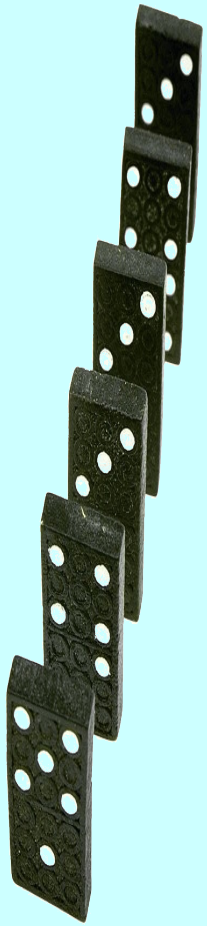
B-K H

Lotus Notes EntwicklerCamp '06



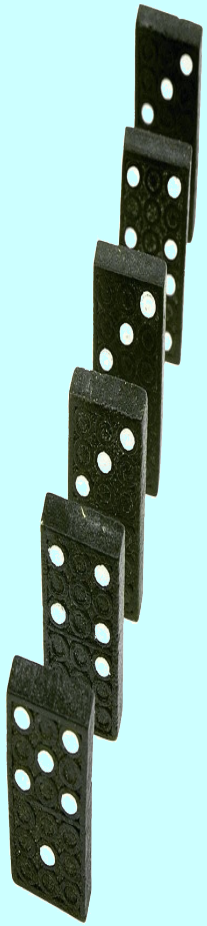
Java Einführung
Hands On Workshop

Bernfried Geiger, Intellisys GmbH, Sindelfingen
www.IntelliSys.de
Jens-B. Augustiny, LIGONET GmbH, Lobsigen bei Bern
www.Ligonet.ch

B-KH

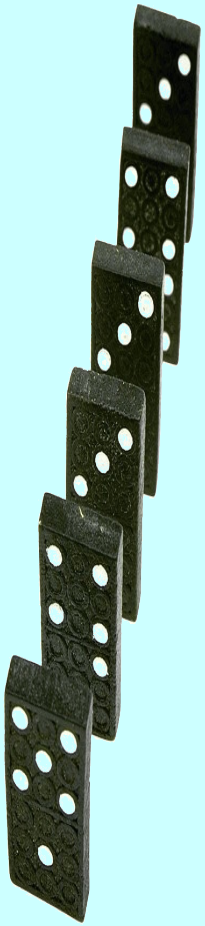
Lotus Notes EntwicklerCamp '06

Agenda



- Ueberprüfung der Installation am Beispielprogramm
- Grundlagen: Objekte
  - Class, Members, Properties, Methods, Setter- und Getter-Methods
- Grundlagen: Sprachstrukturen
  - Syntax
  - If, While
- Erweiterung des Beispiels mit dritter Klasse
  - Tastaturabfrage, Streams
  - Exceptionhandling

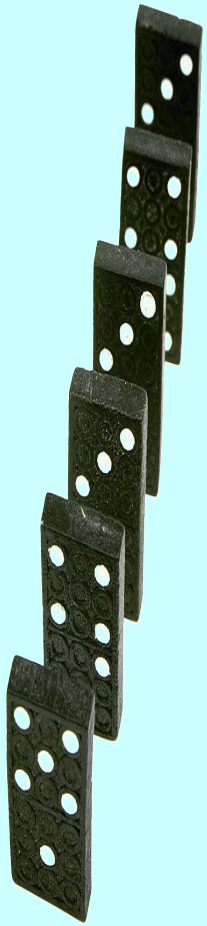
Ausgangsbeispiel



- Eclipse und Java installiert
- Beispiel mit Objekten:
  - DataContainerClass.Java
  - MovingSignStartClass.Java
- Beispiel starten (Funktionskontrolle)
- Beispiel analysieren ...

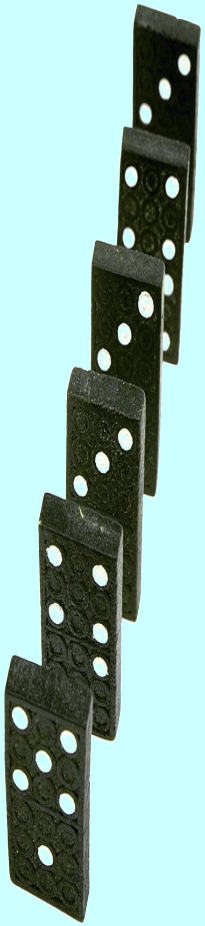
Code talks .....

DataContainerClass.Java



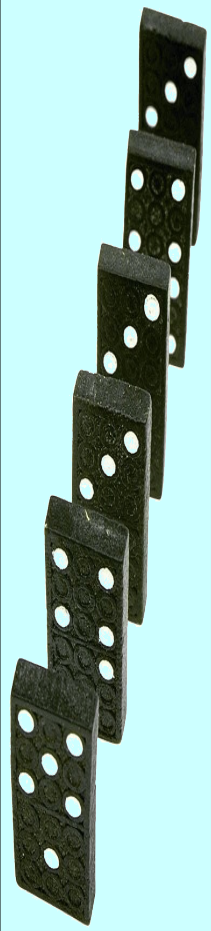
```
public class DataContainerClass {

        private static int Position = 9;

int getPosition() {
        return (Position);
}

void setPosition(int newValue){
        Position = newValue;
}

}
```

Klassen



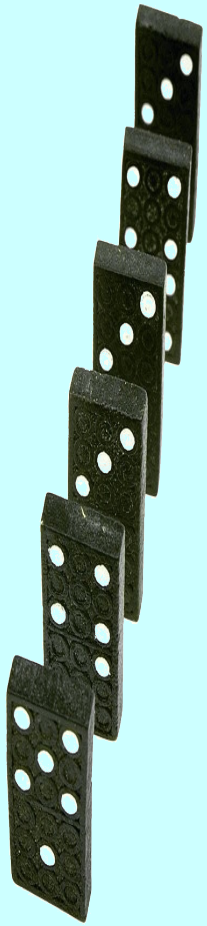
- Jeglicher Javacode steht immer in Klassen
- Ueblicherweise je Klasse ein eigenes File
- Reserviertes Wort "class"
- Dateiendung ".java" (bzw. ".class", wenn compiliert)
- Per default als "public" definiert, kann von anderen Javaklassen aus aufgerufen werden
-

Klasse beinhaltet in der Regel:



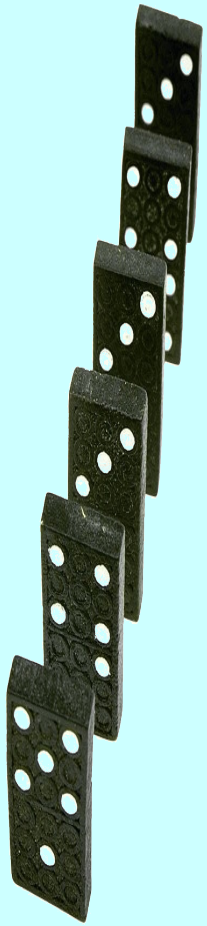
- Methoden (Funktionen)
- Properties/Eigenschaften
- Variablen

Java-Applikation: MovingSignStartClass



- Zwingende Methode, wenn von aussen startbar:
  - public static void main(String[] args) { ... }
  - bei Vorhandensein dieser Methode nach dem Kompilieren direkt vom Interpreter aufrubar:
    - java.exe MovingSignStartClass
    - Gross- Kleinschreibung wird wie immer beachtet
  - Weitere Klassenelemente dürfen vorhanden sein

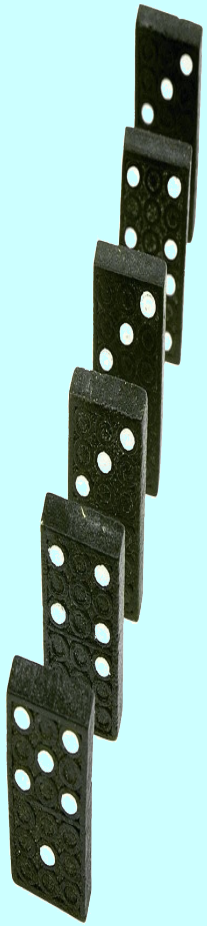
Objekt Deklaration



- Autoklasse MeinAuto;
  (LotusScript: Dim MeinAuto As Autoklasse
  anderes Beispiel: Dim Doc As NotesDocument)

- Speicherplatz reservieren für ein Objekt zu einer definierten Klasse (was muss in den Speicher? Z.B. die Daten eines Autos)

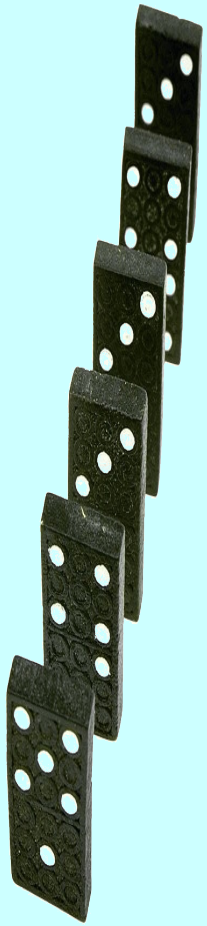
Objekt Instantiierung



- MeinAuto = new AutoKlasse ();
  (LotusScript: Set MeinAuto = New AutoKlasse
  oder: Set Doc = DB.CreateDocument)

- Zuweisung, Instanziierung des Objektes
  (Speicherplatz für TempAuto soll mit den Daten des roten Golfes vor der Türe verknüpft werden).

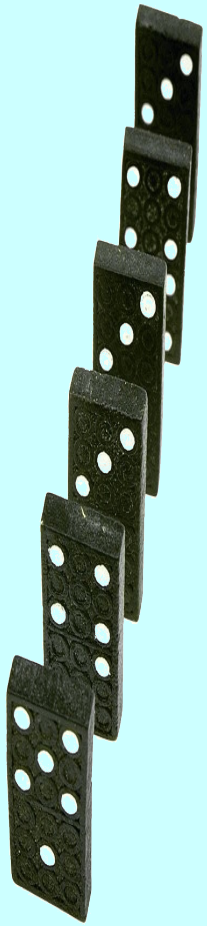
Objekt Nutzung (Zugriff)



- System.out.println (MeinAuto.Farbe);
  (LotusScript: Print MeinAuto.Farbe
  oder: Print Doc.UniversalID)

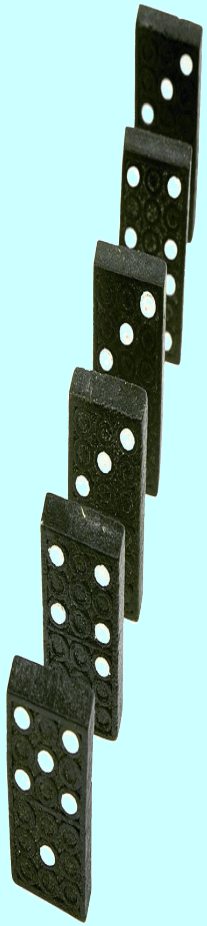
- Ansprechen des Objektes (Daten)

Javaconsole



- Zwingend auf allen Plattformen vorhanden
- Für Lotus Notes:
  6.5 DE: Datei - Extras - Java Debug Console anzeigen
  7.0 EN: File – Tools – Show Java Debug Console
- Dient zur Ausgabe von Texten in die Console:
  - System.out.println ("Hello World");
  - Häufig für einfaches Debugging oder Kontrolle verwendet

void oder nicht void



Beispiel:

```
void setPosition(int newValue){
        Position = newValue;
}
```

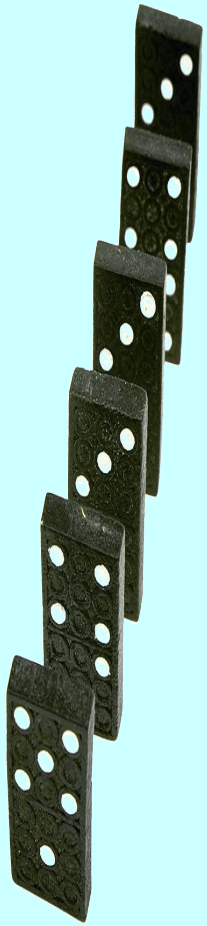
- Methode ("Funktion") liefert keinen Wert zurück. In LotusScript zB.: Sub SetPosition (NewValue As Integer)

```
int getPosition() {
        return (Position);
}
```

- Methode liefert Wert zurück, in LotusScript zB: Function GetPosition ( ) As Integer

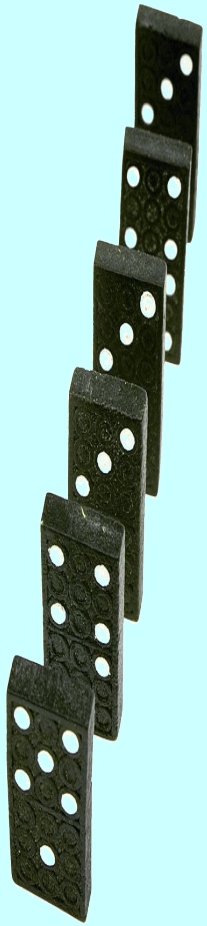
return: Rückgabewert beim Beenden

Erstes Beispiel



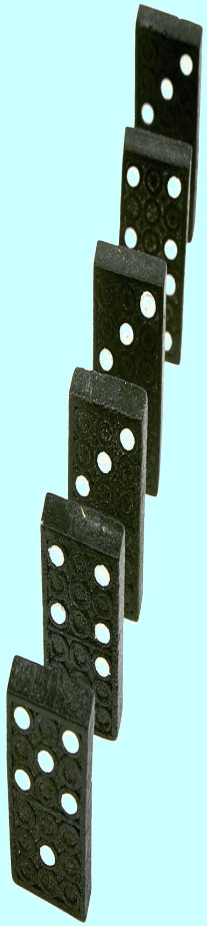
- Eclipse starten:
  - C:\eclipse_starter\eclipse\eclipse.exe
- File – New Project
- JavaProject – Next
- Projectname: Demo1
- Finish

Ausgangsbeispiel laden



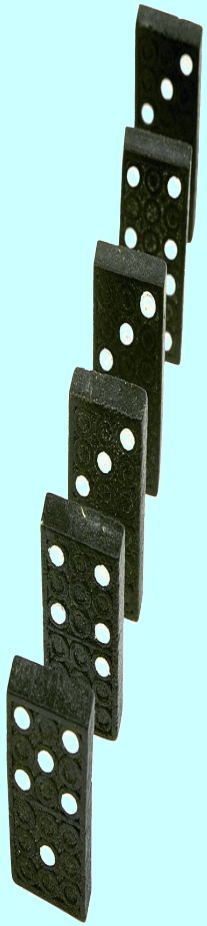
- File – Import – File System – Next
- Browse
- Arbeitsplatz – C-Laufwerk – Eclipse_Starter – Demo2 – 1 – OK
- Alle Files markieren
- Finish

Ausgangsbeispiel starten



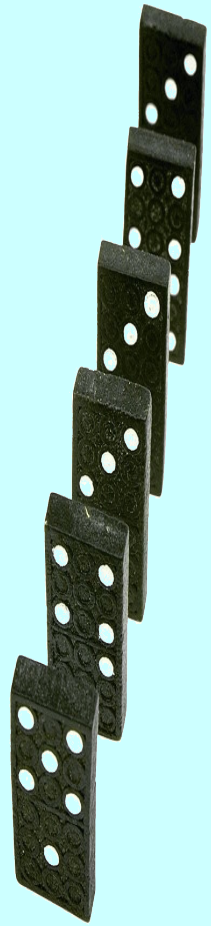
- Projekt Demo1 aufklappen
- Default Package aufklappen
- MovingSignStartClass.java doppelklicken
- Run – Run As – Java Application
  - Application startet
- Ausgabe 9 in Console prüfen

Ausgangsbeispiel nachvollziehen



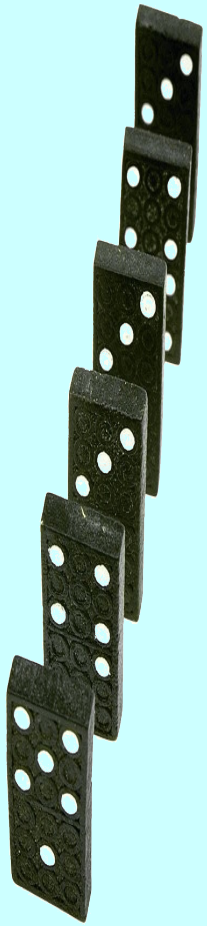
- Was macht der Code?
  - Instanz ActiveData vom Objekt DataContainerClass angelegt
  - ActiveData initialisiert
  - Rückgabewert der Methode getPosition() von dieser Instanz über System.out.println auf die Console ausgegeben

Die Klasse DataContainerClass



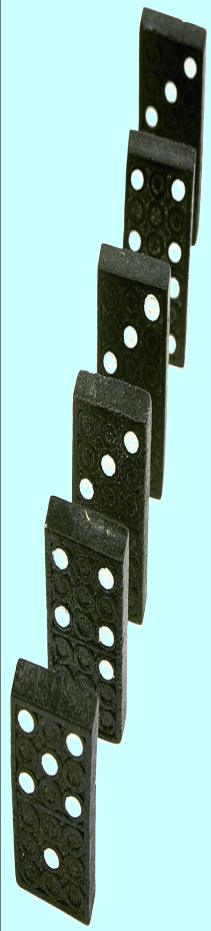
- DataContainerClass.java doppelklicken
- Klasse beinhaltet eine private Variable "Position"
- Variable wird bei Initialisierung auf den Wert 9 gesetzt
- Klasse beinhaltet Methode "getPosition" zum Auslesen dieser Variablen

Auskommentierte Bereiche



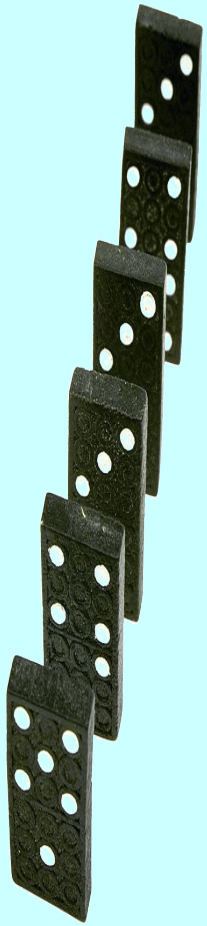
- // (2 mal Slash) dient zur Einleitung einer Kommentarzeile
- /* in Verbindung mit */ umschliessen einen Kommentarblock
- Bitte in beiden Dateien die Kommentarbegrenzungen entfernen
- Ueberlegen, was sich ändert
- Ueberprüfen durch Ausführen

Neue Klasse: Einfaches Ausgabeobjekt



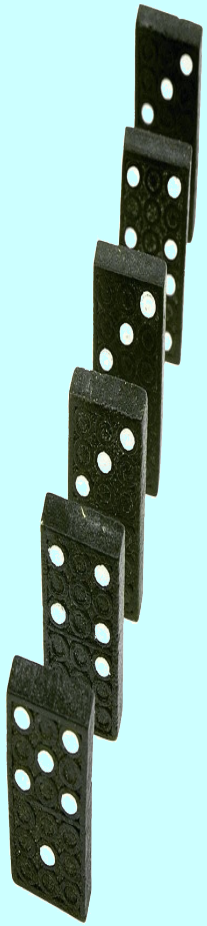
- LineWriterClass / While (Demo 6)
- Modifiziert MovingSignStartClass, um dies zu nutzen

LineWriter Objekt für die Ausgabe



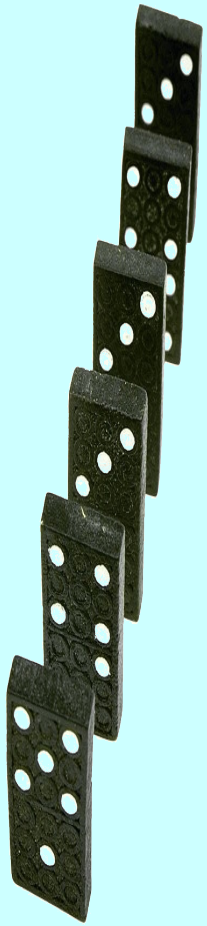
- Stichwort: Kapselung der Ausgabe
- File - New - Class
- Name: LineWriterClass
- Beachten:
  - MovingSignProject als Folder
  - Method Stub "public static void main .... " deselektiert

While Schlaufe: Methode in LineWriterClass



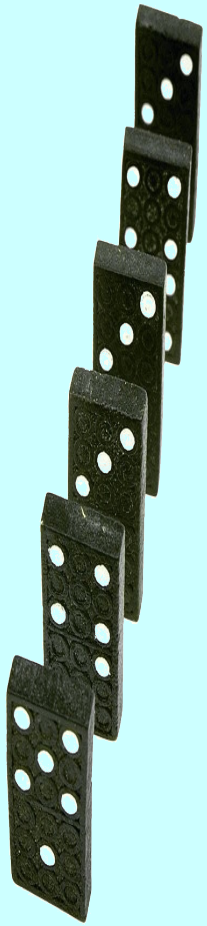
```
void writeLineToConsole(int distance, char visibleChar){
        String resultLine = "";
    int counter = 0;
    int maxCounter;
    maxCounter = distance;
    while (counter < maxCounter) {
        counter = counter + 1;
        resultLine = resultLine + " ";
    }
        resultLine = resultLine + visibleChar + distance;
        System.out.println (resultLine);
}
```

Anpassung der ..StartClass



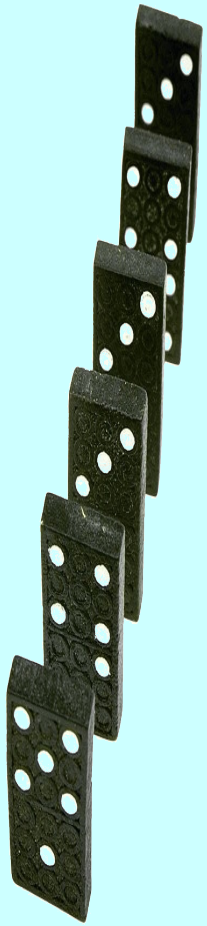
```
DataContainerClass ActiveData;
ActiveData = new DataContainerClass();
LineWriterClass ActiveLine;
ActiveLine = new LineWriterClass();
ActiveLine.writeLineToConsole(ActiveData.getPosition(), '#');
ActiveData.setPosition(10);
ActiveLine.writeLineToConsole(ActiveData.getPosition(),'#');
ActiveData.movePosition(30);
ActiveLine.writeLineToConsole(ActiveData.getPosition(),'#');
ActiveData.movePosition(30);
ActiveLine.writeLineToConsole(ActiveData.getPosition(),'#');
```

Ausbau LineWriterClass: Properties und Methoden



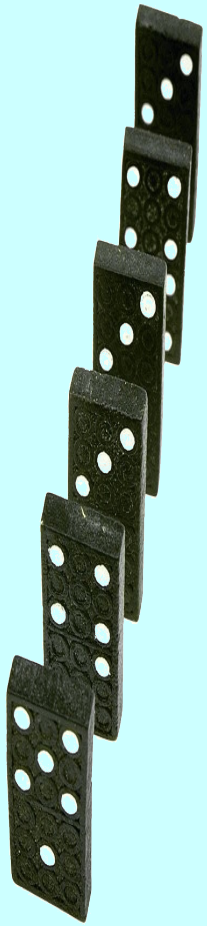
- Property visibleChar
  - Setter: setVisibleChar
  - Getter: getVisibleChar
- Property displayDistance
  - setDisplayDistance
  - getDisplayDistance
- Methode writeLineToConsole

Ergänzung visibleChar in LineWriterClass



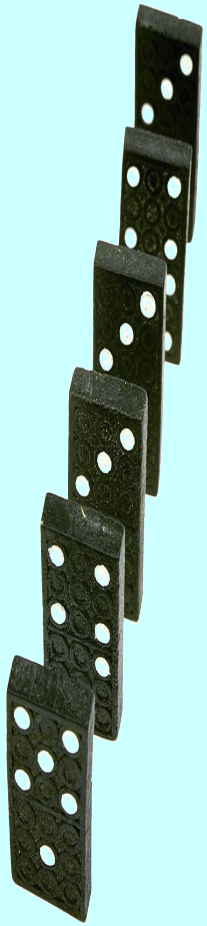
```
char visibleChar = '#';

void setVisibleChar(char newChar) {
        visibleChar = newChar;
}

char getVisibleChar(){
        return(visibleChar);
}
```

Ergänzung displayDistance in LineWriterClass



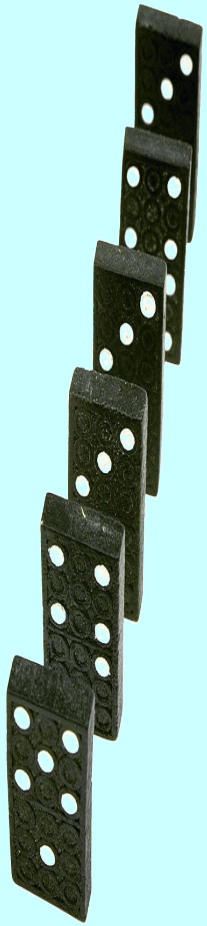
```
boolean displayDistance = false;
void setDisplayDistance (boolean newSetting){
        displayDistance = newSetting;
}

boolean getDisplayDistance () {
        return(displayDistance);
}
```

writeLineToConsole, IF Bedingung



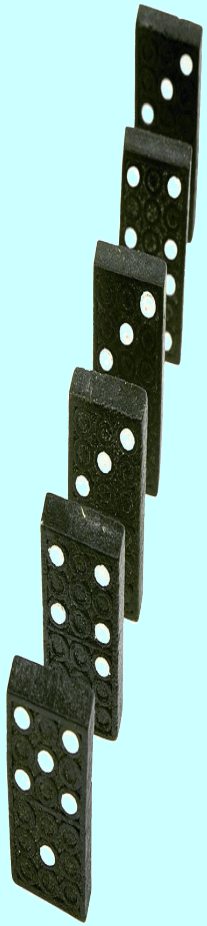
```
void writeLineToConsole(int distance){
    String resultLine = "";
    int counter = 0;
    int maxCounter;
    maxCounter = distance;
    // Add up as many spaces as handed over
    while (counter < maxCounter) {
            counter = counter + 1;
            resultLine = resultLine + " ";
    }
    //generate String to display, add value as number only if flag is set
    resultLine = resultLine + visibleChar;
    if (true == displayDistance) {
            resultLine = resultLine + distance;
    }
            System.out.println (resultLine);
}
```

passende ...StartClass (Ausschnitt) xxxxx



```
ActiveLine.writeLineToConsole(ActiveData.getPosition());
ActiveData.setPosition(10);
ActiveLine.writeLineToConsole(ActiveData.getPosition());
ActiveData.movePosition(30);
ActiveLine.writeLineToConsole(ActiveData.getPosition());
ActiveData.movePosition(30);
ActiveLine.writeLineToConsole(ActiveData.getPosition());
ActiveLine.setVisibleChar('$');
ActiveData.movePosition(-20);
ActiveLine.writeLineToConsole(ActiveData.getPosition());

ActiveLine.setDisplayDistance(true);
ActiveData.movePosition(-20);
ActiveLine.writeLineToConsole(ActiveData.getPosition());
```

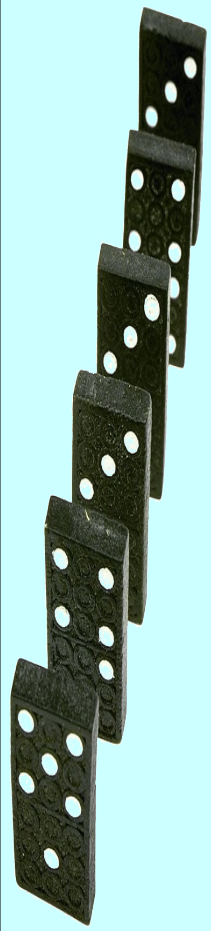
Eingabe-Methode in LineWriterClass



- import java.io.*; // vor Klassenkopf
- int getanInteger() {

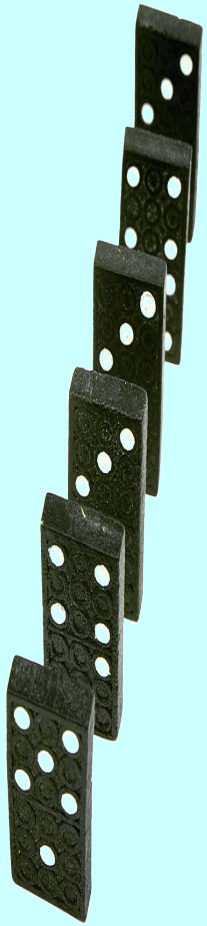
```
        BufferedReader din = new BufferedReader(new

 InputStreamReader(System.in));
        String typedData = "";
        try {
                typedData = din.readLine();
                } catch(Exception e){}
        return Integer.parseInt(typedData);
 }
```

..... StartClass dazu



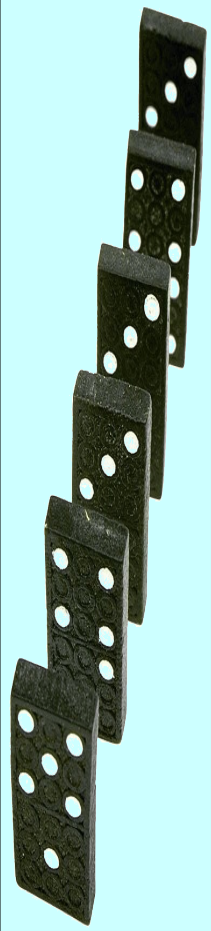
```
DataContainerClass ActiveData;
ActiveData = new DataContainerClass();
LineWriterClass ActiveLine;
ActiveLine = new LineWriterClass();
int inputDistance;
inputDistance = 0;


ActiveLine.writeLineToConsole(ActiveData.getPosition());
inputDistance = ActiveLine.getanInteger();
ActiveData.movePosition(inputDistance);

ActiveLine.writeLineToConsole(ActiveData.getPosition());
```

...StartClass ergänzt mit While-Schleife



```
while ((inputDistance = ActiveLine.getanInteger()) < 100)
{
        ActiveData.movePosition(inputDistance);

ActiveLine.writeLineToConsole(ActiveData.getPosition());

}
```

EchoServer und EchoClient



- Grundlagen der Netzkommunikation (Ansatz als Beispiel)
- Zeigt, dass Java für Netzapplikationen bestens geeignet ist
- Siehe Musterdateien auf der CD und die Beschreibung bei Krüger