Softwarepraktikum: Enigma

Martin Steffen

Sommersemester 2003

Abschnitt I

Softwareentwurf

Bereiche der Softwareentwicklung

- eigentliche Softwareentwicklung
- Projektmanagement
- Konfigurationsmanagement
- Qualitätssicherung
- ...

Hier: i.d.H. Softwareentwicklung

Projektphasen

1. Analyse
2. Entwurf
3. Implementierung
4. Test
5. Inbetriebnahme
6. Wartung

Hilfsmittel für die SE

- Methodologien und Verfahrensweisen
- Formalismen zur Beschreibung der Software, "Design-Syntax", Visualisierungen
  - Klassendiagramme
  - Sequenzdiagramme
  - ...
- Entwicklungsumgebungen (IDE)
  - emacs + jde
  - together
  - ...

Hier: Klassendiagramme in UML und together (später).

Abschnitt II

Hinweise zum Entwurf

Klassen

- bilden eigenen, abgeschlossenen Verantwortlichkeitsbereich
- beschreiben Objekte (u.U. der realen Welt) schematisch
- fassen mehrfach und gemeinsam auftretende Daten zusammen
- können andere erweitern oder spezialisieren/generalisieren (Vererbung)
- stehen miteinander in Beziehung (Assoziationen)

Klassen

Wichtig:

- suggestive Bezeichnungen
- eine feste natürliche Sprache (z.B. durchgängig Deutsch)
- Namen und Kodekonventionen einhalten: Klassennamen beginnen mit Großbuchstaben, Wortgliederung durch Großbuchstaben ...

Klassen: Checkliste

- Werden mehrere Objekte einer Klasse erzeugt?
- Fassen Objekte Daten und Operationen zusammen?
- Gibt es sinnvolle Beziehungen zwischen den Klassen?
- alle Methoden notwendig?
- Sinnvolle Abstraktionen?
- Balance zwischen Allgemeinheit, Erweiterbarkeit etc. vs. kronkreter Umsetzung, Effizients ...
- Schnittstellen "zu öffentlich"?

Klassifizierung von Klassen

- *entity:* Sachverhalt oder Gegenstand
- *control:* Ablauf, Steuerungs- oder Berechnungsvorgang
- *abstract:* ohne Instanzen
- *interface:* abstrakte Schnittstelle, zur logischen Strukturierung
- *boundary:* konkrete Schnittstelle, durch Objekt(e) realisiert
- *primitive:* Standardklassen, Klassen kleiner Objekte
- *enumeration:* Aufzählungen, Wertemengen
- *structure*, *type*, ...

Assoziationen

- Charakteristika
  - Bedeutung
  - Anzahlen der in Relation stehenden Elemente
  - Richtung der Assoziation
- Spezialformen
  - Aggregation: „ist Bestandteil von“
  - Komposition: „ist abhängiger Bestandteil von“

Attribute

- Eigenschaften/Zustand von Objekten
- Wichtig:
  - Kapseln: nicht-statische Attribute private
  - Methoden statt abgeleiteter Attribute verwenden
  - Namenskonventionen einhalten
    - `final-static`-Attribute vollständig in Großbuchstaben, Wortgliederung durch Unterstrich (z.B. `ANZAHL_WALZEN`)
    - alle anderen beginnen mit einem Kleinbuchstaben, Wortgliederung durch Großbuchstaben (z.B. `laengeKlartext`)

Methoden

Operationen auf Objekten.

- geeignete Abstraktionen bilden (interfaces, Sichtbarkeitsstufen)
- get- und set-Methoden kritisch durchsehen
- Namenskonventionen: Imperativ, beginnend mit Kleinbuchstaben, Wortgliederung durch Großbuchstaben (z.B. `tuDiesUndJenes()`)

Allgemeine Bemerkungen

- auf Lesbarkeit achten
- angemessene Dokumentation, `javadoc`
- auf Wiederverwendbarkeit achten
- durch Pakete strukturieren, Ausgangspunkt: Gruppe `x` $\longrightarrow$ Paket `enigma<x>`
- `Makefiles` bereitstellen

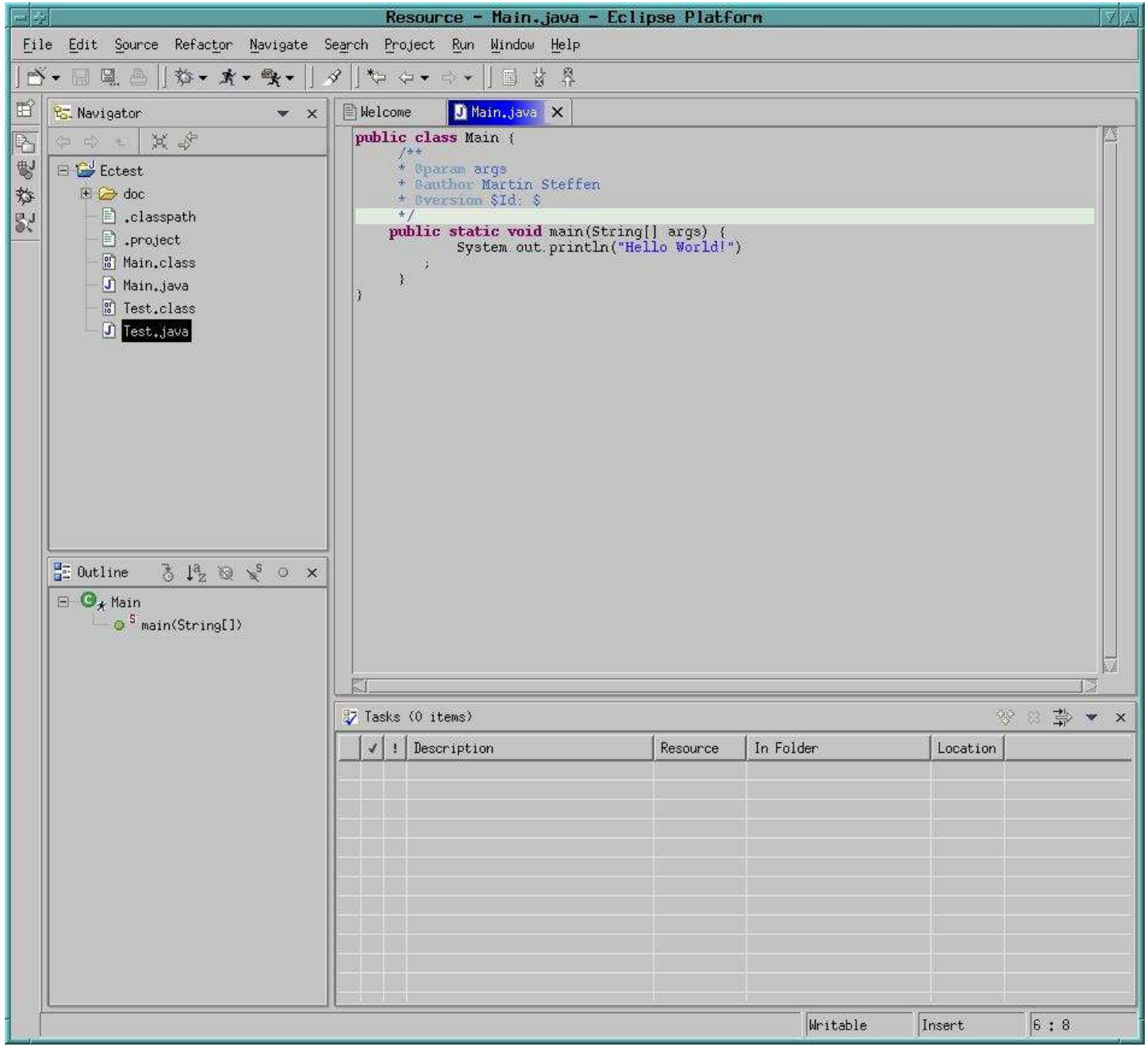
Entwicklungsumgebungen

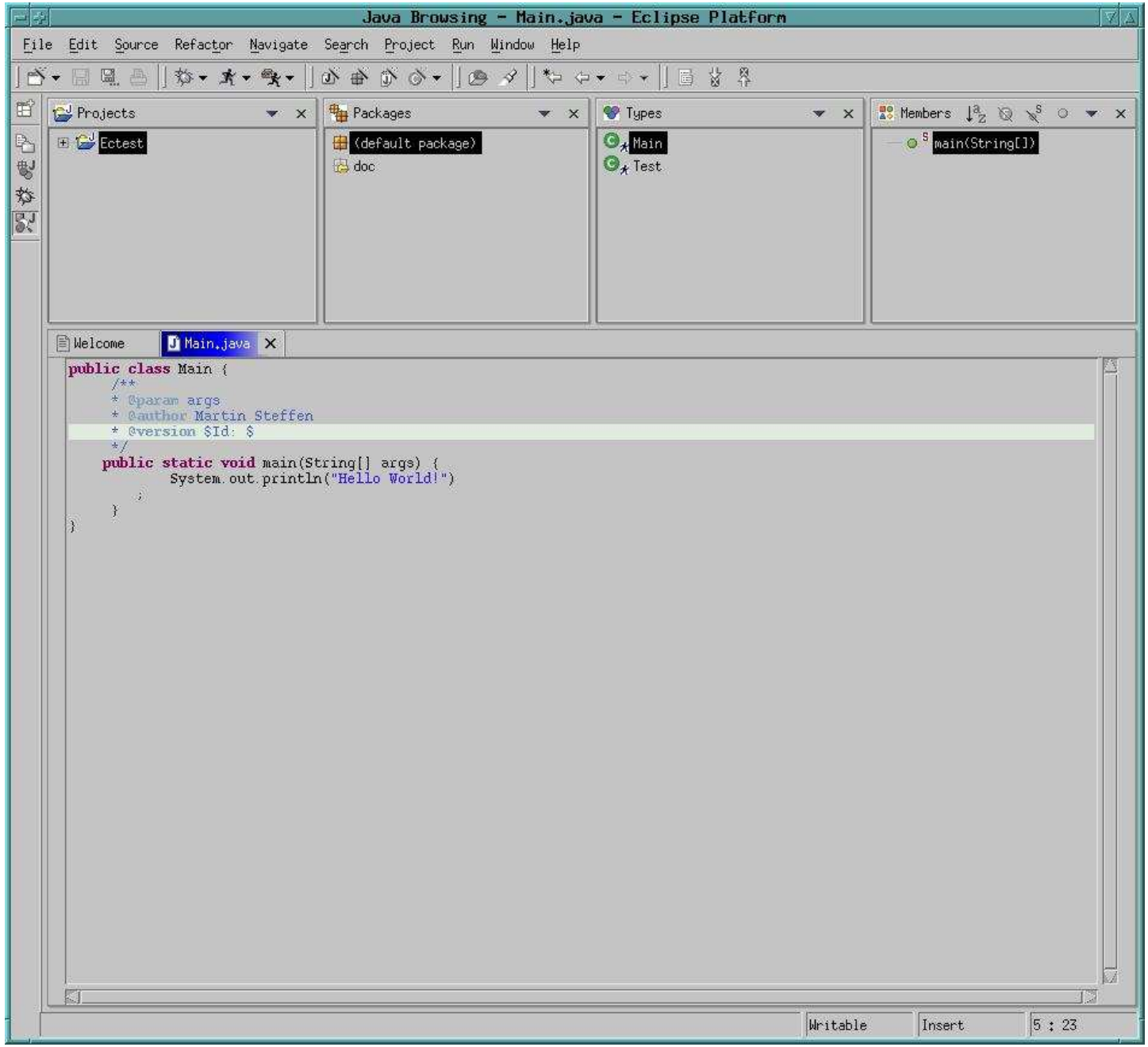
- was können sie:
  - navigieren, visualisieren, Syntaxhighlight ⇒: Spezialeditor
  - auf Fehler hinweisen
  - kennen sich in der Java-Bibliothek aus, kennen "Standardsituationen"[1]
  - stub-code, Tests generieren, Debuggen
  - allgemein: den "Tippen-Speichern-Syntaxfehlersuchen-ändern-" ... Zyklus verwalten
  - Projekte "organisieren"
  - unterstutzen eines Entwurfsmodells, Design, Designnotation (UML)
  - unterstutzen CVS
  - Refactoring
  - Doku-Generierung
  - ...

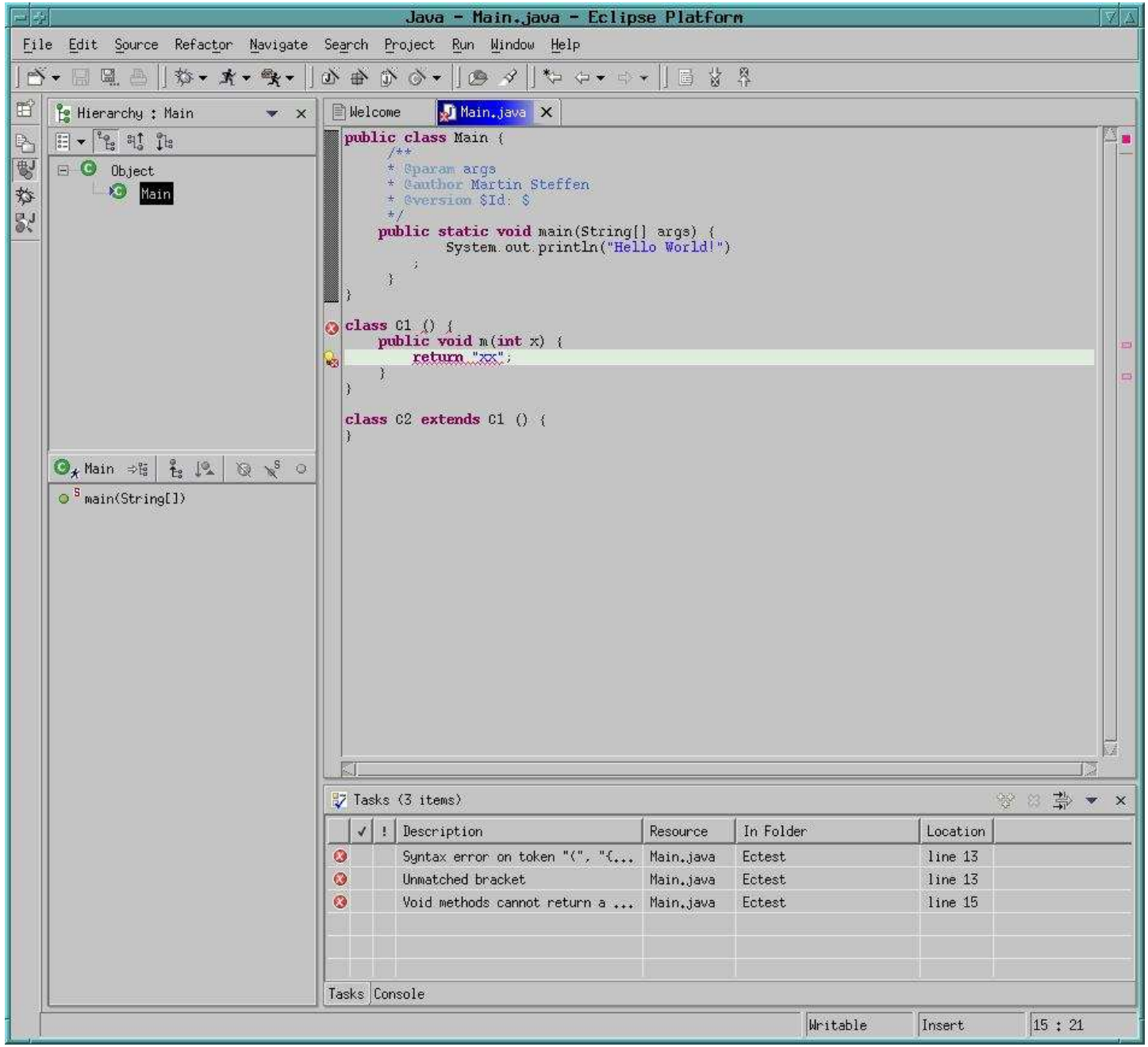
[1] Designpatterns

Entwicklungsumgebungen (II)

- was könnte problematisch sein:
  - erfordern Einarbeitung/Gewöhnung
  - sind eifersüchtig, inkompatible
  - zwingen einem evtl. eine bestimmte Vorgehensweise auf
  - machen Dinge "hinter dem Rücken" des Benutzers
  - sind speicherintensiv
- Bei uns installiert (siehe den *Java*-account)
  - JDE *java development for emacs*
  - BlueJ: für Einsteiger, nicht "industrial-strength"
  - (together)
  - Forte/Sun One Studio


Resource - Main.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Navigator
Ectest
doc
.classpath
.project
Main.class
Main.java
Test.class
Test.java
Welcome
Main.java
public class Main {
/**
* @param args
* @author Martin Steffen
* @version $Id: $
*/
public static void main(String[] args) {
System.out.println("Hello World!")
;
}
}
Outline
Main
main(String[])
Tasks (0 items)
Description
Resource
In Folder
Location
Writable
Insert
6 : 8


Java Browsing - Main.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Projects
Ectest
Packages
(default package)
doc
Types
Main
Test
Members
main(String[])
Welcome
Main.java
public class Main {
/**
* @param args
* @author Martin Steffen
* @version $Id: $
*/
public static void main(String[] args) {
System.out.println("Hello World!")
;
}
}
Writable
Insert
5 : 23


Java - Main.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Hierarchy : Main
Object
Main
Main
main(String[])
Welcome
Main.java
public class Main {
/**
* @param args
* @author Martin Steffen
* @version $Id: $
*/
public static void main(String[] args) {
System.out.println("Hello World!")
;
}
}
class C1 () {
public void m(int x) {
return "xx";
}
}
class C2 extends C1 () {
}
Tasks (3 items)
Description
Resource
In Folder
Location
Syntax error on token "(", "{... Main.java Ectest line 13
Unmatched bracket Main.java Ectest line 13
Void methods cannot return a ... Main.java Ectest line 15
Tasks
Console
Writable
Insert
15 : 21

Klassendiagramme in UML-Notation

Siehe den Merkzettel.
http://www.software-technik.de/microsoft/uml/umschlag.pdf