# EIFFEL STUDIO
# Student Guide

Rolf Bruderer

22. März 2005

# Einleitung

Du hast gerade erst angefangen Informatik zu studieren? Du hast vielleicht noch nie programmiert und verstehst nicht, was "kompilieren" bedeutet? Wenn Du die Entwicklungsumgebung von Eiffel öffnest, dann wird es Dir vor lauter farbigen Knöpfen ganz schlecht? Dann ist dieses kleine Tutorial genau das Richtige für Dich. Es soll Dir dabei helfen, Deine ersten Schritte in der *EiffelStudio* Entwicklungsumgebung zu machen und Dich darin zurecht zu finden.

Die Ausführungen in diesem Leitprogramm sind auf *ISE EiffelStudio 5.5* ausgerichtet. Für die Übungen zur Vorlesung sollte, wenn immer möglich, diese neuste Version der Entwicklungsumgebung verwendet werden. Es könnte sein, dass sonst nicht alles problemlos funktioniert. Es ist jedoch möglich, dass auf gewissen Computern hier an der ETH diese neuste Version noch nicht vollständig installiert ist. In diesem Fall kann auch *EiffelStudio 5.4* verwendet werden. Zumindest unter Windows sollte das Meiste auch mit dieser Version funktionieren.

Beim Durcharbeiten dieses Leitprogramms soll jede und jeder die *EiffelStudio* Programmierumgebung Schritt für Schritt kennen lernen. Ganz nach dem Motto "learning by doing" ist es besonders wichtig, dass Du die Beispiele und Übungen im Tutorial immer gleich selber am Computer ausprobierst. Ein Disketten-Symbol bedeutet jeweils, dass ein "Schritt für Schritt"-Beispiel zum Mitmachen folgt. Falls Du Probleme beim Durchführen eines Beispiels haben solltest, blätterst Du am Besten wieder ein bis zwei Seiten zurück oder wendest Dich im Notfall an Deinen Assistenten.

# Kapitel 1

# Installation

Dieses kurze Kapitel ist lediglich für Leute gedacht, die mit ihrem eigenen Computer arbeiten wollen. Auf den Computern welche den Studenten an der ETH zur Verfügung stehen, sollte *EiffelStudio* bereits installiert sein (zumindest unter Windows). Falls Du also sowieso schon in einem Computerraum an der ETH sitzt, kannst Du dieses Kapitel einfach überspringen.

## 1.1 Downloaden

Eine Free Edition von EiffelStudio kann unter `http://www.eiffel.com/downloads/` heruntergeladen werden. Die Free Edition enthält was Du fürs Programmieren in Eiffel benötigst. Es fehlen lediglich ein paar spezielle Funktionalitäten, die Du für die Übungen nicht brauchen wirst. Lade also *EiffelStudio* runter.

## 1.2 Installieren

Starte den heruntergeladenen Installer um Eiffel zu installieren.

**ACHTUNG:** Wenn dich das Installations-Programm fragt, ob du ***"Precompiled Libraries"*** möchtest, solltest Du unbedingt die Option ***"Base, WEL and Vision2"*** auswählen. Dies wird in den Übungen zur Vorlesung vorausgesetzt.

Ansonsten kannst Du alle anderen Standardeinstellungen der Installation übernehmen.

Falls es Probleme bei der Installation geben sollte, wende Dich einfach an Deinen Assistenten.

# Kapitel 2

# Hello Eiffel

## 2.1 Erstes Eiffel Projekt

Öffne nun *EiffelStudio*, in dem Du auf das entsprechende Icon auf Deinem Desktop klickst (oder unter Linux mit dem Befehl *estudio*):

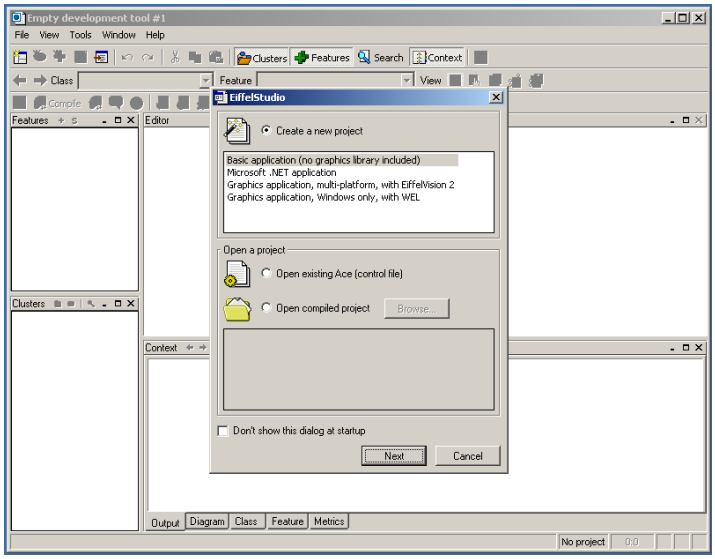


Abbildung 2.1: Herzlich Willkommen in der EiffelStudio Entwicklungsumgebung

Nach dem Aufstarten wird bereits ein Dialog zum Öffnen oder Kreieren eines Projektes angezeigt (siehe *Abbildung 2.1*). Hier hast Du die Auswahl zwischen folgenden drei Möglichkeiten:

- **Create a new project**: Öffnet einen Assistenten zum Kreieren eines neuen Projektes.
- **Open existing Ace**: Öffnet ein bestehendes Projekt.
- **Open compiled project**: Öffnet ein Projekt, welches auf diesem Computer schon ein Mal mit EiffelStudio zum Laufen gebracht wurde.

Falls der Dialog nicht erscheinen sollte oder Du aus Versehen *Cancel* gedrückt hast, so kannst Du aus dem *File*-Menü *New Project ...* auswählen um ein neues Projekt zu erstellen.

## Kick Off

In diesem Beispiel wirst Du lernen, wie Du ein neues Eiffel-Projekt erstellen kannst. Es ist wichtig, dass Du das Beispiel gleich selber Schritt für Schritt am Computer mitmachst. Achte jeweils darauf, dass Du alles genau so wie in der Anleitung beschrieben machst.

- Wir wollen also ein erstes, ganz einfaches Programm erstellen. Dazu wählst Du im eben beschriebenen Dialog die Option "*Create a new project*" und "*Basic application*". Klicke dann auf *Next.*
- Nun erscheint der Dialog zum Erstellen eines neuen Projektes (gemäss *Abbildung 2.2*).

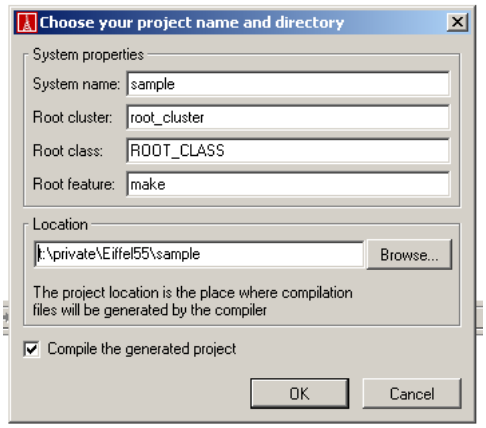


Abbildung 2.2: Dialog zum Erstellen eines neuen Eiffel-Projektes

In diesem Dialog musst Du diverse Angaben zu Deinem neuen Projekt machen. Sie werden im folgenden Schritt für Schritt erklärt.

- **System name:** Hier bestimmst Du, wie Dein Programm heissen soll. Tippe zum Beispiel *mein_erstes_programm.*
  **Achtung**: Für Namen in Eiffel gelten bestimmte Regeln, an die Du Dich halten musst. Ein Name darf keine Sonderzeichen wie Umlaute oder Leerzeichen enthalten. Es sind nur alphabetische Buchstaben (also *a* bis *z*), Ziffern (also *0* bis *9*) und Unterstriche (also _) erlaubt. Das erste Zeichen darf ausserdem kein Unterstrich und keine Zahl sein. Gross-Kleinschreibung spielt keine Rolle. Allerdings schreibt man den Namen eines Programmes üblicherweise nur mit Kleinbuchstaben.
- **Root cluster:** Programme sind in Eiffel in sogenannte *Clusters* unterteilt. Ein Programm besteht also aus mehreren Clustern die lediglich dazu da sind, dass Du in einem grossen Projekt nicht die Übersicht verlierst. Also so ähnlich wie Schubladen in einem Schrank oder Ordner in einem Dateisystem. Hier kannst Du den Namen für das Haupt-cluster angeben. Dein neues Projekt wird am Anfang nur aus diesem einen Cluster bestehen. Du kannst später weitere Cluster zu Deinem Projekt hinzufügen. Darauf werden wir in einem anderen Kapitel noch genauer eingehen.
  Gib also nun einfach einen Namen für das Haupt-cluster Deines Projektes ein, z.B. *haupt-cluster.*

- **Root class:** In einem Cluster befinden sich schliesslich mehrere *Klassen*. Das hat nichts mit Schulklassen zu tun. Klassen sind Beschreibungen für Objekte, aus welchen objekt-orientierte Programme schliesslich aufgebaut werden. Ein Programm muss mindestens aus einer Klasse bestehen. Den Namen für diese Hauptklasse kannst Du hier auswählen. Gib zum Beispiel *MEINE_HAUPTKLASSE* als Klassennamen für Deine erste eigene Klasse ein. Beachte dabei, dass Klassennamen in Eiffel üblicherweise nur aus Grossbuchstaben geschrieben werden.

- **Root feature:** Eine Klasse wiederum besteht aus einem oder mehreren sogenannten *Features*. Ein Feature beschreibt eine gewisse Eigenschaft oder Fähigkeit einer Klasse. Man kann auch sagen, ein Feature ist die Beschreibung eines Befehls, also einfach ein Stück Programmcode. Falls Du das im Moment nicht genau verstehst, braucht Dich das noch nicht weiter zu kümmern. Mit der Zeit wird Dir das dann klar werden.
  Das *Root-feature* ist jener Programmteil, welcher beim Starten des Programmes ausgeführt wird. Von diesem Feature aus können dann weitere Features respektive Befehle aufgerufen werden.
  Wähle nun also einfach einen Namen für Dein Hauptfeature, z.B. *start*.
  Der Name des Features soll beschreiben, was das Feature genau tut. Ausserdem werden Featurenamen wiederum nur aus Kleinbuchstaben gebildet. So kann man Features gut von Klassen unterscheiden.

- **Location:** Jetzt geht es noch darum zu entscheiden, in welchem Verzeichnis auf Deinem Computer das Projekt erstellt werden soll. Wähle also einfach einen sinnvollen Ordner auf Deinem Computer aus. Am Besten lässt Du für Dein Projekt gleich ein neues Verzeichnis erstellen.
  An den Studentenrechnern in den ETH Computerräumen kannst Du Deine Projekte jeweils auf das T-Laufwerk speichern. Hier befinden sich all Deine persönlichen Dateien. Wähle also zum Beispiel *T:\eiffel\mein_erstes_programm* als Projektverzeichnis.

- Nun hast Du alle Angaben gemacht, die es für Dein erstes Projekt braucht. Klicke danach einfach auf *OK* und EiffelStudio wird für Dich Dein erstes Projekt gemäss Deinen Angaben vorbereiten. Diese Vorbereitung kann unter Umständen und je nach Geschwindigkeit des Computers recht lange dauern. Hab also etwas Geduld. Du kannst in dieser Zeit zum Beispiel auch einen Kaffee trinken gehen.

- Wenn der Vorgang zu Ende ist, sollte Dein Bildschirm ungefähr so wie in *Abbildung 2.3* aussehen.

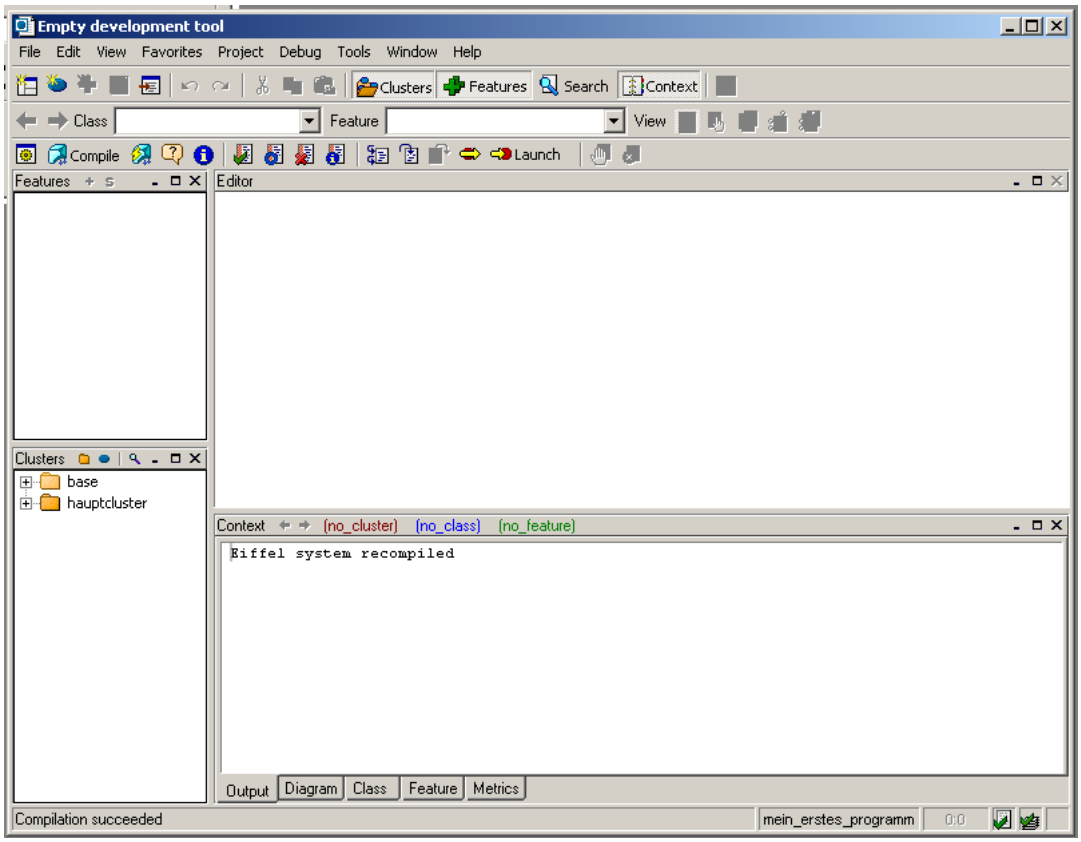


Abbildung 2.3: Entwicklungsumgebung mit erstem eigenem Eiffel-Projekt

Gratulation, Du hast so eben Dein erstes Eiffel-Projekt erstellt.

Eiffel hat nun also für Dich ein Grundgerüst für ein Programm erstellt. Dieses Programm macht momentan noch überhaupt nichts, wenn man es ausführen würde. Bevor wir uns aber damit beschäftigen, wie Du Deinem ersten Programm etwas Leben einhauchen kannst, wollen wir kurz repetieren, was Du in diesem Beispiel schon alles gelernt hast:

- **Namen und andere Bezeichnungen** in Eiffel dürfen nur aus alphabetischen Buchstaben (*a* bis *z* und *A* bis *Z*), Ziffern (*0* bis *9*) und Unterstrichen (_) bestehen. Es dürfen also keine Umlaute, Leerzeichen oder sonstige Sonderzeichen darin vorkommen. Ausserdem muss das erste Zeichen unbedingt ein alphabetischer Buchstabe sein.
- **Der Name eines Eiffel-Projektes** wird üblicherweise in Kleinbuchstaben geschrieben.
- Ein Eiffel-Projekt wird zur besseren Übersichtlichkeit in mehrere **Clusters** eingeteilt. Auch Clusters werden mit Kleinbuchstaben benannt.
- Ein Programm wird durch eine oder mehrere **Klassen** beschrieben. Klassen-Namen werden grundsätzlich in Grossbuchstaben geschrieben.
- Eine Klasse enthält schliesslich sogenannte **Features.** Dies sind die Fähigkeiten oder Eigenschaften einer Klasse. Features werden wiederum mit Kleinbuchstaben benannt.
- Du weisst nun, wie man in Eiffel ein **neues Projekt kreieren** kann.

## 2.2 Die Oberfläche

Nun wollen wir uns die Eiffel-Entwicklungsoberfläche mal etwas genauer unter die Lupe nehmen, damit Du nachher ungefähr weisst, was wo ist. Grundsätzlich ist das Fenster in vier Teile unterteilt, welche wir uns im Folgenden kurz anschauen wollen.

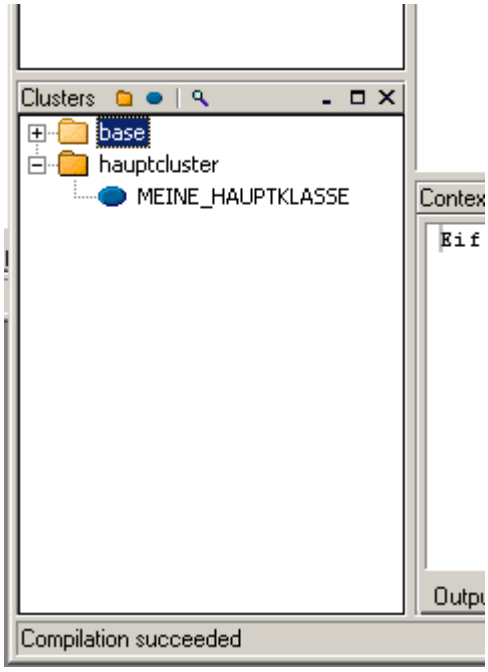


Abbildung 2.4: Clusters

Ganz links unten siehst Du eine Verzeichnisstruktur. Dies sind die Clusters (siehe *Abbildung 2.4*). Wie gesagt, ein Projekt wird in Clusters unterteilt. Ein Cluster kann weitere Unter-cluster haben. Du kannst durch die Clusters navigieren wie durch die Verzeichnisse Deines Betriebssystem. In den Clustern befinden sich die Klassen aus welchen Dein Projekt besteht. Wenn Du eine Klasse anklickst, so wird sie im Editor angezeigt.


Abbildung 2.5: Editor

Rechts oben befindet sich der Editor. Hier bearbeitest Du den Programmtext, auch Code genannt. In der *Abbildung 2.5* siehst Du den Editor mit dem Programmtext Deiner Hauptklasse, wie sie von Eiffel erstellt wurde. Bis jetzt besteht die Klasse *MEINE_HAUPTKLASSE* lediglich aus einem Feature *start*.

## Eine weitere Fähigkeit

Im Editor kannst Du also Deine Klassen bearbeiten. Das heisst, Du kannst definieren was die Features respektive Fähigkeiten einer Klasse sein sollen. Nun werden wir der Klasse *MEINE_HAUPTKLASSE* ein neues Feature hinzufügen.

- Öffne die Klasse *MEINE_HAUPTKLASSE* im Editor. Du musst dazu unten links in den Clusters zuerst das Cluster *hauptcluster* aufmachen, indem Du es anklickst. Danach kannst Du auf *MEINE_HAUPTKLASSE* klicken. Der Programmtext der Klasse sollte nun im Editorfenster erscheinen (wie in *Abbildung 2.5*)
- Füge nun den folgenden Programmtext vor dem letzten *end* ein.

```
sag_hallo is
    do
        io.put_string ("Hallo Eiffel!")
        io.new_line
    end
```

Achte darauf, dass Du den Text bis ins letzte Detail genau so wie abgebildet abschreibst. Alle Klammern, Punkte und Anführungszeichen müssen genau stimmen. Wenn Du nur ein Zeichen falsch abtippst, wird Eiffel Dein Programm unter Umständen nicht verstehen können. Du solltest Dir auch gleich angewöhnen, die Einrückungen und Zeilenumbrüche gleich wie in diesen Beispielen zu machen. Für Einrückungen benutzt man übrigens die Tabulator-Taste (*Tab*).

- Nun sollte der ganze Programmtext Deiner Klasse wie folgt aussehen:

```
indexing
    description: "System's root class"
    note: "Initial version automatically generated"

class
    MEINE_HAUPTKLASSE

create
    start

feature -- Initialization

    start is
            -- Creation procedure.
        do
            --| Add your code here
        end

    sag_hallo is
        do
            io.put_string ("Hallo Eiffel!")
            io.new_line
        end

end -- class MEINE_HAUPTKLASSE
```

Das letzte *end* markiert das Ende einer Klasse. Alle Features müssen zwischen dem Wort *feature* und diesem letzten *end*, welches die Klasse abschliesst, stehen.

- Drücke nun auf den Knopf *Compile* ganz oben in der Entwicklungsumgebung. Der Knopf sieht wie folgt aus: 

Dies bewirkt, dass Eiffel Deine Änderungen überprüft und übernimmt. Falls nach dem Drücken des Knopfes eine Fehlermeldung im Fenster unter dem Editor-Fenster erscheinen sollte, hast Du wahrscheinlich etwas falsch abgetippt. Eiffel kann Deinen Programmtext nicht verstehen. Überprüfe in diesem Fall ob Du alles genau so geschrieben hast, wie oben abgedruckt und korrigiere es gegebenenfalls. Drücke den *Compile* Knopf dann erneut.

Wenn keine Fehlermeldung erscheint hast Du das neue Feature erfolgreich hinzugefügt und es sollte nun bei den Features links vom Editor aufgelistet werden (siehe *Abbildung 2.6*).

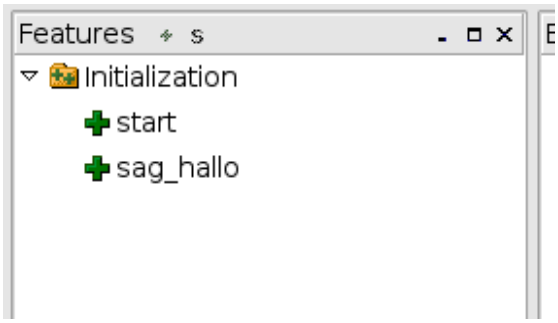


Abbildung 2.6: Features

Links vom Editor befindet sich die Auflistung aller Features der aktuellen Klasse (siehe *Abbildung 2.6*). Falls nicht alle Features einer Klasse aufgelistet werden, musst Du vermutlich nochmals den *Compile* Knopf drücken, damit EiffelStudio alle Features übernimmt. Durch anklicken eines Features wird im Editor automatisch an die Stelle gesprungen, wo das Feature definiert ist.

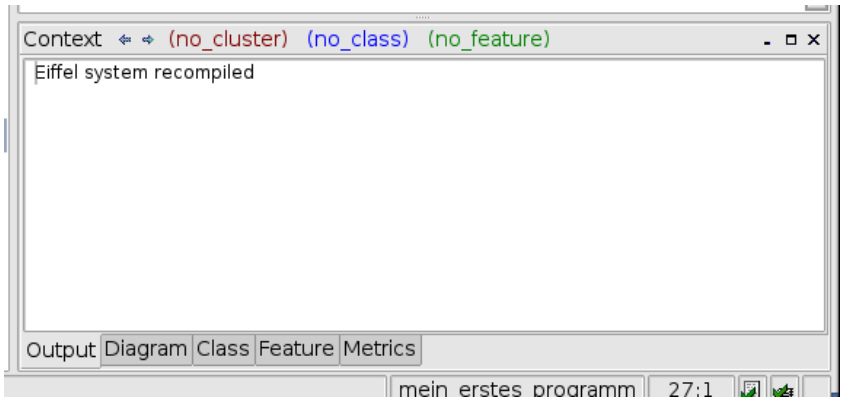


Abbildung 2.7: Context

Unten rechts schliesslich befindet sich das Context Fenster (siehe *Abbildung 2.7*). Dieser Teil enthält viele verschiedene Zusatz-Tools von Eiffel Studio, das erkennt man nur schon an der Unterteilung am unteren Rand in *Output*, *Diagram*, *Class*, *Feature* und *Metrics*. Aber das ist ein anderes Kapitel. Im Moment reicht es, wenn Du weisst, dass EiffelStudio Dich hier bei *Output* informiert, falls es nach dem Drücken des *Compile* Knopfes allfällige Fehler in Deinem Programm gefunden hat. Wenn das Programm okay ist, sollte hier der Text “Eiffel system recompiled” angezeigt werden.

## 2.3 Programm zum Laufen bringen

So, nun wollen wir aber endlich unser erstes Programm zum Laufen bringen. Um ein Programm laufen zu lassen, sind zwei wesentliche Schritte nötig:

1. **Kompilieren:** Damit ein Programm auf einem Computer ausgeführt werden kann, muss es zuerst übersetzt werden, so dass es der Computer versteht. Diesen Vorgang nennt man Kompilieren (oder auf Englisch eben *Compile*). Um Dein Eiffel-Programm in die Maschinensprache Deines Computers zu

übersetzen, musst Du auf den Compile Knopf (siehe *Abbildung 2.8*) drücken, bevor Du das Programm ausführen kannst.


Abbildung 2.8: Compile und Launch Knopf

2. **Starten:** Wenn das Programm erfolgreich kompiliert werden konnte, wenn es also keine Fehlermeldung gab, so kannst Du das Programm durch Drücken auf den *Launch* Knopf starten (siehe *Abbildung 2.8*).

Nach jeder Änderung Deines Programmes musst Du es wieder neu kompilieren, bevor Du es ausführen kannst. Ansonsten haben die Änderungen im Programmtext keinen Einfluss.

### Jetzt geht's los!

Da Dein Hauptfeature *start* bis jetzt noch überhaupt nichts macht, passiert im Moment auch noch nichts, wenn Du das Programm ausführst. Wir werden deshalb in diesem Beispiel etwas Leben in unser Programm bringen.

- Ergänze den Programmtext des Features *start* so, dass das Feature *sag_hallo* aufgerufen wird. Das Feature *start* muss also wie folgt abgeändert werden:

```
start is
        -- Creation procedure.
    do
        sag_hallo
    end
```

- Drücke auf den *Compile* Knopf um das Programm mit den Änderungen neu zu kompilieren. Falls es Fehlermeldungen im Context Fenster geben sollte, hast Du wahrscheinlich etwas falsch geschrieben, dann musst Du die Fehler korrigieren und erneut kompilieren.
- Wenn das Programm fehlerfrei kompiliert wurde, erscheint der Text "Eiffel system recompiled" im Context Fenster. Dann kannst Du den *Launch* Knopf drücken um das Programm auszuführen.

Das Programm sollte den Text *Hallo Eiffel!* auf einer Konsole ausgeben.

## 2.4 Compilerfehler

Ein *Compilerfehler* ist nicht etwa ein Fehler im Eiffel-Compiler, sondern ein Fehler in Deinem Programm. Wenn der Compiler Dein Programm nicht versteht und somit nicht übersetzen kann, dann meldet er Dir einen sogenannten Compilerfehler. Man könnte in Analogie zu natürlichen Sprachen im Prinzip davon sprechen, dass jeder Compilerfehler entweder einem Grammatikfehler oder einem Rechtschreibefehler in Deinem Programm entspricht. Ein möglicher Grammatikfehler wäre ein vergessenes Satzzeichen (zum Beispiel eine schliessende

Klammer) oder eine falsche Satzstellung (zum Beispiel ein Feature nach dem letzten *end*, welches die Klasse beendet). Ein Rechtschreibefehler entspräche dann zum Beispiel einem falsch geschriebenen und somit dem Compiler unbekannten Befehlsnamen.

Weil ein Computer ein nicht besonders intelligentes Wesen ist, reicht bereits ein vergessenes Zeichen, damit der Compiler ein Programm nicht mehr versteht. Hier hinkt also der Vergleich mit der menschlichen Sprache, wo ein fehlendes Satzzeichen oder ein kleiner Rechtschreibefehler nur selten zur Unverständlichkeit eines Textes führt.

Immerhin ist der Compiler so nett, uns bei einem Compilerfehler mitzuteilen, wo er den Code nicht mehr verstehen konnte. Leider bedeutet das nicht unbedingt, dass die Ursache für den Fehler genau an der angegebenen Stelle zu finden ist. Es kann durchaus auch sein, dass der eigentliche Fehler im Code an einer Stelle vor der angegebenen Fehlerposition ist. So muss zum Beispiel ein vergessenes Zeichen nicht unmittelbar zur Unverständlichkeit des Codes an der entsprechenden Stelle führen. Es kann auch sein, dass dadurch der Code für den Compiler erst einiges später keinen Sinn mehr ergibt.

## Wir basteln einen Compilerfehler

- Zuerst verunstalten wir unser Programm ein bisschen in dem wir das schliessende *end* am Ende des Features *sag_hallo* löschen. Ausserdem entfernen wir den Unterstrich beim Aufruf des Features *sag_hallo* im Feature *start*. Der Code sollte dann wie folgt aussehen:

```
indexing
    description: "System's root class"
    note: "Initial version automatically generated"

class
    MEINE_HAUPTKLASSE

create
    start

feature -- Initialization

    start is
            -- Creation procedure.
        do
            saghallo
        end

    sag_hallo is
        do
            io.put_string ("Hallo Eiffel!")
            io.new_line


end -- class MEINE_HAUPTKLASSE
```

- Kompiliere dieses Programm. Dies sollte eine Fehlermeldung im Context Fenster ergeben: “Syntax error ...”. Ausserdem wird der Cursor automatisch an der Stelle im Programmtext platziert, wo EiffelStudio den Fehler vermutet.

- In unserem Fall ist der Fehler eine Zeile vor der markierten Zeile, dort fehlt nämlich ein schliessendes *end* für das Feature *sag_ hallo*. Korrigiere diesen Fehler, in dem Du dort wieder ein *end* hinschreibst. Versuche das Programm erneut zu kompilieren.
- Nun gibt es einen anderen Compilerfehler im Context Fenster: "Error: unknown identifier". Das bedeutet, dass der Compiler auf einen Namen respektive einen Bezeichner (eben einen "identifier") gestossen ist, den er nicht kennt. Dieses Mal springt der Compiler nicht automatisch in die Zeile wo der Fehler vermutet wird. Stattdessen wird im Context Fenster eine ausführliche Fehlermeldung angezeigt (siehe *Abbildung 2.9*). In der Fehlermeldung enthalten ist auch der Name des Features, in welchem der Fehler aufgetreten ist (grün markierter Featurename *start*).

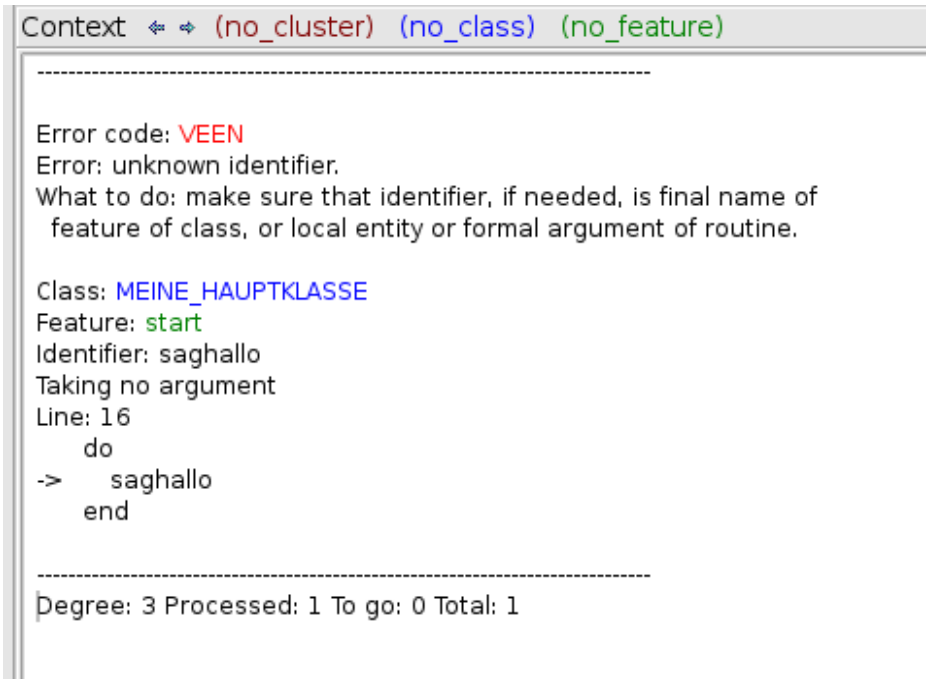


Abbildung 2.9: Ein Kompilerfehler

- Klicke mit der rechten Maustaste im Context Fenster auf den grün markierten Featurenamen *start*. Ziehe nun die Maus in das Editorfenster und klicke nochmals auf die rechte Maustaste. Es wird nun die Stelle im Editor angezeigt und markiert, wo der Fehler vermutet wird. Es handelt sich offensichtlich um einen Tippfehler, es gibt kein Feature *saghallo*, nur ein Feature *sag_ hallo*.
- Korrigiere also den Fehler und füge den fehlenden Unterstrich wieder ein. Nun sollte das Programm wieder fehlerfrei kompilieren.

## 2.5 Compiler-Troubleshooting

EiffelStudio hat manchmal so seine liebe Mühe beim Kompilieren eines Programmes. Es kann zum Beispiel vorkommen, dass plötzlich absolut unerklärliche Fehlermeldungen auftreten. In diesem Fall kannst Du folgendes tun:

1. Drücke einfach erneut auf den *Compile* Knopf. Der Compiler braucht manchmal einfach einen zweiten Anlauf, dann geht es plötzlich.

2. Falls das immer noch nichts nützen sollte und Du sicher bist, dass Dein Programm kompilierbar sein sollte, hilft es manchmal EiffelStudio zu beenden und nochmals neu zu starten. Dein schon mal kompiliertes Projekt kannst Du durch öffnen der entsprechenden Datei mit der Endung *.epr* erneut laden. Wähle dazu *Open Project ...* aus dem Menü *File*. Anschliessend nochmals den *Compile* Knopf drücken.

3. Wenn das immer noch nichts hilft, ist Dein Projekt eventuell korrupt. Es hilft dann unter Umständen wenn Du das Projekt nochmals ganz frisch von EiffelStudio erstellen und neu durchkompilieren lässt. Dazu gibt es zu jedem Projekt eine Datei mit der Endung *.ace*. Starte Eiffel nochmals neu. Wähle nun *New Project ...* aus dem Menü *File* und dann *Open existing Ace*. Öffne die passende Ace-Datei zu Deinem Projekt. Es erscheint noch ein Dialog, wo du entscheiden kannst, wo das Projekt neu erstellt werden soll. Normalerweise musst Du dort nichts ändern. Du kannst einfach Dein altes Projekt überschreiben lassen. Achte darauf, dass die Checkbox "*Compile the generated project*" ausgewählt ist. Danach wird *EiffelStudio* Dich noch fragen, ob Du das bestehende Projekt wirklich überschreiben willst. Diese Frage kannst Du getrost mit *Yes* beantworten, da Dein Programmtext nicht verloren gehen wird. Lediglich das lauffähige Programm wird völlig neu erzeugt, was natürlich wieder eine Weile dauern kann. Aber danach sollte hoffentlich alles wieder tip-top funktionieren.

Ausserdem ist noch zu beachten, dass der Kompiliervorgang von Eiffel aus zwei Teilen besteht:

1. **Normale Compilation:** Du siehst eine *Compilation Progress* Anzeige in der ein blauer Balken den Fortschritt der Compilation anzeigt. Diese Übersetzung ist bei jedem Kompiliervorgang nötig und verläuft normalerweise unproblematisch (ausser natürlich wenn es Compilerfehler in Deinem Programmtext hat).

2. **C Compilation:** Manchmal startet nach der normalen Kompilation noch ein weiterer Compilations-Vorgang. Diese *C Compilation* ist nötig, wenn ein neues Projekt kompiliert wird oder wenn am Projekt besonders starke Änderungen vorgenommen wurden. In diesem Fall erscheint nach der normalen Compilation der Text "Background C compilation launched" im Context Fenster. Diese Compilation läuft im Hintergrund und es öffnet sich eine Konsole, in welcher die Ausgabe des Compilers angezeigt wird (siehe Abbildung 2.10). Es kann vorkommen, dass es bei diesem Teil der Compilation zu einem Fehler kommt, obwohl die normale Compilation erfolgreich war. Dies ist kein Fehler in Deinem Programm, sondern ein Problem des Systems.

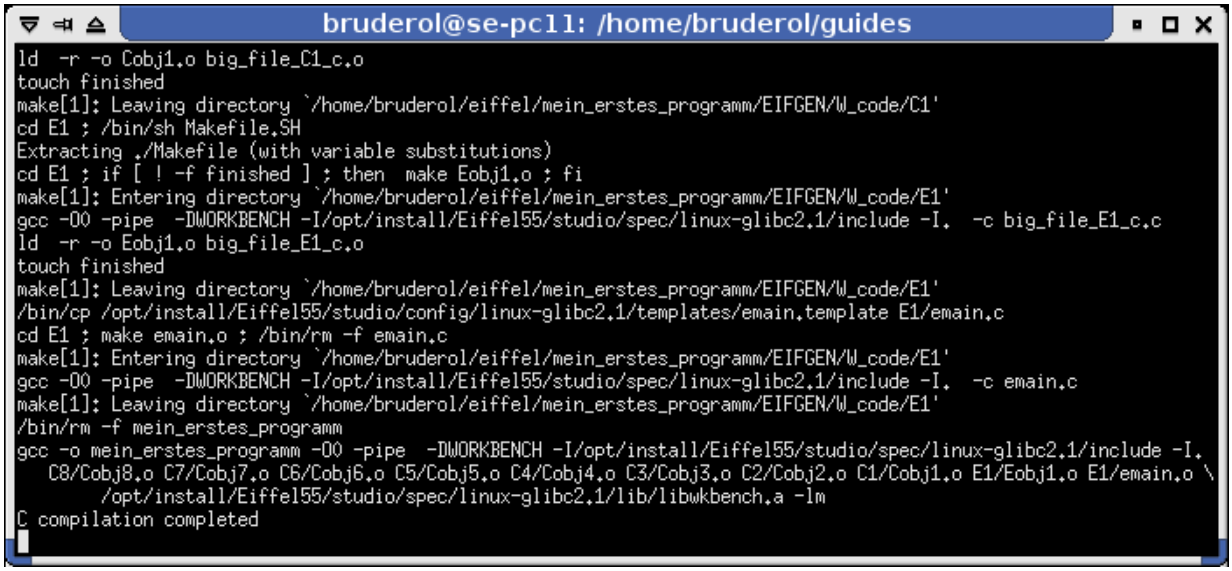


Abbildung 2.10: C Compilation Ausgabe in Konsole

Falls bei der C Compilation ein Fehler auftritt, obwohl es bei der normalen Compilation keinen Fehler gab, so kann man folgendes tun:

1. Manchmal braucht die C Compilation einfach einen zweiten Anlauf und plötzlich geht es. Dazu gibt es im Menü *Project* einen Menüpunkt *Compile Workbench C Code.* Starte mit diesem Befehl die C Compilation erneut und habe etwas Geduld.
2. Falls das nichts bringt, musst Du die selben Schritte probieren, wie am Anfang dieses Kapitels beschrieben (EiffelStudio neu starten, Projekt neu von Ace-File erstellen).

Falls all diese Tipps nichts nützen sollten, wendest Du Dich bei Problemen am Besten an Deinen Assistenten.

## 2.6 Laufzeitfehler

Leider heisst es noch lange nicht, dass ein Programm richtig und fehlerfrei funktioniert, wenn es ein Mal fehlerfrei compiliert. Der Compiler konnte das Programm zwar in maschinen-verständliche Sprache übersetzen, aber ob das Programm nun auch fehlerfrei läuft ist natürlich nicht sicher. So ist der Satz “Spring aus dem Fenster!” zwar ein korrekter deutscher Satz, den jeder deutschsprachige Mensch versteht. Es gibt aber wohl nur sehr wenige Leute, welche diesem Aufruf ohne Widerrede Folge leisten würden. Oder anders gesagt, jene welche diesen Befehl ausführen, springen geradewegs ins Verderben, respektive stürzen ab. Ganz ähnlich kann ein Programm zwar compilieren, aber bei der Ausführung stürzt es trotzdem ab. Grundsätzlich können wir dabei drei Arten von Fehlverhalten einer Software unterscheiden:

- **Abbruch**: Das Programm bricht plötzlich ab und es erscheint eine Fehlermeldung. Dies nennt man einen Laufzeitfehler. In EiffelStudio erscheint in diesem Fall ein Fenster, welches einem den Ort im Programmtext anzeigt, an welchem der Fehler aufgetreten ist. Mehr dazu später.
- **Endlosschleife**: Das Programm läuft in eine Endlosschleife und reagiert nicht mehr. In diesem Falle kann in EiffelStudio das Programm durch Klicken auf den Pause-Knopf (siehe *Abbildung 2.11*) angehalten werden, um heraus zu finden, wo das Programm hängen bleibt.

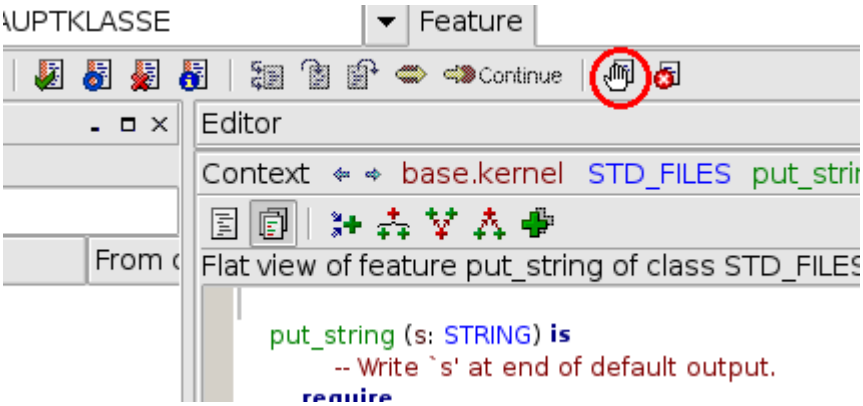


Abbildung 2.11: Der Pause-Knopf zum Anhalten einer Applikation

- **Anderes Fehlverhalten:** Die Software läuft zwar ohne Abbrüche oder Abstürze, sie macht aber irgendwie nicht das, was sie eigentlich sollte. Hier liegt das Problem in der eigentlichen Bedeutung des Programmes und man muss nochmals über die Bücher oder eben hinter den Programmcode.

Es folgt ein kleines Beispiel, damit Du verstehst, wie man heraus finden kann, wo im Programmtext der Laufzeitfehler aufgetreten ist.

## Wo ist denn der Böög?

- Basteln wir uns also mal ein Programm welches einen Laufzeitfehler produziert. Verändere dazu die Klasse *MEINE_HAUPTKLASSE* wie folgt:

```
indexing
    description: "System's root class"
    note: "Initial version automatically generated"

class
    MEINE_HAUPTKLASSE

create
    start

feature -- Initialization

    start is
            -- Creation procedure.
        local
            text: STRING
        do
            sag_hallo
            sag_hallo
            io.put_string (text)
        end

    sag_hallo is
            -- Schreibe Begruessung auf Bildschirm.
        do
            io.put_string ("Hallo Eiffel!")
            io.new_line
        end

end -- class MEINE_HAUPTKLASSE
```

- Kompiliere das Programm und lasse es laufen. Es sollte ein Laufzeitfehler auftreten.

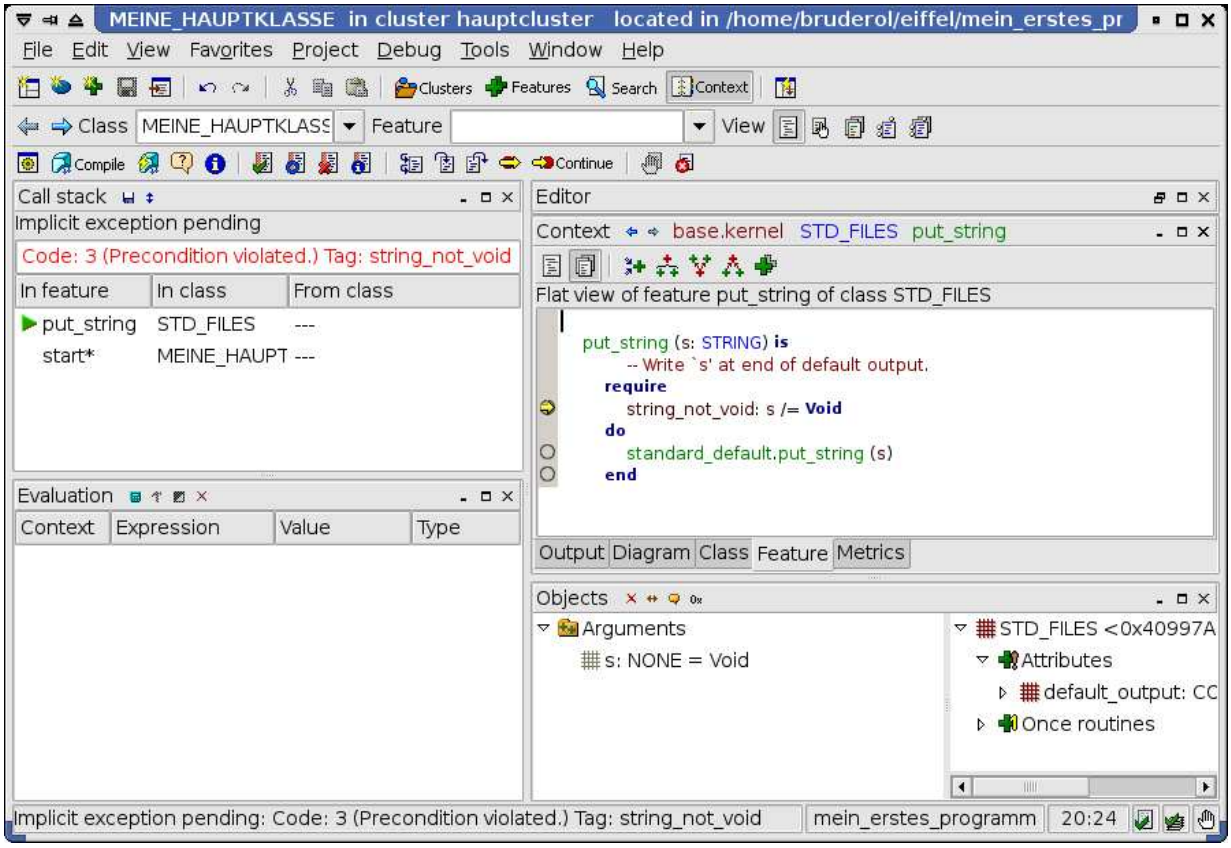


Abbildung 2.12: Das Debugger Fenster mit einem Laufzeitfehler

- Im sich öffnenden Debugger Fenster (siehe *Abbildung 2.12*) wird angezeigt, wo der Fehler aufgetreten ist. Der Fehler passierte anscheinend beim Aufruf des Features *put_string*. Die Zeile wo der Fehler passiert ist wird mit einem gelben Pfeil markiert. In unserem Fall handelt es sich um eine Precondition-Violation. Das bedeutet, dass das Feature *put_string* auf unerlaubte Art und Weise aufgerufen wurde. Der Fehler liegt also gar nicht im Feature *put_string*, sondern dort wo das Feature aufgerufen wurde. Im *Call Stack* auf der linken Seite kann man sehen, welches Feature von wo aus aufgerufen wurde. Das oberste Feature ist das Feature, wo der Fehler aufgetreten ist. Ein Feature wurde jeweils vom Feature unten dran im Call Stack aufgerufen. In unserem Fall wurde das Feature *put_string* also direkt aus dem Feature *start* aufgerufen. Klicke also auf das Feature *start* im Call Stack.

- Der Fehler liegt hier im Feature *start*. Die lokale Variable *text* wurde gar nie initialisiert (das heisst sie hat keinen Wert), bevor das Feature *put_string* aufgerufen wurde. Wir müssen das Feature *start* deshalb entsprechend ändern und die Variable *text* zuerst initialisieren, bevor sie an das Feature *put_string* übergeben wird.

- Um wieder in den Editor zu gelangen, musst Du das laufende Programm abbrechen. Dazu klickst Du auf den Abbruch-Knopf (siehe Abbildung 2.13).

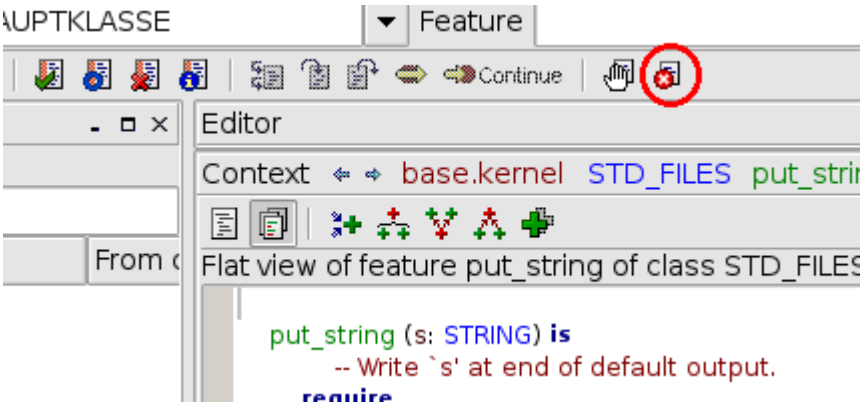


Abbildung 2.13: Der Abbruch Knopf

- Passe das Feature *start* also wie folgt an, damit die lokale Variable *text* einen Wert hat, der ausgegeben werden kann:

```
start is
        -- Creation procedure.
    local
        text: STRING
    do
        sag_hallo
        sag_hallo
        text := "Good bye!"
        io.put_string (text)
    end
```

- Kompiliere erneut und lasse das Programm laufen.

Jetzt sollte alles funktionieren und auf dem Bildschirm zwei mal “Hallo Eiffel!” und einmal “Good bye!” erscheinen.