

llo

Handbuch zum "JUnit-Backend"

Christian Baumann und Julia Preusse

21. November 2007

Inhaltsverzeichnis

1	Voraussetzungen:	3
2	Erstellen einer Aufgabe:	4
2.1	Erstellen eines EC-Ordners:	4
2.2	Erstellen einer Aufgabe:	4
2.3	Das JUnit-Backend:	5
3	Beispiel:	5
3.1	Backend-Einstellungen:	6
3.1.1	Importe:	6
3.1.2	Unit-Tests:	6
3.1.3	Hilfsfunktionen:	7
3.2	Einreichung:	7
3.2.1	Mit syntaktischem Fehler:	7
3.2.2	Mit semantischem Fehler:	8
3.2.3	Fehlerfreie Einreichung:	8

1 Voraussetzungen:

- Plone
- ECAutoAssessmentBox
- ECSpooler
- JUnit-Backend

Nachdem alle Komponenten installiert wurden, können Sie das Backend verwenden, indem Sie unter **Konfiguration -> Auto Assessment Settings** den verfügbaren (Aviable) Backends "JUnit (1.0)" den gewählten (Selected) hinzufügen.

Spooler/Backend-Einstellungen

[← Zurück zur Plone-Konfiguration](#)

Einstellungen, die die Benutzung von ECSpooler/Backends für die automatische Überprüfung von Aufgaben betreffen.

Verbindungseinstellungen
Host ■
Die Adresse Ihres Spooler-Dienstes (z. B. aix.cs.uni-magdeburg.de).

Port ■
Der Port Ihres Spooler-Dienstes (z. B. 5050)

Benutzername ■
Der Benutzername mit dem Sie sich am Spooler-Dienst authentifizieren.

Passwort ■
Das Passwort für den angegebenen Benutzernamen.

Verfügbare und ausgewählte Backends
In der linken Liste finden Sie alle Backends, die über den Spooler-Dienst auf 'Laptop:5050' verfügbar sind. Wählen Sie ein oder mehrere Backends aus und fügen Sie diese der Liste der für Ihre Seite verfügbaren Backends hinzu.
Available: Selected:

Abbildung 1: Auto Assessment Settings

2 Erstellen einer Aufgabe:

2.1 Erstellen eines EC-Ordners:

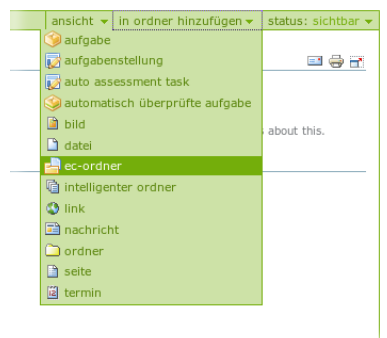


Abbildung 2: EC-Ordner

Aufgaben werden in Ordnern organisiert (vergleichbar mit einem Aufgabenblatt), die man wie folgt anlegt:

Auf der Plone-Startseite befindet sich das Menü "in ordner hinzufügen", in dem der Punkt "ec-ordner" gewählt werden muss.

Es wird zunächst ein Titel für den Ordner eingegeben, gefolgt von einer kurzen Beschreibung des Inhalts und optionalen Hinweisen.



Abbildung 3: Speichern

Durch einen Klick auf "speichern" werden die Eingaben bestätigt und ein neuer Ordner angelegt, in den automatisch gewechselt wird.

2.2 Erstellen einer Aufgabe:

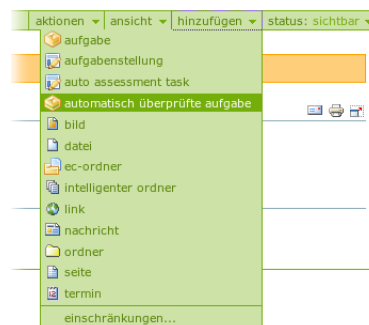


Abbildung 4: Automatisch überprüfte Aufgabe

Zur Erstellung einer Aufgabe, wählen Sie aus dem Menü "hinzufragen" den Punkt "automatisch überprüfte Aufgabe" und füllen auf der erscheinenden Seite die Eingabefelder wie gefordert aus.

Um das JUnit-Backend für diese Aufgabe nutzen zu können, wählen Sie unter dem Punkt "Backend" "JUnit (1.0)" und bestätigen die Eingaben mit einem Klick auf "nächster".



Abbildung 5: Nächster

2.3 Das JUnit-Backend:

Für die automatische Überprüfung von Java-Aufgaben, sind folgende Informationen notwendig:

- Imports

Das Ausfüllen dieses Feldes ist optional. Hier können Sie gegebenenfalls für die Unit-Tests benötigte Klassen und Archive angeben, die im Bibliothekenordner ¹ des Backends liegen.

- Unit-Tests

Das Ausfüllen dieses Feldes ist notwendig. Hier geben Sie die Unit-Tests ein. Bitte beachten Sie, dass nur Unit-Tests auf Basis der im Bibliothekenordner hinterlegten JUnit-Version verarbeitet werden können. Standardmäßig ist JUnit 4.3.1 mitgeliefert. Sollten Sie eine neuere Version nutzen wollen, tauschen Sie die entsprechenden Dateien aus.

- Help functions

Das Ausfüllen dieses Feldes ist optional. Hier können Sie gegebenenfalls benötigte Hilfsfunktionen definieren.

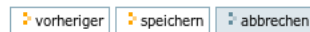


Abbildung 6: Speichern

Nach einem Klick auf "speichern" ist das Erstellen der Aufgabe abgeschlossen und das Backend ist in der Lage Einreichungen zu verarbeiten.

3 Beispiel:

Beispielhaft soll nun folgende Aufgabe erstellt werden:

Caesar-Chiffre:

Entwerfen Sie eine Klasse mit der das chiffrieren von Texten nach dem Caesar-Chiffre realisiert wird.

Die Klasse soll folgende Funktionen enthalten:

- **encode:** Nimmt einen Character und einen Int-Wert auf und gibt den um den Wert verschobenen Buchstaben zurück.
- **code:** Nimmt einen String und einen Int-Wert als Parameter auf und gibt den um den Wert chiffrierten String zurück.

Diese Aufgabe einzustellen geschieht in den folgenden Schritten:

- Erstellen eines EC-Ordners:

Erstellen Sie, wie in Abschnitt 2.1 beschrieben, einen EC-Ordner.

- Erstellen einer Aufgabe:

Erstellen Sie nun, wie in Abschnitt 2.2 beschrieben, eine Aufgabe und bestätigen Sie Ihre Eingaben mit "nächster", um zu den Einstellungen des Backends zu gelangen.

¹Zusätzliche Bibliotheken müssen in den Ordner `ECSpooler/backends/junit/junit.libs/` kopiert werden.

3.1 Backend-Einstellungen:

3.1.1 Importe:



Abbildung 7: Imports

Der Import von Bibliotheken ist für diese Aufgabe nicht notwendig.

3.1.2 Unit-Tests:

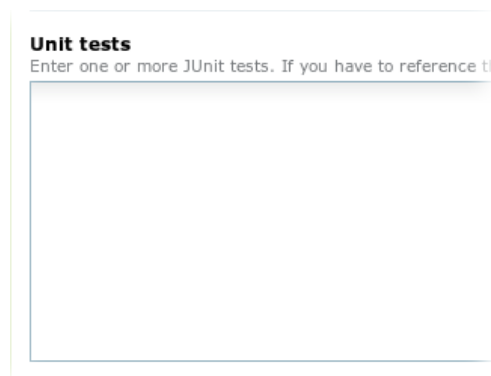


Abbildung 8: Unit tests

Unit-Tests für dieses Beispiel könnten beispielsweise so aussehen:

```
1  @Test public void encodeTestSingleChar(){
2      ${CLASS} submission = new ${CLASS}();
3      assertEquals('b',submission.encode('a',1));
4  }
5
6  @Test public void encodeEncode(){
7      ${CLASS} submission = new ${CLASS}();
8      assertEquals('a',submission.encode(submission.encode('a',5),-5));
9  }
10
11 @Test public void codeTestSingleString(){
12     ${CLASS} submission = new ${CLASS}();
13     String str = "abc";
14     assertEquals("bcd",submission.code(str,1));
15 }
16
17 @Test public void codeCode(){
18     ${CLASS} submission = new ${CLASS}();
19     String str = "Test";
20     assertEquals(str, submission.code(submission.code(str,4),-4));
21 }
```

3.1.3 Hilfsfunktionen:

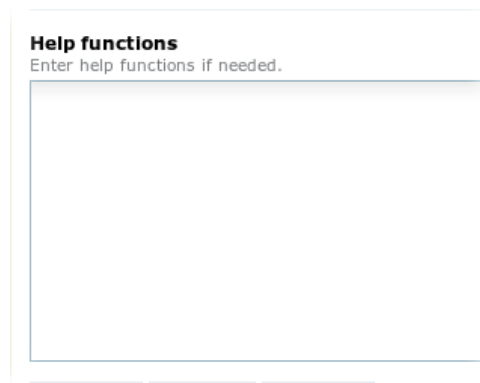


Abbildung 9: Help functions

Für die Unit-Tests wurden keine Hilfsfunktionen verwendet. Mit einem Klick auf **speichern** ist das Erstellen der Aufgabe beendet und das Backend ist in der Lage Java-Programme auf syntaktische und gemäß der angegebenen Unit-Tests auf ihre semantische Korrektheit hin zu überprüfen.

3.2 Einreichung:

Die Rückmeldungen des Backends sollen anhand von drei Einreichungen dargestellt werden:

3.2.1 Mit syntaktischem Fehler:

```
1 public class Caesar {
2     public static char encode(char buchstabe, int n){
3         buchstabe += n;
4         return buchstabe;
5     }
6
7     public static String code(String text, int n){
8         String txt = new String("")
9         for(int i = 0; i < text.length(); i++){
10            txt += encode(text.charAt(i), n);
11        }
12        return txt;
13    }
14 }
```

Der syntaktische Fehler in dieser Einreichung liegt in Zeile 8, da das abschließende Semikolon fehlt. Bei dieser Einreichung generiert das Backend folgende Rückmeldung:

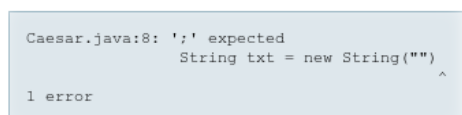


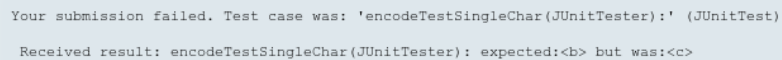
Abbildung 10: Ein Syntaxfehler wurde gefunden.

Der Studierende kann anhand dieser Ausgabe seinen Fehler nachvollziehen und ihn berichtigen.

3.2.2 Mit semantischem Fehler:

```
1 public class Caesar {
2     public static char encode(char buchstabe, int n){
3         buchstabe += n+1;
4         return buchstabe;
5     }
6
7     public static String code(String text, int n){
8         String txt = new String("");
9         for(int i = 0; i < text.length(); i++){
10             txt += encode(text.charAt(i), n);
11         }
12         return txt;
13     }
14 }
```

Der semantische Fehler liegt hier in Zeile 3, in der die Verschiebungslogik fehlerhaft ist. Das Backend liefert folgende Ausgabe:



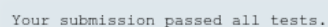
Your submission failed. Test case was: 'encodeTestSingleChar(JUnitTester):' (JUnitTest)
Received result: encodeTestSingleChar(JUnitTester): expected: but was:<c>

Abbildung 11: Ein semantischer Fehler wurde gefunden.

3.2.3 Fehlerfreie Einreichung:

```
1 public class Caesar {
2     public static char encode(char buchstabe, int n){
3         buchstabe += n;
4         return buchstabe;
5     }
6
7     public static String code(String text, int n){
8         String txt = new String("");
9         for(int i = 0; i < text.length(); i++){
10             txt += encode(text.charAt(i), n);
11         }
12         return txt;
13     }
14 }
```

Eine fehlerfreie Einreichung wird folgendermaßen angezeigt:



Your submission passed all tests.

Abbildung 12: Eine fehlerfreie Einreichung.