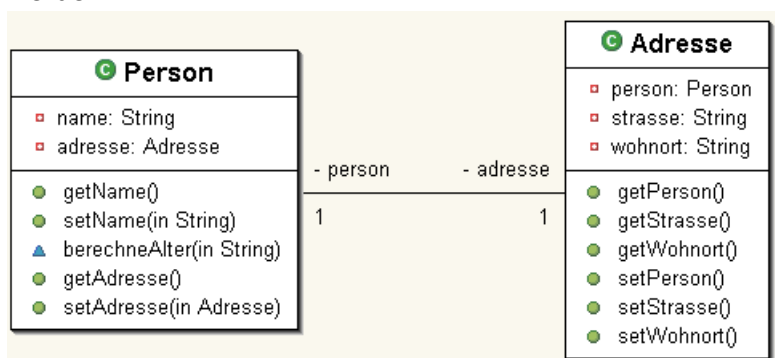
Unten im Dialogfenster kann mit "Use accessors" festgelegt werden, ob für das Attribut get- und set-Methoden erzeugt werden.

Eine Methode erzeugt man wieder über das Kontextmenü der Klasse, New -> Method.

Die Methode soll `berechneAlter` heißen und braucht als Parameter den Geburtstag. Also bei parameters Add wählen, dann als Name `geburtstag` eingeben und als Typ `String` wählen.

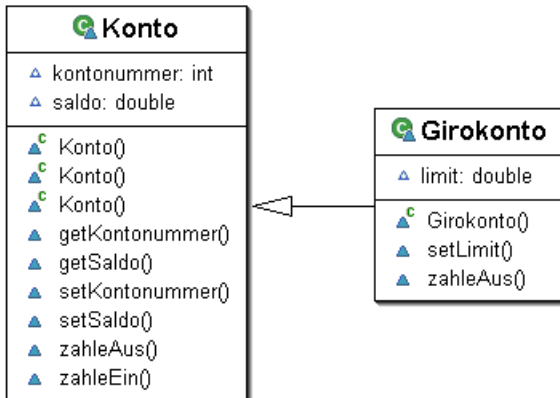


Um eine Assoziation darzustellen muss als erstes eine neue Klasse erzeugt werden, hier die Klasse `Adresse` mit den Attributen `strasse` und `wohnort` sowie den jeweiligen get- und set-Methoden. Dann in der Toolbar Association wählen und nacheinander `Person` und `Adresse` anklicken. Im aufklappendem Dialogfenster können dann z.B. die Multiplizitäten angegeben werden.



3.1.2. Klassendiagramm für bestehende Pakete

Für das `bank` Paket wird ein Klassendiagramm folgendermaßen erzeugt. Im Package-Explorer `bank` anklicken und im Kontextmenü `UML -> Classdiagramm Editor` wählen. Im Dialogfenster die Klassen abwählen, die im Klassendiagramm nicht erscheinen sollen, in diesem Fall die beiden Testklassen. Mit `<OK>` bestätigen und das Klassendiagramm wird angezeigt.



Hinweise:

- Verschieben kann man die Diagrammelemente mit `Strg+Alt+Cursortasten` oder mit der Maus.
- Doppelklick auf beliebiges Element im Package-Explorer oder im Klassendiagramm bewirkt einen Sprung an die entsprechende Stelle im Quelltext.
- Damit Änderungen im Quelltext auch im Diagramm sichtbar werden, muss die Quelldatei erst abgespeichert werden.
- Wenn das Klassendiagramm umformatiert wird, sobald man es neu öffnet, wurde beim Erstellen des Klassendiagramms kein Package angegeben.

Sollen die Testklassen `KontoTest` und `GirokontoTest` auch im Diagramm dargestellt werden, kann man sie per Drag&Drop aus dem Package-Explorer in das Diagramm ziehen oder man wählt im Kontextmenü des Diagramms `Insert -> Class` und gibt im Dialogfenster dann den Klassennamen an.



Klickt man jetzt `Girokontotest` an, sieht man an dem schwarzen Pfeil, dass diese Klasse von einer anderen Klasse abgeleitet ist. Solche bereits bestehenden Vererbungen kann man sich über das Kontextmenü `Inheritance -> Insert supertype` darstellen lassen.

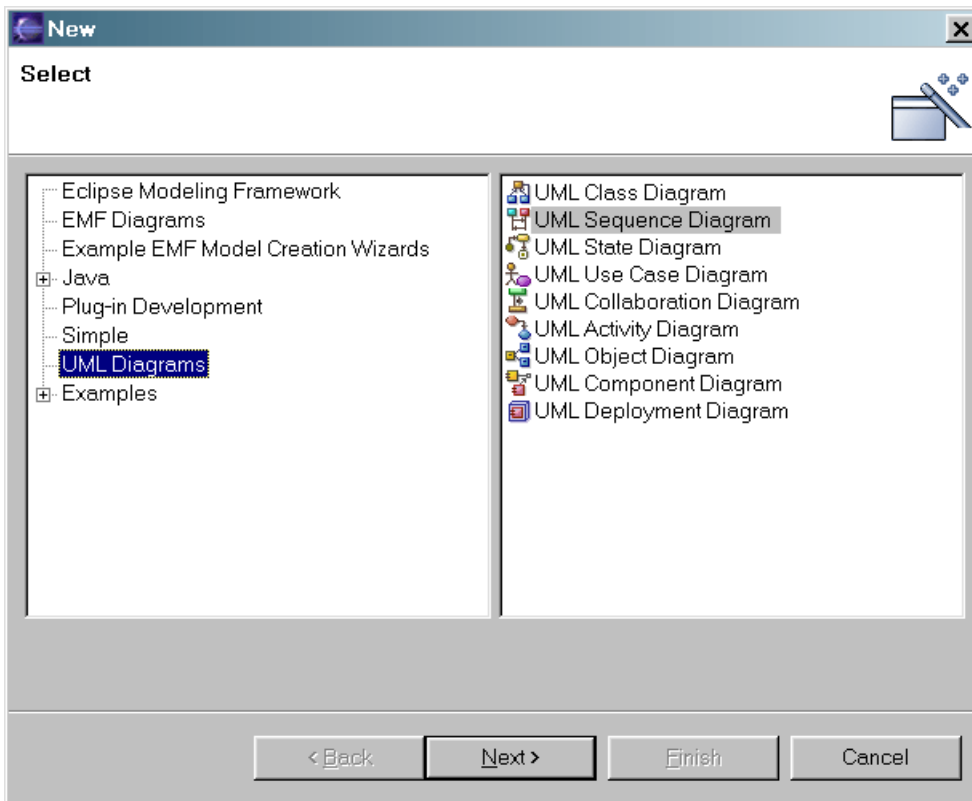
Kontextmenü der Klassen:

- `Collapse View <-> Normalize View`
Ändert die Klassendarstellung mit Methoden und Attributen in eine Darstellung nur mit dem Klassennamen.
- `Maximize` stellt alle Attribute und Methoden der Klasse dar.
- `Minimize` zeigt nur Klassennamen an.

- Möchte man eine Klasse im Diagramm nicht mehr angezeigt bekommen, auf **keinen** Fall `Delete` wählen, das würde die gesamte Klasse auch im Quelltext löschen, sondern `Hide` anklicken.
- Viewselector verändert die Ansichten im Klassendiagramm. z.B. ob Parameter und Rückgabewerte der Methoden angezeigt werden.
- Wenn man das Diagramm als Grafik speichern möchte: Rechtsklick auf leeren Bereich im Diagramm -> Export -> Dateityp wählen

3.2. Andere UML-Diagramme

Die anderen UML-Diagramme, die für Softwaretechnik benötigt werden, erstellt man über das Menü `File -> New -> Others`.



In diesem Dialogfenster kann man dann auswählen, ob man ein

- Anwendungsfalldiagramm (UML Use Case Diagram), ein
 - Sequenzdiagramm (UML Sequence Diagram) oder ein
 - Zustandsdiagramm (UML State Diagram) erstellen möchte.
- Für ein Zustandsdiagramm muss allerdings eine Klasse ausgewählt werden.

4. Projekte anderen zur Verfügung stellen:

4.1. Unter Windows:

4.1.1. Exportieren eines Projekts:

Aus Eclipse per Drag&Drop das Projekt in den Windows-Explorer ziehen. Es wird ein Verzeichnis mit dem Projektnamen angelegt.

4.1.2. Importieren des Projekts:

In Eclipse ein Java-Projekt erstellen. Aus dem Windows-Explorer den Inhalt des Verzeichnisses mit dem Projektnamen, der exportiert wurde, per Drag&Drop in das Eclipse-Projekt ziehen. Eventuell ein `Project` -> `Rebuild all` durchführen.

4.2. Unter Linux:

4.2.1. Exportieren eines Projekts:

Das Projekt markieren und im Kontextmenü `Export` -> `File System` wählen, dann Zielverzeichnis angeben und mit `Finish` bestätigen. Es wird ein Verzeichnis mit dem Projektnamen und den dazu gehörenden Dateien im Zielverzeichnis angelegt.

4.2.2. Importieren des Projekts:

In Eclipse ein Java-Projekt mit dem entsprechenden Projektnamen erstellen. Dann im Kontextmenü des Projektes `Import` -> `File System` auswählen. Das Quellverzeichnis angeben und mit `Select all` und `Finish` wird das Projekt in Eclipse geladen.

5. Anmerkungen oder Probleme

5.1. Shortcuts

Die Shortcuts sind unter `Window` -> `Preferences` -> `Workbench` -> `Key` zu finden. Dort kann man auch eigene einstellen.

`Ctrl+Shift F`: Formatiert einen markierten Codebereich nach den Javarichtlinien (Einrückungen, Klammersetzung u.Ä.). Wenn nichts markiert ist, wird alles formatiert.

`Alt+Shift+R`: Umbenennen eines Elements.

5.2. Omondo geht nicht

Falls Omondo UML beim nach dem Installieren nicht ausgeführt werden kann, scheinbar nicht vorhanden ist, hilft eventuell folgendes Vorgehen:

Eclipse mit dem Parameter `-clean` starten.

Beispiel: `C:\Programme\eclipse\eclipse -clean`