

Einheit 02: Einführung in JUnit Test

L.Raed

Ludwig-Maximilians-Universität München
Institut für Informatik
Programmierung und Softwaretechnik
Prof. Wirsing

1. September 2013



Die Lernziele der heutigen Einheit

- 1 Testen mit Print Statement und Scrapbook
- 2 Testen mit JUnit

Die Lernziele der heutigen Einheit

- 1 Testen mit Print Statement und Scrapbook
- 2 Testen mit JUnit

Die Lernziele der heutigen Einheit

- 1 Testen mit Print Statement und Scrapbook
- 2 Testen mit JUnit

Gliederung

1 Testen mit Print Statement

- Testen in Main Methode
- Testen in Scrapbook

2 Einführung in JUnit

- Prinzip und Installation
- Assertion Methoden
- Äquivalenzklassen: Testdaten auswählen
- JUnit Beispiel

Print Statement mit Main Methode

- **Testen mit Print Statements: `System.out.println(object)`**

- 1 Schreibe in jeder Klasse eine Main Methode.
- 2 Drucke fehlerverdächtige Variablen an mehreren Stellen aus.
- 3 Rufe die zu testenden Methoden in Main auf.
- 4 Vergleiche den SOLL-Wert mit dem IST-Wert der verdächtigen Variable

- **Vorteile vom Print Statement mit Main Methode**

- 1 + leicht anzuwenden: pflanze `System.out.println` im Code.
- 2 + sehr praktisch für viele (einfache) Fälle.

- **Nachteile vom Print Statement mit Main Methode**

- 1 - ohne main Methode ist nicht durchführbar.
- 2 - bei jeder Änderung muss alles neu kompiliert werden.
- 3 - nicht konfigurierbar: Output-Ziel nur die Konsole.
- 4 - alle `System.out.println()` müssen nachher entfernt werden.

Die System.out.println(Object o) Methode

- Die System.out.println(Object o) Methode

- Ergibt: package.klasse@hashCode
- Ruft automatisch toString() auf.
- Der Parameter o wird in ein String konvertiert.

- Die toString() Methode

- Ist eine Methode der Wurzelklasse Object
- Konvertiert einen object Parameter in ein String.
- Für einen lesbaren Objektzustand muss toString() überschrieben werden.
- Erinnerung: Objektzustand ist die aktuelle Wertbelegung der Feldvariablen.

Die System.out.println(Object o) Methode

- Die System.out.println(Object o) Methode

- Ergibt: package.klasse@hashCode
- Ruft automatisch toString() auf.
- Der Parameter o wird in ein String konvertiert.

- Die toString() Methode

- Ist eine Methode der Wurzelklasse Object
- Konvertiert einen object Parameter in ein String.
- Für einen lesbaren Objektzustand muss toString() überschrieben werden.
- Erinnerung: Objektzustand ist die aktuelle Wertbelegung der Feldvariablen.

Die System.out.println(Object o) Methode

- Die System.out.println(Object o) Methode

- Ergibt: package.klasse@hashCode
- Ruft automatisch toString() auf.
- Der Parameter o wird in ein String konvertiert.

- Die toString() Methode

- Ist eine Methode der Wurzelklasse Object
- Konvertiert einen object Parameter in ein String.
- Für einen lesbaren Objektzustand muss toString() überschrieben werden.
- Erinnerung: Objektzustand ist die aktuelle Wertbelegung der Feldvariablen.

Die System.out.println(Object o) Methode

- Die System.out.println(Object o) Methode

- Ergibt: package.klasse@hashCode
- Ruft automatisch toString() auf.
- Der Parameter o wird in ein String konvertiert.

- Die toString() Methode

- Ist eine Methode der Wurzelklasse Object
- Konvertiert einen object Parameter in ein String.
- Für einen lesbaren Objektzustand muss toString() überschrieben werden.
- Erinnerung: Objektzustand ist die aktuelle Wertbelegung der Feldvariablen.

Beispiel für Print Statement mit Main Methode

1 Programmieren Sie das calculateFactorial Beispiel

- Erzeugen Sie eine Klasse Factorial im Package eclipse.ke02
- Schreiben Sie die Methode: `public int calculateFactorial(int number)`
- Factorial von n ist die Fakultät von $n! = n * (n-1) * (n-2) * \dots * 1$
- Factorial von 0 ist 1 und für negative Zahlen nicht definiert.
- Bauen Sie bewusst den Fehler ein, dass $n! = n * (n-1) * (n-2) * \dots * 0$

2 Programmieren Sie Print Statement Test mit Main Methode

- Schreiben Sie eine Main Methode für die Factorial Klasse
- Erzeugen Sie drin ein testObject von der Factorial Klasse.
- Führen Sie zwei Testfälle durch: `number=5` und `number=-5`.
- Rufen Sie die `calculateFactorial(5)` und `calculateFactorial(-5)` auf.
- Vergleichen Sie durch `System.out.println()` das SOLL/IST Ergebnis.

3 Korrigieren Sie die Fehler

- Testen Sie am Anfang, ob die `number` kleiner als 0 ist
- Benutzen Sie die richtige Formel: $n! = n * (n-1) * (n-2) * \dots * 1$

Beispiel für Print Statement mit Main Methode

1 Programmieren Sie das calculateFactorial Beispiel

- Erzeugen Sie eine Klasse Factorial im Package eclipse.ke02
- Schreiben Sie die Methode: `public int calculateFactorial(int number)`
- Factorial von n ist die Fakultät von $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1$
- Factorial von 0 ist 1 und für negative Zahlen nicht definiert.
- Bauen Sie bewusst den Fehler ein, dass $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 0$

2 Programmieren Sie Print Statement Test mit Main Methode

- Schreiben Sie eine Main Methode für die Factorial Klasse
- Erzeugen Sie drin ein testObject von der Factorial Klasse.
- Führen Sie zwei Testfälle durch: `number=5` und `number=-5`.
- Rufen Sie die `calculateFactorial(5)` und `calculateFactorial(-5)` auf.
- Vergleichen Sie durch `System.out.println()` das SOLL/IST Ergebnis.

3 Korrigieren Sie die Fehler

- Testen Sie am Anfang, ob die `number` kleiner als 0 ist
- Benutzen Sie die richtige Formel: $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1$

Beispiel für Print Statement mit Main Methode

1 Programmieren Sie das calculateFactorial Beispiel

- Erzeugen Sie eine Klasse Factorial im Package eclipse.ke02
- Schreiben Sie die Methode: `public int calculateFactorial(int number)`
- Factorial von n ist die Fakultät von $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1$
- Factorial von 0 ist 1 und für negative Zahlen nicht definiert.
- Bauen Sie bewusst den Fehler ein, dass $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 0$

2 Programmieren Sie Print Statement Test mit Main Methode

- Schreiben Sie eine Main Methode für die Factorial Klasse
- Erzeugen Sie drin ein testObject von der Factorial Klasse.
- Führen Sie zwei Testfälle durch: `number=5` und `number=-5`.
- Rufen Sie die `calculateFactorial(5)` und `calculateFactorial(-5)` auf.
- Vergleichen Sie durch `System.out.println()` das SOLL/IST Ergebnis.

3 Korrigieren Sie die Fehler

- Testen Sie am Anfang, ob die `number` kleiner als 0 ist
- Benutzen Sie die richtige Formel: $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1$

Beispiel für Print Statement mit Main Methode

1 Programmieren Sie das calculateFactorial Beispiel

- Erzeugen Sie eine Klasse Factorial im Package eclipse.ke02
- Schreiben Sie die Methode: `public int calculateFactorial(int number)`
- Factorial von n ist die Fakultät von $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1$
- Factorial von 0 ist 1 und für negative Zahlen nicht definiert.
- Bauen Sie bewusst den Fehler ein, dass $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 0$

2 Programmieren Sie Print Statement Test mit Main Methode

- Schreiben Sie eine Main Methode für die Factorial Klasse
- Erzeugen Sie drin ein testObject von der Factorial Klasse.
- Führen Sie zwei Testfälle durch: `number=5` und `number=-5`.
- Rufen Sie die `calculateFactorial(5)` und `calculateFactorial(-5)` auf.
- Vergleichen Sie durch `System.out.println()` das SOLL/IST Ergebnis.

3 Korrigieren Sie die Fehler

- Testen Sie am Anfang, ob die `number` kleiner als 0 ist
- Benutzen Sie die richtige Formel: $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1$

Beispiel für Print Statement mit Main Methode

1 Programmieren Sie das calculateFactorial Beispiel

- Erzeugen Sie eine Klasse Factorial im Package eclipse.ke02
- Schreiben Sie die Methode: `public int calculateFactorial(int number)`
- Factorial von n ist die Fakultät von $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1$
- Factorial von 0 ist 1 und für negative Zahlen nicht definiert.
- Bauen Sie bewusst den Fehler ein, dass $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 0$

2 Programmieren Sie Print Statement Test mit Main Methode

- Schreiben Sie eine Main Methode für die Factorial Klasse
- Erzeugen Sie drin ein testObject von der Factorial Klasse.
- Führen Sie zwei Testfälle durch: `number=5` und `number=-5`.
- Rufen Sie die `calculateFactorial(5)` und `calculateFactorial(-5)` auf.
- Vergleichen Sie durch `System.out.println()` das SOLL/IST Ergebnis.

3 Korrigieren Sie die Fehler

- Testen Sie am Anfang, ob die `number` kleiner als 0 ist
- Benutzen Sie die richtige Formel: $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1$

Beispiel für Print Statement mit Main Methode

1 Programmieren Sie das calculateFactorial Beispiel

- Erzeugen Sie eine Klasse Factorial im Package eclipse.ke02
- Schreiben Sie die Methode: `public int calculateFactorial(int number)`
- Factorial von n ist die Fakultät von $n! = n * (n-1) * (n-2) * \dots * 1$
- Factorial von 0 ist 1 und für negative Zahlen nicht definiert.
- Bauen Sie bewusst den Fehler ein, dass $n! = n * (n-1) * (n-2) * \dots * 0$

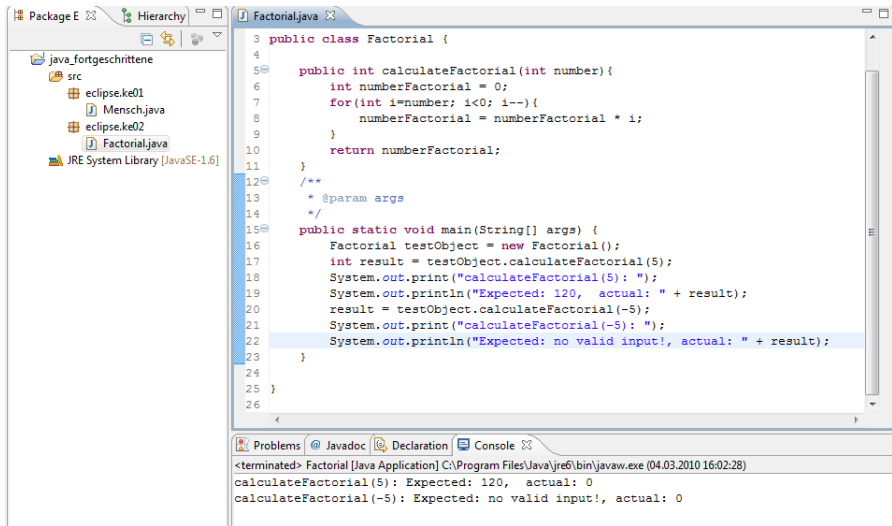
2 Programmieren Sie Print Statement Test mit Main Methode

- Schreiben Sie eine Main Methode für die Factorial Klasse
- Erzeugen Sie drin ein testObject von der Factorial Klasse.
- Führen Sie zwei Testfälle durch: `number=5` und `number=-5`.
- Rufen Sie die `calculateFactorial(5)` und `calculateFactorial(-5)` auf.
- Vergleichen Sie durch `System.out.println()` das SOLL/IST Ergebnis.

3 Korrigieren Sie die Fehler

- Testen Sie am Anfang, ob die `number` kleiner als 0 ist
- Benutzen Sie die richtige Formel: $n! = n * (n-1) * (n-2) * \dots * 1$

Lösung: Beispiel für Print Statement mit Main Methode



The screenshot displays the Eclipse IDE interface. On the left, the 'Package Explorer' shows a project named 'java_fortgeschrittene' with a source folder 'src' containing files 'eclipse.ke01', 'Mensch.java', 'eclipse.ke02', and 'Factorial.java'. The 'JRE System Library [JavaSE-1.6]' is also listed. The main editor window shows the code for 'Factorial.java'.

```
3 public class Factorial {
4
5     public int calculateFactorial(int number) {
6         int numberFactorial = 0;
7         for(int i=number; i<0; i--){
8             numberFactorial = numberFactorial * i;
9         }
10        return numberFactorial;
11    }
12    /**
13     * @param args
14     */
15    public static void main(String[] args) {
16        Factorial testObject = new Factorial();
17        int result = testObject.calculateFactorial(5);
18        System.out.print("calculateFactorial(5): ");
19        System.out.println("Expected: 120, actual: " + result);
20        result = testObject.calculateFactorial(-5);
21        System.out.print("calculateFactorial(-5): ");
22        System.out.println("Expected: no valid input!, actual: " + result);
23    }
24
25 }
26
```

The bottom of the IDE shows the 'Console' tab with the following output:

```
<terminated> Factorial [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (04.03.2010 16:02:28)
calculateFactorial(5): Expected: 120, actual: 0
calculateFactorial(-5): Expected: no valid input!, actual: 0
```

Print Statement in Scrapbook

● Die Scrapbook-Technik

- Führt den zu testenden Code oder Teil davon on the fly.
- Benutzt eine Scrapbook-Page als Editor.
- In dem Editor kommt der zu testende Code oder Teildavon.
- Ist sehr hilfreich, wenn der Code zu lang ist.
- Eine Main-Methode ist dafür nicht erforderlich.

● Erstellen von Scrapbook

- Package markieren → RM → other → Java Run/Debug → Scrapbook page
- Next → Enter or select the Parent folder → Parent Folder auswählen
- File name → Scrapbook-Page Name eingeben. → Finish.

● Scrapbook Ausführen

- Der zu testende Code in Scrapbook Editor auswählen.
- Run → Excute menu item. Oder das Excute Symbol betätigen.
- Run → Excute zeigt das Ergebnis in der Console.
- Run → Display zeigt das Ergebnis in Scrapbook.

● Aufgabe: Erzeugen Sie ein Scrapbook Page für Factorial Programm

- Testen Sie die calculateFactorial für 5 und -5.

Print Statement in Scrapbook

● Die Scrapbook-Technik

- Führt den zu testenden Code oder Teil davon on the fly.
- Benutzt eine Scrapbook-Page als Editor.
- In dem Editor kommt der zu testende Code oder Teildavon.
- Ist sehr hilfreich, wenn der Code zu lang ist.
- Eine Main-Methode ist dafür nicht erforderlich.

● Erstellen von Scrapbook

- Package markieren → RM → other → Java Run/Debug → Scrapbook page
- Next → Enter or select the Parent folder → Parent Folder auswählen
- File name → Scrapbook-Page Name eingeben. → Finish.

● Scrapbook Ausführen

- Der zu testende Code in Scrapbook Editor auswählen.
- Run → Excute menu item. Oder das Excute Symbol betätigen.
- Run → Excute zeigt das Ergebnis in der Console.
- Run → Display zeigt das Ergebnis in Scrapbook.

● Aufgabe: Erzeugen Sie ein Scrapbook Page für Factorial Programm

- Testen Sie die calculateFactorial für 5 und -5.

Print Statement in Scrapbook

● Die Scrapbook-Technik

- Führt den zu testenden Code oder Teil davon on the fly.
- Benutzt eine Scrapbook-Page als Editor.
- In dem Editor kommt der zu testende Code oder Teildavon.
- Ist sehr hilfreich, wenn der Code zu lang ist.
- Eine Main-Methode ist dafür nicht erforderlich.

● Erstellen von Scrapbook

- Package markieren → RM → other → Java Run/Debug → Scrapbook page
- Next → Enter or select the Parent folder → Parent Folder auswählen
- File name → Scrapbook-Page Name eingeben. → Finish.

● Scrapbook Ausführen

- Der zu testende Code in Scrapbook Editor auswählen.
- Run → Excute menu item. Oder das Excute Symbol betätigen.
- Run → Excute zeigt das Ergebnis in der Console.
- Run → Display zeigt das Ergebnis in Scrapbook.

● Aufgabe: Erzeugen Sie ein Scrapbook Page für Factorial Programm

- Testen Sie die calculateFactorial für 5 und -5.

Print Statement in Scrapbook

● Die Scrapbook-Technik

- Führt den zu testenden Code oder Teil davon on the fly.
- Benutzt eine Scrapbook-Page als Editor.
- In dem Editor kommt der zu testende Code oder Teildavon.
- Ist sehr hilfreich, wenn der Code zu lang ist.
- Eine Main-Methode ist dafür nicht erforderlich.

● Erstellen von Scrapbook

- Package markieren → RM → other → Java Run/Debug → Scrapbook page
- Next → Enter or select the Parent folder → Parent Folder auswählen
- File name → Scrapbook-Page Name eingeben. → Finish.

● Scrapbook Ausführen

- Der zu testende Code in Scrapbook Editor auswählen.
- Run → Excute menu item. Oder das Excute Symbol betätigen.
- Run → Excute zeigt das Ergebnis in der Console.
- Run → Display zeigt das Ergebnis in Scrapbook.

● Aufgabe: Erzeugen Sie ein Scrapbook Page für Factorial Programm

- Testen Sie die calculateFactorial für 5 und -5.

Print Statement in Scrapbook

● Die Scrapbook-Technik

- Führt den zu testenden Code oder Teil davon on the fly.
- Benutzt eine Scrapbook-Page als Editor.
- In dem Editor kommt der zu testende Code oder Teildavon.
- Ist sehr hilfreich, wenn der Code zu lang ist.
- Eine Main-Methode ist dafür nicht erforderlich.

● Erstellen von Scrapbook

- Package markieren → RM → other → Java Run/Debug → Scrapbook page
- Next → Enter or select the Parent folder → Parent Folder auswählen
- File name → Scrapbook-Page Name eingeben. → Finish.

● Scrapbook Ausführen

- Der zu testende Code in Scrapbook Editor auswählen.
- Run → Excute menu item. Oder das Excute Symbol betätigen.
- Run → Excute zeigt das Ergebnis in der Console.
- Run → Display zeigt das Ergebnis in Scrapbook.

● Aufgabe: Erzeugen Sie ein Scrapbook Page für Factorial Programm

- Testen Sie die calculateFactorial für 5 und -5.

Print Statement in Scrapbook

● Die Scrapbook-Technik

- Führt den zu testenden Code oder Teil davon on the fly.
- Benutzt eine Scrapbook-Page als Editor.
- In dem Editor kommt der zu testende Code oder Teildavon.
- Ist sehr hilfreich, wenn der Code zu lang ist.
- Eine Main-Methode ist dafür nicht erforderlich.

● Erstellen von Scrapbook

- Package markieren → RM → other → Java Run/Debug → Scrapbook page
- Next → Enter or select the Parent folder → Parent Folder auswählen
- File name → Scrapbook-Page Name eingeben. → Finish.

● Scrapbook Ausführen

- Der zu testende Code in Scrapbook Editor auswählen.
- Run → Excute menu item. Oder das Excute Symbol betätigen.
- Run → Excute zeigt das Ergebnis in der Console.
- Run → Display zeigt das Ergebnis in Scrapbook.

● Aufgabe: Erzeugen Sie ein Scrapbook Page für Factorial Programm

- Testen Sie die calculateFactorial für 5 und -5.

Print Statement in Scrapbook

● Die Scrapbook-Technik

- Führt den zu testenden Code oder Teil davon on the fly.
- Benutzt eine Scrapbook-Page als Editor.
- In dem Editor kommt der zu testende Code oder Teildavon.
- Ist sehr hilfreich, wenn der Code zu lang ist.
- Eine Main-Methode ist dafür nicht erforderlich.

● Erstellen von Scrapbook

- Package markieren → RM → other → Java Run/Debug → Scrapbook page
- Next → Enter or select the Parent folder → Parent Folder auswählen
- File name → Scrapbook-Page Name eingeben. → Finish.

● Scrapbook Ausführen

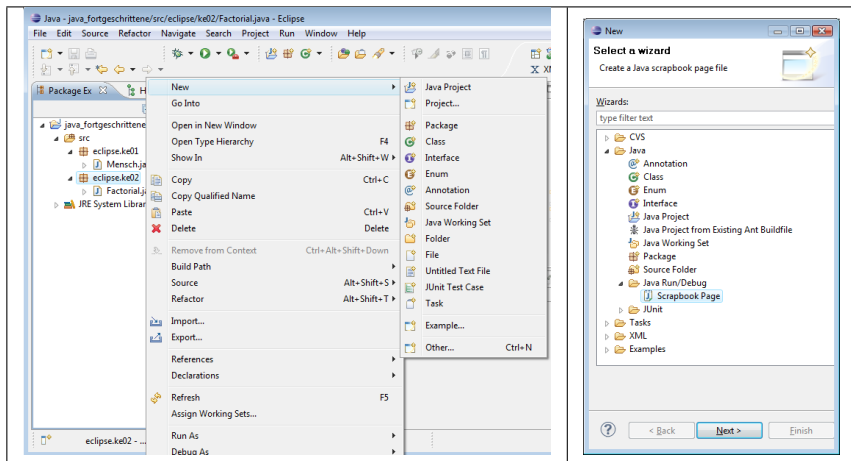
- Der zu testende Code in Scrapbook Editor auswählen.
- Run → Excute menu item. Oder das Excute Symbol betätigen.
- Run → Excute zeigt das Ergebnis in der Console.
- Run → Display zeigt das Ergebnis in Scrapbook.

● Aufgabe: Erzeugen Sie ein Scrapbook Page für Factorial Programm

- Testen Sie die calculateFactorial für 5 und -5.

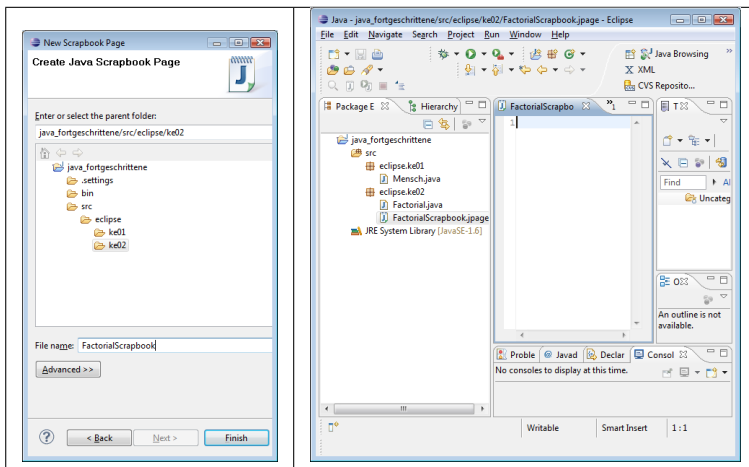
Beispiel: Scrapbook erstellen

1 eclipse.ke02 → RM → New → other → Java Run/Debug → Scrapbook page



Beispiel: Print Statement in Scrapbook

- ② Enter or select the parent folder → unverändert übernehmen.
- ③ File name → FactorialScrapbook eintippen → Finish

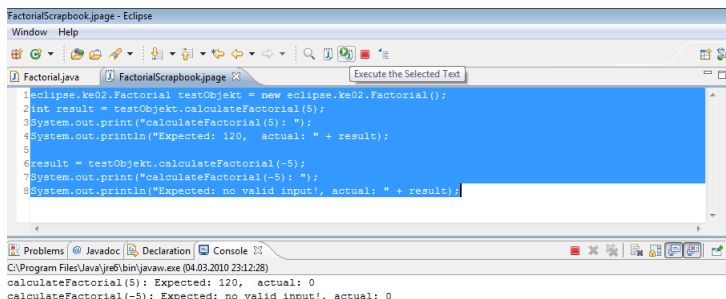


Beispiel: Print Statement in Scrapbook

4 Den zu testenden Code eintippen

- Ein Factorial testObjekt erzeugen (Fra eintippen + STRG + SPACE)
- Mit dem testObjekt die Methode calculateFactorial(5) aufrufen.
- Mit dem testObjekt die System.out.println() aufrufen.
- Mit dem testObjekt die Methode calculateFactorial(-5) aufrufen.

5 Den zu testenden Code markieren → Excuted the selected Text betätigen.



```
FactorialScrapbook.jpage - Eclipse
Window Help

Factorial.java  FactorialScrapbook.jpage  Execute the Selected Text

1 eclipse.ke02.Factorial testObjekt = new eclipse.ke02.Factorial();
2 int result = testObjekt.calculateFactorial(5);
3 System.out.print("calculateFactorial(5): ");
4 System.out.println("Expected: 120, actual: " + result);
5
6 result = testObjekt.calculateFactorial(-5);
7 System.out.print("calculateFactorial(-5): ");
8 System.out.println("Expected: no valid input!, actual: " + result);

Problems  Javadoc  Declaration  Console
C:\Program Files\Java\jre6\bin\javaw.exe (04.03.2010 23:12:28)
calculateFactorial(5): Expected: 120, actual: 0
calculateFactorial(-5): Expected: no valid input!, actual: 0
```

Prinzip und Installation

- JUnit ist ein „Open Source Testing Framework“

- Mit JUnit kann der Entwickler-Sourcecodes getestet werden.
- Die Grundidee von JUnit ist der IST/SOLL Vergleich.
- IST/SOLL Vergleich: was liefert die Methode und was wird erwartet?
- IST/SOLL Vergleich wird durch die JUnit Assertion Methods erreicht

- Installation von JUnit

- JUnit ist mit Eclipse als Plug-in mitgeliefert.
- Man kann aber die neuste JUnit Version downloaden und einbinden.
- Download Seite: <http://www.junit.org>. Zwei JARs sind dabei wichtig:
- junit-x.y.jar und junit-x.y-src.jar wobei x.y für Version-Nr stehen.
- Beide JARs im Projekt lib Ordner speichern und anschliessend einbinden.
- JARs als Classpath-Variable (JUNIT_HOME und JUNIT_SRC) einbinden.

Prinzip und Installation

- JUnit ist ein „Open Source Testing Framework“

- Mit JUnit kann der Entwickler-Sourcecodes getestet werden.
- Die Grundidee von JUnit ist der IST/SOLL Vergleich.
- IST/SOLL Vergleich: was liefert die Methode und was wird erwartet?
- IST/SOLL Vergleich wird durch die JUnit Assertion Methods erreicht

- Installation von JUnit

- JUnit ist mit Eclipse als Plug-in mitgeliefert.
- Man kann aber die neuste JUnit Version downloaden und einbinden.
- Download Seite: <http://www.junit.org>. Zwei JARs sind dabei wichtig:
- junit-x.y.jar und junit-x.y-src.jar wobei x.y für Version-Nr stehen.
- Beide JARs im Projekt lib Ordner speichern und anschliessend einbinden.
- JARs als Classpath-Variable (JUNIT_HOME und JUNIT_SRC) einbinden.

Prinzip und Installation

- JUnit ist ein „Open Source Testing Framework“

- Mit JUnit kann der Entwickler-Sourcecodes getestet werden.
- Die Grundidee von JUnit ist der IST/SOLL Vergleich.
- IST/SOLL Vergleich: was liefert die Methode und was wird erwartet?
- IST/SOLL Vergleich wird durch die JUnit Assertion Methods erreicht

- Installation von JUnit

- JUnit ist mit Eclipse als Plug-in mitgeliefert.
- Man kann aber die neuste JUnit Version downloaden und einbinden.
- Download Seite: <http://www.junit.org>. Zwei JARs sind dabei wichtig:
- junit-x.y.jar und junit-x.y-src.jar wobei x.y für Version-Nr stehen.
- Beide JARs im Projekt lib Ordner speichern und anschliessend einbinden.
- JARs als Classpath-Variable (JUNIT_HOME und JUNIT_SRC) einbinden.

Prinzip und Installation

- JUnit ist ein „Open Source Testing Framework“

- Mit JUnit kann der Entwickler-Sourcecodes getestet werden.
- Die Grundidee von JUnit ist der IST/SOLL Vergleich.
- IST/SOLL Vergleich: was liefert die Methode und was wird erwartet?
- IST/SOLL Vergleich wird durch die JUnit Assertion Methods erreicht

- Installation von JUnit

- JUnit ist mit Eclipse als Plug-in mitgeliefert.
- Man kann aber die neuste JUnit Version downloaden und einbinden.
- Download Seite: <http://www.junit.org>. Zwei JARs sind dabei wichtig:
- junit-x.y.jar und junit-x.y-src.jar wobei x.y für Version-Nr stehen.
- Beide JARs im Projekt lib Ordner speichern und anschliessend einbinden.
- JARs als Classpath-Variable (JUNIT_HOME und JUNIT_SRC) einbinden.

Wie funktioniert JUnit Test?

- Die Hauptsäulen von JUnit sind die Assertion Methoden

- JUnit besteht aus „assertion methods“
- Assertion Methoden testen verschiedene Konditionen.
- JUnit ist so designed, um den Code des Entwickler zu testen.
- Die erste JUnit Version hat 7 Asseration Methoden.
- Die neueren Versionen haben viel mehr. Das Prinzip ist dasgleiche!

- JUnit Test in Eclipse durchführen

- Aus klasse K wird die JUnit Klasse „KTest“ generiert.
- Die K-Methoden werden in KTest-Methoden aufgerufen.
- Mit Assertion Methoden wird der SOLL/IST Vergleich getestet.
- `public void testMethodX(){assertionMethode(Aufruf von methodX());}`

- Alle JUnit Tests werden unabhängig von einander ausgeführt

- Eine Methode m1 erzeugt ein Array.
- Eine Methode m2 benutzt das Array.
- Dann muss m2 in JUnit Code m1 aufrufen!
- Da m1, m2 in JUnit unabhängig von einander getestet werden!

Wie funktioniert JUnit Test?

- Die Hauptsäulen von JUnit sind die Assertion Methoden

- JUnit besteht aus „assertion methods“
- Assertion Methoden testen verschiedene Konditionen.
- JUnit ist so designed, um den Code des Entwickler zu testen.
- Die erste JUnit Version hat 7 Asseration Methoden.
- Die neueren Versionen haben viel mehr. Das Prinzip ist dasgleiche!

- JUnit Test in Eclipse durchführen

- Aus klasse K wird die JUnit Klasse „KTest“ generiert.
- Die K-Methoden werden in KTest-Methoden aufgerufen.
- Mit Assertion Methoden wird der SOLL/IST Vergleich getestet.
- `public void testMethodX(){assertionMethode(Aufruf von methodX());}`

- Alle JUnit Tests werden unabhängig von einander ausgeführt

- Eine Methode m1 erzeugt ein Array.
- Eine Methode m2 benutzt das Array.
- Dann muss m2 in JUnit Code m1 aufrufen!
- Da m1, m2 in JUnit unabhängig von einander getestet werden!

Wie funktioniert JUnit Test?

- Die Hauptsäulen von JUnit sind die Assertion Methoden
 - JUnit besteht aus „assertion methods“
 - Assertion Methoden testen verschiedene Konditionen.
 - JUnit ist so designed, um den Code des Entwickler zu testen.
 - Die erste JUnit Version hat 7 Asseration Methoden.
 - Die neueren Versionen haben viel mehr. Das Prinzip ist dasgleiche!
- JUnit Test in Eclipse durchführen
 - Aus klasse K wird die JUnit Klasse „KTest“ generiert.
 - Die K-Methoden werden in KTest-Methoden aufgerufen.
 - Mit Assertion Methoden wird der SOLL/IST Vergleich getestet.
 - `public void testMethodX(){assertionMethode(Aufruf von methodX());}`
- Alle JUnit Tests werden unabhängig von einander ausgeführt
 - Eine Methode m1 erzeugt ein Array.
 - Eine Methode m2 benutzt das Array.
 - Dann muss m2 in JUnit Code m1 aufrufen!
 - Da m1, m2 in JUnit unabhängig von einander getestet werden!

Wie funktioniert JUnit Test?

- Die Hauptsäulen von JUnit sind die Assertion Methoden
 - JUnit besteht aus „assertion methods“
 - Assertion Methoden testen verschiedene Konditionen.
 - JUnit ist so designed, um den Code des Entwickler zu testen.
 - Die erste JUnit Version hat 7 Asseration Methoden.
 - Die neueren Versionen haben viel mehr. Das Prinzip ist dasgleiche!
- JUnit Test in Eclipse durchführen
 - Aus klasse K wird die JUnit Klasse „KTest“ generiert.
 - Die K-Methoden werden in KTest-Methoden aufgerufen.
 - Mit Assertion Methoden wird der SOLL/IST Vergleich getestet.
 - `public void testMethodX(){assertionMethode(Aufruf von methodX());}`
- Alle JUnit Tests werden unabhängig von einander ausgeführt
 - Eine Methode m1 erzeugt ein Array.
 - Eine Methode m2 benutzt das Array.
 - Dann muss m2 in JUnit Code m1 aufrufen!
 - Da m1, m2 in JUnit unabhängig von einander getestet werden!

Wie funktioniert JUnit Test?

- Die Hauptsäulen von JUnit sind die Assertion Methoden
 - JUnit besteht aus „assertion methods“
 - Assertion Methoden testen verschiedene Konditionen.
 - JUnit ist so designed, um den Code des Entwickler zu testen.
 - Die erste JUnit Version hat 7 Asseration Methoden.
 - Die neueren Versionen haben viel mehr. Das Prinzip ist dasgleiche!
- JUnit Test in Eclipse durchführen
 - Aus klasse K wird die JUnit Klasse „KTest“ generiert.
 - Die K-Methoden werden in KTest-Methoden aufgerufen.
 - Mit Assertion Methoden wird der SOLL/IST Vergleich getestet.
 - `public void testMethodX(){assertionMethode(Aufruf von methodX());}`
- Alle JUnit Tests werden unabhängig von einander ausgeführt
 - Eine Methode m1 erzeugt ein Array.
 - Eine Methode m2 benutzt das Array.
 - Dann muss m2 in JUnit Code m1 aufrufen!
 - Da m1, m2 in JUnit unabhängig von einander getestet werden!

Wie funktioniert JUnit Test?

- Die Hauptsäulen von JUnit sind die Assertion Methoden
 - JUnit besteht aus „assertion methods“
 - Assertion Methoden testen verschiedene Konditionen.
 - JUnit ist so designed, um den Code des Entwickler zu testen.
 - Die erste JUnit Version hat 7 Asseration Methoden.
 - Die neueren Versionen haben viel mehr. Das Prinzip ist dasgleiche!
- JUnit Test in Eclipse durchführen
 - Aus klasse K wird die JUnit Klasse „KTest“ generiert.
 - Die K-Methoden werden in KTest-Methoden aufgerufen.
 - Mit Assertion Methoden wird der SOLL/IST Vergleich getestet.
 - `public void testMethodX(){assertionMethode(Aufruf von methodX());}`
- Alle JUnit Tests werden unabhängig von einander ausgeführt
 - Eine Methode m1 erzeugt ein Array.
 - Eine Methode m2 benutzt das Array.
 - Dann muss m2 in JUnit Code m1 aufrufen!
 - Da m1, m2 in JUnit unabhängig von einander getestet werden!

Asseration Methoden

Asseration Methoden	Test-Beschreibung
<code>assertEquals(a, b)</code>	Ist a gleich b?
<code>assertTrue(a)</code>	Ist boolean a true?
<code>assertNotTrue(a)</code>	Ist boolean a false?
<code>assertNull(a)</code>	Ist das Objekt a gleich null?
<code>assertNotNull(a)</code>	Ist das Objekt a ungleich null?
<code>asserSame(a, b)</code>	Referenziert a und b auf dasgleiche Objekt?
<code>asserNotSame(a, b)</code>	Referenziert a und b nicht auf dasgleiche Objekt?

JUnit Anwendungsregel Beispiele

1 Methode X(), die ein Array erzeugt, initialisiert und zurück liefert:

- x() ist richtig, wenn der Rückgabewert ungleich null ist.
- `public void testX(){assertNotNull(testobject.x()); }`

2 Methode X(), die einen Wert zurück liefert:

- x() ist richtig, wenn sie den erwarteten Wert zurück liefert.
- `void testX(){Expected, assertEquals(testobject.x()); }`

3 Methode X(), die einen Wert setzt und boolean liefert

- x() ist richtig, wenn sie true liefert.
- `void testX(){assertTrue(testobject.x()); }`

4 Methode X(), die eine Feldvariable setzt und void Rückgabewert hat

- x() ist richtig, wenn der neue Wert in die Ziel-Variable V gesetzt wird.
- x() wird aufgerufen, setzt einen Wert, der mit getV() zurückgeliefert wird.
- `void testX(){testObjekt.x(); assertEquals(Expected, testObjekt.getV())}`

JUnit Anwendungsregel Beispiele

1 Methode X(), die ein Array erzeugt, initialisiert und zurück liefert:

- x() ist richtig, wenn der Rückgabewert ungleich null ist.
- `public void testX(){assertNotNull(testobject.x()); }`

2 Methode X(), die einen Wert zurück liefert:

- x() ist richtig, wenn sie den erwarteten Wert zurück liefert.
- `void testX(){Expected, assertEquals(testobject.x()); }`

3 Methode X(), die einen Wert setzt und boolean liefert

- x() ist richtig, wenn sie true liefert.
- `void testX(){assertTrue(testobject.x()); }`

4 Methode X(), die eine Feldvariable setzt und void Rückgabewert hat

- x() ist richtig, wenn der neue Wert in die Ziel-Variable V gesetzt wird.
- x() wird aufgerufen, setzt einen Wert, der mit getV() zurückgeliefert wird.
- `void testX(){testObjekt.x(); assertEquals(Expected, testObjekt.getV())}`

JUnit Anwendungsregel Beispiele

1 Methode X(), die ein Array erzeugt, initialisiert und zurück liefert:

- x() ist richtig, wenn der Rückgabewert ungleich null ist.
- `public void testX(){assertNotNull(testobject.x()); }`

2 Methode X(), die einen Wert zurück liefert:

- x() ist richtig, wenn sie den erwarteten Wert zurück liefert.
- `void testX(){Expected, assertEquals(testobject.x()); }`

3 Methode X(), die einen Wert setzt und boolean liefert

- x() ist richtig, wenn sie true liefert.
- `void testX(){assertTrue(testobject.x()); }`

4 Methode X(), die eine Feldvariable setzt und void Rückgabewert hat

- x() ist richtig, wenn der neue Wert in die Ziel-Variable V gesetzt wird.
- x() wird aufgerufen, setzt einen Wert, der mit getV() zurückgeliefert wird.
- `void testX(){testObjekt.x(); assertEquals(Expected, testObjekt.getV())}`

JUnit Anwendungsregel Beispiele

1 Methode X(), die ein Array erzeugt, initialisiert und zurück liefert:

- x() ist richtig, wenn der Rückgabewert ungleich null ist.
- `public void testX(){assertNotNull(testobject.x()); }`

2 Methode X(), die einen Wert zurück liefert:

- x() ist richtig, wenn sie den erwarteten Wert zurück liefert.
- `void testX(){Expected, assertEquals(testobject.x()); }`

3 Methode X(), die einen Wert setzt und boolean liefert

- x() ist richtig, wenn sie true liefert.
- `void testX(){assertTrue(testobject.x()); }`

4 Methode X(), die eine Feldvariable setzt und void Rückgabewert hat

- x() ist richtig, wenn der neue Wert in die Ziel-Variable V gesetzt wird.
- x() wird aufgerufen, setzt einen Wert, der mit getV() zurückgeliefert wird.
- `void testX(){testObjekt.x(); assertEquals(Expected, testObjekt.getV())}`

JUnit Anwendungsregel Beispiele

1 Methode X(), die ein Array erzeugt, initialisiert und zurück liefert:

- x() ist richtig, wenn der Rückgabewert ungleich null ist.
- `public void testX(){assertNotNull(testobject.x()); }`

2 Methode X(), die einen Wert zurück liefert:

- x() ist richtig, wenn sie den erwarteten Wert zurück liefert.
- `void testX(){Expected, assertEquals(testobject.x()); }`

3 Methode X(), die einen Wert setzt und boolean liefert

- x() ist richtig, wenn sie true liefert.
- `void testX(){assertTrue(testobject.x()); }`

4 Methode X(), die eine Feldvariable setzt und void Rückgabewert hat

- x() ist richtig, wenn der neue Wert in die Ziel-Variable V gesetzt wird.
- x() wird aufgerufen, setzt einen Wert, der mit getV() zurückgeliefert wird.
- `void testX(){testObjekt.x(); assertEquals(Expected, testObjekt.getV())}`

JUnit Anwendungsregel Beispiele

1 Methode X(), die ein Array erzeugt, initialisiert und zurück liefert:

- x() ist richtig, wenn der Rückgabewert ungleich null ist.
- `public void testX(){assertNotNull(testobject.x()); }`

2 Methode X(), die einen Wert zurück liefert:

- x() ist richtig, wenn sie den erwarteten Wert zurück liefert.
- `void testX(){Expected, assertEquals(testobject.x()); }`

3 Methode X(), die einen Wert setzt und boolean liefert

- x() ist richtig, wenn sie true liefert.
- `void testX(){assertTrue(testobject.x()); }`

4 Methode X(), die eine Feldvariable setzt und void Rückgabewert hat

- x() ist richtig, wenn der neue Wert in die Ziel-Variable V gesetzt wird.
- x() wird aufgerufen, setzt einen Wert, der mit getV() zurückgeliefert wird.
- `void testX(){testObjekt.x(); assertEquals(Expected, testObjekt.getV())}`

JUnit Anwendungsregel Beispiele

1 Methode X(), die ein Array erzeugt, initialisiert und zurück liefert:

- x() ist richtig, wenn der Rückgabewert ungleich null ist.
- `public void testX(){assertNotNull(testobject.x()); }`

2 Methode X(), die einen Wert zurück liefert:

- x() ist richtig, wenn sie den erwarteten Wert zurück liefert.
- `void testX(){Expected, assertEquals(testobject.x()); }`

3 Methode X(), die einen Wert setzt und boolean liefert

- x() ist richtig, wenn sie true liefert.
- `void testX(){assertTrue(testobject.x()); }`

4 Methode X(), die eine Feldvariable setzt und void Rückgabewert hat

- x() ist richtig, wenn der neue Wert in die Ziel-Variable V gesetzt wird.
- x() wird aufgerufen, setzt einen Wert, der mit getV() zurückgeliefert wird.
- `void testX(){testObjekt.x(); assertEquals(Expected, testObjekt.getV())}`

JUnit Anwendungsregel Beispiele

❶ Methode X(), die ein Array erzeugt, initialisiert und zurück liefert:

- x() ist richtig, wenn der Rückgabewert ungleich null ist.
- `public void testX(){assertNotNull(testobject.x()); }`

❷ Methode X(), die einen Wert zurück liefert:

- x() ist richtig, wenn sie den erwarteten Wert zurück liefert.
- `void testX(){Expected, assertEquals(testobject.x()); }`

❸ Methode X(), die einen Wert setzt und boolean liefert

- x() ist richtig, wenn sie true liefert.
- `void testX(){assertTrue(testobject.x()); }`

❹ Methode X(), die eine Feldvariable setzt und void Rückgabewert hat

- x() ist richtig, wenn der neue Wert in die Ziel-Variable V gesetzt wird.
- x() wird aufgerufen, setzt einen Wert, der mit getV() zurückgeliefert wird.
- `void testX(){testObjekt.x(); assertEquals(Expected, testObjekt.getV())}`

Testdaten auswählen

1 Ermittle die Äquivalenzklassen des Wertebereiches

- Äquivalenzklassen: nichtleere und disjunkte Teilmengen des Wertebereiches
- Beispiel: i ist int Zahl mit $0 < i < 10$
- Äquivalenzklasse 1: alle Werte kleiner 0 (bis int Minusgrenze)
- Äquivalenzklasse 2: der untere Grenzwert also 0
- Äquivalenzklasse 3: alle Werte zwischen 0 und 10 (außer 0, 10)
- Äquivalenzklasse 4: der obere Grenzwert also 10.
- Äquivalenzklasse 5: alle Werte größer als 10 (bis int Plusgrenze)

2 Wähle von jeder Äquivalenzklasse ein Datum als Stellvertreter

- Ein Datum ist ein beliebiger Wert aus der Äquivalenzklasse
- Stellvertreter für die Äquivalenzklassen 1 bis 5 sind z.B. $\{-5, 0, 4, 10, 35\}$

3 Führe die JUnit Test mindestens einmal für jede Äquivalenzklasse

JUnit Aufgabe

- Übertragen Sie den Code unten in Eclipse

```
package eclipse.ke02;

public class Lotto {
    private int[] myFeld;

    public int[] zieheLotto() {
        myFeld[0] = 5; myFeld[1] = 35;
        myFeld[2] = 17; myFeld[3] = 49;
        myFeld[4] = 28; myFeld[5] = 7;
        return myFeld; }

    public int liefereZahl(int position) {
        return myFeld[position]; }

    public boolean ersetzeZahl(int position, int neu) {
        if (position > 0 && position < myFeld.length) {
            myFeld[position] = neu;
            return true; }
        return false; } }
```

- Schreiben Sie für Lotto die JUnit Testklasse

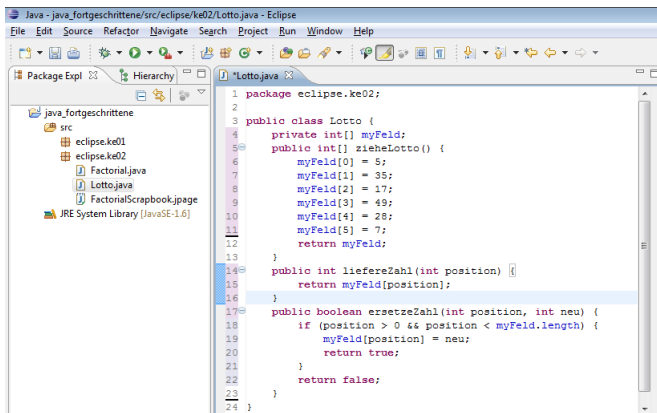
JUnit Aufgabe Lösungsschritte

- 1 Code in Eclipse übertragen.
- 2 JUnit Testklasse erstellen.
- 3 SetUp() und tearDown() aktivieren.
- 4 Die zu testenden Methoden auswählen.
- 5 JUnit Testklasse Skelett-Implementation.
- 6 JUnit Testklasse Methoden implementieren
- 7 JUnit ausführen.
- 8 Fehler korrigieren.
- 9 JUnit Test erneut ausführen.

Lösung: 1-Code in Eclipse übertragen

• Lotto-Code in Eclipse übertragen

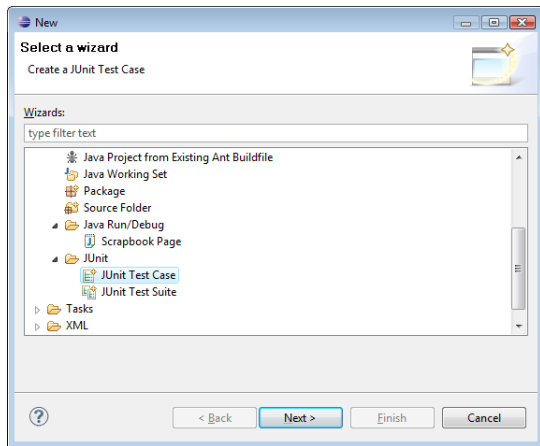
- Erstellen Sie die Klasse Lotto unter dem Package ke02
- Tippen Sie den Code in der Folie davor in Ihrer Klasse ein.



```
1 package eclipse.ke02;
2
3 public class Lotto {
4     private int[] myFeld;
5     public int[] zieheLotto() {
6         myFeld[0] = 5;
7         myFeld[1] = 35;
8         myFeld[2] = 17;
9         myFeld[3] = 49;
10        myFeld[4] = 28;
11        myFeld[5] = 7;
12        return myFeld;
13    }
14    public int liefereZahl(int position) {
15        return myFeld[position];
16    }
17    public boolean ersetzeZahl(int position, int neu) {
18        if (position > 0 && position < myFeld.length) {
19            myFeld[position] = neu;
20            return true;
21        }
22        return false;
23    }
24 }
```

Lösung: 2-JUnit Testklasse erstellen

- Lotto in Package Explorer → RM → NewOther... → JUnit → JUnit Test Case



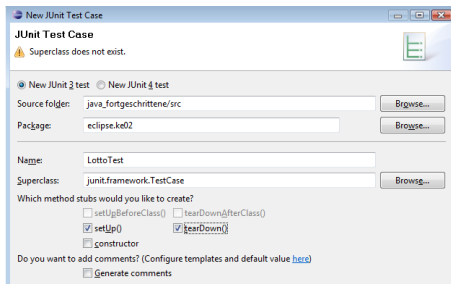
Lösung: 3- stub SetUp(),tearDown() aktivieren

- Next→ **SetUp()** aktivieren.

- **SetUp()**: wird vor jedem Methodenaufwurf implizit aufgerufen.
- **SetUp()**: wird zum Initialisieren z.B von Testobjekten benutzt.

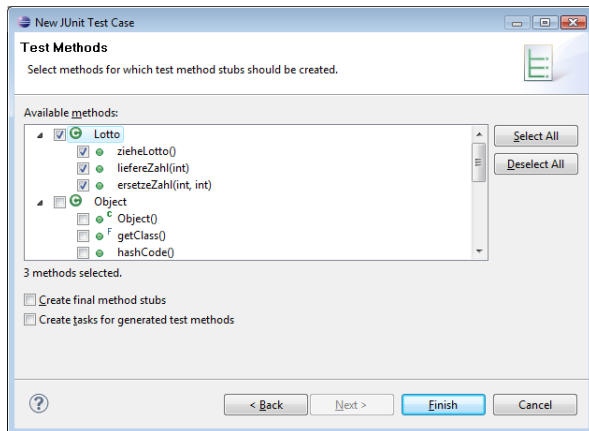
- **tearDown()** aktivieren

- **tearDown()**: wird nach jedem Methodenaufwurf implizit aufgerufen.
- **tearDown()**: wird zum Reseten benutzt. Z.B Löschen von Einträgen.



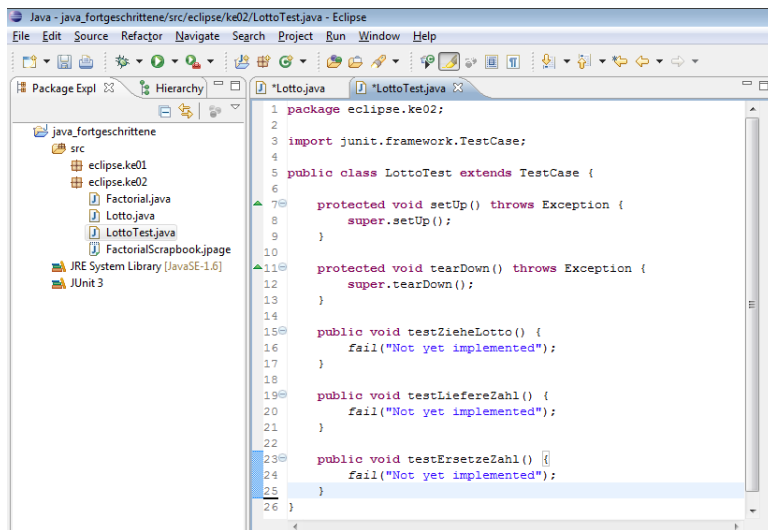
Lösung: 4-Die zu testenden Methoden auswählen

- Next → Lotto Methoden auswählen → Finish



Lösung: 5- Die LottoTest Skelett-Implementation

- Finish → Eclipse erstellt die LottoTest Skelett-Implementation.



```
1 package eclipse.ke02;
2
3 import junit.framework.TestCase;
4
5 public class LottoTest extends TestCase {
6
7     protected void setUp() throws Exception {
8         super.setUp();
9     }
10
11     protected void tearDown() throws Exception {
12         super.tearDown();
13     }
14
15     public void testZieheLotto() {
16         fail("Not yet implemented");
17     }
18
19     public void testLiefereZahl() {
20         fail("Not yet implemented");
21     }
22
23     public void testErsetzeZahl() {
24         fail("Not yet implemented");
25     }
26 }
```

Lösung: 6-Methoden-Implementation

1 `public void SetUp()`

- initialisiert ein testObjekt der Klasse Lotto.
- ruft also den Lotto Default-Konstruktor auf.

2 `public void testZieheLotto()`

- teste die zieheLotto() Methode der Lotto-Klasse
- zieheLotto() ist erfolgreich, wenn das Array ungleich null ist.
- Also ist die passende Assertion: `assertNotNull(a)`.

3 `public void testLiefereZahl()`

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die liefereZahl() Methode der Lotto-Klasse.
- liefereZahl() ist richtig, wenn sie die richtige Zahl liefert.
- Also ist die passende Assertion: `assertEquals(expected, actual)`

4 `public void testErsetzeZahl()`

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die ersetzeZahl() Methode der Lotto-Klasse.
- ersetzeZahl() ist richtig, wenn sie true liefert.
- Also ist die passende Assertion: `assertTrue(a)`

Lösung: 6-Methoden-Implementation

1 public void SetUp()

- initialisiert ein testObjekt der Klasse Lotto.
- ruft also den Lotto Default-Konstruktor auf.

2 public void testZieheLotto()

- teste die zieheLotto() Methode der Lotto-Klasse
- zieheLotto() ist erfolgreich, wenn das Array ungleich null ist.
- Also ist die passende Assertion: assertNotNull(a).

3 public void testLiefereZahl()

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die liefereZahl() Methode der Lotto-Klasse.
- liefereZahl() ist richtig, wenn sie die richtige Zahl liefert.
- Also ist die passende Assertion: assertEquals(expected, actual)

4 public void testErsetzeZahl()

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die ersetzeZahl() Methode der Lotto-Klasse.
- ersetzeZahl() ist richtig, wenn sie true liefert.
- Also ist die passende Assertion: assertTrue(a)

Lösung: 6-Methoden-Implementation

1 public void SetUp()

- initialisiert ein testObjekt der Klasse Lotto.
- ruft also den Lotto Default-Konstruktor auf.

2 public void testZieheLotto()

- teste die zieheLotto() Methode der Lotto-Klasse
- zieheLotto() ist erfolgreich, wenn das Array ungleich null ist.
- Also ist die passende Assertion: assertNotNull(a).

3 public void testLiefereZahl()

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die liefereZahl() Methode der Lotto-Klasse.
- liefereZahl() ist richtig, wenn sie die richtige Zahl liefert.
- Also ist die passende Assertion: assertEquals(expected, actual)

4 public void testErsetzeZahl()

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die ersetzeZahl() Methode der Lotto-Klasse.
- ersetzeZahl() ist richtig, wenn sie true liefert.
- Also ist die passende Assertion: assertTrue(a)

Lösung: 6-Methoden-Implementation

1 public void SetUp()

- initialisiert ein testObjekt der Klasse Lotto.
- ruft also den Lotto Default-Konstruktor auf.

2 public void testZieheLotto()

- teste die zieheLotto() Methode der Lotto-Klasse
- zieheLotto() ist erfolgreich, wenn das Array ungleich null ist.
- Also ist die passende Assertion: assertNotNull(a).

3 public void testLiefereZahl()

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die liefereZahl() Methode der Lotto-Klasse.
- liefereZahl() ist richtig, wenn sie die richtige Zahl liefert.
- Also ist die passende Assertion: assertEquals(expected, actual)

4 public void testErsetzeZahl()

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die ersetzeZahl() Methode der Lotto-Klasse.
- ersetzeZahl() ist richtig, wenn sie true liefert.
- Also ist die passende Assertion: assertTrue(a)

Lösung: 6-Methoden-Implementation

1 public void SetUp()

- initialisiert ein testObjekt der Klasse Lotto.
- ruft also den Lotto Default-Konstruktor auf.

2 public void testZieheLotto()

- teste die zieheLotto() Methode der Lotto-Klasse
- zieheLotto() ist erfolgreich, wenn das Array ungleich null ist.
- Also ist die passende Assertion: assertNotNull(a).

3 public void testLiefereZahl()

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die liefereZahl() Methode der Lotto-Klasse.
- liefereZahl() ist richtig, wenn sie die richtige Zahl liefert.
- Also ist die passende Assertion: assertEquals(expected, actual)

4 public void testErsetzeZahl()

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die ersetzeZahl() Methode der Lotto-Klasse.
- ersetzeZahl() ist richtig, wenn sie true liefert.
- Also ist die passende Assertion: assertTrue(a)

Lösung: 6-Methoden-Implementation

1 public void SetUp()

- initialisiert ein testObjekt der Klasse Lotto.
- ruft also den Lotto Default-Konstruktor auf.

2 public void testZieheLotto()

- teste die zieheLotto() Methode der Lotto-Klasse
- zieheLotto() ist erfolgreich, wenn das Array ungleich null ist.
- Also ist die passende Assertion: assertNotNull(a).

3 public void testLiefereZahl()

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die liefereZahl() Methode der Lotto-Klasse.
- liefereZahl() ist richtig, wenn sie die richtige Zahl liefert.
- Also ist die passende Assertion: assertEquals(expected, actual)

4 public void testErsetzeZahl()

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die ersetzeZahl() Methode der Lotto-Klasse.
- ersetzeZahl() ist richtig, wenn sie true liefert.
- Also ist die passende Assertion: assertTrue(a)

Lösung: 6-Methoden-Implementation

1 `public void SetUp()`

- initialisiert ein testObjekt der Klasse Lotto.
- ruft also den Lotto Default-Konstruktor auf.

2 `public void testZieheLotto()`

- teste die zieheLotto() Methode der Lotto-Klasse
- zieheLotto() ist erfolgreich, wenn das Array ungleich null ist.
- Also ist die passende Assertion: `assertNotNull(a)`.

3 `public void testLiefereZahl()`

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die liefereZahl() Methode der Lotto-Klasse.
- liefereZahl() ist richtig, wenn sie die richtige Zahl liefert.
- Also ist die passende Assertion: `assertEquals(expected, actual)`

4 `public void testErsetzeZahl()`

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die ersetzeZahl() Methode der Lotto-Klasse.
- ersetzeZahl() ist richtig, wenn sie true liefert.
- Also ist die passende Assertion: `assertTrue(a)`

Lösung: 6-Methoden-Implementation

1 `public void SetUp()`

- initialisiert ein testObjekt der Klasse Lotto.
- ruft also den Lotto Default-Konstruktor auf.

2 `public void testZieheLotto()`

- teste die zieheLotto() Methode der Lotto-Klasse
- zieheLotto() ist erfolgreich, wenn das Array ungleich null ist.
- Also ist die passende Assertion: `assertNotNull(a)`.

3 `public void testLiefereZahl()`

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die liefereZahl() Methode der Lotto-Klasse.
- liefereZahl() ist richtig, wenn sie die richtige Zahl liefert.
- Also ist die passende Assertion: `assertEquals(expected, actual)`

4 `public void testErsetzeZahl()`

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die ersetzeZahl() Methode der Lotto-Klasse.
- ersetzeZahl() ist richtig, wenn sie true liefert.
- Also ist die passende Assertion: `assertTrue(a)`

Lösung: 6-Methoden-Implementation

1 public void SetUp()

- initialisiert ein testObjekt der Klasse Lotto.
- ruft also den Lotto Default-Konstruktor auf.

2 public void testZieheLotto()

- teste die zieheLotto() Methode der Lotto-Klasse
- zieheLotto() ist erfolgreich, wenn das Array ungleich null ist.
- Also ist die passende Assertion: assertNotNull(a).

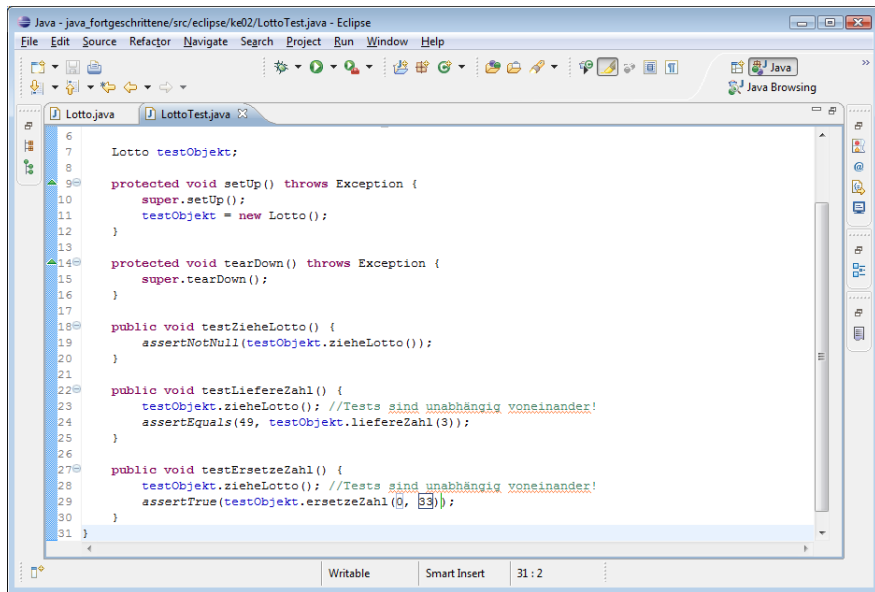
3 public void testLiefereZahl()

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die liefereZahl() Methode der Lotto-Klasse.
- liefereZahl() ist richtig, wenn sie die richtige Zahl liefert.
- Also ist die passende Assertion: assertEquals(expected, actual)

4 public void testErsetzeZahl()

- Ruft zieheLotto() auf, da die Tests unabhängig sind.
- Teste die ersetzeZahl() Methode der Lotto-Klasse.
- ersetzeZahl() ist richtig, wenn sie true liefert.
- Also ist die passende Assertion: assertTrue(a)

Lösung: 6-Methoden-Implementierung



```
Java - java_fortgeschrittene/src/eclipse/ke02/LottoTest.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

LottoTest.java
6
7    Lotto testObjekt;
8
9    protected void setUp() throws Exception {
10        super.setUp();
11        testObjekt = new Lotto();
12    }
13
14    protected void tearDown() throws Exception {
15        super.tearDown();
16    }
17
18    public void testZieheLotto() {
19        assertNotNull(testObjekt.zieheLotto());
20    }
21
22    public void testLiefereZahl() {
23        testObjekt.zieheLotto(); //Tests sind unabhängig voneinander!
24        assertEquals(49, testObjekt.liefereZahl(3));
25    }
26
27    public void testErsetzeZahl() {
28        testObjekt.zieheLotto(); //Tests sind unabhängig voneinander!
29        assertTrue(testObjekt ersetzeZahl(0, 33));
30    }
31 }
```

Writable Smart Insert 31:2

Lösung: 7- JUnit ausführen

- Run → Run As → 1 Junit Test

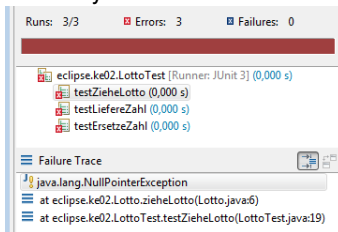
The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Displays the test results for 'eclipse.ke02.LottoTest'. It shows three tests: 'testZieheLotto (0,000 s)', 'testLiefereZahl (0,000 s)', and 'testErsetzeZahl (0,000 s)'. The 'Failure Trace' section shows a 'java.lang.NullPointerException' at 'eclipse.ke02.Lotto.zieheLotto(Lotto.java:6)' and 'eclipse.ke02.LottoTest.testZieheLotto(LottoTest.java:19)'.
- JUnit View:** Shows the test results summary: 'Finished after 0,063 sec', 'Runs: 3/3', 'Errors: 3', and 'Failures: 0'.
- Source Editor:** Displays the source code of 'LottoTest.java'. The code includes imports for 'junit.framework.TestCase', a class declaration 'public class LottoTest extends TestCase', and methods 'testObject', 'setUp', and 'tearDown'.
- Console:** Shows the output of the test run: '<terminated> LottoTest [JUnit] C:\Prc'.

Lösung: 8- Fehler korrigieren

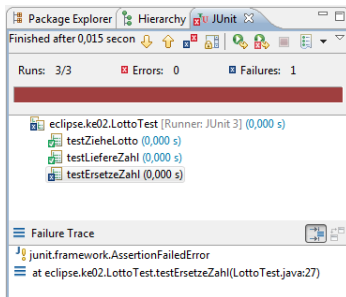
• Fehler im Code korrigieren

- Failure Trace gibt gute Hinweise über den Fehler
- `java.lang.NullPointerException`
- at `eclipse.ke02.Lotto.zieheLotto(Lotto.java:6)`
- TIPP: `myFeld` muss vorher initialisiert werden!



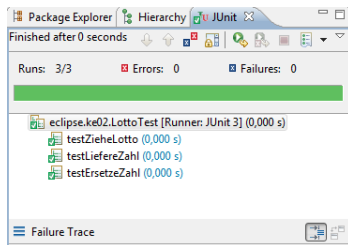
Lösung: 9- JUnit Test erneut durchführen

- Fehler im Code korrigieren
 - `private int myFeld = new int[6];`
- JUnit Test erneut durchführen
 - Run → Run As → 1 Junit Test



Lösung: 9- JUnit Test erneut durchführen

- Fehler im Code korrigieren
 - if (position >= 0 && position < myFeld.length)
- JUnit Test erneut durchführen
 - Run → Run As → 1 Junit Test



Programmieraufgabe: JUnit Test

- **Programmieren Sie für Factorial die JUnit Testklasse**
 - Implementieren Sie die Testmethode `testCalculateFactorial(int number)`
 - Benutzen Sie dafür Testfälle aus den Äquivalenzklassen.
 - Äquivalenzklassen sind z.B.: $\{[1.., -1]\}$, 0 , $[1, ..]$
 - Testdaten: `number = -5, 0, 3` (Vertreter aus den Äquivalenzklassen)
- **Erstellen Sie ein JUnit Test Suite**
 - Zielpackage \rightarrow RM \rightarrow New \rightarrow other \rightarrow JUnit \rightarrow JUnit Test Suite
 - Führen Sie Ihr JUnit Test Suite aus.