Tutorial 4

Enterprise Generation Language
HelloWorld Tutorial

Enterprise Generation Language, oder EGL, ist eine Programmiersprache der 4.Generation, die ursprünglich in den 1980er Jahren entwickelt wurde. Die Sprache ist keine Neuerung. IBM hat die Programmiersprache EGL (Enterprise Generation Language) als Fortführung von Informix 4GL und IBM Visual Age Generator entwickelt, erweitert um zusätzliche Funktionen.

EGL ist eine prozedurale 4GL- Sprache und zugleich eine Entwicklungsumgebung. EGL ist vielleicht am meisten bekannt als eine Funktion innerhalb des IBM WebSphere Studio Enterprise Developer, die mainframe-orientierte Werkzeuge auf Basis der Eclipse-Entwicklungsumgebung bereitstellt. Mit EGL lassen sich schnell vollwertige Applikationen erstellen, ohne dass sich der Entwickler mit den Details der Middleware oder der Programmierung in Java/J2EE herumschlagen muss. 4GL-Sprachen haben zum Ziel, dieselbe Funktionalität mit weniger Code-Zeilen als etwa C++oder Java zu erreichen. Und nachdem die Entwickler mit EGL die Business-Logik geschrieben haben, können sie daraus direkt Java-oder Cobol-Code erzeugen. Die resultierenden Anwendungen und Komponenten laufen dann auf beliebigen Plattformen.

Die Sprache folgt dem IBM Rational-Ansatz, dass sich Code bereits in sehr frühen Entwicklungsphasen testen lässt, nicht erst im fertigen Produkt. Alle großen IBM Entwicklungszentren arbeiten mit Rational-Tools und -Frameworks.

IBM schuf EGL, um prozedurale Programmierer zu unterstützen, insbesondere solche mit RPG und COBOL-Erfahrung, damit diese die Konzepte und Anwendung der objektorientierten Programmierung leichter verstehen. EGL ist für RPG- und COBOL-Entwickler einfach zu erlernen und innerhalb weniger Wochen mit sehr guter Produktivität einsetzbar.

Mit EGL unterstützt IBM offene Standards und eine offene Architektur.

Aufgabe: *Beschäftigen Sie sich mit diesem Tutorial. Vollziehen Sie alle hier vorgestellten praktischen Schritte (Erstellen des EGL-Programms, Generieren der JAR- und CLASS-Datei etc.) nach.*

Gliederung

Das Tutorial wird mit Hilfe der Entwicklungsumgebung WebSphere Studio Developer for zSeries (WDz) erstellt. Bein WDz handelt es sich um ein Eclipse Plug-In, d.h., es werden Eclipse Funktionen als Basis verwendet.

1. VMWare Player Virtuelle Maschine

In diesem Tutorial wird die komplette Entwicklung innerhalb einer virtuellen Maschine auf der Basis von VM-Ware durchgeführt. So kann den Teilnehmern die umfangreiche Installation der WDz Software abgenommen werden. Zum Ausführen des Images muss lediglich der kostenlose VMWare-Player installiert werden. Die aktuelle Version des VMWare-Players ist unter [VMP] im Internet zu finden. Er steht im Augenblick für Linux und Windows zu Verfügung.

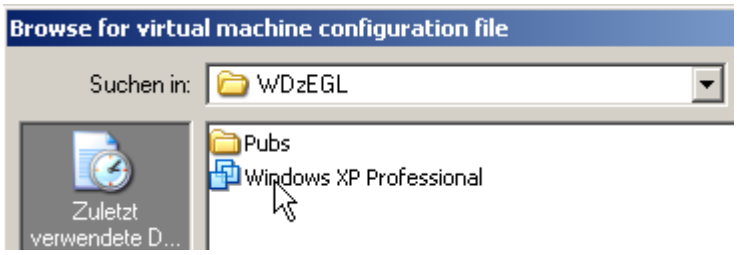
Die DVD von Herrn Erras enthält eine self-extracting compressed file. Das Image enthält die VM mit

- Windows XP Professional mit Service Pack 2 , Lizenz der Universität Leipzig
- WDz (IBM Rational Websphere Developer for zSeries Version 6.016.0) von Herrn Storz,
- DB2 Express Edition 9.0 for Windows,
- SQL Script mit vorgefertigten Daten um DB2 zu füllen,
- Apache EOF,
- XML Code und Java Code von Herrn Erras.

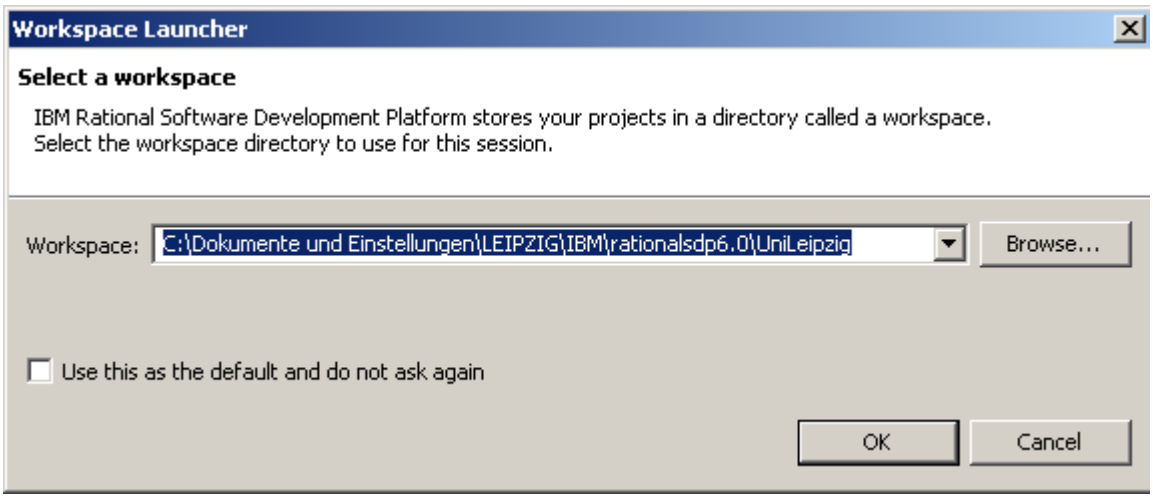
Nach der Installation wird der Inhalt der beigelegten DVD auf die Festplatte kopiert und die virtuelle Maschine durch Doppelklick auf die Datei „Windows XP Professional.vmx“ gestartet.

Insert DVD und extract nach Verzeichnis C;\WdzEGL . Start VM, login als unilp, PW = unilp. 1kr auf Bildschirm, Eigenschaften – Einstellungen 1024 x 768

Start VMWare

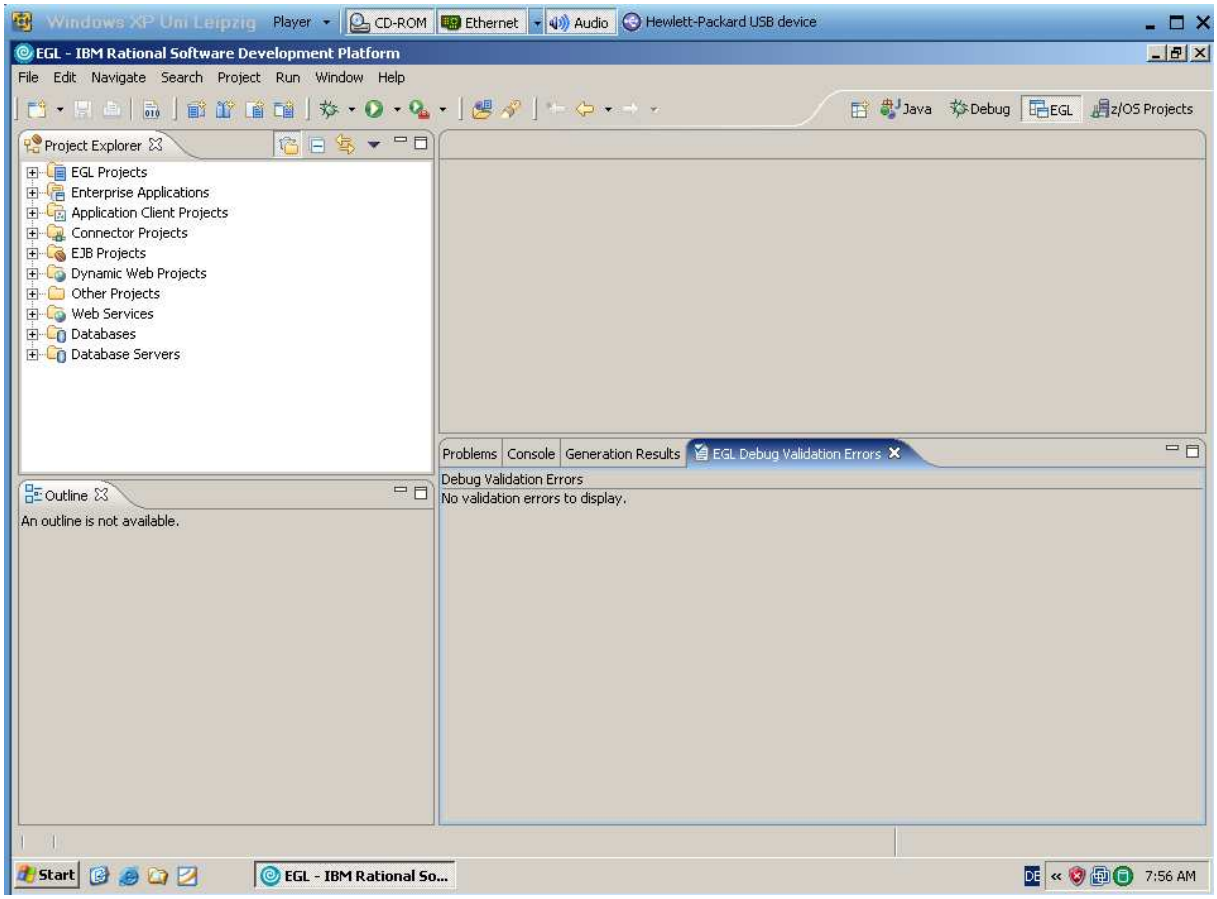


1k auf UNILP, PW = unilp
2k auf WebSphere Developer for zSeries Icon
Es erscheint dieser Icon auf der Bildschirmoberfläche der virtuellen Maschine



no change, ok

warten. Der WDz Entry Bildschirm erscheint.



Epfehlungen: Wenn z.B. die Standard Auflösung des Bildschirms 1024 x 786 ist, empfiehlt es sich, in der virtuellen Maschine mit Start → Einstellungen → Systemsteuerung → Anzeige → Einstellungen ebenfalls 1024 x 786 einzustellen. Hierzu den Bildschirm der virtuellen Maschine mit den Scroll Bars nach unten und nach links bewegen, um Zugriff auf den Start Button zu erhalten.

Weiterhin wird empfohlen, das Fenster mit dem Bildschirm der virtuellen Maschine so einzustellen, dass ein kleiner Teil des Bildschirms der realen Maschine sichtbar bleibt. Mit einem Mausklick kann jetzt zwischen der virtuellen und der realen Maschine gewechselt werden.

In der virtuellen Maschine kann mit Alt Gr + Druck ein Screen Shot des aktuellen Fensters in die Zwischenablage gestellt werden. Mit Strg + Druck kann ein Screen Shot des gesamten Bildschirms der virtuellen Maschine in die Zwischenablage gestellt werden. In beiden Fällen kann die reale Maschine anschließend auf den Inhalt der Zwischenablage zugreifen.

2. EGL 4th Generation Language

In diesem Tutorial wird ein Hello-World-Programm in der EGL Sprache erstellt, welches eine Ausgabe auf der Konsole erzeugt.

Bei der EGL handelt es sich um eine so genannte 4th Generation Language (4GL), also eine Sprache der 4. Generation. Im Gegensatz zu Sprachen der dritten Generation (wie C++, Java oder Cobol) beschreibt der Programmierer in einer solchen Sprache nicht „WIE“ ein bestimmtes Problem zu lösen ist, sondern „WAS“ der Rechner tun soll, um ein Problem zu lösen. Ziel ist es, Programme mit möglichst wenig Code auf leicht verständliche Weise zu erstellen, wobei die konkrete technische Umsetzung in den Hintergrund tritt. Die Entwicklung wird so vereinfacht und beschleunigt und geht mit einer gleichzeitigen Verbesserung der Lesbarkeit des Codes einher, was zusätzlich die Wartbarkeit erleichtert.

Um dies zu erreichen, sind Sprachen der 4. Generation häufig auf ein bestimmtes Themengebiet zugeschnitten. Sie sollten deshalb nur zur Lösung von Problemen aus dem jeweiligen Themengebiet verwendet werden, da sich Probleme anderer Gebiete mit einer spezialisierten Sprache häufig nur schwer oder gar nicht lösen lassen.

Beispiele für solche Sprachen sind:

- **Structured Query Language (SQL):** Abfragesprache für relationale Datenbanken.
- **MATLAB:** Bietet eine Sprache zur Implementierung von mathematischen Algorithmen.
- **Macrosprachen:** Zur Automatisierung von Abläufen in Desktopanwendungen.

EGL wurde von der Firma IBM entwickelt. Die Sprache eignet sich besonders zur gezielten Verarbeitung von Datenbankinhalten und kann z. B. in Kombination mit Java Server Faces (JSF) zum dynamischen Befüllen von Internetseiten verwendet werden. Zur Erstellung von Webauftritten mit der EGL finden sich deshalb auf der Internetseite von IBM einige englischsprachige Beispiele.

Da für die Entwicklung von dynamischen Internetseiten eine Reihe von Frameworks (z. B. Struts oder JSF) und Sprachen (z. B. Visual Basic Script, PHP[1], Java) zur Verfügung stehen, die mittlerweile einen hohen Verbreitungsgrad erreicht haben, sollte das Hauptaugenmerk bei der EGL-Entwicklung auf die Implementierung von Businesslogik gelegt werden.

In vielen Bereichen der Wirtschaft sind große Datenmengen in relationalen Datenbanken abgelegt, die nach komplizierten Regeln verarbeitet werden müssen. Werden diese Verarbeitungsregeln mit einer Sprache der dritten Generation gelöst, enthält der Code, bezogen auf die reinen Anwendungslogik, häufig einen großen Anteil an technischem „Overhead“, d. h. Codezeilen, die nicht direkt mit der eigentlichen Logik in Verbindung stehen, sondern die technischen Rahmenbedingungen realisieren. Als Beispiel sei an dieser Stelle etwa das Aufbauen einer Datenbankverbindung mittels Java Database Connectivity (JDBC) genannt. Bei der Programmierung mit EGL wird bewusst versucht, diesen technischen Anteil im Code auf ein Minimum zu reduzieren und so die reine Anwendungslogik in den Vordergrund zu rücken.

EGL stellt eine Reihe von Funktionen zur Verfügung, um Daten in relationalen Datenbanken mit Hilfe so genannter SQLRecords zu verarbeiten oder abzurufen. Dies kann vielfach sogar ohne die Verwendung von SQL-Statements erfolgen.

Im Gegensatz zu anderen Programmiersprachen wird EGL-Code nicht direkt kompiliert. Stattdessen wird der Code in eine andere Sprache umgewandelt. Die WDz Entwicklungsumgebung stellt derzeitig Java und Cobol als Zielsprachen zur Verfügung. Wir verwenden Java als Zielsprache in dem vorliegenden Tutorial.

3. Abgrenzung von EGL zu Stored Procedures.

Stored Procedures sind Prozedueren, die auf einem Datenbanksystem abgelegt werden und direkt dort zur Ausführung kommen. Zur Definition dieser Prozeduren bieten einige Datenbanksysteme spezielle Sprachen an, die besonders zur Manipulation von Daten ausgelegt sind. Beispiele hierfür sind die proprietären Sprachen PL/SQL (Oracle-Datenbanksysteme) und Transact-SQL (Microsoft SQL-Server).

EGL bietet eine Reihe von Möglichkeiten, um Datenbankabfragen effizient zu formulieren. Im Vergleich zu Stored Procedures arbeiten sie unabhängig vom jeweiligen Datenbanksystem, während Sprachen zur Definition von Stored Procedures häufig an ein bestimmtes Datenbanksystem gebunden sind und nur schwer auf Systeme anderer Hersteller übertragen werden können.

Außerdem können mit der EGL sowohl Client- als auch Server-Anwendungen erstellt werden, da sie nicht direkt an ein Datenbanksystem gebunden ist. Stored Procedures hingegen stehen in der Regel nur serverseitig zur Verfügung.

4. Aufbau eines EGL-Projekts für Java

Grundsätzlich besteht ein EGL-Projekt aus mehreren Hauptbestandteilen:

EGL-Source-Code Dateien: Diese enthalten die eigentliche Logik.

EGL-Build-Descriptor: Enthält eine Beschreibung der Umgebung. Hier wird beispielsweise festgelegt, welche Zielplattform verwendet wird oder welche Parameter zum Ansprechen einer evtl. genutzten Datenbank verwendet werden.

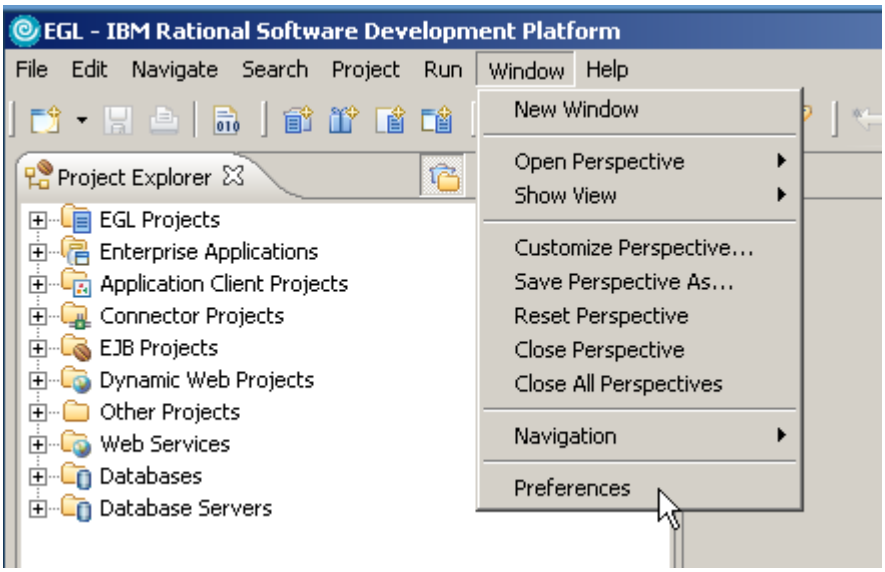
Java-Source-Code Dateien: Werden nach der Generierung automatisch aus dem EGL-Source-Code und der dazugehörigen EGL-Beschreibung erstellt.

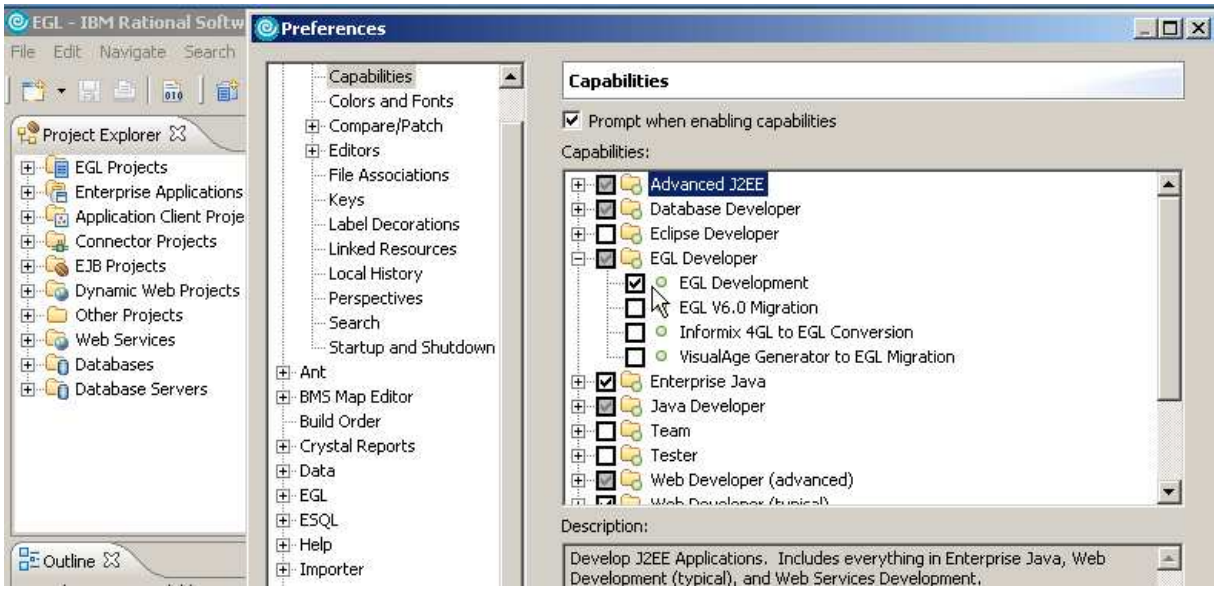
5. Vorbedingungen und Konfiguration

Aufgrund des hohen Funktionsumfangs des WDz hat IBM die Funktionen in einzelne Rollen untergliedert, die sich auf bestimmte Arbeitsgebiete beziehen (z. B. Java- oder EGL-Entwicklung).

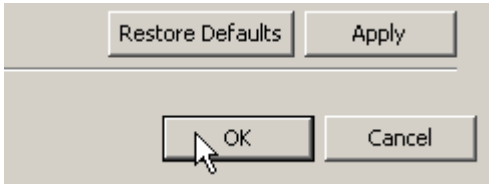
Wird eine solche Rolle aktiviert, werden alle Funktionen, die zur Entwicklung auf einem bestimmten Arbeitsgebiet notwendig sind, in die Entwicklungsumgebung integriert. Der Nutzer wird so nicht von einer Unzahl überflüssiger Funktionen „erschlagen“.

Für die Entwicklung mit der EGL muss deshalb die Rolle „EGL Development“ aktiviert werden. Dies geschieht durch Öffnen der Einstellungen im Menü Window → Preferences, unter Workbench → Capabilites → EGL Developer. Dort ist ein Haken bei EGL Development zu setzen und mit OK zu bestätigen.





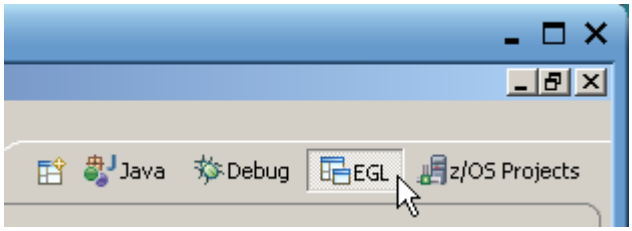
unter Workbench → Capabilites → EGL Developer. Dort ist ein Haken bei EGL Development zu setzen.



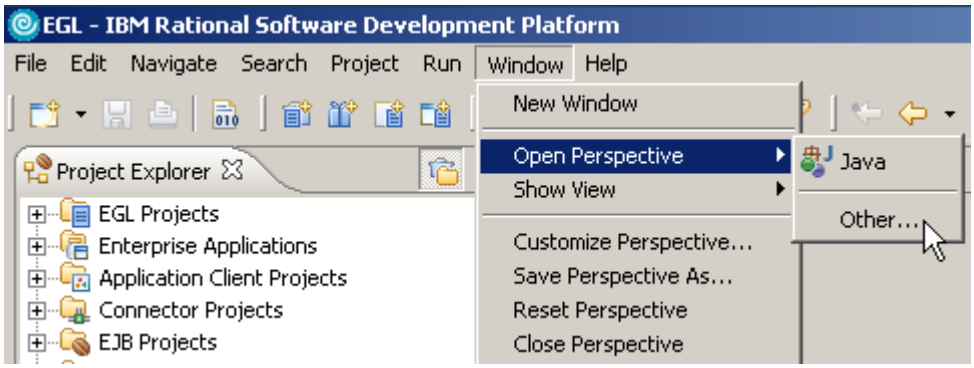
und mit OK zu bestätigen.

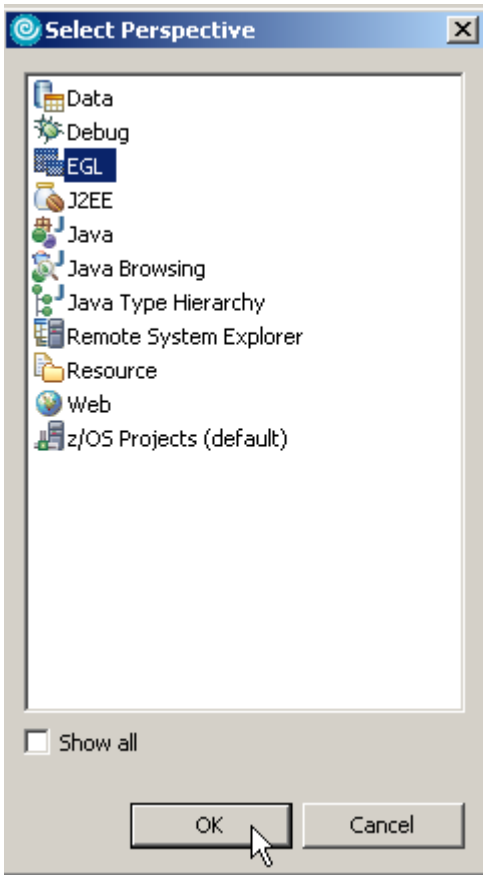
Jetzt wird ein Hello-World-Programm erstellt, welches eine Ausgabe auf der Konsole erzeugt.

Um das Tutorial zu starten, muss zunächst in die EGL-Perspektive gewechselt werden. Dies geschieht (wie bei Eclipse üblich) durch einen Klick auf das Wort EGL in der rechten Menüliste oberhalb des Editorfensters.



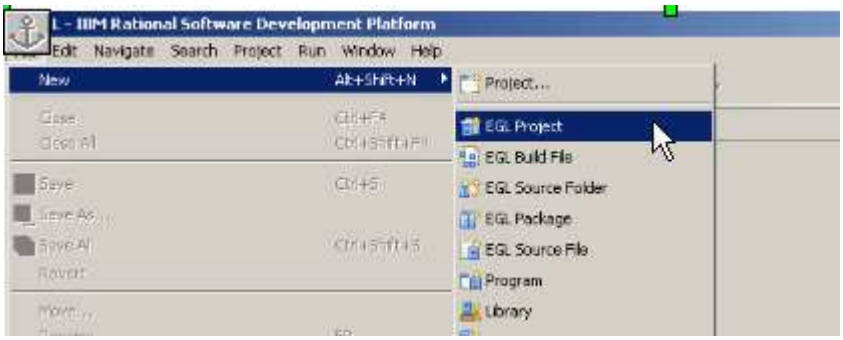
Alternativ: Window → Open Perspective → Other… → EGL.

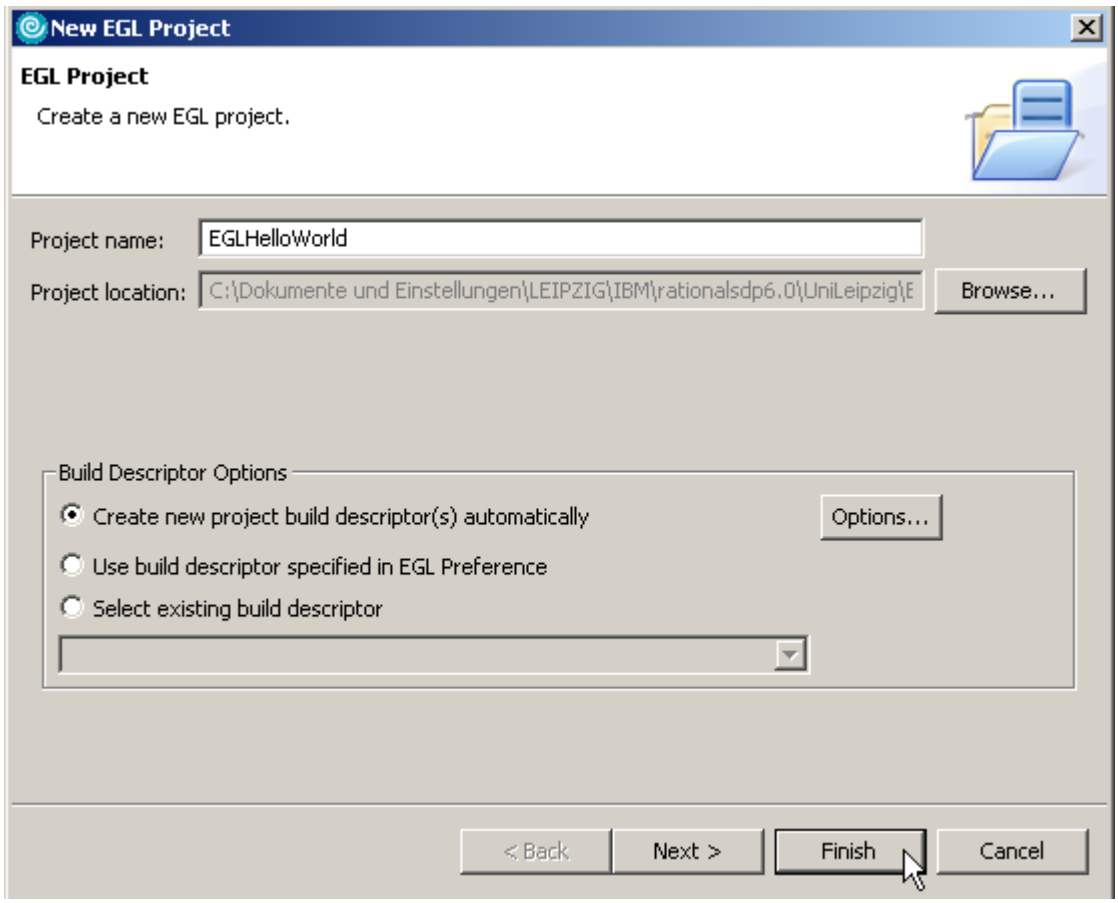




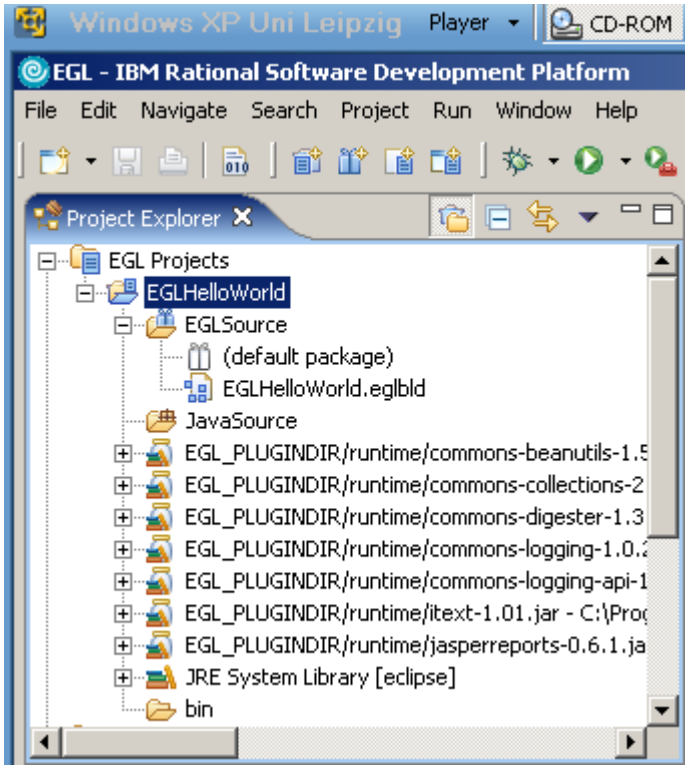
6. Projekt erstellen

Nun kann ein neues EGL-Projekt erstellt werden: File → New → EGL Project.



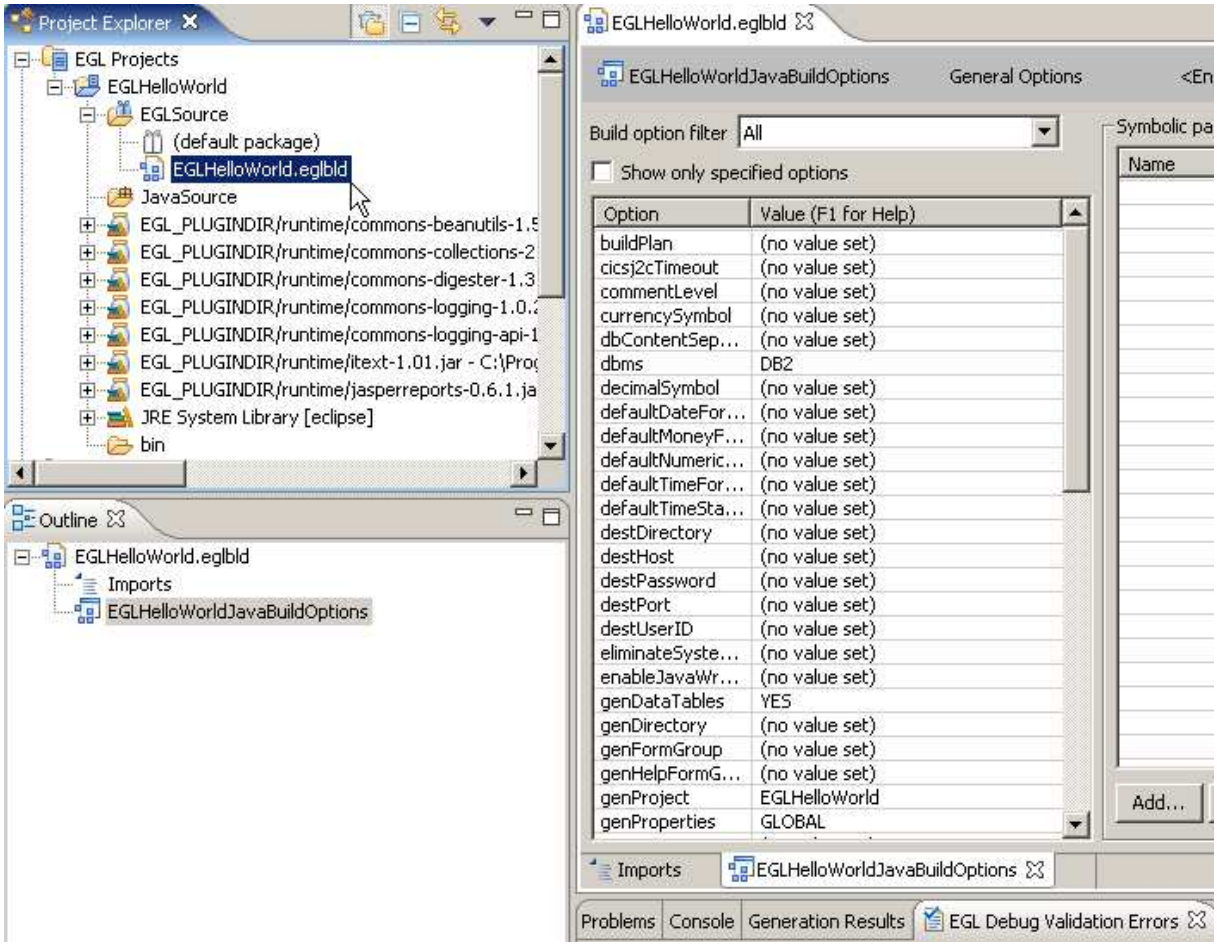


Es öffnet sich ein Wizzard. Dort wird der Name des Projekts angegeben. In diesem Fall beispielsweise „EGLHelloWorld“. Weiterhin sollte darauf geachtet werden, dass die Option „Create new project build descriptor(s) automatically“ aktiviert ist. Durch einen weiteren Klick auf Finish wird das Projekt erzeugt.

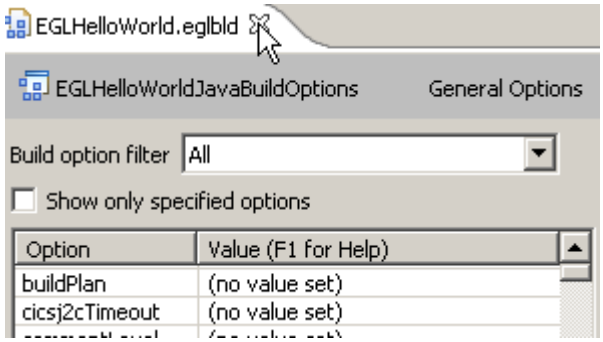


Das neue Projekt ist nun im Project Explorer auf der linken Bildschirmseite unter EGL Projects zu finden.

Es enthält zahlreiche Java-Bibliotheken und zwei Ordner für EGL- und Java-Quellcode. Im EGLSource-Ordner ist bereits die Datei EGLHelloWorld.eglbld enthalten. Hierbei handelt es sich um den erwähnten EGL-Build-Descriptor. In dieser Datei sind zahlreiche Einstellungen, die zum Generieren des Java-Codes notwendig sind, enthalten. Diese Einstellungen wurden vom WDz teilweise automatisch vorbelegt und können nun angepasst werden. Um die Einstellungen zu ändern, genügt ein Doppelklick auf den Dateinamen.



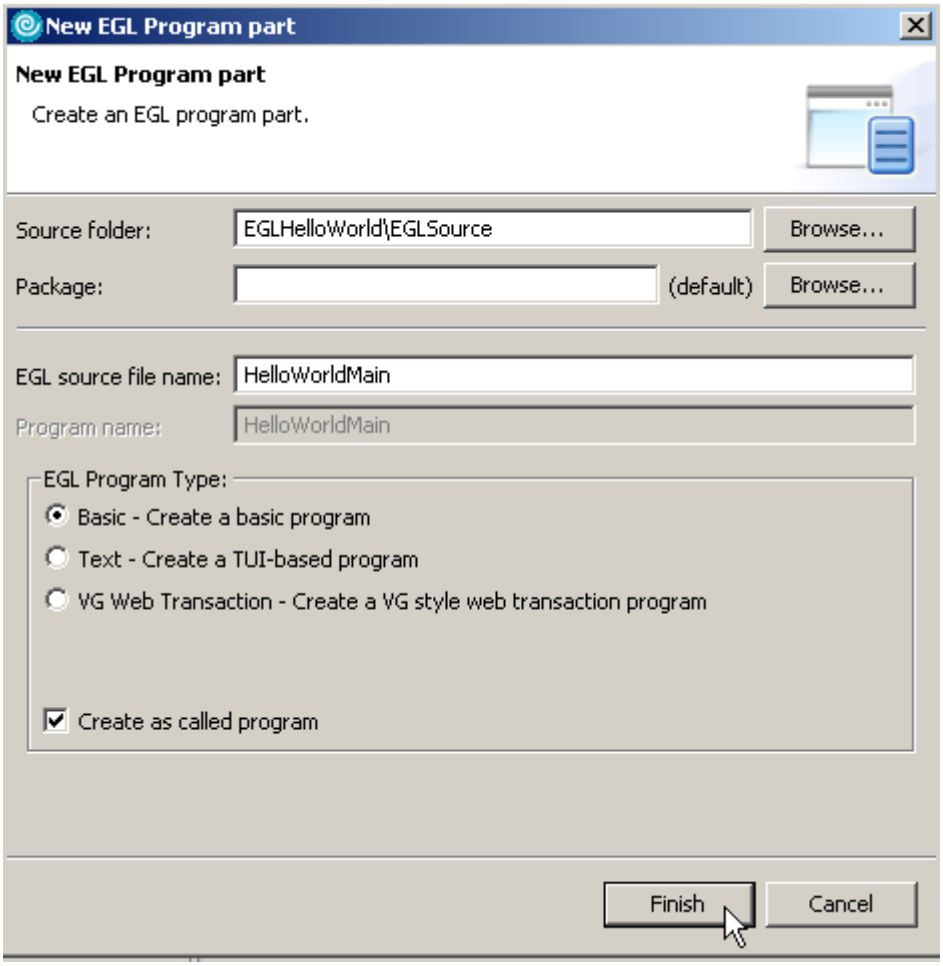
Es öffnet sich die dargestellte Ansicht der Buildoptionen. Da in diesem Fall keine Einstellungen nötig sind, kann die Datei ohne Änderungen wieder geschlossen werden.

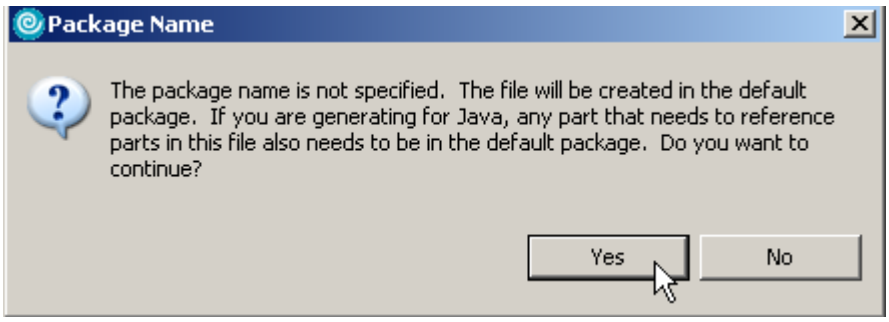


7. Programm erstellen

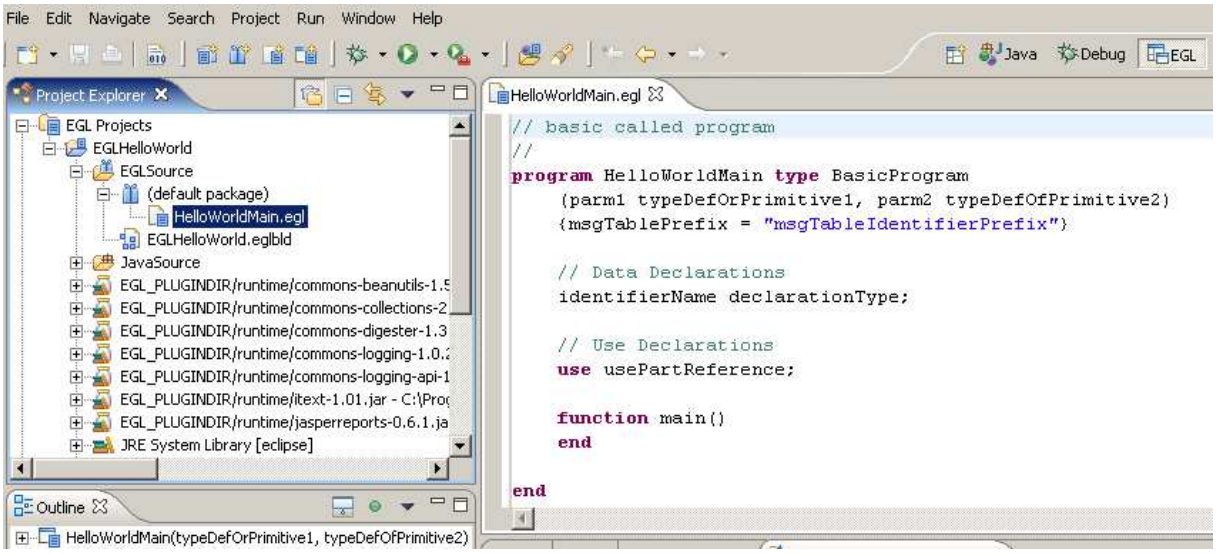
Als nächstes muss eine neue Source-Code-Datei erstellt werden, die später die eigentliche Ausführungslogik enthalten soll: File → New → Program .

Es werden nun Dateiname (EGL source file name = HelloWorldMain) und Programmtyp (EGL Program Type = Basic – Create a basic program) festgelegt. Weitere Einstellungs-möglichkeiten sind der Zielordner, in dem die Datei abgelegt werden soll, und das Packa-ge. Bei einem Package handelt es sich (wie bei Java) um eine Art Namespace, der Funktionen enthalten kann. Gleichartige Funktionen können so zur besseren Strukturierung in einem gemeinsamen Package zusammengefasst werden. Für das „Hello World"-Beispiel ist keine Änderung der Einstellungen notwendig. Häkchen vor „Create as called program". Der Dialog ist über Finish zu schließen.





Bevor die Source-Code-Datei erstellt wird, wird beim Nutzer noch einmal nachgefragt, ob wirklich das Default-Package verwendet werden soll. Diese Frage ist mit Yes zu beantworten.



Die neue Datei (HelloWorldMain.egl) wird jetzt automatisch im Source-Code-Editor geöffnet und ist ebenso im Project Explorer sichtbar

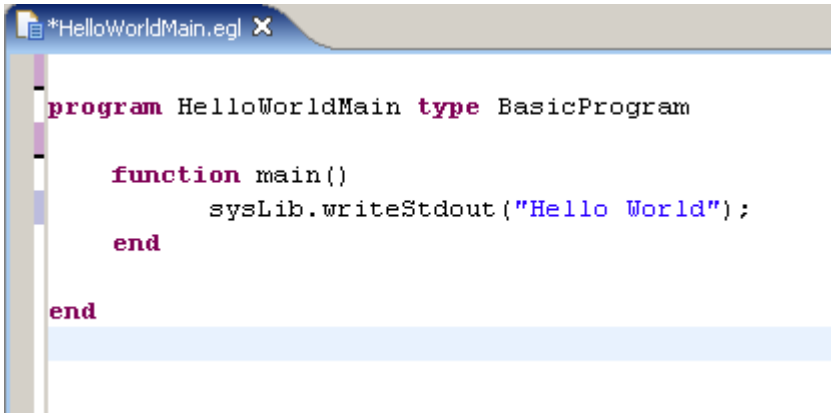
Die Datei enthält ein sog. Skeleton, also einen Vorschlag von WDz, wie das main Programm aussehen könnte.Das skeleton hat folgenden Inhalt:

```
// basic called program
//
program HelloWorldMain type BasicProgram
   (parm1 typeDefOrPrimitive1, parm2 typeDefOfPrimitive2)
   {msgTablePrefix = "msgTableIdentifierPrefix"}

   // Data Declarations
   identifierName declarationType;
   // Use Declarations
   use usePartReference;

   function main()
   end
end
```

Dieser Code ist nicht lauffähig und enthält unnötige Bestandteile, die der WDz zur Verdeutlichung der Syntax generiert. Der Code wird deshalb durch den folgenden ersetzt und stellt unser HelloWorld Programm dar:

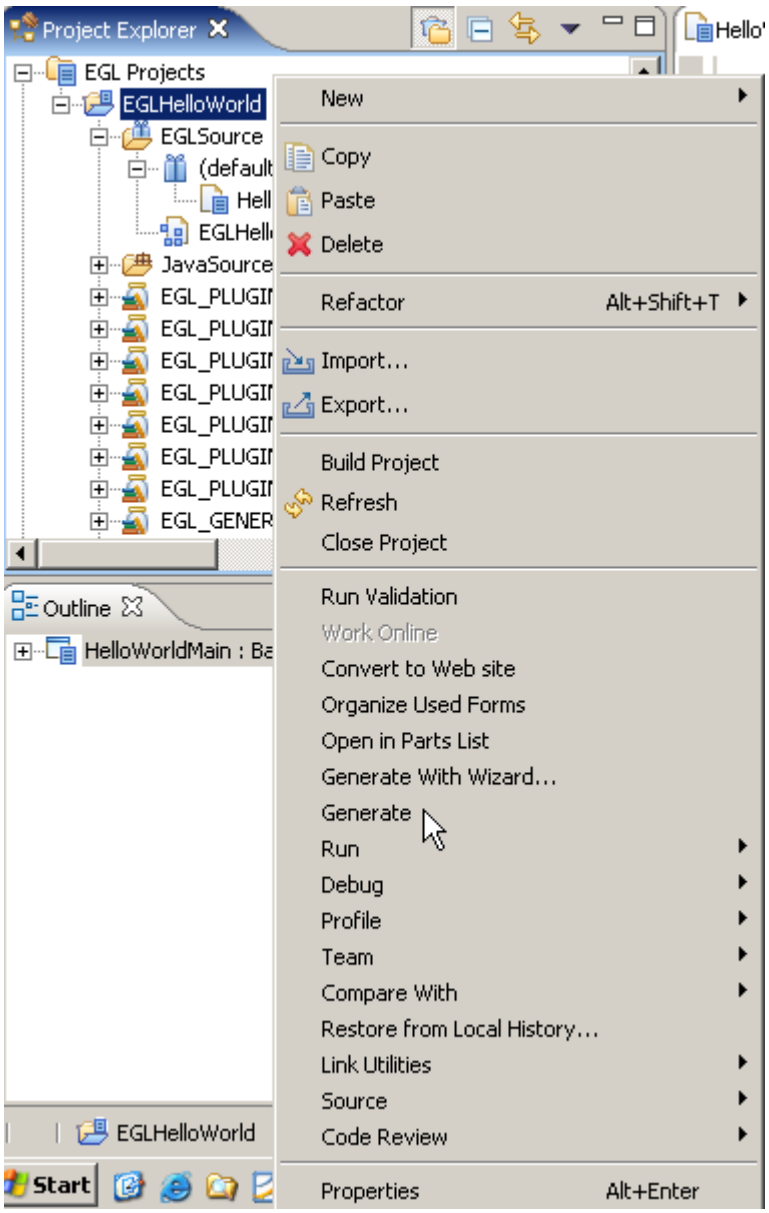
```
program HelloWorldMain type BasicProgram

   function main()
         sysLib.writeStdout("Hello World");
   end

end
```

Dieses Programm besteht lediglich aus einer Hauptfunktion (main()), die beim Starten des Programms ausgeführt wird. Innerhalb der Funktion wird mit Hilfe der Standardbibliothek sysLib ein String in den Standard-Out-Stream ausgegeben.



Um das Programm kompilieren zu können, wird es in Java-Source-Code umgewandelt. Dies geschieht durch einen Rechtsklick auf den Ordner EGLSource im Project Explorer. Im daraufhin geöffneten Kontextmenü wird Generate gewählt.

1kr Rechtsklick auf EGLHelloWorld

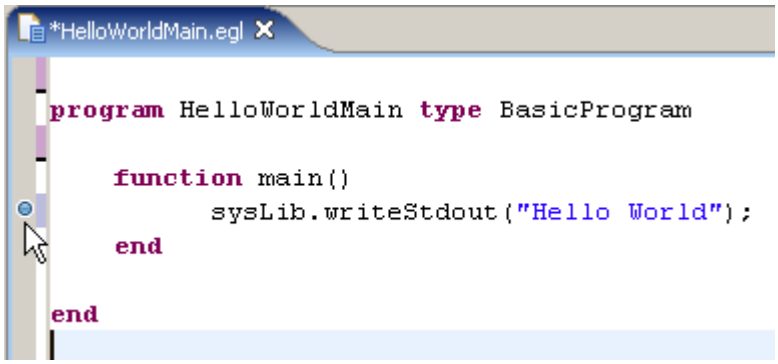


Falls die Source-Code-Datei zu diesem Zeitpunkt noch nicht gespeichert wurde, wird vom WDz gefragt, ob die Datei jetzt gespeichert werden soll. Diese Frage ist mit OK zu beantworten. Der WDz erzeugt nun selbstständig den Java-Source-Code und legt ihn im Ordner JavaSource (default package) ab. In diesem Fall handelt es sich dabei um die Datei namens HelloWorldMain.java.

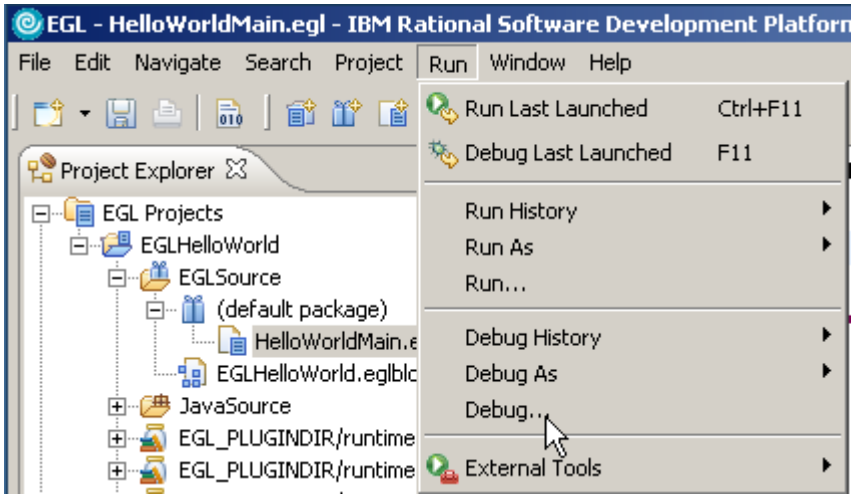
8. Programm im Debugger ausführen

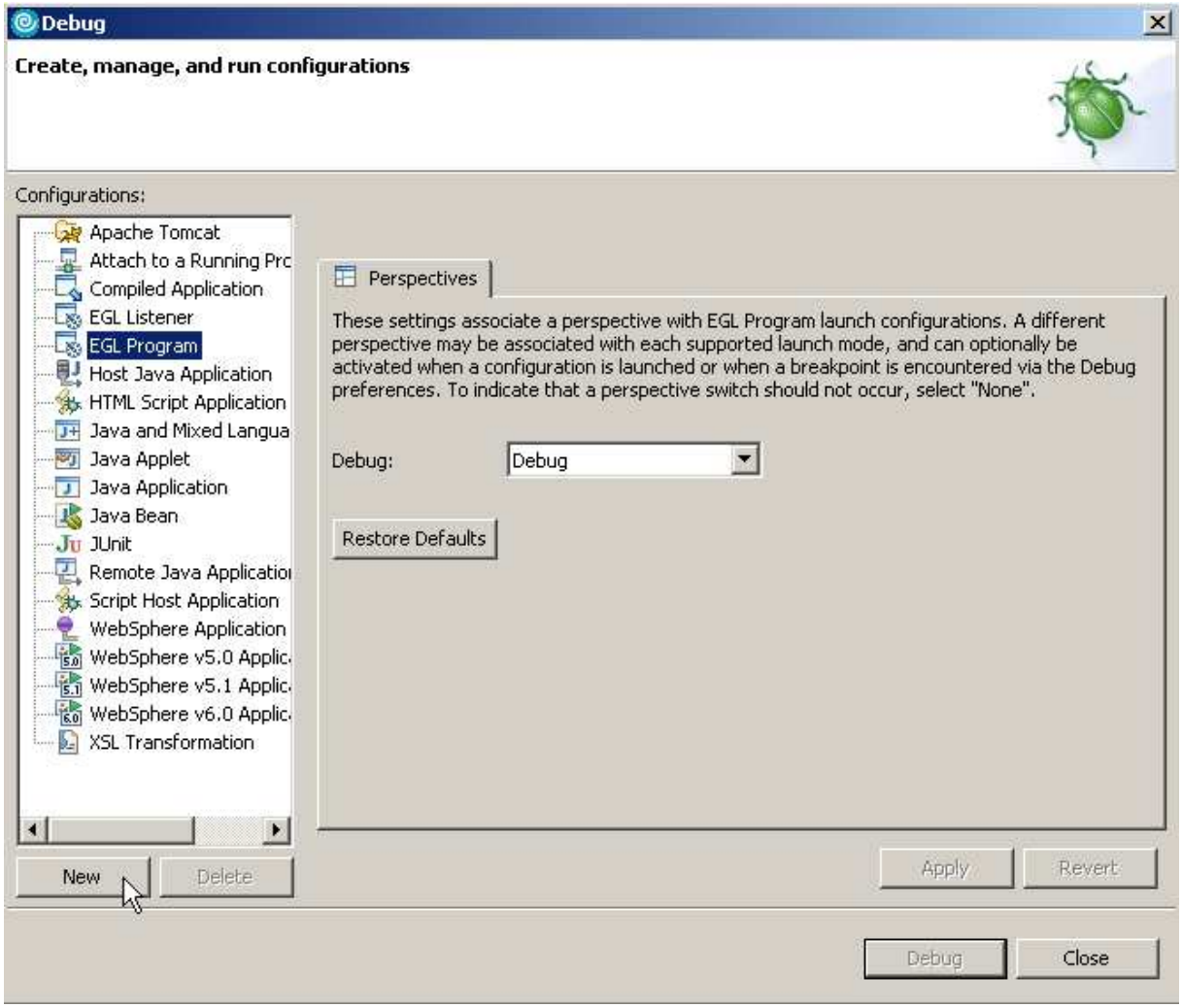
Als nächstes wird das Programm im Debugger ausgeführt und ein Breakpoint gesetzt, um es an einer bestimmten Stelle anzuhalten.

Das Setzen eines Breakpoints geschieht durch Doppelklick auf die graue Leiste auf der linken Seite des Source-Code-Editors. Bei späteren Debuggerdurchläufen wird der Programmablauf in jeder Zeile angehalten, die einen solchen Breakpoint enthält. Hierbei ist zu beachten, dass nur in den Zeilen Breakpoints gesetzt werden können, die ausführbare Logik enthalten. In diesem Beispiel ist dies nur in der Zeile mit dem Aufruf von sysLib.writeStdout() möglich.



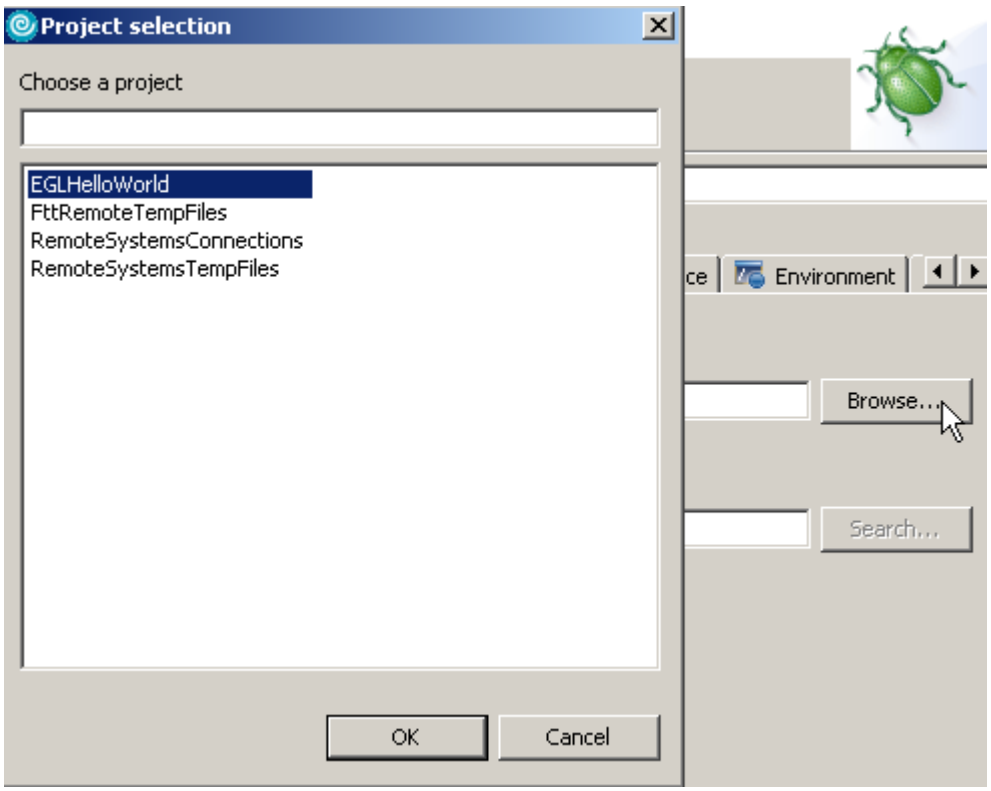
Nach Setzen des Breakpoints wird eine neue Debugger-Konfiguration angelegt. Das geschieht durch Run → Debug ... und muss nur einmal durchgeführt werden, da die Konfiguration im WDz gespeichert wird.





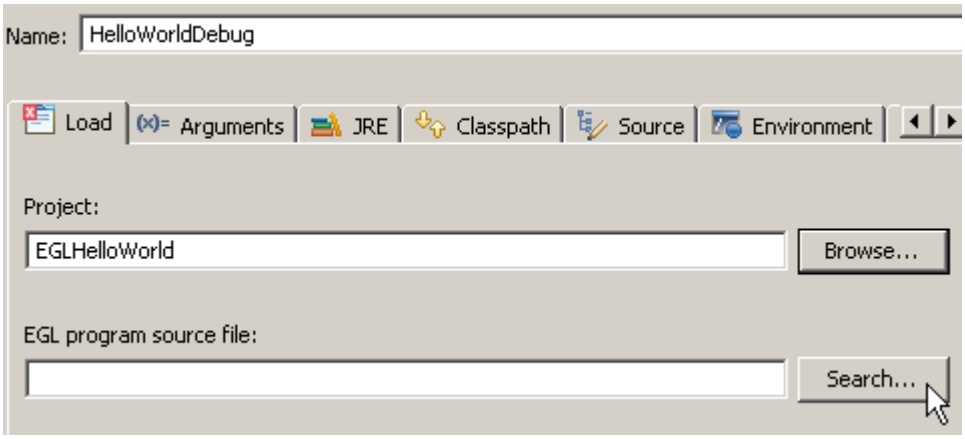
Man wählt nun EGL-Program und klickt auf New.

Über das Eingabefeld Name kann jetzt der Name der Konfiguration eingeben werden (z. B. HelloWorldDebug). Dann wird das Projekt ausgesucht, welches ausgeführt werden soll. Zu diesem Zweck wird der Browse-Button gedrückt und das Projekt EGLHelloWorld selektiert.

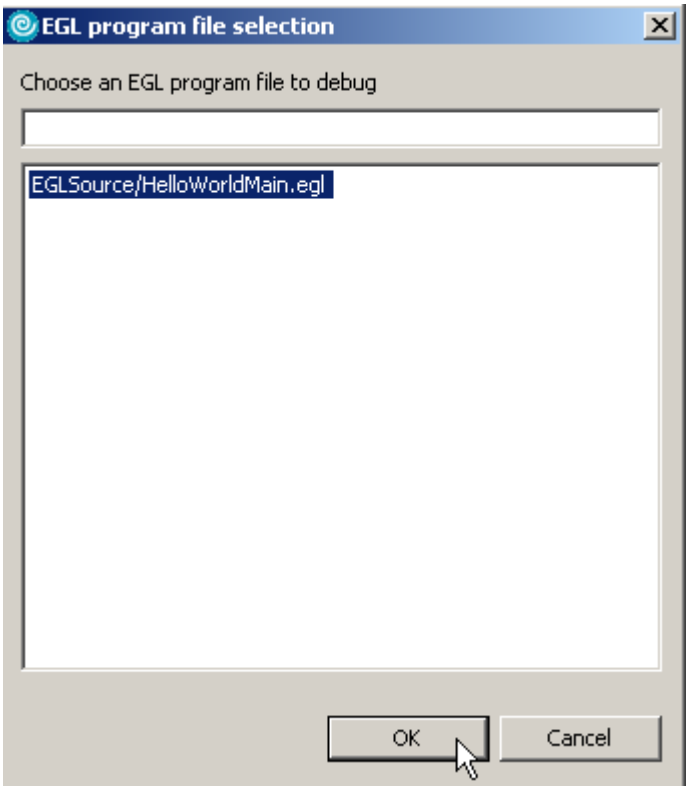


1k auf ok.

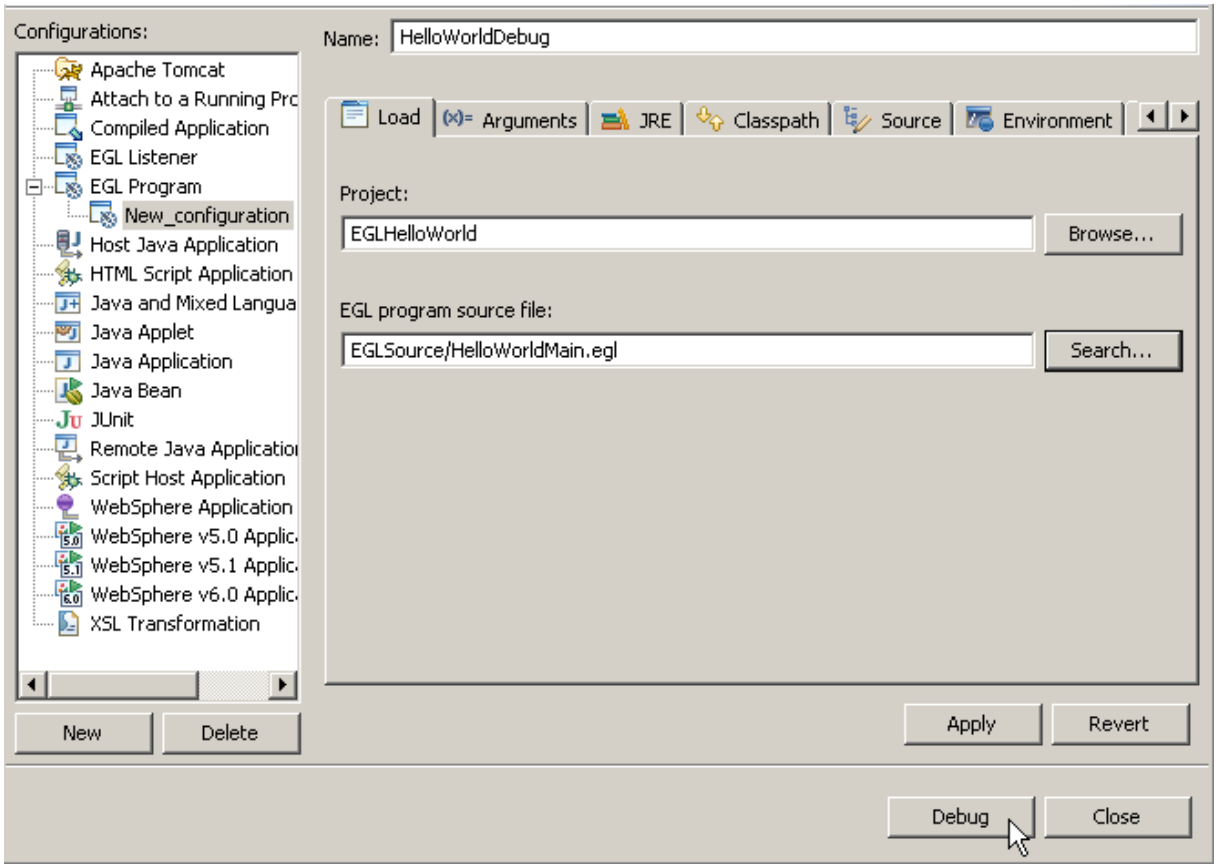
Abschließend muss der Einstiegspunkt in das Programm angegeben werden. Hierzu wird der Search-Button angeklickt.



EGLSource/HelloWorldMain.egl ausgewählt und OK angeklickt.

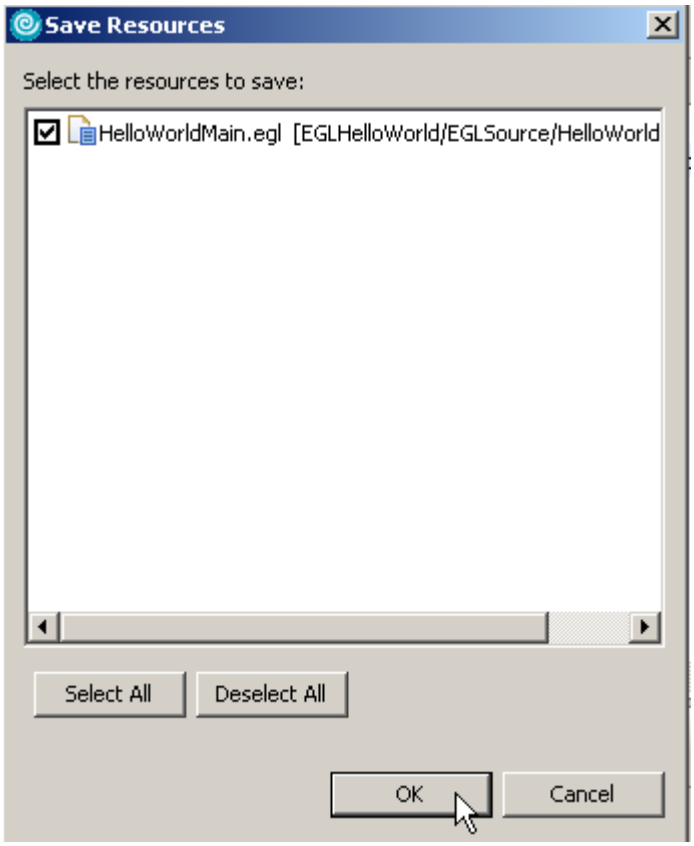


Weitere Einstellungen sind nicht nötig.



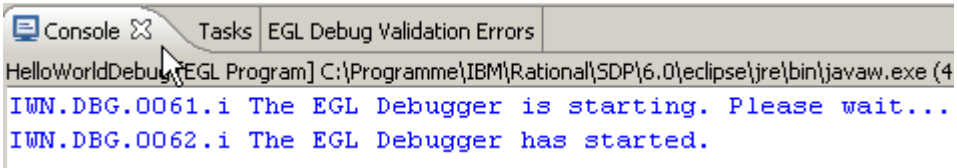
Ein Klick auf Debug startet den Debugger mit der neuen Konfiguration.

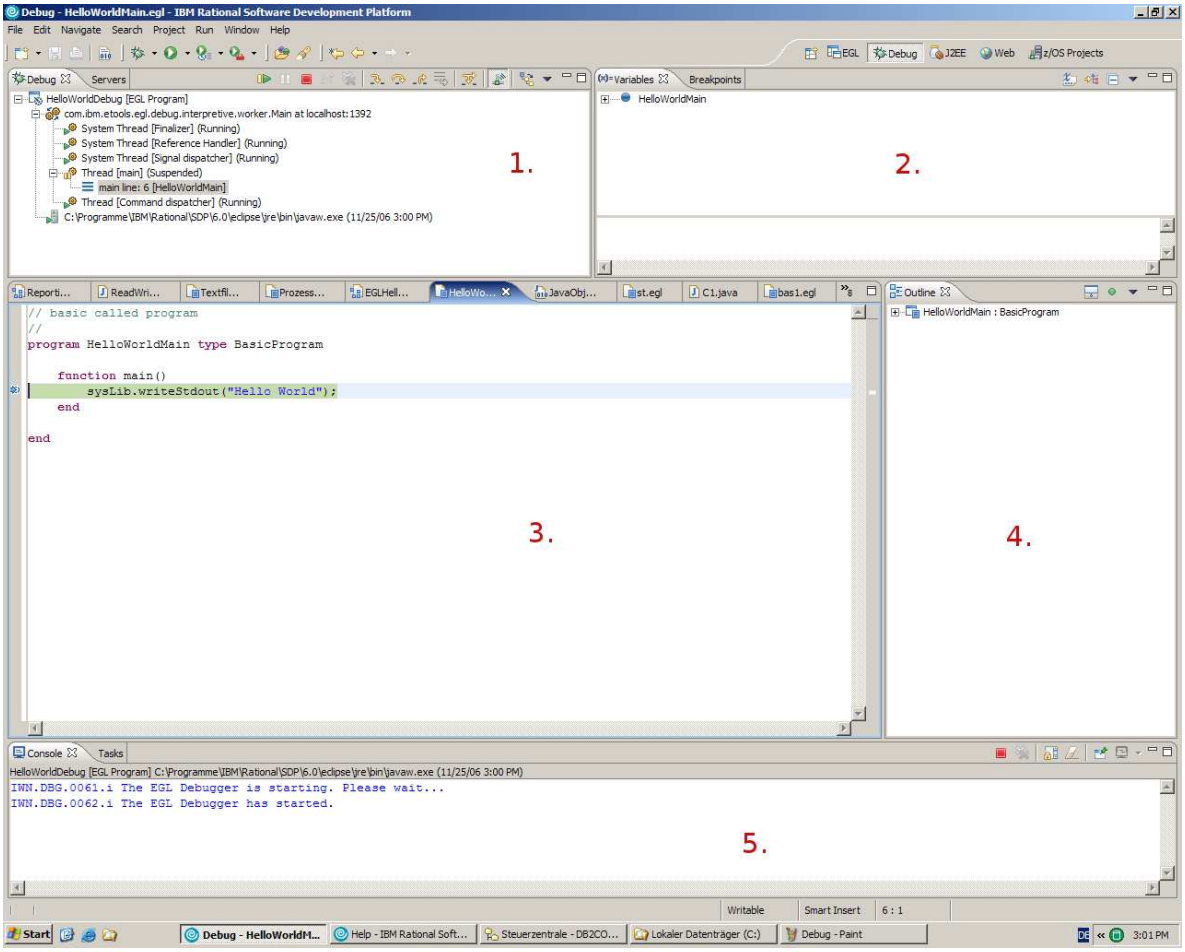
Falls die Source-Code-Datei noch nicht gespeichert wurde, wird nachgefragt, ob diese gespeichert werden soll. Dies wird mit OK bejaht.



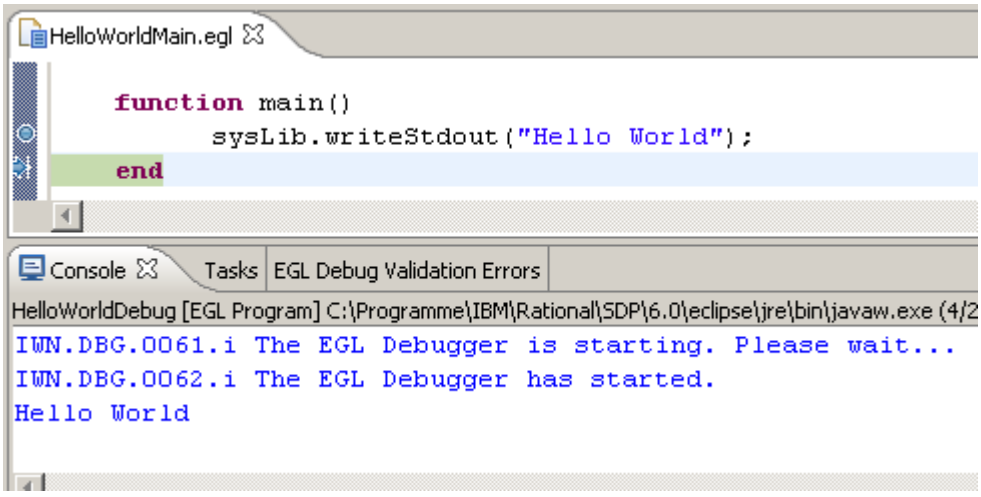
Der WDz wechselt jetzt die Perspektive von EGL nach Debug. Diese Perspektive gliedert sich in 5 Hauptbestandteile:

1. **Links oben:** Enthält Informationen über den DebugThread
2. **Rechts oben**: Informationen über Variablen
3. **Mitte links:** Ansicht des Quellcodes, an dem sich die Ausführung gerade befindet. Die aktuelle Zeile ist dabei mit einem kleinen Pfeil markiert.
4. **Mitte rechts**: Struktur der geöffneten Source-Code-Datei.
5. **Unten:** Ausgabe auf der Konsole (das Register Console muss ausgewählt sein).

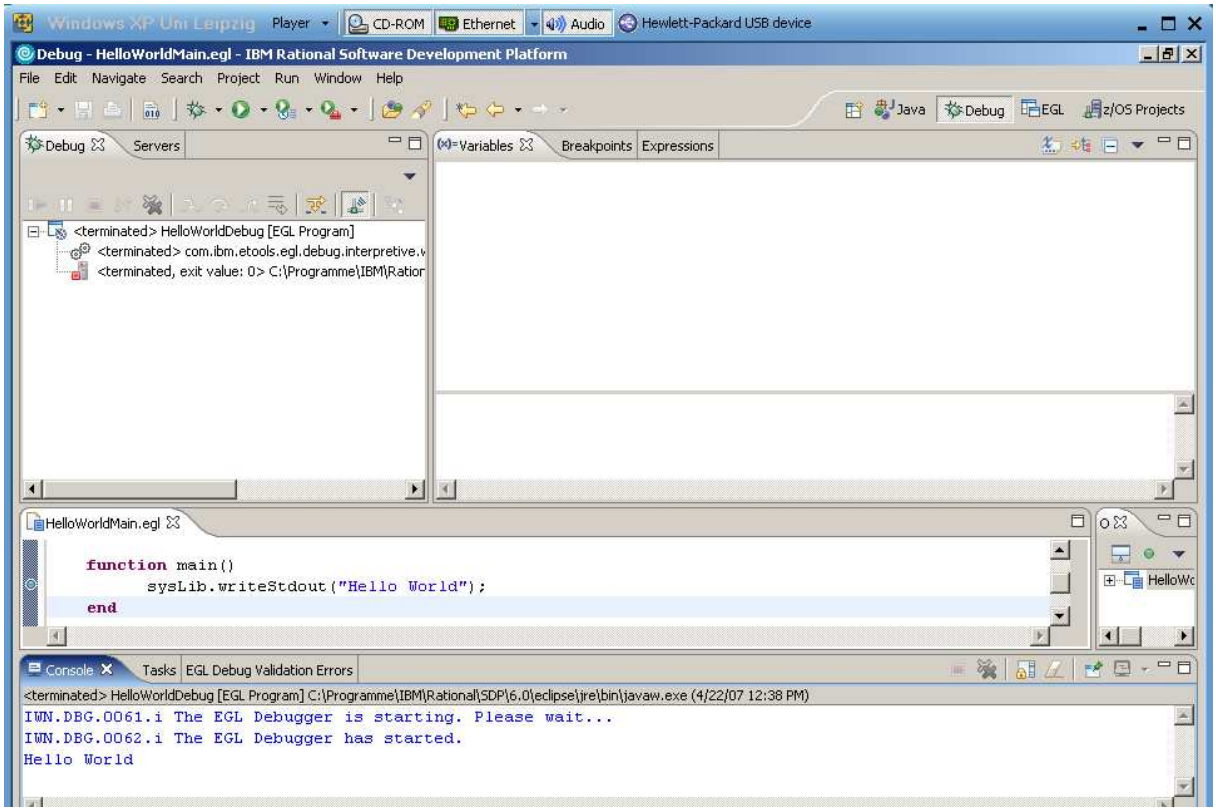




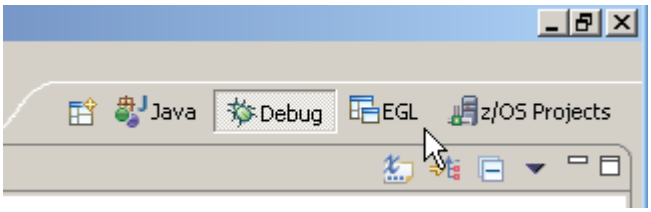
In der Quellcodeansicht (3.) ist bereits das Hauptprogramm geöffnet. Außerdem ist die Codezeile, in der der Breakpoint gesetzt wurde, markiert. Durch Drücken der Taste F6 kann zur nächsten Anweisung gesprungen werden.

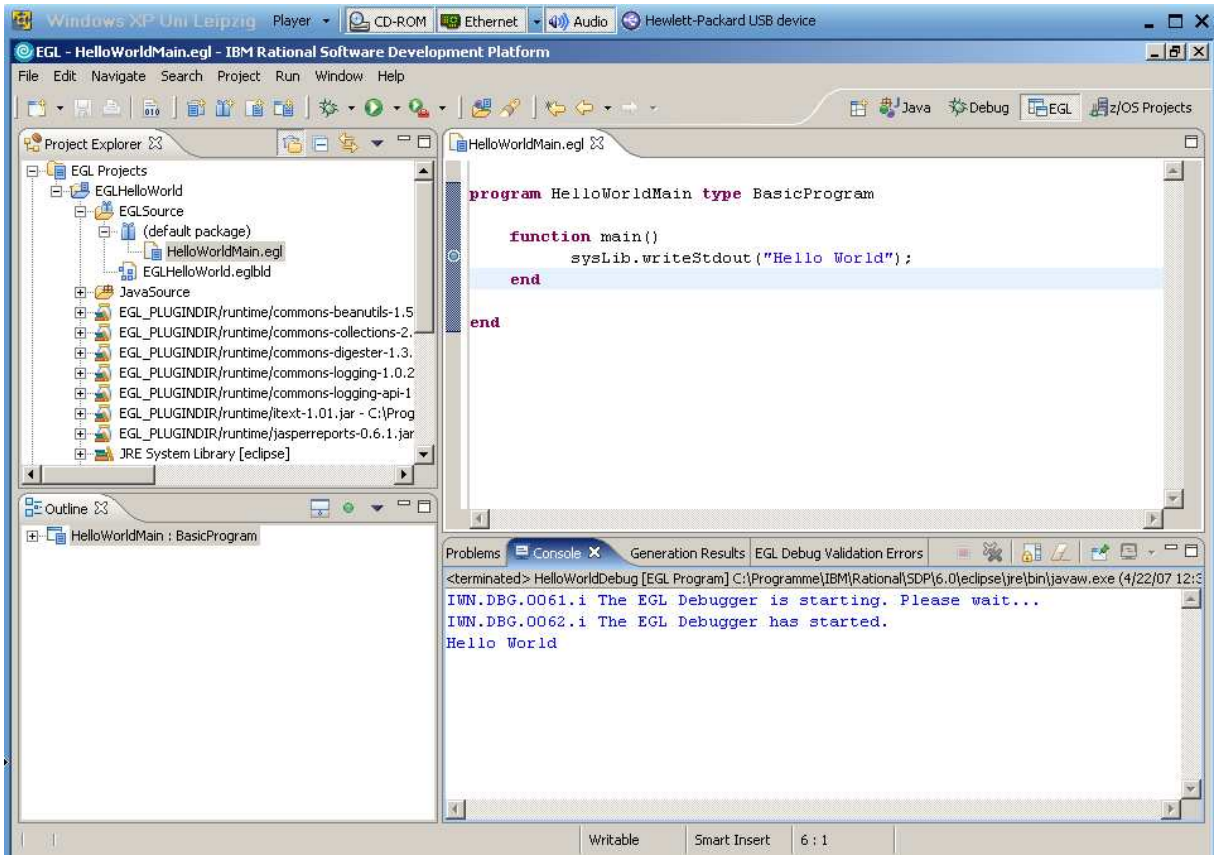


Wie erwartet wird "Hello World" auf der Konsole (Bereich 5) ausgegeben. Ein weiterer Tastendruck auf F6 beendet das Programm, da die letzte Anweisung ausgeführt wurde



Nach Beendigung des Durchlaufs kann die Perspektive wieder gewechselt werden. Hierzu muss man den Knopf "EGL" in der Leiste am oberen Bildschirmrand drücken.

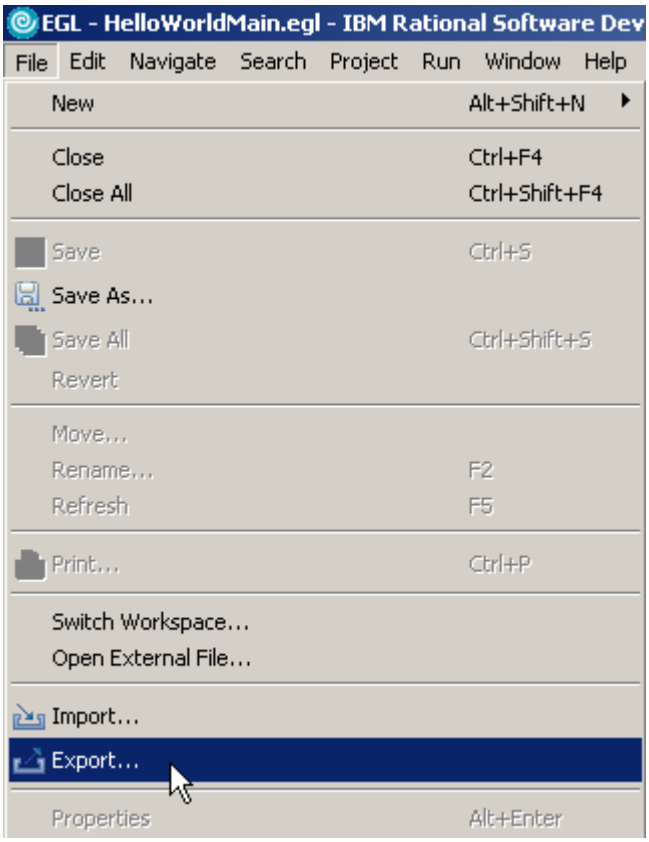


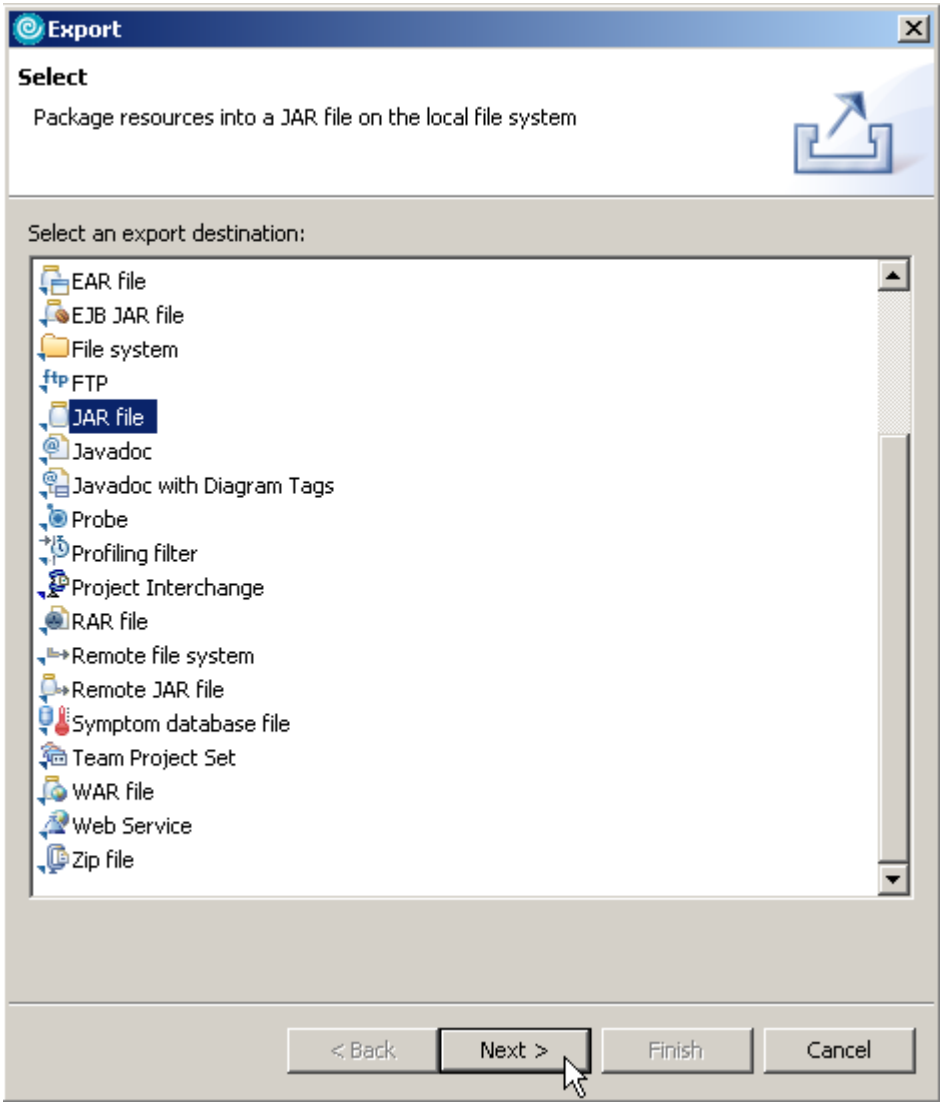

EGL - HelloWorldMain.egl - IBM Rational Software Development Platform
File Edit Navigate Search Project Run Window Help
Java Debug EGL z/OS Projects
Project Explorer
EGL Projects
EGLHelloWorld
EGLSource
(default package)
HelloWorldMain.egl
EGLHelloWorld.eglbld
JavaSource
HelloWorldMain.egl
program HelloWorldMain type BasicProgram
function main()
sysLib.writeStdout("Hello World");
end
end
Outline
HelloWorldMain : BasicProgram
Problems Console Generation Results EGL Debug Validation Errors
IWN.DBG.0061.i The EGL Debugger is starting. Please wait...
IWN.DBG.0062.i The EGL Debugger has started.
Hello World
Writable Smart Insert 6 : 1

9. Exportieren des Programms auf einen anderen Computer

Um das Programm auf einem beliebigen Computer ausführen zu können, muss eine (für Java übliche) JAR-Datei erzeugt werden (- im Grunde handelt es sich hierbei um eine ZIP-Datei.). Diese Datei enthält den kompilierten Bytecode der Anwendung.

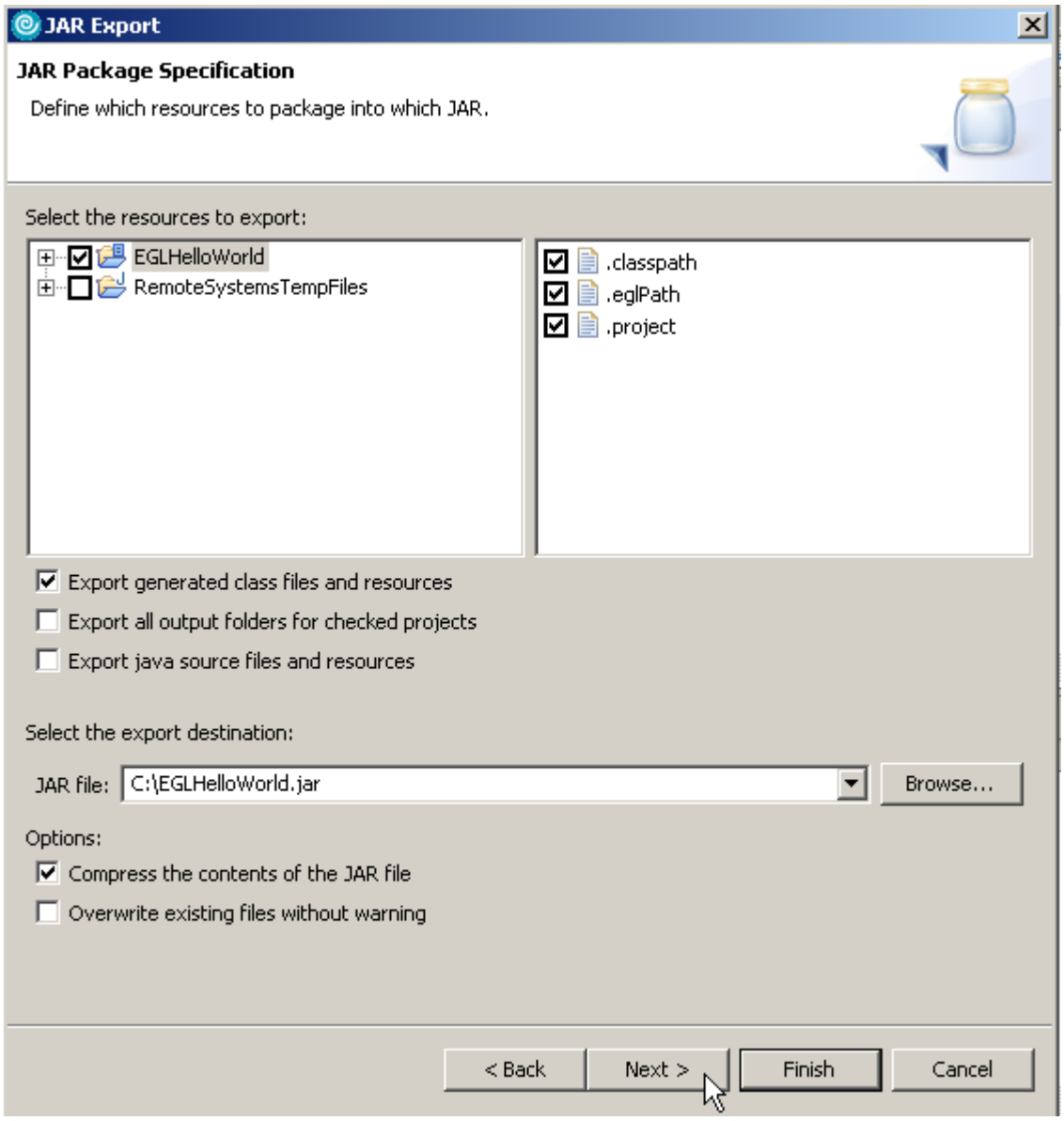
Zum Erzeugen der JAR-Datei wird File → Export ausgewählt.





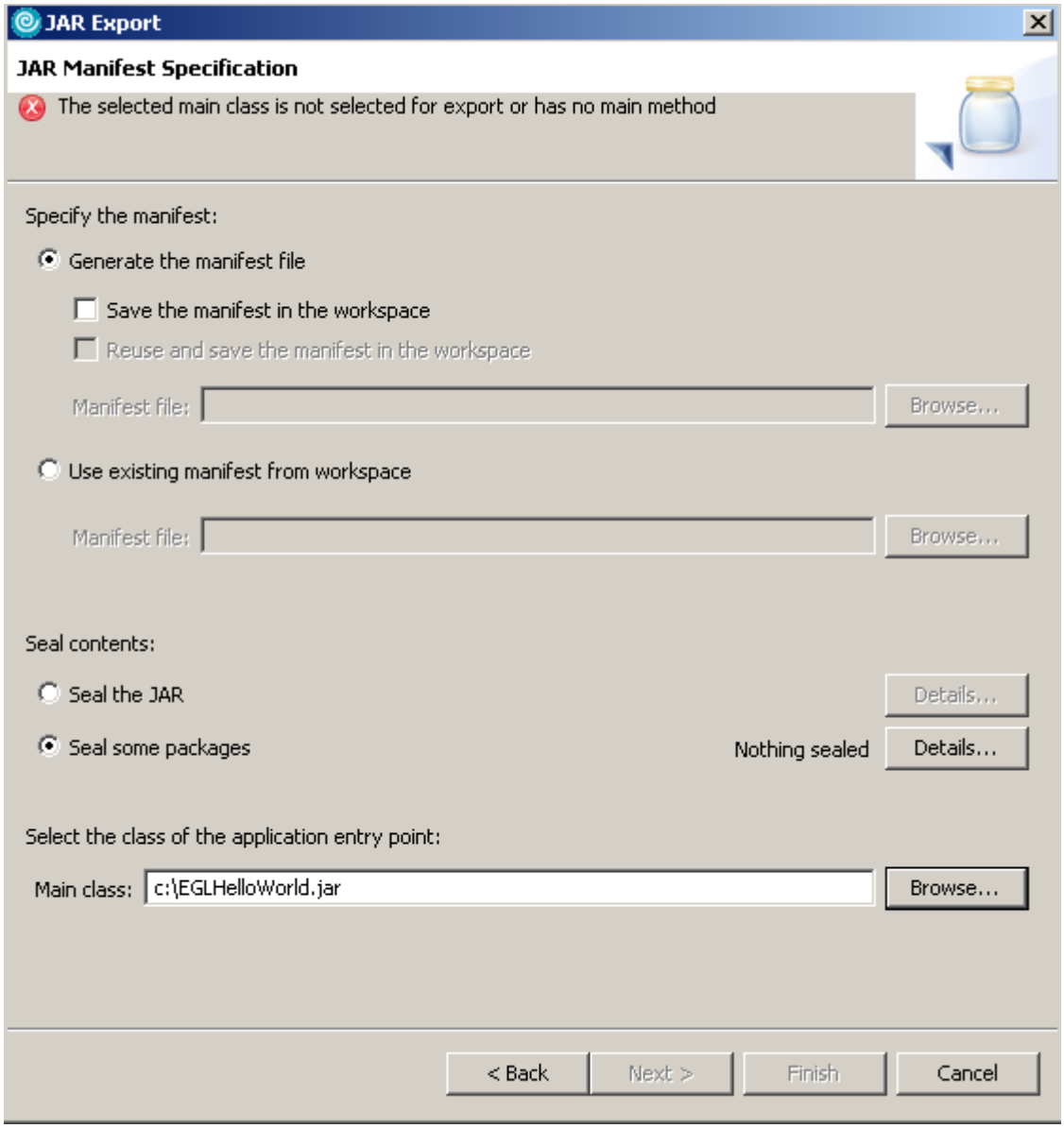
Es öffnet sich ein neuer Dialog, in dem JAR File gewählt und Next gedrückt wird.

Als nächstes wird das Projekt ausgesucht, das exportiert werden soll (vor EGLHelloWorld ist ein Haken zu setzen).

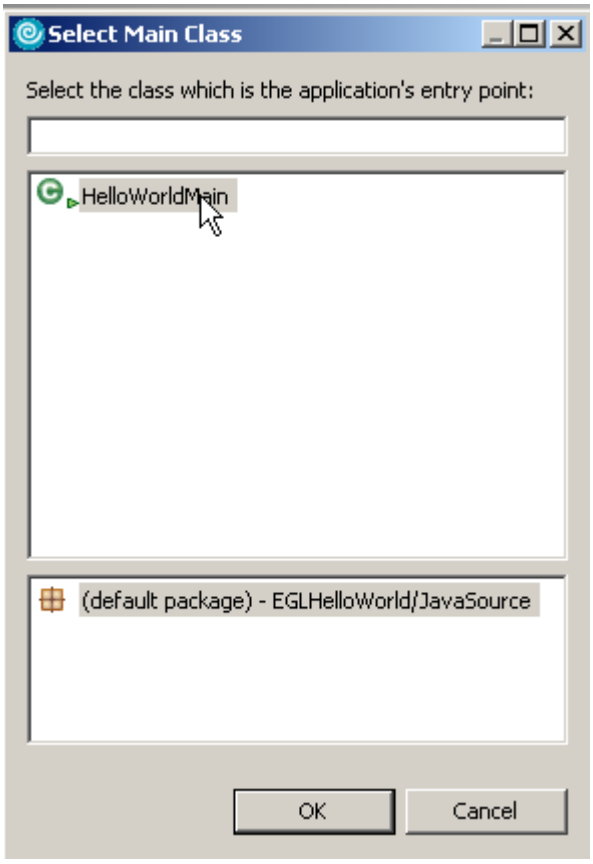


Über den Browse Button werden Zielverzeichnis und Name der JAR-Datei festgelegt (beispielsweise c:\EGLHelloWorld.jar).

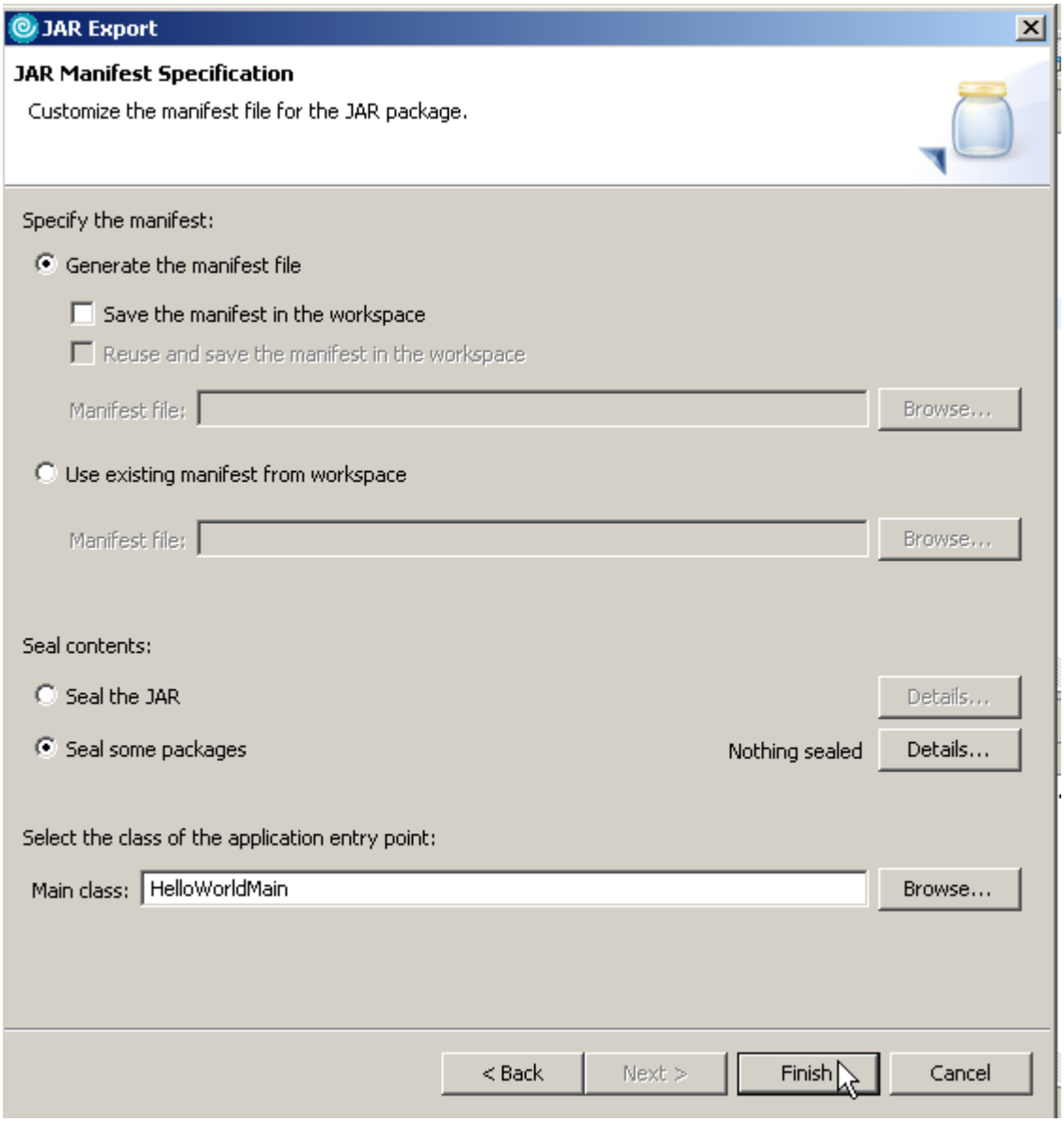
Man klickt nun zweimal auf Next.



und wählt die Funktion, die beim Programmstart als erstes ausgeführt werden soll (Main class). Dies geschieht durch einen Klick auf den Browse Button und durch Auswahl von HelloWorldMain.



Doppelklick auf den Namen oder alternativ Namen auswählen und ok drücken

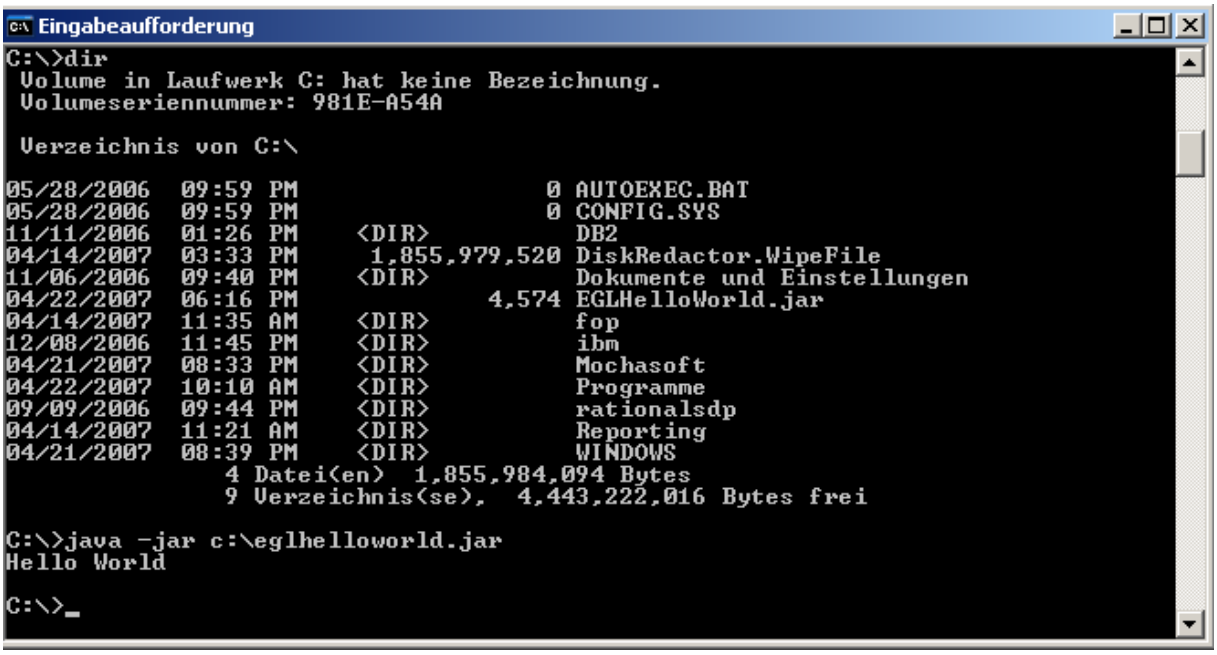


Mit einem finalen Klick auf Finish wird die JAR-Datei erzeugt.

Um das Programm ohne WDz auszuführen, öffnet man die DOS Eingabeaufforderung in der virtuellen Maschine. Im Verzeichnis c:\ befindet sich jetzt die Datei EGLHelloWorld.jar. Eine Eingabe des folgenden Befehls:

java –jar c:\EGLHelloWorld.jar

erzeugt das erwartete Ergebnis.



Aufgabe: *Legen Sie nun ein weiteres EGL-Projekt an. Der Projektname soll Ihre PRAK(T)-User-ID sowie Ihren Nachnamen enthalten; z.B. "PRAKT20(Müller)".*

Aufgabe: *Erstellen Sie in diesem neuen EGL-Projekt ein selbst entwickeltes EGL-Programm. Zur Programmentwicklung kann nachstehender Anhang "Übersicht der grundlegenden Programmkonstrukte der EGL" genutzt werden.*

Aufgabe: *Generieren Sie aus dem EGL-Code den Java- und den Bytecode.*

Aufgabe: *Exportieren Sie die JAR-Datei in folgendes Verzeichnis auf dem virtuellen Windows Server "websphere" (IP 139.18.8.211): "C:\WDz-Tutorien\Tutorial-16d(EGL)". Geben Sie dieser JAR-Datei einen Namen, der Ihre PRAK(T)-User-ID und Ihren Nachnamen enthält, wie z.B. "PRAKT20-Müller.jar". Falls sich in diesem Verzeichnis schon eine JAR-Datei mit Ihrer PRAK(T)-User-ID im Dateinamen befindet, ist diese alte Datei zu löschen.*

Aufgabe: *Exportieren Sie den Programm-Code Ihres EGL-Programms in folgendes Verzeichnis auf dem virtuellen Windows Server "websphere" (IP 139.18.8.211): "C:\WDz-Tutorien\Tutorial-16d(EGL)". Geben Sie dieser Text-Datei einen Namen, der Ihre PRAK(T)-User-ID und Ihren Nachnamen enthält, wie z.B. "PRAKT20Müller.egl". Falls sich in diesem Verzeichnis schon solch eine Datei mit Ihrer PRAK(T)-User-ID im Dateinamen befindet, ist diese alte Datei zu löschen.*

Aufgabe: *Fertigen Sie zwei Screenshots entsprechend den beiden nachfolgenden Abbildungen an. Die Screenshots müssen folgendes nachweisen:*

1. Screenshot (Projekt-Explorer, EGL-Programmcode, Sicht "Ergebnisse der Generierung")

1) Dass Sie die beiden EGL-Projekte "EGLHelloWorld" und "<Ihre Prakt-ID><Ihr Nachname>" angelegt haben

2) Die Existenz Ihrer EGL-Datei sowie des daraus generierten Java-Codes sowie Java-Bytecodes

3) Ihren VOLLSTÄNDIGEN EGL-Programmcode. Reicht ein Screenshot dafür nicht aus, dann sind mehrere Screenshots anzufertigen.

4) Die Sicht "Ergebnisse der Generierung" muss im Screenshot enthalten sein und nachweisen, dass die Generierung aus dem eigenen EGL-Code fehlerfrei beendet wurde.

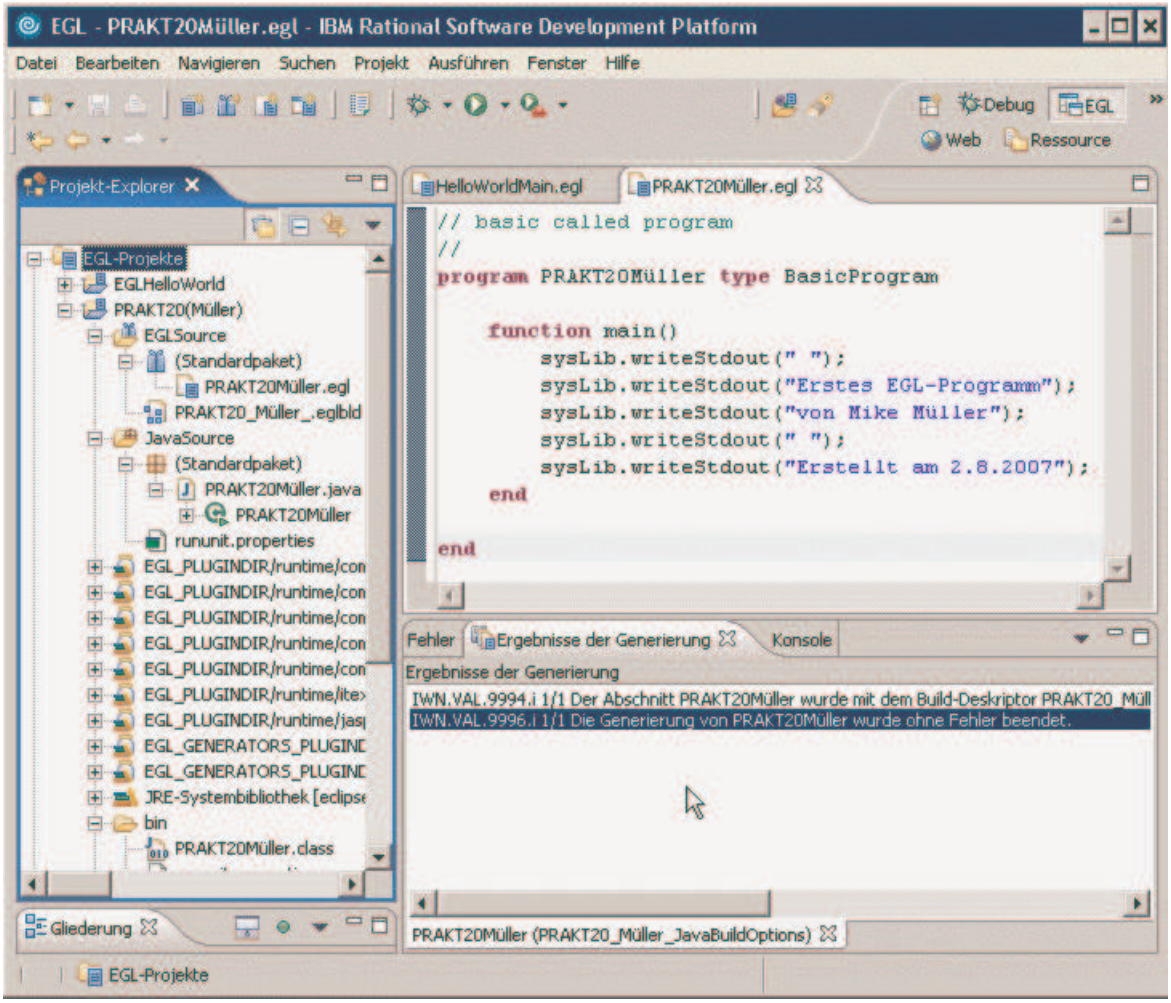
2. Screenshot (DOS-Fenster des virtuellen Windows Servers mit IP 139.18.8.211)

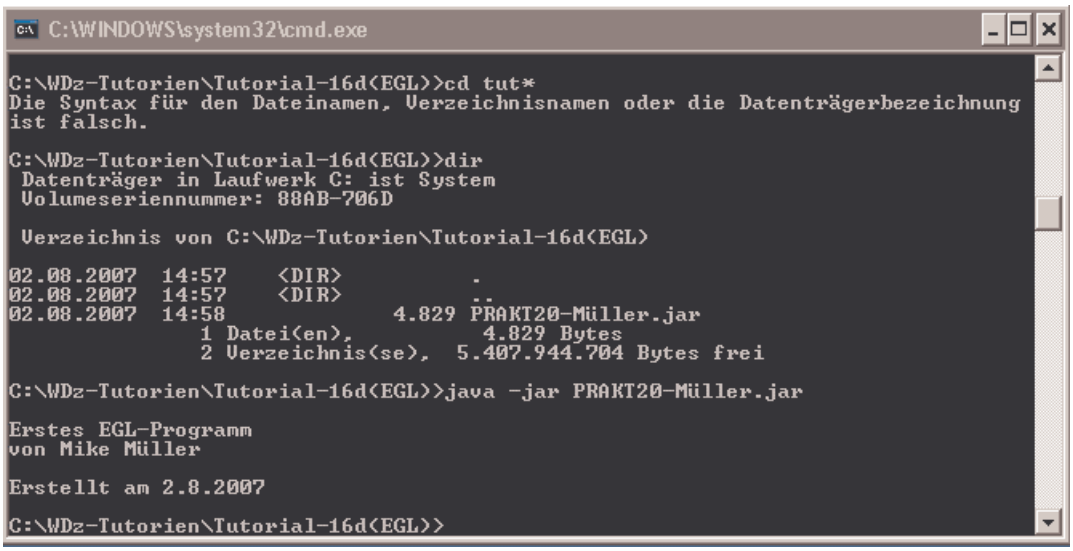
1) Befehl, mit dem das eigene Programm aufgerufen wurde: java –jar ...

2) Alle Ein- und Ausgaben Ihres Programms

Die Screenshots müssen im JPEG-Format erstellt werden und dürfen eine Größe von 200 Kbyte nicht überschreiten.

Aufgabe: *Schicken Sie eine Abgabe-E-Mail an Ihren Betreuer mit genau zwei Anhängen, nämlich den beiden Screenshots. Wenn Sie eigenständig ein erheblich komplexeres EGL-Programm, als in diesem Tutorial vorgestellt, entwickelt haben, dann erhalten Sie dafür eventuell einen Bewertungspluspunkt. Bedingung: Sie müssen in der Abgabe-E-Mail erklärt haben, dass Sie das Programm vollständig oder teilweise selbst entwickelt haben und dies ausschließlich für die Übung "z/OS und OS/390" getan haben.*


EGL - PRAKT20Müller.egl - IBM Rational Software Development Platform
Datei Bearbeiten Navigieren Suchen Projekt Ausführen Fenster Hilfe
Debug
EGL
Web
Ressource
Projekt-Explorer
EGL-Projekte
EGLHelloWorld
PRAKT20(Müller)
EGLSource
(Standardpaket)
PRAKT20Müller.egl
PRAKT20_Müller_.eglbld
JavaSource
(Standardpaket)
PRAKT20Müller.java
PRAKT20Müller
rununit.properties
bin
PRAKT20Müller.class
HelloWorldMain.egl
PRAKT20Müller.egl
// basic called program
//
program PRAKT20Müller type BasicProgram
function main()
sysLib.writeStdout(" ");
sysLib.writeStdout("Erstes EGL-Programm");
sysLib.writeStdout("von Mike Müller");
sysLib.writeStdout(" ");
sysLib.writeStdout("Erstellt am 2.8.2007");
end
end
Fehler
Ergebnisse der Generierung
Konsole
Ergebnisse der Generierung
IWN.VAL.9994.i 1/1 Der Abschnitt PRAKT20Müller wurde mit dem Build-Deskriptor PRAKT20_Müll
IWN.VAL.9996.i 1/1 Die Generierung von PRAKT20Müller wurde ohne Fehler beendet.
Gliederung
PRAKT20Müller (PRAKT20_Müller_JavaBuildOptions)
EGL-Projekte


C:\WINDOWS\system32\cmd.exe
C:\WDz-Tutorien\Tutorial-16d(EGL)>cd tut*
Die Syntax für den Dateinamen, Verzeichnisnamen oder die Datenträgerbezeichnung ist falsch.
C:\WDz-Tutorien\Tutorial-16d(EGL)>dir
Datenträger in Laufwerk C: ist System
Volumeseriennummer: 88AB-706D
Verzeichnis von C:\WDz-Tutorien\Tutorial-16d(EGL)
02.08.2007 14:57 <DIR> .
02.08.2007 14:57 <DIR> ..
02.08.2007 14:58 4.829 PRAKT20-Müller.jar
1 Datei(en), 4.829 Bytes
2 Verzeichnis(se), 5.407.944.704 Bytes frei
C:\WDz-Tutorien\Tutorial-16d(EGL)>java -jar PRAKT20-Müller.jar
Erstes EGL-Programm
von Mike Müller
Erstellt am 2.8.2007
C:\WDz-Tutorien\Tutorial-16d(EGL)>

Anhang
Übersicht der grundlegenden Programmkonstrukte der EGL

- **Variablendeklaration**

Syntax	Erklärung
Variablenname Typ;	Die Deklaration erfolgt durch Angabe des gewünschten Namens und des Typs der Variablen.

- **Bedingte Ausführung**

Syntax	Erklärung
If (*boolscher Ausdruck***)** *Anweisungen…* **End**	Der Code zwischen *If* und *End* wird ausgeführt, wenn der eingeklammerte boolsche Ausdruck *true* zurückliefert. Mögliche Operatoren: ==, !=, >=, <=, >, <, in, is, like, matches, not.

- **Schleifen**

Syntax	Erklärung
for (*Zählvariable* **from** *Startwert* **to** *Endwert* **by** *Inkrement***)** *Anweisungen…* **End** **While (***boolscher Ausdruck***)** *Anweisungen…* **End**	Wie in anderen Sprachen üblich, wird bei der for-Schleife eine Zählvariable inkrementiert, bis ein Zielwert erreicht ist. Die while-Schleife wird hingegen so lange ausgeführt, bis der boolsche Ausdruck im Schleifenkopf *false* ist.

- **Funktionsdeklaration**

Syntax	Erklärung
Function *Funktionsname* **(***Parameterliste***) [returns** *Datentyp***]** *Anweisungen…* **End**	Deklaration einer Funktion. Die Parameter werden per Referenz übergeben und können sowohl als Ein- als auch als Ausgabeparameter fungieren. Falls die Funktion einen Wert zurückgibt, muss das optionale Schlüsselwort returns gefolgt vom Datentyp des Rückgabewerts angegeben werden. Innerhalb der Funktion wird der Rückgabewert durch das Schlüsselwort return gefolgt vom gewünschten Wert, gesetzt. Der Funktionsaufruf endet nach der return-Anweisung.

- **Ende eines Aufrufs**

Syntax	Erklärung
;	Grundsätzlich wird jede vollständige Anweisung mit einem Semikolon abgeschlossen.

- **Aufruf einer Funktion, die sich in einer Bibliothek befindet**

Syntax	Erklärung
Bibliotheksname.*Funktionsname*	Um eine Funktion, die sich in einer Bibliothek befindet, aufzurufen, muss vor dem Funktionsnamen der Name der Bibliothek angegeben werden. Die beiden Bezeichner werden dabei durch einen Punkt voneinander getrennt.

- **Das Schlüsselwort „in“**

Syntax	Erklärung
If (*Suchwert* **in** *Array***)** Anweisungen… **end**	In vielen Programmen muss häufig untersucht werden, ob ein bestimmter Wert in einem Array enthalten ist. Bei den meisten Programmiersprachen muss hierzu eine Schleife implementiert werden, die jedes Array-Element mit dem Suchwert vergleicht. In der EGL kann dieses Problem durch Verwendung des Schlüsselworts „in“ ohne Verwendung einer Schleife gelöst werden. Zusätzlich wird der Array-Index der Fundstelle in eine globale Variable geschrieben (sysVar.ArrayIndex), um an späterer Stelle auf das Array-Element zugreifen zu können.

- **Einbinden eines Packages**

Syntax	Erklärung
Import *Packagename*;	Um den Inhalt eines Packages, das Teil eines Projektes ist, in einer Quellcodedatei eines anderen Packages verfügbar zu machen, wird an den Anfang der Datei das Schlüsselwort Import, gefolgt vom Packagenamen gesetzt. Das Importieren mehrerer Packages ist durch mehrfaches Einfügen des Importstatements möglich.

Weitere Informationen finden sich in der Onlinehilfe des WDz, oder in
IBM WebSphere Developer for zSeries EGL Reference Guide, IBM Form No. SC31-6837-03

Übersicht der wichtigsten Debuggerbefehle:

F8 Resume: Das Programm wird bis zum nächsten Breakpoint ausgeführt (in einem Programm können mehrere Breakpoints gesetzt werden).

F6 Step Over: Die aktuelle Anweisung wird ausgeführt. Das Programm wird nach der Ausführung angehalten.

F5 Step Into: Falls es sich bei der aktuellen Anweisung um eine Funktion handelt, für die der Source-Code zur Verfügung steht, wird die entsprechende Source-Code Datei geöffnet. Als nächster Schritt wird die erste Anweisung innerhalb dieser Funktion angezeigt.