Interreg
EUROPÄISCHE UNION
Österreich-Tschechische Republik
Europäischer Fonds für regionale Entwicklung

INFORMATIK

Objektorientierte Programmierung mit Java

UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA
VŠTE
EUROPÄISCHE UNION

Inhalt

1. KLASSEN

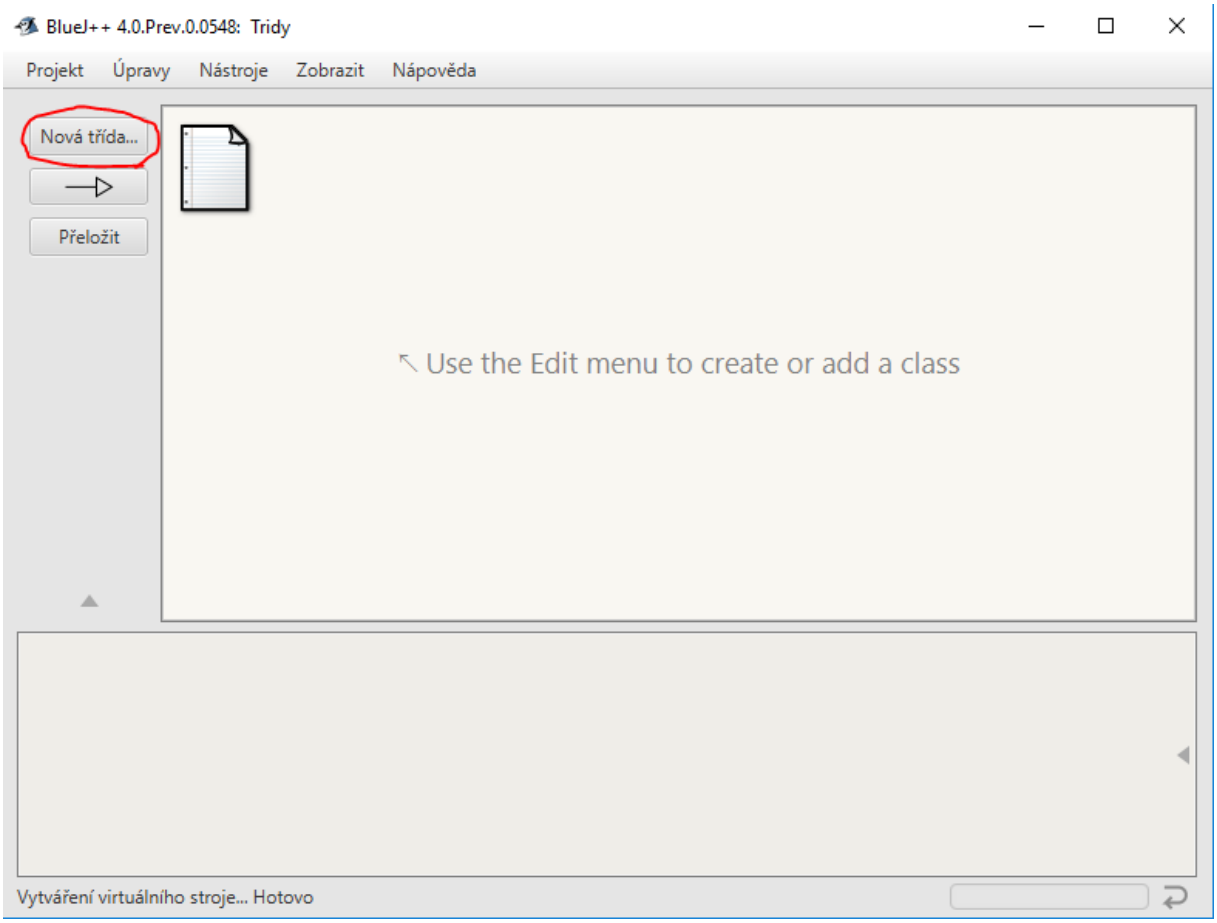
Im wirklichen Leben zum Beispiel wird das Wort Stuhl als Möbelstück bezeichnet. Die Funktion ist, darauf zu sitzen. Es handelt sich um eine Beschreibung des Objekts, die durch verschiedene Attribute oder Funktionen gekennzeichnet ist. Eine Analogie in der Programmierung ist eine Klasse. Die Klasse gruppiert Objekte mit einigen gemeinsamen Eigenschaften. Im wirklichen Leben gibt es viele verschiedene Stühle, die in Material und Farbe variieren können. Bei der Programmierung entspricht ein bestimmter Stuhl einer Instanz der entsprechenden Klasse, manchmal auch als Objekt bezeichnet. So kann die Klasse mit einem Formular verglichen werden, in das eine bestimmte Sache instanziiert wird. Typischerweise können Klassen beliebig viele Instanzen haben.

Einzelne Objekte können miteinander kommunizieren, sich gegenseitig verschiedene Nachrichten senden, in denen sie verschiedene Informationen oder Dienste anfordern können. Ein Beispiel kann ein Taschenrechner sein, den wir für viele Aufgaben benötigen, wie z.B. das Hinzufügen von zwei Zahlen. Jede der Funktionen dieses Rechners wird professionell als Methode bezeichnet. Das Anfordern der Verwendung einer bestimmten Methode wird als Methodenaufruf bezeichnet. Somit ist die Methode Teil des Programms, das eine Instanz als Antwort auf einen Methodenaufruf verwendet (eine Nachricht empfangen). Der Methoden-Builder definiert, wie das Objekt auf die Nachricht reagieren soll.

Das gesamte objektorientierte Programm Pecinovsky (2004) beschreibt als in einer Programmiersprache geschrieben eine Beschreibung der verwendeten Klassen, Objekte und Nachrichten, die diese Objekte senden, ergänzt durch komplexere Programme, indem es die Platzierung von Programmen auf einzelnen Computern beschreibt und diese der Verwaltung der jeweiligen Dienstprogramme zuordnet (z.B. Betriebssystem oder Anwendungsserver).

1.1. Erstellen von Klassen

Jetzt werden wir die Klassen selbst bauen. Also eine Art von Mustern, die von einer bestimmten Instanz (Objekt) erzeugt werden. Im BlueJ-Programm, das wir während dieses Kurses einsetzen werden, wählen wir ein neues Projekt aus. Wählen Sie dann die Schaltfläche auf der linken Seite, wie im Bild unten gezeigt.



Im sich öffnenden Fenster wählen wir die einfachste mögliche Definition einer Klasse - leere Klasse. Dann wählen wir den Namen der Klasse. In unserem Projekt haben wir uns für ErsteKlasse entschieden.

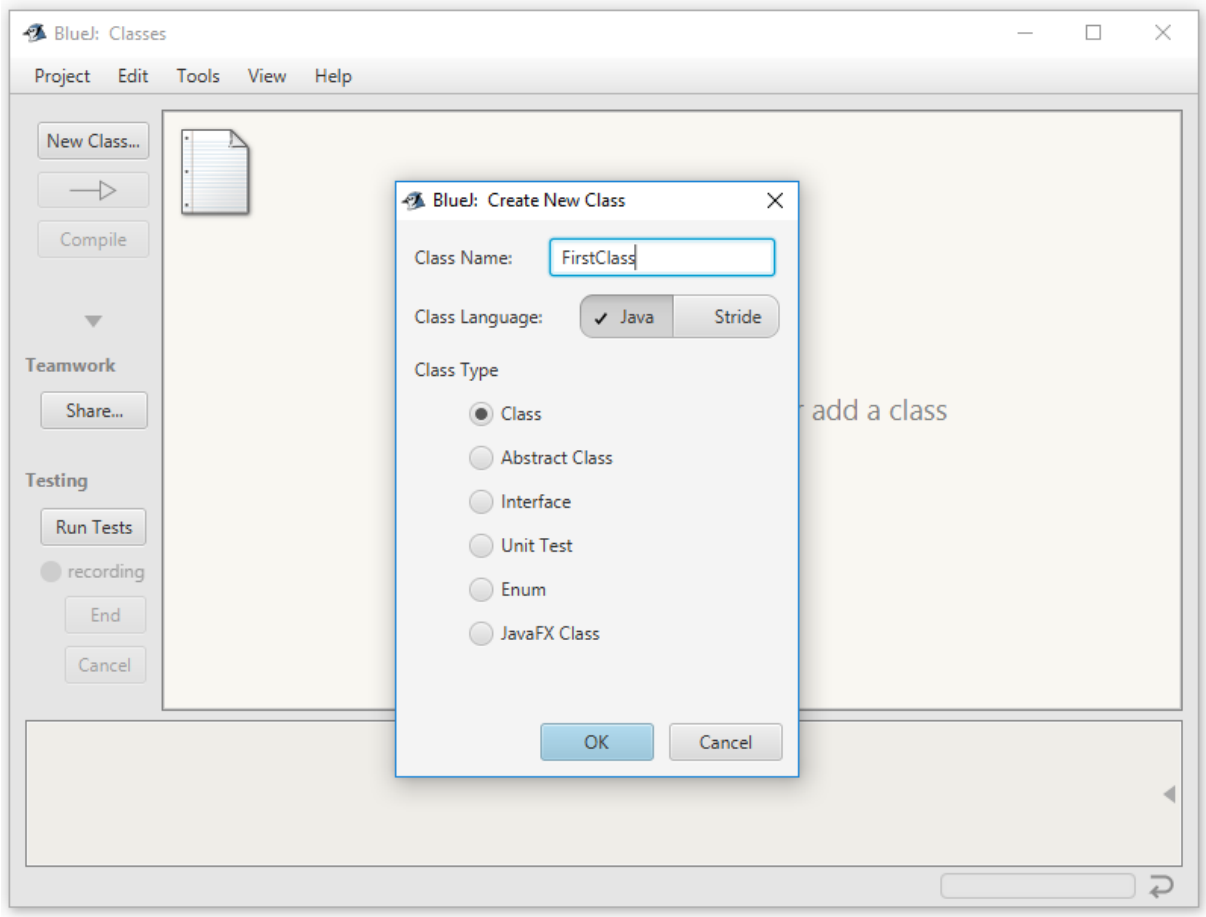
Der Name der Klasse hängt von einigen Faktoren ab:

Regeln für die Erstellung eines Identifikators

Links, Klassen und andere müssen für die korrekte Identifizierung benannt werden. Diesen Namen nennen wir Identifikatoren. Diese Identifikatoren müssen eine Reihe von Regeln erfüllen:

- Es kann alle Zeichen enthalten, die in einem Satz von UNICODE enthalten sind.
- Lücken dürfen nicht verwendet werden.
- Es ist üblich, mehrwertige Klassennamen ohne Leerzeichen zu schreiben. Einzelne Wörter beginnen dann mit Großbuchstaben wie: MeineErsteKlasse
- Groß- und Kleinschreibung werden als unterschiedlich angesehen.
- Die Länge ist unbegrenzt
- Es darf nicht mit einer Ziffer beginnen.

- Kann nicht mit den Keywords übereinstimmen



Nach der Erstellung und Übersetzung haben wir die erste Klasse. Wenn wir in den Ordner schauen, in dem wir das gesamte Projekt gespeichert haben, werden wir feststellen, dass sich mehrere Dateien im Ordner befinden.

FirstClass.java:	Wir nennen diese Datei Quelldatei. Wir werden ein Programm schreiben, welches das Verhalten der Klasse und damit ihre Instanzen beschreibt. Diese Datei ist nur eine Textdatei. Sie können es in jedem beliebigen Texteditor bearbeiten.
FirstClass.class:	Ein übersetzter Bytecode wird in dieser Datei gespeichert.
FirstClass.ctxt BlueJ:	zusätzliche Datei. Wenn Sie diese Datei löschen, wird BlueJ sie bei der nächsten Übersetzung erneut erstellen.

1.2. Arbeiten mit der Klasse

Nach einem Doppelklick auf die Klasse öffnen sich ein Textfenster, in dem Sie den Quellcode der Klasse eingeben oder bearbeiten können. Siehe Bild unten:

Im Textfenster werden wir sehen:

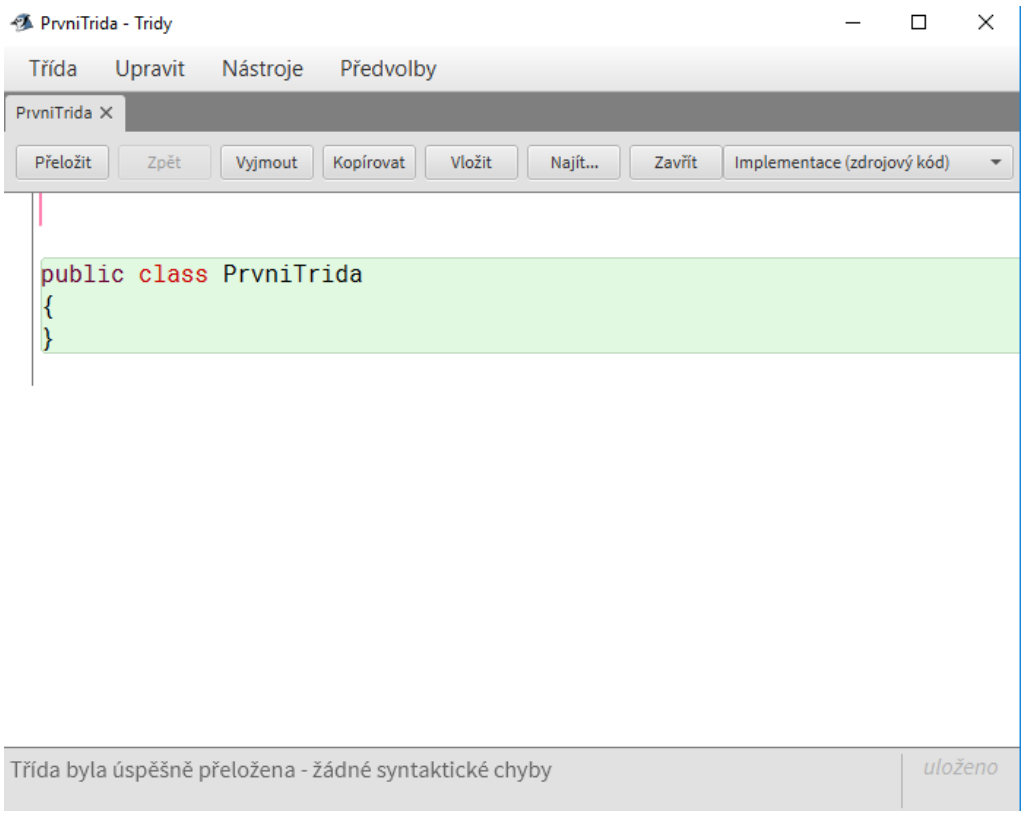
- Klassendefinition FirstClass
- Konstrukteur FirstClass ()
- Methode sampleMethod()

Die Klassendefinition enthält Wörter

Public: Ein Schlüsselwort, das angibt, dass jeder mit dieser Klasse arbeiten kann.

Class Ein Schlüsselwort, das die Klassendefinition angibt.

FirstClass: Klassenname (sein Identifikator)
Der Inhalt der Klasse ist in den folgenden Klammern enthalten. In unserem Beispiel enthält sie noch keine Informationen.



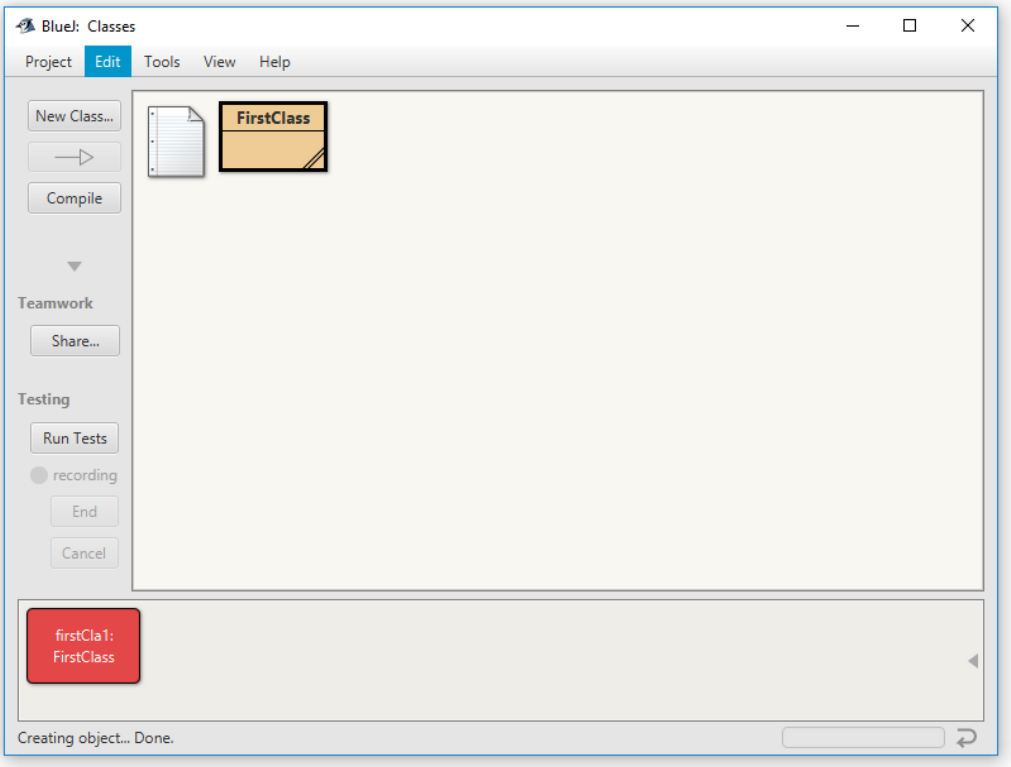
1.3. Erstellen einer ersten Instanz

Schließen Sie das Textfenster. Wenn Sie mit der rechten Maustaste auf unsere erste Klasse klicken. Achtung! Die betreffende Klasse muss kompiliert werden! Wählen Sie im Popup-Menü den neuen Befehl ErsteKlasse (). BlueJ wird uns dann nach dem Namen der zu erstellenden Instanz fragen. Gleichzeitig werden wir es vor dem ausgewählten Namen anbieten. Alles, was du tun musst, ist zu genehmigen. So haben wir die erste Klasse erstellt. Wie im Bild unten gezeigt.

Durch Auswählen des neuen Befehls ErsteKlasse () erstellen wir eine Instanz der Klasse, indem wir eine Nachricht senden, die aus dem neuen Schlüsselwort kompiliert wurde, gefolgt vom Namen der Klasse, aus der wir ein Paar runde Klammern erstellen möchten. Also rufen wir eine spezielle Methode namens Konstruktor auf. Der Konstruktor erstellt die angeforderte Instanz und gibt einen Link zurück, über den wir die erstellte Instanz ansprechen.

In unserem Fall haben wir keine Konstruktormethode konstruiert. Aber der Konstruktor muss jede Klasse haben! Wenn wir keinen Konstruktor erstellen, erstellt der Compiler automatisch den einfachsten Konstruktor, den wir bestimmen - den Standardkonstruktor. Im Moment erstellen wir den ersten Konstruktor in der Klasse, der Compiler stoppt mit der Erstellung des impliziten Konstruktors.

Die erzeugte Instanz der Klasse ist in der folgenden Abbildung dargestellt. Es ist unten, in dem Bereich, den wir die Objektbank nennen. Die Instanz der Klasse ist rot.



1.4. Eine Klasse entfernen

Sehr einfach. Klicken Sie mit der rechten Maustaste auf die Klasse und wählen Sie Löschen. Nach der Bestätigung wird die Klasse tatsächlich gelöscht. Die Instanz der Klasse selbst blieb jedoch erhalten. Die Anforderung, eine Klasse zu löschen, ist eigentlich eine Anforderung, die entsprechenden Dateien von der Festplatte zu löschen. Die Klasseninstanz selbst wird im Speicher abgelegt. Wenn wir die Instanz der Klasse entfernen wollen, müssen wir die virtuelle Maschine neu starten.

1.5. Neustart der virtuellen Maschine

Beim Neustart der virtuellen Maschine werden alle Referenzen im Link-Stapel gelöscht. Also löschen wir alle Instanzen. Wir können dies entweder mit der Tastenkombination Strg + Umschalt + r oder durch einen Rechtsklick auf das Rechteck unten rechts und die Auswahl des Befehls tun: Starten Sie die virtuelle Maschine neu.

2. KONSTRUKTEURE

Im vorherigen Kapitel hat der Compiler einen impliziten Konstruktor für uns erstellt. Nun zeigen wir Ihnen, wie Sie Ihre eigenen Konstruktoren erstellen können.

2.1. Nichtparametrischer Konstruktor

Erstellen Sie eine Klasse “Student” ähnlich derjenigen, die wir gelöscht haben. Jetzt erstellen wir einen nicht-parametrischen Konstruktor. Nichtparametrisch bedeutet, dass es für seinen Lauf keine Informationen - Parameter - benötigt. Die Klassen- und Konstruktordeklaration selbst könnte in etwa so aussehen. Der Konstruktor selbst ist gelb:
public class Student

```
    {
          int math =3;
                int english =3;
                int ICT =3;
          public Student ()
          {
                .........
          }
    }
```

Nun gehen wir die Bedeutung der einzelnen Textteile durch.

Wir haben zuerst die Klassen Student und Variable Mathematik, Englisch und ICT erklärt.

Der Header des Konstruktors sieht dem Klassenkopf sehr ähnlich. Das Keyword class wird weggelassen. Der Name des Konstruktors muss mit dem Klassennamen übereinstimmen, in dem der Konstruktor aufgeführt ist. Hier sind die Klammern, in die die Parameter geschrieben werden, dann der parametrische Konstruktor, wenn keine Klammern geschrieben werden, wird es ein parametrischer Konstruktor sein. Der Körper des Konstruktors selbst steht in Klammern.

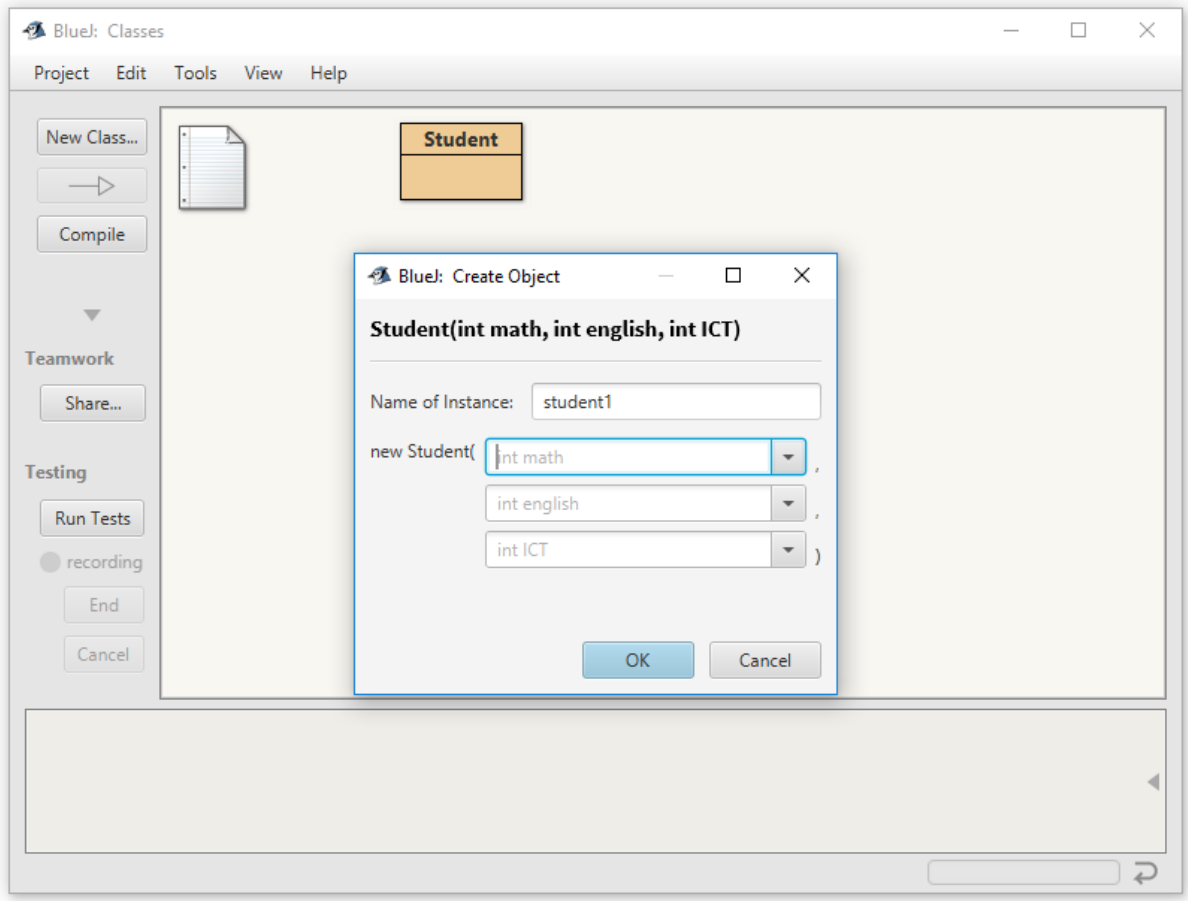
2.2. Konstruktor mit Parametern

Wenn wir eine Studenteninstanz wären, hätte jeder Student die gleichen Noten. Deshalb erstellen wir diesmal einen anderen Konstruktor. Diesmal mit den Parametern.

```
public class Student
{
      int math =3;
      int english =3;
      int ICT =3;
      public Student ()
      {
            .........
      }
      public Student (int math,int english,int ICT)
      {
            .........
      }
}
```

Der neue parametrische Konstruktor wurde gelb markiert. Beide Konstruktoren haben den gleichen Namen. Der Übersetzer unterscheidet sie nach Anzahl und Art der Parameter. Dem parametrischen Konstruktor geben wir zunächst den Typ und damit die Kennung an, mit der das Programm den Parameter im Körper des Konstruktors aufruft. Daher können wir keine Konstruktoren mit gleichen Parametertypen konstruieren. Der Übersetzer unterscheidet nicht einmal die Rückgabewerte für einzelne Konstruktoren. Die Verwendung mehrerer Konstruktoren wird als Überladung bezeichnet.

Nun, da wir eine neue Instanz erstellen wollen - einen Schüler mit einem parametrischen Konstruktor -, werden wir aufgefordert, ganzzahlige Parameter einzugeben, wie in der folgenden Abbildung gezeigt.



2.3. This

Viele Konstrukteure sind einander ähnlich. Häufig wollen wir in einem Konstruktor einen anderen Konstruktor verwenden. Dieses Umschreiben des Body-Konstruktors kann durch die Verwendung des “This” Schlüsselwortes vermieden werden, gefolgt von einer Liste von Parametern. Hüten Sie sich davor, einen anderen Konstruktor aufzurufen, indem Sie This verwenden. This muss der allererste Konstruktorbefehl sein! Die Verwendung des This Konstruktors ist ein weiteres Beispiel.

```
public class Student
{
      int math =3;
      int english =3;
      int ICT =3;
      public Student ()
      {
            this (0, 0, 0);
      }
      public Student (int math,int english,int ICT)
      {
            .........
      }
}
```

3. METHODEN

Die Methode ist ein spezifisches Unterprogramm, das eine bestimmte Funktion ausführt. Auch ist es eines der am häufigsten verwendeten Werkzeuge (fast) jeder Programmiersprache. Wir könnten die Methoden mit den Werkzeugen vergleichen, mit denen dann jede Instanz der Klasse ausgestattet ist. Das Verfahren selbst besteht aus mehreren Teilen:

- Zugriffsspezifikation - die bestimmt, wer die Methode aufrufen darf. Die häufigsten sind öffentlich und privat. Der Spezifikator ist optional.
- Art des Rückgabewertes - Obligatorisch. Wenn die Methode nicht zurückkehrt, schreiben wir void.
- Methodenname - die gleichen Regeln wie in Konstruktoren
- Methodenparameterliste - auch das gleiche Prinzip wie bei den Konstruktoren

Der Körper des Verfahrens selbst ist in Verbundklammern eingeschlossen. Wo wir einzelne Befehle schreiben können. Wenn wir wollen, müssen wir nichts in den Körper schreiben.

```
public void Hello()
{
      System.out.println(”Hello”);
}
```

3.1. Methoden, die einen Wert zurückgeben

Wenn die Methode einen Wert zurückgeben soll, müssen wir ihren Typ in der Kopfzeile angeben, und im Body der Methode müssen wir die Anweisung zurückgeben, gefolgt von der Variablen, die wir zurückgeben möchten. Nach der Return-Anweisung wird die Methode sofort beendet. Daher wird der Code, der auf die Return-Anweisung folgt, nicht im Body der Methode ausgeführt. Das Beispiel ist unten dargestellt.

```
public double averageGrade ()
{
      double average= (math + english + ICT)/3;
      return average;
}
```

3.2. Aufruf von Methoden

Durch einfaches Schreiben von Codes wird kein Code ausgeführt. Es wird geschehen, wenn wir es bestimmen. Sie können die Methode nur an der Stelle aufrufen, an der die Methode zugänglich ist. Rufen Sie die Methode durch Eingabe des Methodennamens auf und geben Sie die Parameter der Methode in Klammern ein. Wenn die Methode in einer anderen Klasse aufgeführt ist, müssen wir zuerst die Klasse benennen. Der Methodenname wird durch einen Punkt getrennt. Wenn wir eine Instanznachricht senden, müssen wir zuerst eine Referenz auf diese Instanz schreiben. Bei Attributen ist die Situation ähnlich.

3.3. Parameterübergabe an Methoden

Werte primitiver Typen wie Zeichen, logische Werte oder Zahlen werden übergeben, so dass der Wert in die lokale Methodenvariable kopiert wird.

Objekttypwerte werden über die Verknüpfung übergeben. Daher ein Link auf das Objekt, in dem der Aktualparameter gerade in die lokale Methodenvariable kopiert wird.

4. STATISCHE ATTRIBUTE

Statische Elemente gehören zu einer Klasse und nicht zu einer Instanz. Statische Attribute werden mit "static" bezeichnet. Da es zu einer Klasse gehört, haben alle Methoden dieser Klasse Zugriff darauf. Wir können statische Attribute lesen, auch wenn es keine Klasseninstanz gibt. Die Deklaration der statischen Attribute ist unten aufgeführt und gelb markiert.

```
class Group {
      private static int number = 15;
      public void New(int count) {
            number = number + count;
      }
}
```

4.1. Statische Klassen

Klassenmethoden werden in der Klasse aufgerufen. Dies sind die Hilfsmethoden, die wir oft verwenden, aber wir wollen keine Instanz speziell für diesen Zweck erstellen. Als Beispiel kann eine statische Methode verwendet werden, um zu testen, ob eine bestimmte Zahl ein Plus ist.

```
public static boolean isPlus(int number) {
      if (number >= 0) {
            return true;
      }
      return false;
}
```

Achtung! Da die statische Methode zur Klasse gehört, können wir nicht auf Instanzattribute in ihr zugreifen. Diese Attribute existieren nicht innerhalb einer Klasse, sondern innerhalb einer Instanz.

4.2. Lokale Variablen

Manchmal müssen wir uns an etwas in der Methode erinnern. Zu diesem Zweck werden lokale Variablen verwendet. Wir deklarieren sie innerhalb der Methode. Über die Methode hinaus kann nicht auf sie zugegriffen werden. Dies ermöglicht es uns, eine weitere lokale Variable mit dem gleichen Namen in einer anderen Methode zu definieren. Wir verwenden in ihrer Deklaration keine Zugriffsmodifikatoren (z.B. öffentlich und privat) oder static. Wir müssen ihrer Deklaration einen gewissen Wert beimessen. Beim Verlassen der Methode wird die lokale Variable gelöscht. Deshalb gibt es in ihnen nichts, was wir zwischen den verschiedenen Methoden brauchen. Häufige Gründe für die Verwendung lokaler Variablen sind, das Programm transparenter zu machen und die Anzahl der Fehler zu reduzieren, die durch die wiederholte Typisierung komplexer Ausdrücke verursacht werden. Die Deklaration der lokalen Variablen erfolgt wie im folgenden Beispiel:

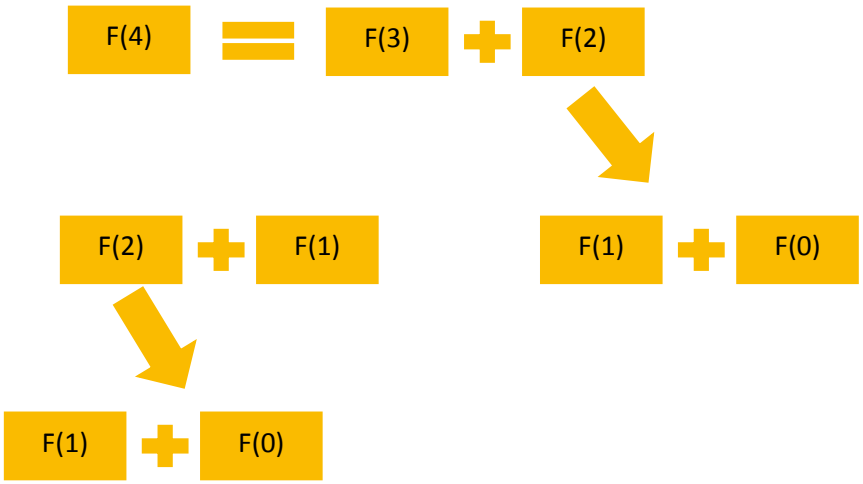
```
public void totalPrice (int pieces)
{
      int totalPrice = pieces *15;
}
```

4.3. Rekursion

Wiederkehrend ist die Definition eines Objekts (mathematisch verstanden) durch dich selbst. Rekursive Merkmale müssen einen Mechanismus beinhalten, der die Rekursion zu gegebener Zeit beendet. Wenn das nicht geschehen würde, würde die Rekursion bis zur "Unendlichkeit" gehen. Das klassische Beispiel für das Ende der Rekursion ist das Einfügen von Stopps.

Ein klassisches Beispiel für Rekursion ist die Fibonacci-Sequenz. Bei der Fibonacci-Sequenz ist jedes Element der Sequenz die Summe seiner beiden vorherigen Elemente. A F (0) = 0 und F (1) = 1.

Ein Beispiel für die Berechnung des vierten Elements ist in der folgenden Abbildung dargestellt. Aus der Abbildung ist ersichtlich, dass wir, um F (4) zu berechnen, zunächst F (3) und F (2) rekursiv neu berechnen müssen. Um F (3) zu berechnen, müssen wir zuerst F (2) + F (1) berechnen, und für F (2) müssen wir F (1) + F (0) rekursiv wiederholen.



Ein großer Teil der Berechnung und Rekursion erfolgt wiederholt, was sie rechenintensiv macht. Aus diesem Grund ist es oft besser, klassische Zyklen zu verwenden. In einigen Fällen wird der gesamte Algorithmus rekursiv definiert (z.B. Fibonacci-Sequenz), oder die Rekursion kann die Arbeit mit einigen Datenstrukturen erleichtern.

Ein konkretes Beispiel für einen rekursiven Funktionsaufruf ist unten dargestellt. In unserem Beispiel haben wir 2 gelb markierte Stationen. Der rekursive Aufruf ist dann mit einer grünen Farbe markiert.

```
public static int fib (int n){
     if(n == 0) return 0;
     else if(n == 1) return 1;
     else return fib (n - 1) + fib (n - 2);
}
```

5.VERKAPSELUNG

Wikipedia (vom 7. Mai 2018) definiert die Verkapselung:

Die Verkapselung ist eine der Grundlagen der OOP (Objektorientierte Programmierung). Es bezieht sich auf die Bündelung von Daten mit den Methoden, die mit diesen Daten arbeiten.[5] Die Kapselung wird verwendet, um die Werte oder den Zustand eines strukturierten Datenobjekts innerhalb einer Klasse zu verbergen und den direkten Zugriff Unbefugter auf diese zu verhindern. Öffentlich zugängliche Methoden werden in der Regel in der Klasse (sogenannte Getter und Setter) für den Zugriff auf die Werte bereitgestellt, und andere Client-Klassen rufen diese Methoden auf, um die Werte innerhalb des Objekts abzurufen und zu modifizieren.

Die Verkapselung ist sehr wichtig. Wenn wir ein kompliziertes Programm hätten, könnten wir versehentlich den Verlauf oder einige seiner Teile beeinflussen. Das Auffinden und Beheben solcher Probleme würde uns viel Zeit kosten. Die Verkapselung ist eines der grundlegenden Konzepte der Objektprogrammierung.

Die Verkapselung in Java ist ein Mechanismus zum Packen von Daten (Variablen) und Code. In der Verkapselung werden die Klassenvariablen vor anderen Klassen verborgen und können nur mit Methoden ihrer aktuellen Klasse angesprochen werden. Daher wird es auch als Ausblenden von Daten bezeichnet.

Dies erreichen wir, indem wir die Teile, auf die andere Funktionen zugreifen sollen, als öffentlich auswählen. Dieser öffentliche Teil der Klasse wird als Klassenschnittstelle bezeichnet. In der Klassenschnittstelle ist es nur sinnvoll, das aufzunehmen, was die anderen Abschnitte des Programms wirklich über die Klasse wissen müssen. Für alle anderen wollen wir die anderen Teile des Programms nicht verwenden und setzen sie privat.

Wenn einige Teile des Programms einige Abschnitte der öffentlichen Klasse verwenden, die später geändert werden, kann dies die Funktionalität beeinträchtigen. Daher ist es besser, seine öffentlich zugänglichen Teile nach der Veröffentlichung der Klasse nicht zu ändern.

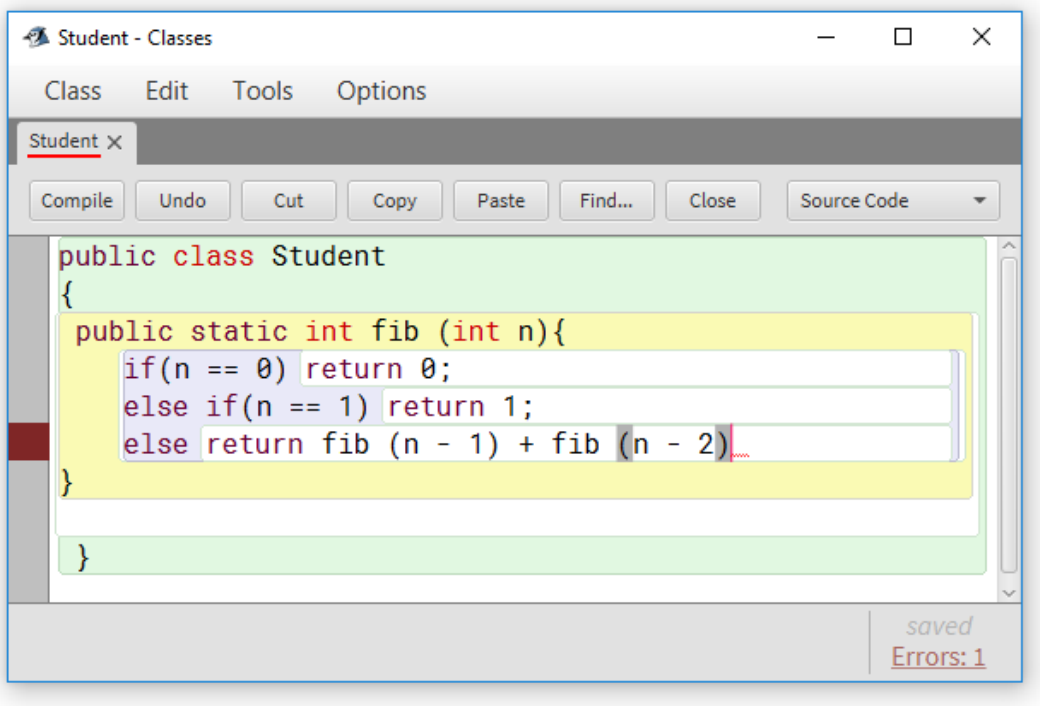
Häufig werden alle Klassenattribute als privat deklariert. Nach unserem Ermessen können wir Zugriffsmethoden (Setter, Getter) veröffentlichen, mit denen sichergestellt werden kann, dass das betreffende Attribut nur gelesen und nicht bearbeitet werden kann oder mit einigen einschränkenden Bedingungen gesetzt werden kann. (Zum Beispiel ist das Alter nur eine positive ganze Zahl). Wenn der Attributname mit dem Namen des Parameters übereinstimmt und ich mit beiden arbeiten muss, verwende ich das Schlüsselwort this. Dieses Wort wird als Klassenname verwendet, in dem die Methode enthalten ist.

6.DEBUGGEN DES PROGRAMMS

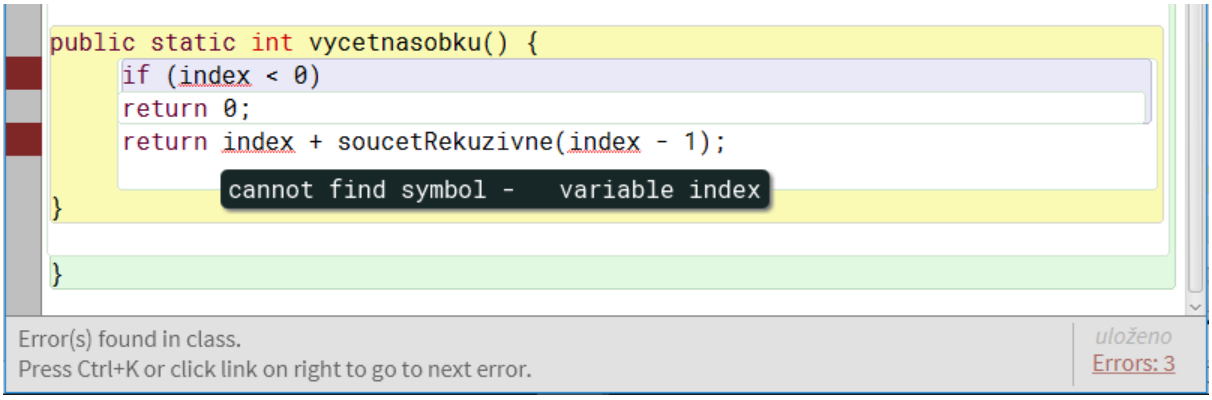
Wir machen oft einen Fehler, wenn wir ein Programm schreiben. Diese Fehler können in mehrere Kategorien eingeteilt werden.

Häufige Probleme sind syntaktische Fehler. Wann überwinden wir die Sprachsyntax? Also gegen die Regeln, wie man einzelne Teile zusammensetzt. Typischerweise wird z.B. ein vergessenes Semikolon oder eine Klammer verwendet.

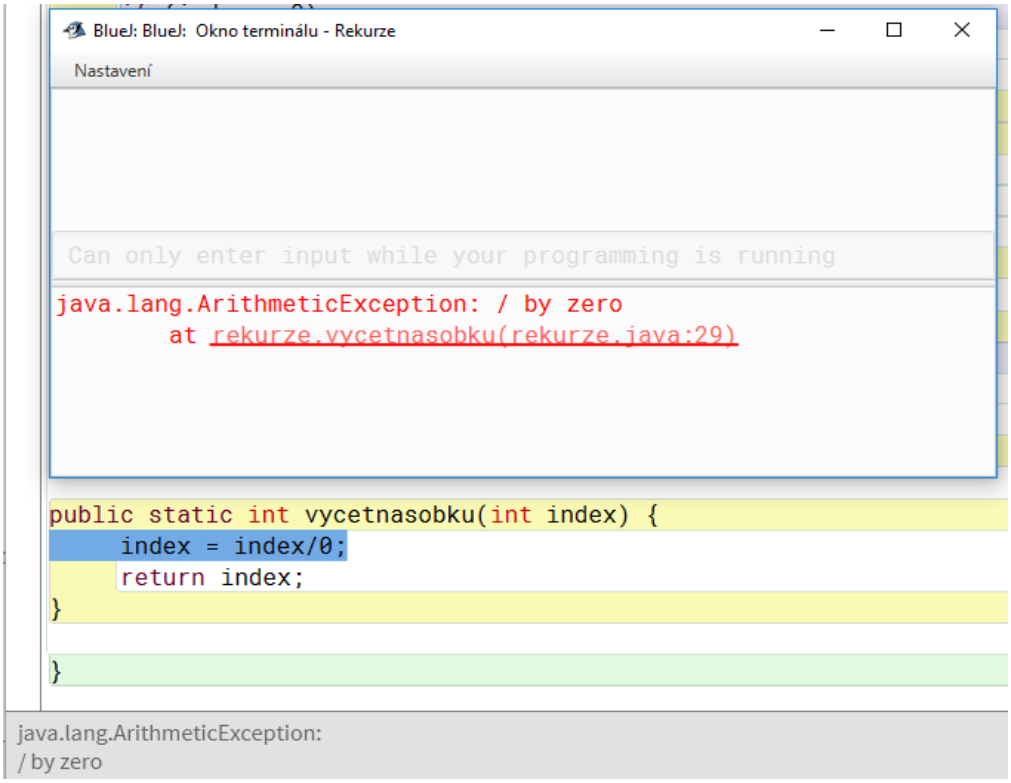
Dieses Problem wird sehr oft durch den Compiler vor der Kompilierung selbst gekennzeichnet. Gleichzeitig werden wir die Linie, in der sie das Problem annimmt, bunt markieren. Wie im Bild zu sehen:



Andere Fehler treten beim Kompilieren auf. Für weitere Informationen über den Fehler können wir unten links zu dem Wort Errors springen. Nach dem Anklicken erhalten wir weitere Hilfe zum Fehler.



Andere Fehler können als Laufzeitfehler bezeichnet werden. Diese Fehler werden vom Compiler nicht gefunden. Aber in einigen Phasen des Programmlaufs können sie auftreten. Ein typisches Beispiel ist die Nullteilung. Wenn das Programm stoppt, wenn der Nullbruch eintritt, öffnet sich das Terminalfenster, in dem es als Fehler angezeigt wird. BlueJ markiert auch die Zeile, in der das Problem aufgetreten ist, wie im Bild unten gezeigt. Um solche Fehler zu beseitigen ist es notwendig, alle möglichen potentiell problematischen Zustände des Programms im Idealfall auszuprobieren. So, dass der Benutzer nicht auf ähnliche Probleme stößt. Im Idealfall bereiten wir im Vorfeld eine Reihe von Testaufgaben vor.



Semantische Fehler sind die heimtückischste Art von Fehlern. Wenn der Compiler nicht erscheint, scheint das Programm nahtlos zu funktionieren. In einigen unerwarteten Momenten zeigt das Programm jedoch einen Fehler an. Was ein großes Problem sein kann.

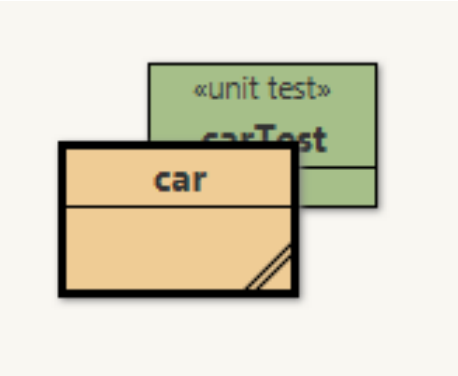
Ein Beispiel könnte sein: Mars Landing Marine Crash (1999) Problem der Kommunikation zwischen den Komponenten - der Benutzer der Schnittstelle erwartete den Wert in Kilometern, der Anbieter gab ihn in Meilen an. Statt der geplanten 140-150 Kilometer führte er nur 57 Kilometer über die Oberfläche. In dieser Höhe ist die Marsatmosphäre auf der Sonde jedoch zu dicht. Der Climate Orbiter brannte auf etwa 80 Kilometern. (Quelle: Technet.cz)

Diese Art von Fehler wird durch Software-Engineering gelöst.

6.1. Testgetriebene Entwicklung

Zusätzlich zum Testen des fertigen Programms können wir eine andere Philosophie der Softwareentwicklung wählen - Test Driven Development. Als Teil dieses Ansatzes definieren wir zunächst eine Reihe von Tests und schreiben dann das Programm selbst, in dem wir uns nur darauf konzentrieren, den Code durch diese Tests zu leiten. (Wir sprechen z.B. nicht von Code-Effizienz) Das Refactoring folgt. Duplikate werden aus dem Code entfernt, und der Code wird in der Regel in der akzeptabelsten Form bearbeitet. Durch die Wiederholung von Tests wird sichergestellt, dass die Funktionalität des Codes während des Refactorings nicht beeinträchtigt wird.

BlueJ bietet zu diesem Zweck eine Reihe von Tools an. In BlueJ erstellen wir einen Komponententest für diese Klasse, indem wir mit der rechten Maustaste auf die zu testende Klasse klicken und die Option zum Erstellen einer Testklasse auswählen. Die gegebene Klasse erhält dann einen untrennbaren Partner - einen Komponententest. Die Testklasse wird farblich differenziert. Wie im Bild zu sehen.



Wenn wir die Tests mit der gleichen Menge von Objekten durchführen wollen, können wir einen Komponententest erstellen. Wir erstellen das Tool, indem wir nur die Objekte erstellen, die sie im Testwerkzeug im Objektstapel enthalten sollen. Klicken Sie nun mit der

rechten Maustaste auf die Testklasse und wählen Sie Object Bench to Test Fixture. Beachten Sie, dass alle Nachrichten, die wir seit dem letzten Neustart der virtuellen Maschine gesendet haben, aufgezeichnet wurden. Wenn wir also sicher sein wollen, was getan wird, starten Sie zuerst die virtuelle Maschine neu und erstellen Sie dann Objekte. Wenn wir ein bereits gespeichertes Objekt mit anderen Objekten überschreiben wollen, erstellen wir einfach diese Objekte und wählen dann Object Bench to Test Fixture. Das Produkt wird überschrieben.

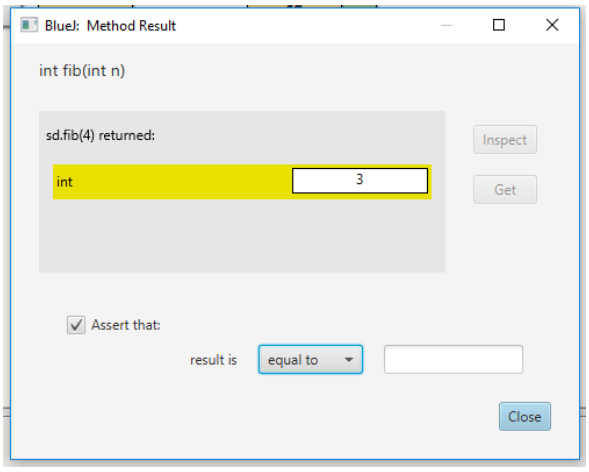
Alternativ können wir die Testklasse auch manuell bearbeiten.

Um anschließend Objekte aufzurufen, klicken Sie mit der rechten Maustaste auf die Testklasse und wählen Sie Test Fixture to Object Bench.

6.2. Einen Test erstellen

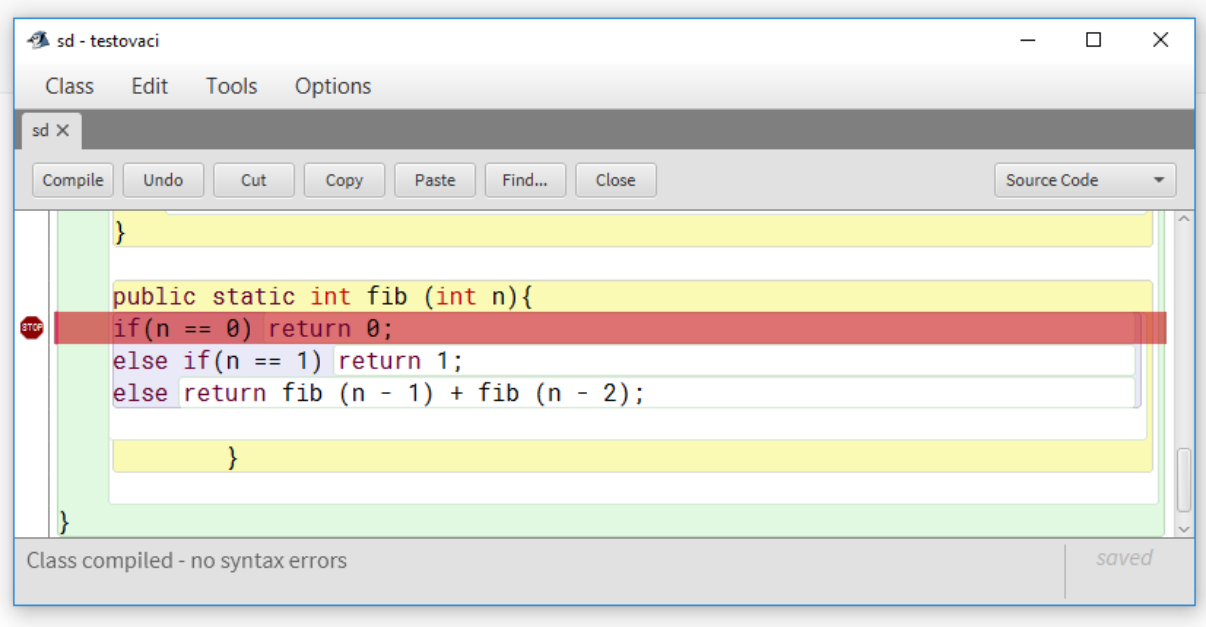
Im Idealfall starten wir zuerst die virtuelle Maschine neu. Klicken Sie nun mit der rechten Maustaste auf den Komponententest und wählen Sie Testmethode erstellen. Wählen Sie zunächst den Namen der Testmethode selbst. Jetzt zeichnet BlueJ unser Verfahren zur Prüfung des Produkts auf. Wir werden die erforderlichen Maßnahmen ergreifen. Während der Aufzeichnung wird BlueJ uns bitten, den Rückgabewert für ausgewählte Methoden zu testen. Es ist auf dem Bild unten dargestellt. Während der Aufnahme leuchtet das rote Licht links. Wenn wir die Aufnahme stoppen wollen, ohne zu speichern, wählen wir: Abbrechen. Um zu beenden und zu speichern, wählen wir den Ausgang. Beide befinden sich direkt unter dem roten Knopf.

Wenn wir den Test testen wollen, klicken Sie mit der rechten Maustaste auf den Komponententest, wo wir den Test im Menü auswählen. Wenn wir im Laufe des Tests anders reagieren als der Rückgabewert des Tests, wird der Test unterbrochen. Es öffnet sich ein Testergebnisfenster, in dem wir erfahren können, in welchem Teil des Codes der Fehler aufgetreten ist.



7.DEBUGGER

Es ist ein Werkzeug, das dem Programmierer hilft, Fehler im Programm zu erkennen. Um den Debugger wiederherzustellen, müssen wir zuerst einen Stopp im Code setzen. Was wir tun werden, indem wir im Code-Editor auf die linke Spalte klicken. Auf der entsprechenden Zeile. Hier erscheint die rote Stoppmarke und die Linie ist rot markiert. Wie im Bild unten gezeigt. Wenn das Programm ausgeführt wird, stoppt das Programm an der Stoppstelle und es erscheint ein Debuggerfenster.



Am unteren Rand des Debugger-Fensters (unten) befinden sich insgesamt 5 Schaltflächen.

- **stopp**, um das Programm zu stoppen. Es ist z.B. nützlich, wenn sich das Programm aufgehängt hat.

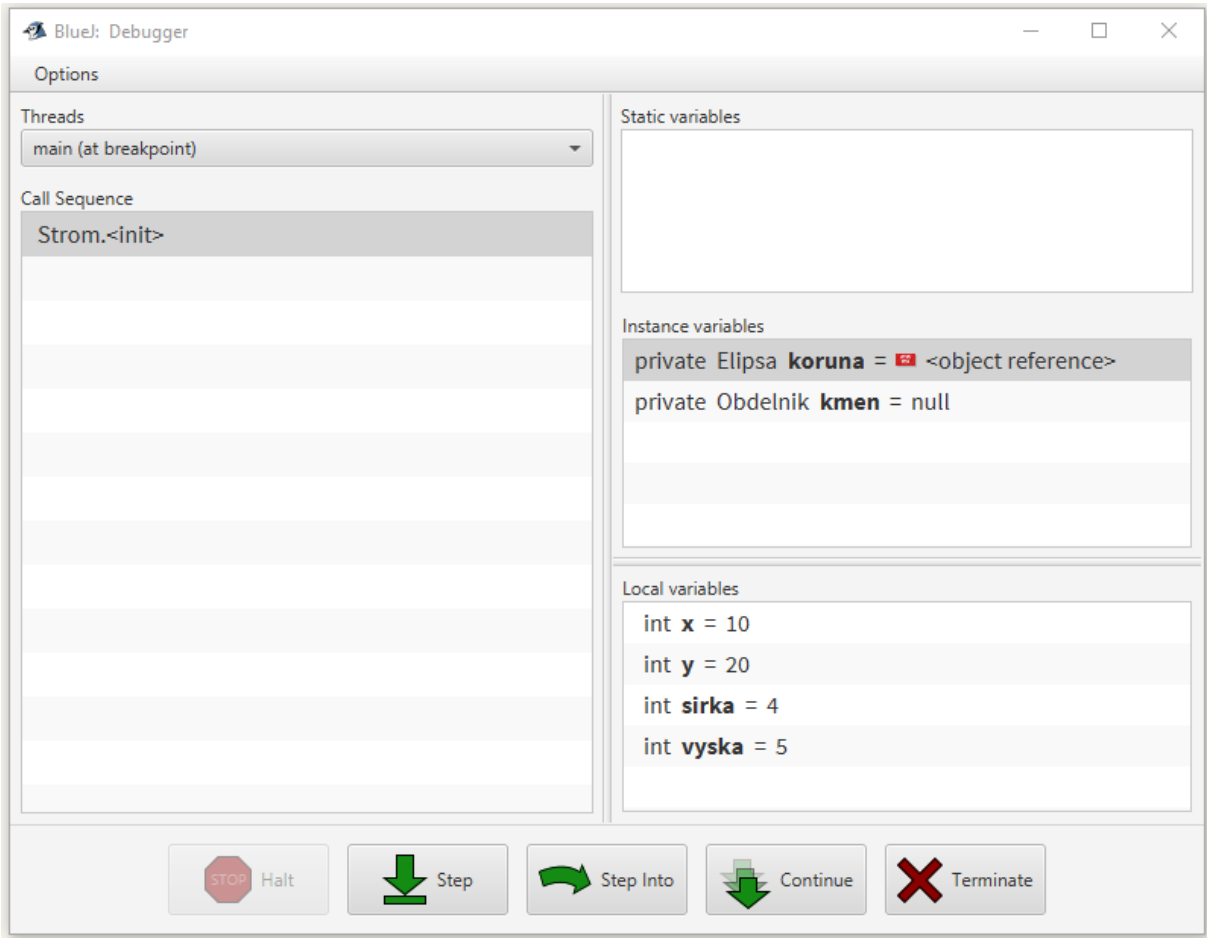
- **step**, um den nächsten Schritt des Codes auszuführen. Der jeweilige Schritt wird auch grafisch durch die grüne Farbe im Code-Editor angezeigt.

- **step into** ist dem step sehr ähnlich. Der Unterschied besteht darin, dass, wenn Sie die step aufrufen, die gesamte Methode ausgeführt wird, aber wenn Sie step into wählen, die aufgerufenen Methode Schritt für Schritt ausgeführt wird.

- **continue**, das Programm wird normalerweise bis zu seinem Ende oder bis zum nächsten Halt fortgesetzt.

- **terminate**, um die Programmausführung zu beenden. Alternativ können wir das Programm beenden, indem wir die virtuelle Maschine neu starten.

Wir können diese neuen Stopps auch während des Prozesses dem Code hinzufügen. Das Debugger-Fenster selbst ist in mehrere Teile unterteilt. Oben links können wir einen bestimmten Thread auswählen. Unter der Wahl des Threads ist der Bereich Calling Order. Dieser Bereich könnte als Rückadress-Stapel bezeichnet werden. Hier sind alle Aufrufe während des Programmlaufs aufgelistet. Neue Anrufe werden am höchsten eingestuft. Wenn wir auf einen bestimmten Aufruf klicken, öffnet sich der Code-Editor mit der Klasse, die die aufgerufene Methode enthält. Die Farbe wird durch die aktuell ausgeführte Zeile hervorgehoben.



Im rechten Teil des Debuggerfensters werden die Klassenattribute aufgelistet. Dies sind Klassenattribute, die zu dem aktuell ausgewählten Ordner in der Call Order gehören. Nachfolgend finden Sie die Attribute von Instanzen. Hier werden die Instanzattribute angezeigt, die zum ausgewählten Element im Abrufauftrag gehören. Wenn wir uns die Instanzattribute in unserem Beispiel genauer ansehen, stellen wir fest, dass diese Instanz zwei Attribute enthält. Das Kronenattribut enthält eine Referenz auf das Objekt. Wenn Sie darauf tippen, erscheint ein Fenster mit allen Attributen der primitiven Typen der Instanz. Attribut “kmen” zu einem bestimmten Zeitpunkt hat nirgendwo eine Verbindung. (zeigt Null an)

Wenn die lokalen Variablen, die auch zu dem aktuell ausgewählten Element im Abrufauftrag gehören, unten angezeigt werden, zeigt der Debugger hier nur Variablen an, die bereits über einen dedizierten Speicher und den zugewiesenen Startwert verfügen, d.h. sie sind definiert.

8. AUSNAHMEN

Bisher gingen wir davon aus, dass das gesamte Programm "ausfällt", sobald unvorhergesehene Ereignisse eintreten. Dieses Problem kann jedoch durch Ausnahmen vermieden werden. In solchen Situationen tritt eine Ausnahme auf, die den Programmablauf und den Übergang des Threads an die Stelle, an der die Situation behandelt wird, sofort unterbricht. Wenn das nicht der Fall ist, wird der Thread beendet und da wir nur einen Thread verwenden, wird die gesamte Anwendung abstürzen. Es gibt mehrere Arten von Ausnahmen, die nach der Ursache der unerwarteten Situation aufgeschlüsselt werden können:

- Fehler - kritischer Fehler, verursacht z.B. durch Ressourcenmangel für die virtuelle Maschine - OutOfMemoryError, Stapelüberlauf - StackOverflowError oder ähnliches. Diese Ausnahmen werden in der Regel nicht behandelt.

- RuntimeException - oft ein Fehler des Programmierers, (zB Null Division - ArithmeticException, ungültiger Index - ArrayIndexOutOf BoundsException). Um diese Bedingung aufzurufen, müssen wir nicht die Fähigkeit deklarieren, die Methode im Kopf aufzurufen.

- Ausnahme des Benutzers (z.B. anstelle einer Telefonnummer gibt er Buchstaben ein), oder eine nicht vorhandene Datei oder einer Datei eines anderen Typs. Auf solche Ausnahmen können wir oft sehr leicht reagieren. Lassen Sie beispielsweise den Benutzer die Datei noch einmal auswählen. Für solche Ausnahmen müssen wir immer das Schlüsselwort throws und die Liste der aufgerufenen Ausnahmeklassen angeben.

8.1. Geschützter Modus

Der potenziell problematische Teil des Codes kann mit Try-and-Compound-Klammern in den "protected mode" versetzt werden. Dieser Modus ist etwas langsamer. Tritt jedoch ein Fehler auf - eine Ausnahme -, wird der im Catch enthaltene Teil ausgeführt. Der optionale Block wird schließlich immer ausgeführt. Schließlich wird es oft für die Bereinigung von Job-Ausnahmen verwendet, wenn wir zu einer Ausnahme zurückkehren, wie z.B. das Schließen von Dateien, das Freigeben von Speicherplatz und so weiter.

```
public static void exception () {
      try {
            int a = 5 / 0; //create exception
      }
      catch (Exception e)
      {
            System.out.println("An exception was taken ");
      }
      finally
      {
            System.out.println("The contents of the block will
            always be processed.");
      }
}
```

8.2. Throw

Alternativ können wir die obige Ausnahme mit dem Schlüsselwort throw hinzufügen. Dies gilt nur für die geprüften Ausnahmen, falls sie die Ausnahmebehandlung an die aufrufende Methode übergeben wollen.
Wie dieses Beispiel zeigt:

```
if (index == null) {
      throw new NullPointerException();
}
```

8.3. Throws

Wenn wir eine Methode erstellen, die eine Ausnahme erzeugen kann, welche wir nicht behandeln wollen oder können. Wir teilen dem Übersetzer explizit mit, dass wir diese Ausnahme an die Top-Level-Behandlung mit der Ausnahme throws + class übergeben.

```
public static void  read ( ) throws IOException {
      ...
}
```

9. ARBEITEN MIT DATEIEN

Alle im Speicher gespeicherten Informationen, Daten oder Objekte werden beim Beenden des Programms gelöscht. Um sie zu bewahren und neu zu laden, müssen wir sie in einer Datei speichern.

Für eine reibungslose Dateneingabe ist es am besten, die Anwendungsdaten im Ordner appdata zu speichern. Die AppData oder Anwendungsdaten oder Anwendungsdaten enthalten Daten, die von Programmen erstellt wurden. In diesem Ordner erstellt es seinen eigenen Ordner praktisch jedes Programm, das auf dem Computer installiert ist, und speichert dann verschiedene Daten darin. Der einfachste Weg, um zu diesem Ordner zu gelangen ist, Dateien in den Roaming-Unterordner im Datei-Explorer %appdata% einzufügen, die Daten in diesem Ordner sollten den Benutzern auf verschiedenen Computern innerhalb der Domäne folgen.

Das Paket java.io enthält eine File-Klasse, die uns alle wichtigen Werkzeuge zur Dateiverwaltung zur Verfügung stellt. Viele Methoden der Klasse File benötigen als Argument einen Dateinamen. Als Argument können Sie die Zeichenkette String oder die Instanz der Klasse File verwenden. Dies ist oft die optimale Lösung, mit der wir einige Informationen finden oder im Voraus eine Operation über die Datei durchführen können.

Sie können ein Dateiobjekt auf verschiedene Arten anlegen:

- Der Name der Datei - wir erstellen ihn aus einem absoluten oder relativen Pfad, der in einen abstrakten Pfad umgewandelt wird.
- Der Name der Datei in Bezug auf den übergeordneten Ordner - der abstrakte Pfad wird in Bezug auf den übergeordneten Ordner erstellt.
- Uniform Resource Identifier (URI) - Bestimmte Anforderungen müssen erfüllt sein. Z.B. darf der Pfad nicht leer sein.

So kann beispielsweise die Datei selbst über die URI erstellt werden.

```
import java.io.File;
import java.io.IOException;
...
public void createFile() throws IOException{
      File soubor = new File("C:\\Temp\\hello.txt");
      soubor.createNewFile();
}
```

Das Dateisystem kann Einschränkungen für bestimmte Operationen am aktuellen Dateisystemobjekt implementieren, wie z.B. Lesen, Schreiben und Booten, die wir Zugriffsrechte nennen.

Instanzen der Klasse File sind invariant, d.h. sobald die Instanz erstellt wurde, ändert sich der abstrakte Pfad, den das File-Objekt repräsentiert, verändert sich nie.

Die Dateiklasse bietet verschiedene Methoden zum Arbeiten mit Dateien. Zum Beispiel können wir arbeiten mit:

- **Dateipfad** - Das Dateiobjekt ist ein abstrakter Pfad zur Datei. Auf diese Weise können wir anders arbeiten. Rückgabemethoden, die den Pfad zu einer Datei zurückgeben, haben typischerweise einen Rückgabewert vom Typ Textzeichenkette, z.B. durch Verwendung von:
    - GetPath () ruft einen abstrakten Pfad zur Datei ab.
    - getName () Ermittelt den Namen der Datei.

- **Dateiinformationen** - Es gibt mehrere Methoden, die Dateiinformationen zurückgeben, wie z.B.: Länge (); canRead () oder, zum Beispiel, lastModified ().

- **Verzeichnisinformationen** - Wir haben mehrere Methoden nur für das Verzeichnis, wie z.B. list () - eine Liste aller Dateien im Verzeichnis oder Unix SheetRoots () Rückgabefelder aller Verzeichnisbaum-Root.

- **Arbeiten mit Dateien** - Wir haben verschiedene Methoden, um mit Dateien zu arbeiten:
    - RenameTo (File k) als Parameter gibt die nächste Instanz des File-Objekts an.
    - löschen ()
    - mkdir () Verzeichniserstellung
    - setReadOnly ()

10. GRAFISCHE BENUTZEROBERFLÄCHE (GUI)

Es ist die grafische Umgebung, mit der der normale Benutzer konfrontiert wird und mit der er arbeitet. Wir alle kennen die einzelnen Komponenten dieser Umgebung sehr gut. Dazu gehören beispielsweise Knöpfe, Scroller, Schieberegler und dergleichen. Verschiedene Grafikbibliotheken wie AWT (Abstract Windowing Toolkit), JFC (Java Foundation Classes) sind in Java verfügbar. Für BlueJ können Sie die Simple GUI Designer Extension installieren. Das kann die GUI-Erstellung vereinfachen. In diesem Beitrag wird diskutiert, wie man mit JFC, auch bekannt als Swing, arbeitet.

10.1. Erstanwendung

Beim Erstellen einer Anwendung mit einer grafischen Benutzeroberfläche, in der die Anwendung ausgeführt wird müssen wir zuerst ein "Fenster" (Rahmen) schaffen. Wir werden die JFrame-Klasse verwenden. Es gibt mehrere Möglichkeiten, ein Fenster zu erstellen. Wir haben die ConstructGui-Klasse erstellt, die von der JFrame-Klasse erbt. In dieser Klasse haben wir einen nicht-parametrischen Konstruktor erstellt. Wir haben einen Knopf und ein Label in der Klasse platziert.

Eine weitere gängige Komponente, die wir in diesem Beispiel nicht verwendet haben, ist das Schreibfeld (JTextField). Es würde wie folgt deklariert werden:

```
JTextField someName = new JTextField(”Text”, 6)
```

Geben Sie als Parameter den Text ein, der im Feld angezeigt wird. Als nächstes gibt es eine Zahl, die angibt, wie viele Zeichen das de Feld passen soll.

Wir haben das Layout der Komponenten in FlowLayout festgelegt. Aufgrund von FlowLayout war es notwendig, java.awt zu importieren.

Die einzelnen Komponenten mussten dem Fenster hinzugefügt werden. Was wir mit der add () Methode gemacht haben, die den Namen des hinzugefügten Objekts als Parameter angibt.

```
import java.awt.*;
import javax.swing.*;

public class ConstructionGui extends JFrame {
      private JButton button;
      private JLabel value;

      public ConstructionGui()
      {
            FlowLayout layout = new FlowLayout();
            setLayout(layout);
            value = new JLabel(”Total:”);
            add(value);
            button = new JButton(”Add 1”);
            add(button);
      }
}
```

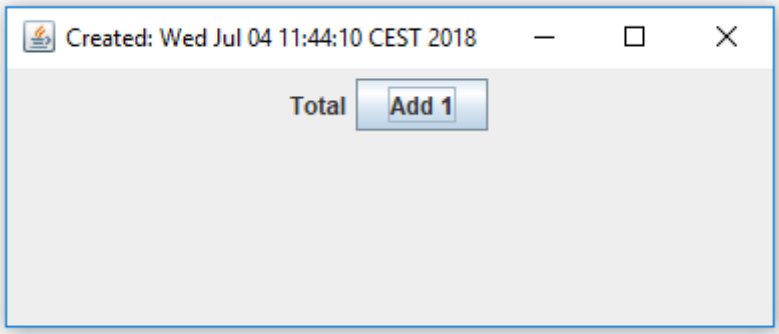
Anschließend haben wir eine weitere Klasse mit dem Namen FirstGUI erstellt. Mit der Methode main. In der Methode main haben wir Objekte mit der Klasse ConstructionGUI erstellt. In diesem Objekt haben wir verschiedene grundlegende Methoden ausgewählt. Wir haben eine Bibliothek von Daten importiert, um die Funktion date() zu verwenden, die uns das heutige Datum in das Titelfenster schreibt.

```
import java.awt.*;
import javax.swing.*;

public class ConstructionGui extends JFrame {
      private JButton button;
      private JLabel value;

      public ConstructionGui()
      {
            FlowLayout layout = new FlowLayout();
            setLayout(layout);
            value = new JLabel(”Total”);
            add(value);
            button = new JButton(”Add 1”);
      add(button);
}}
```

Das Ergebnis sieht so aus:



Die einzelnen Komponenten sind zentriert. Wenn wir das Fenster skalieren, werden sie untergeordnet.

11. EREIGNISSE

Jetzt haben wir ein einfaches Single-Label-Fenster erstellt. Es passiert jedoch nichts, wenn die Taste gedrückt wird. Deshalb verwenden wir Ereignisse, um unserem Fenster Funktionalität hinzuzufügen. Das Prinzip der Ereignisse ist, dass wir in der GUI z.B. ein Ereignis auf dem Taster aufrufen. Das Ereignis, für das wir das Ereignis erstellt haben, ist ein Ereignis-Listener. Die Methode, etwas zu tun, wird beim Ereignis-Listener aufgerufen. In unserem Fall erstellen wir die Schaltflächen Button und Button Counter aus dem Label. Der vollständige Code befindet sich auf der nächsten Seite.

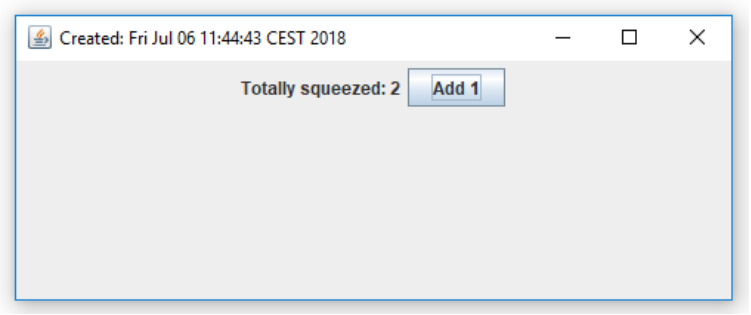
Wir haben der ConstructionGui-Klasse eine weitere Klasse namens EventCost hinzugefügt. Dadurch hat es Zugriff auf ConstructionGui-Komponenten. Wir haben den ActionListener in der Klasse EventCost implementiert. Es ist notwendig, java.awt.event zu importieren. *;

Die ActionListener-Klasse enthält nur eine Rückgabemethode actionPerformed (ActionEvent e), in der wir definieren, was passiert, wenn ein Ereignis aufgerufen wird. In unserem Fall ist das Zählen ein Knopfdruck.

Im Konstruktor haben wir Ereignishörer erstellt - ein Objekt namens MyCount. Wir haben der Nummer des Zuhörers einen Button zugeordnet.

```
EventCost MyCount = new EventCost();
MyButton.addActionListener(MyCount);
```

Das Ergebnis sieht so aus:



```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class ConstructionGui extends JFrame {
      private JButton MyButton;
      private JLabel MyValue;
      int MyNumber = 0;

      public ConstructionGui()
      {
            FlowLayout layout = new FlowLayout();
            setLayout(layout);
            MyValue = new JLabel("Not squeezed ");
            add(MyValue);
            MyButton = new JButton("Add 1");
            add(MyButton);
            EventCost MyCount = new EventCost();
            MyButton.addActionListener(MyCount);
      }

      public class EventCost implements ActionListener {
            public void actionPerformed(ActionEvent e) {
                  MyNumber = MyNumber + 1;
                  MyValue.setText("Totally    squeezed:    "    +
                  MyNumber);
            }
      }
}
```