Testen

Zentrale Konzepte:

- ⊙ Modultests
  - ⊙ Sammlungen testen
  - ⊙ Positives Testen
  - ⊙ Negatives Testen
- ⊙ Tests automatisieren
  - ⊙ Regressionstests
- ⊙ Testen mit Junit
  - ⊙ Testfälle
  - ⊙ Zusicherungen
  - ⊙ Testgerüste

```
public class TagTest
{
    private Tag tag1;
    private Verabredung termin1;
    private Verabredung termin2;
    private Verabredung termin3;

    @Before
    protected void setUp()
    {
       tag1 = new Tag(1);
       termin1 = new Verabredung("Trudi", 1);
       termin2 = new Verabredung("Hans", 1);
       termin3 = new Verabredung("Mama", 1);
    }

    @Test
    public void dreiTermineVereinbaren()
    {
       assertEquals(true,tag1.setzeTermin(9, termin1));
       assertEquals(true,tag1.setzeTermin(13, termin2));
       assertEquals(true,tag1.setzeTermin(17, termin3));
    }
}
```


Debugging: Fehlerursachen finden

Zentrale Konzepte:

- ⊙ Mündliche Ausführung
- ⊙ Ausgabeanweisungen
- ⊙ Debugger
- ⊙ Dokumentation von Klassen

Mündliche Ausführungen

- Man selber erklärt einer anderen Person den Code

  => Der Zuhörer entdeckt Fehler

  => Man selber entdeckt Fehler beim Erklären

- Für solche Codeinspektionen gibt es formale gruppenbasierte Verfahren, die in großen Softwareprojekten eingesetzt werden.

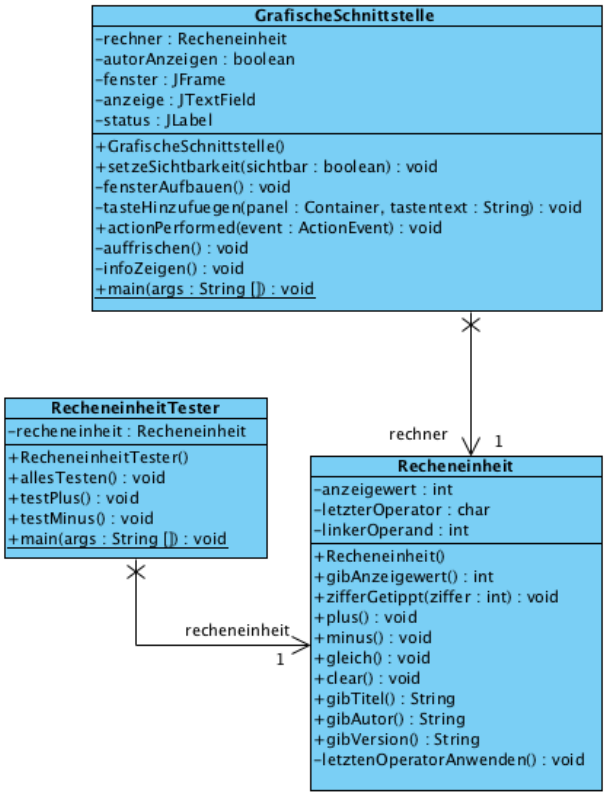
Projekt *Taschenrechner*

Klasse *Recheneinheit*: enthält Methoden, die Operationen eines Taschenrechners simuliere

- *zifferGetippt()*
- *clear()*
- *plus()*
- *gleich()*
- *minus()*

Klasse *GrafischeSchnittstelle*:

- nimmt Nutzereingaben entgegen
- ruft die zugehörigen Methoden der Klasse Recheneinheit auf.



Ausgabeanweisungen

- Wird sehr häufig eingesetzt.
- Man versieht sein Programm mit Ausgabeanweisungen, die zusätzliche Informationen während des Ablaufs ausgeben:
  - Welche Methoden aufgerufen wurden
  - Werte von Parametern
  - Reihenfolge der Methodenaufrufe
  - Werte von Attributen an wichtigen Stellen
- Man benötigt keine besonderen Werkzeuge und kann damit in allen Programmiersprachen arbeiten.
- Wesentlich ist, dass man die richtigen Methoden an den richtigen Stellen mit den wesentlichen Ausgabeanweisungen versieht.

Bsp: Ausgabeanweisungen in der Klasse *Recheneinheit*

Ausgabe signalisiert den Start der Methode und gibt Eingabewert an.

```
public void zifferGetippt(int ziffer)
{
   System.out.println("Aufruf: zifferGetippt mit: " + ziffer);

   anzeigewert = anzeigewert * 10 + ziffer;

   System.out.println( "linker Operand: "+ linkerOperand +
                       " letzter Operator: " + letzterOperator +
                        " anzeigewert: " + anzeigewert +
                        " am Ende von zifferGetippt.");
}
```

Ausgabe gibt den Zustand des Objekts (Attributwerte) nach Ausführung der Methode aus.

Ausgabeanweisung für den Zustand des Objekts wird potentiell in jeder Methode benötigt.

=> Private Methode für die Ausgabe des Objektzustands bereitstellen

Der Name der Methode wird am Aufrufort als Argument übergeben.

```
private void zustandAusgeben(String methodenName)
{
    System.out.println( "linker Operand: "+ linkerOperand +
                        " letzter Operator: " + letzterOperator +
                        " anzeigewert: " + anzeigewert +
                        " am Ende von " + methodenName);
}
```

```
public void zifferGetippt(int ziffer)
{
   System.out.println("Aufruf: zifferGetippt mit: " + ziffer);

   anzeigewert = anzeigewert * 10 + ziffer;

   zustandAusgeben("zifferGetippt");
}
```

Hier wird die Ausgabemethode aufgerufen.

Nachteile von Ausgabeanweisungen

- Ausgabeanweisungen sind nur dann nützlich, wenn die richtigen Methoden an den richtigen Stellen mit den richtigen Ausgaben angereichert werden.
- Zu viele Ausgabeanweisungen führen zu einer Informationsflut. Insbesondere bei Ausgabeanweisungen in Schleifen muss man vorsichtig sein.
- Sobald sie ihren Zweck erfüllt haben, kann es recht umständlich sein, sie wieder zu entfernen.
- Es kann auch sein, dass man nach dem Entfernen merkt, dass man sie doch wieder benötigt ☹.

Ein- und Ausschalten von Ausgabeanweisungen

Durch ein zusätzliches Attribut *testausgaben* vom Typ *boolean* und das Auslagern der Ausgabe in Methoden, lassen sich Ausgabeanweisungen nach Belieben ein- und ausschalten.

```
public class Recheneinheit {
    ...
    private boolean testausgaben = false;

    ...

    private void testausgabe(String info) {
      if(testausgaben) {
          System.out.println(info);
      }
    }

    private void zustandAusgeben(String methodenName) {
      if(testausgaben) {
          System.out.println( "linker Operand: "+ linkerOperand +
                              " letzter Operator: " + letzterOperator +
                              " anzeigewert: " + anzeigewert +
                              " am Ende von " + methodenName);
      }
    }
}
```

Über dieses Attribut lassen sich die Ausgabeanweisungen ein- und ausschalten.

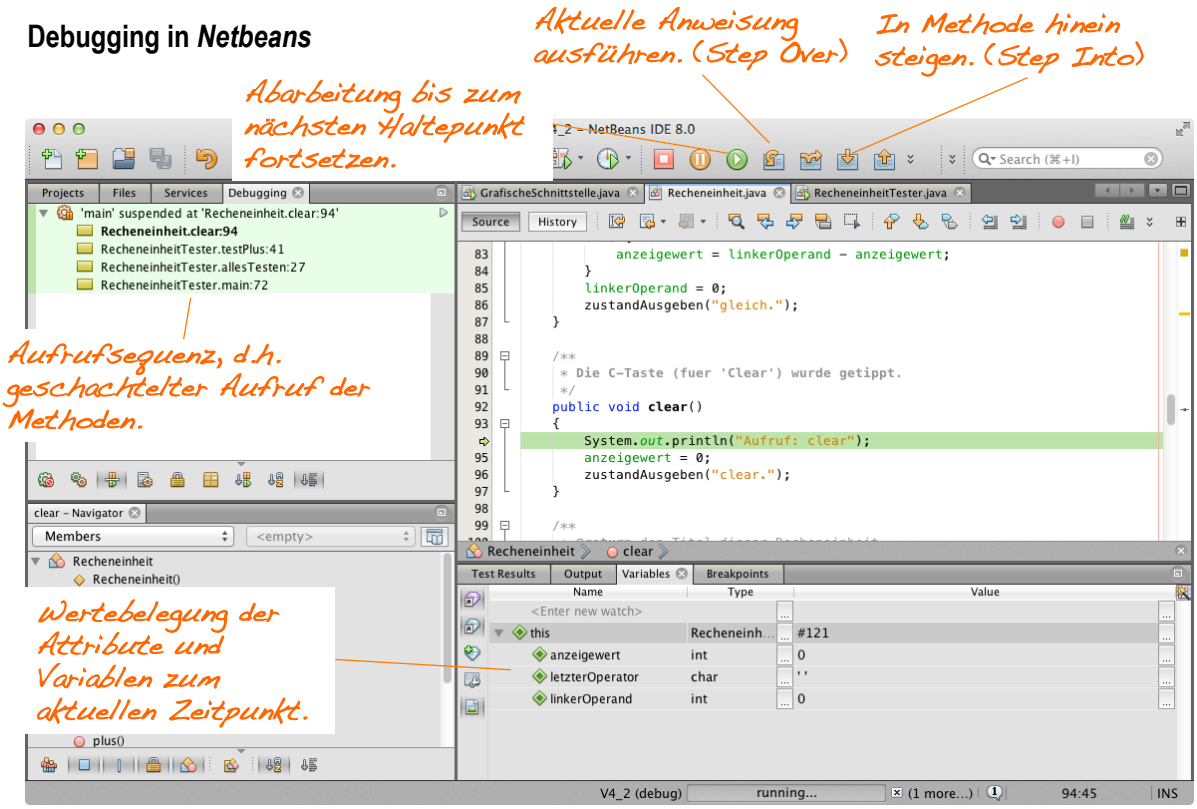
Mit dieser Methode lassen sich beliebige Zeichenketten ausgeben.

Mit dieser Methode lässt sich der Zustand des Objekts ausgeben.

Debugger

- Ermöglicht es, Anweisungssequenzen Schritt für Schritt zu durchlaufen und Methodenaufrufen zu folgen.
- Um an einer bestimmten Stelle des Programms mit der schrittweisen Ausführung zu beginnen, muss ein Haltepunkt gesetzt werden.
- Ermöglicht den Zugriff auf die Werte aller Variablen der Klasse, d.h. insbesondere auf die Attribute (Zustand) des Objekts.
- Liefert Informationen über die Aufrufsequenz zu jedem Zeitpunkt. Die Schachtelung der Methodenaufrufe wird hier deutlich.
- Die Werte der lokalen Variablen in den Methoden der Aufrufsequenz kann abgerufen werden.
- Kann in *Netbeans* aktiviert werden über die Auswahl der Klasse mit der main-Methode und im Kontextmenü: *Debug File*

Debugging in *Netbeans*



Kommentierung von Klassen

- Das Bearbeiten von Programmen anderer setzt voraus, dass man das Programm versteht.
- Das Verstehen eines Programms ist allein auf der Basis der Quellcodes oft sehr schwierig.
- Daher sollte das Programm von der Programmiererin ausführlich kommentiert werden.
- Die Kommentare sollten alle Bestandteile einer Klasse nachvollziehbar beschreiben.
- Einzeilige Kommentare können mit `//` beginnen, mehrzeilige Kommentare sollten in `/*` und `*/` eingeschlossen sein.

```
/*
 * Wird aufgerufen, wenn eine Zifferntaste getippt wurde.
 *
 */
public void zifferGetippt(int ziffer) {
        //anzeigewert um 10er Stelle verschieben und ziffer addieren
        anzeigewert = anzeigewert * 10 + ziffer;
}
```

Dokumentation von Klassen

- Das Nutzen von anderen Klassen setzt voraus, dass man die von Ihr angebotenen Dienstleistungen (Methoden) versteht.
- Für die Nutzung reicht es aus, die Schnittstelle der Klasse zu kennen, die Implementierung muss nicht bekannt sein.
- Eigene Klassen in einem größeren Projekt sollten dokumentiert werden wie die Java-Bibliotheksklassen.
- Die Dokumentation einer Klasse sollte genau die Informationen bieten, die andere Programmierer benötigen, um die Klasse ohne einen Blick auf Ihre Implementierung nutzen zu können

Elemente einer Dokumentation

Die Dokumentation einer Klasse sollte die folgenden Informationen umfassen:

- den Klassennamen
- Einen Kommentar, der den allgemeinen Zweck und die Eigenschaften der Klasse beschreibt.
- eine Versionsnummer.
- Den/die Autorennamen
- Eine Dokumentation für jeden Konstruktor und jede Methode, d.h.
    - den Namen der Methode/des Konstruktors
    - Eine Beschreibung des Zwecks und der Arbeitsweise der Methode
    - Den Ergebnistyp und eine Beschreibung des Ergebnisses
    - Namen und Typen der Parameter, sowie eine Beschreibung der Parameter

Javadoc Kommentare

- Java kommt mit dem Werkzeug *javadoc*, mit dem sich aus den Kommentaren aus dem Quellcode automatisch eine HTML-Dokumentation der Klassen erstellen lässt.
- *Javadoc*-Kommentare beginnen mit `/**` , enden mit `*/` und enthalten sog. *Javadoc-Tags* die dazu dienen z.B. Klassen, Methoden und Attribute näher zu beschreiben.
- *Javadoc-Tags* beginnen mit @. Hier ein Ausschnitt an Tags:

Tag & Parameter	Ausgabe	Verwendung in
`@author` name	Beschreibt den Autor.	Klasse
`@version` version	Gibt die Versionsnummer an	Klasse
`@see` referenz	Erzeugt einen Link auf ein anderes Element der Dokumentation.	Klasse, Attribut, Methode
`@param` name beschreibung	Parameterbeschreibung einer Methode.	Methode
`@return` beschreibung	Beschreibung des Rückgabewerts einer Methode.	Methode

Javadoc in *Netbeans* nutzen

- Die Dokumentation für ein Projekt lässt sich bei Auswahl des Projekts mit *Generate JavaDoc (*Kontextmenü*)* erstellen. Die Dokumentation wird im Projektordner erzeugt (Ordner *dist/javadoc*).
- Wählt man einen Klassennamen oder eine Methode im Editor aus und wählt im Kontextmenü *Show Javadoc*, so wird die Dokumentation des gesamten Projekts (sofern man diese vorher erstellt hat) im Webbrowser angezeigt.
- Mit *Window->IDE Tools->Javadoc Documentation* aus dem Hauptmenü lässt sich unter dem Editor ein Reiter öffnen, in dem die Dokumentation der im Editor gewählten Methode eingeblendet wird.