Einführung in JAVA und Open Source Physics

Ph. Cain

Übung CSM1

Teil I: JAVA

Was ist JAVA?

Programmiersprache

- einfach, objektorientiert, C-ähnlich
- Compiler erzeugt portablen (plattformunabh.) Byte-Code

Interpreter (JAVA Virtual Machine)

- plattformabhängige Verarbeitung des Byte-Codes

Plattform

- Vordefinierte Klassen (Pakete) z.B. für Grafik, I/O, etc.
- Programmierschnittstelle (API)

Warum JAVA?

“Write once, run anywhere!”

- einfach und trotzdem umfangreich
- modern und sehr weit verbreitet
- unabhängig von Hardware und Betriebssystem,
  nur JAVA Plattform notwendig
- sicher, netzwerktauglich, dynamisch, erweiterbar, schnell!?

Für die Übung:

- Vergleichbarkeit der Übungsergebnisse
- einfache grafische Darstellung mit Open Source Physics
- Unterstützung durch Lehrmaterial

Warum JAVA?

“Write once, run anywhere!”

- einfach und trotzdem umfangreich
- modern und sehr weit verbreitet
- unabhängig von Hardware und Betriebssystem, nur JAVA Plattform notwendig
- sicher, netzwerktauglich, dynamisch, erweiterbar, schnell!?

Für die Übung:

- Vergleichbarkeit der Übungsergebnisse
- einfache grafische Darstellung mit Open Source Physics
- Unterstützung durch Lehrmaterial

Programmstruktur

Beispielprogramm 1: *Helloworld.java*

```
/**                                          // Kommentare
  *  Helloworld.java - Helloworld in JAVA
 */
public class  Helloworld {                   // Klassendefinition
  public static void main( String[] args ) { // Programmstart main()
    System.out.println("Hello World!");      // Textausgabe
  }                                          // Ende der Methode main()
}
```

- Programmquelltext $\equiv$ Klassendefinitionen
- **Übersetzen:** `javac Helloworld.java`
- **Starten:** `java Helloworld`

Programmstruktur

Beispielprogramm 1: *Helloworld.java*

```
/**                                          // Kommentare
  *  Helloworld.java - Helloworld in JAVA
 */
public class  Helloworld {                   // Klassendefinition
  public static void main( String[] args ) { // Programmstart main()
    System.out.println("Hello World!");      // Textausgabe
  }                                          // Ende der Methode main()
}
```

- Programmquelltext $\equiv$ Klassendefinitionen
- **Übersetzen:** `javac Helloworld.java`
- **Starten:** `java Helloworld`

Programmstruktur

Beispielprogramm 1: *Helloworld.java*

```
/**                                          // Kommentare
  *  Helloworld.java - Helloworld in JAVA
 */
public class  Helloworld {                   // Klassendefinition
  public static void main( String[] args ) { // Programmstart main()
    System.out.println("Hello World!");      // Textausgabe
  }                                          // Ende der Methode main()
}
```

- Programmquelltext $\equiv$ Klassendefinitionen
- **Übersetzen:** `javac Helloworld.java`
- **Starten:** `java Helloworld`

Programmstruktur

Beispielprogramm 2: *Factorial.java* (aus *JAVA in a Nutshell*)

```
/**
 * This program computes the factorial of a number
 */
public class Factorial {                    // Define a class
  public static void main(String[] args) { // The program starts here
    int input = Integer.parseInt(args[0]); // Get the user's input
    double result = factorial(input);      // Compute the factorial
    System.out.println(result);            // Print out the result
  }                                        // The main() method ends here

  public static double factorial(int x) {  // This method computes x!
    if (x < 0)                             // Check for bad input
      return 0.0;                          // If bad, return 0
    double fact = 1.0;                     // Begin with an initial value
    while(x > 1) {                         // Loop until x equals 1
      fact = fact * x;                     // Multiply by x each time
      x = x - 1;                           // And then decrement x
    }                                      // Jump back to start of loop
    return fact;                           // Return the result
  }                                        // factorial() ends here
}
```

Einfache Datentypen

Typ	Inhalt	Bits	Wertebereich
boolean	*true* oder *false*	1	
char	Unicode Zeichen	16	$\backslash u0000$ bis $\backslash uFFFF$
byte	Ganze Zahl	8	-128 bis 127
short	Ganze Zahl	16	-32768 bis 32767
int	Ganze Zahl	32	-2147483648 bis 2147483647
long	Ganze Zahl	64	-2^{63} bis $2^{63}-1$
float	Fließkommazahl	32	$\pm 1.4 \times 10^{-45}$ bis $\pm 3.4028235 \times 10^{38}$
double	Fließkommazahl	64	$\pm 4.9 \times 10^{-324}$ bis $\pm 1.79769313\ldots \times 10^{308}$

Einfache Typkonvertierung

Erweiterung des Wertebereichs

- automatische Konvertierung in größeren Wertebereich

```
int i=13;
double d=i;
```

Einschränkung des Wertebereichs

- Konvertierung muss durch *cast* erzwungen werden

```
double d=13.456;
int i=(int) d;
```

Komplexe Datentypen

String

- Zeichenkette, z.B. `"Hello world!"`
- eigene Klasse

Array

- Felder aus identische Datentypen, z.B.
  `byte[] b= new byte[1024];`
- auch mehrdimensional, z.B. `int[][] a= {{1,2,3},{4,5}};`

Referenz

- "Zeiger" auf Objekt, z.B. Array, Klasse, Interface

Ausdrücke (expressions)

- Einfache Ausdrücke (variables,literals) , z.B. `1.7`, `true`, `sum`
- Komplexe Ausdrücke mit Operatoren, z.B.

```
sum=1+3*1.435-(int)(d+33.4)
```

Einige Operatoren

- Arithmetisch: `+,-,*,/,%,!,++,--`
- Vergleich: `<,>,<=,>=,==,!=,?:`
- Logisch: `&&,||`
- Bitweise: `&,|,^,<<,>>,~`
- Zuweisung: `=,*=,+=,-=,/=,%=`
- Andere: `.,new, instanceof,(`*type*`)`

Anweisungen (Statements)

Befehl	Zweck	Syntax
expression	Ausdruck	`var=expr;expr++;method();`
compound	Gruppierung	`{`*statements* `}`
labeled	Label	*label: statement*
variable	Variablendekl.	`[final]` *type name* `[=`*value*`];`
if	Bedingung	`if ( expr ) statement [ else statement]`
switch	Bedingung	`switch ( expr ) { [ case expr : statements ]` `... [ default: statements ] }`
while	Schleife	`while ( expr ) statement`
do	Schleife	`do statement while ( expr );`
for	Schleife	`for ( init ; test ; increment ) statement`

Anweisungen (Forts.)

Befehl	Zweck	Syntax
break	Block beenden	`break [ label ];`
continue	Schleife fortsetzen	`continue [ label ] ;`
for/in	Iteration	`for ( variable : iterable ) statement`
return	Methode beenden	`return [ expr ] ;`
synchronized	Objekt blockieren	`synchronized ( expr ) { statements }`
throw	Ausnahme erzeugen	`throw expr ;`
try	Ausnahme fangen	`try { statements } [ catch ( type name ) { statements } ] ... [ finally { statements } ]`
assert	Annahme testen	`assert invariant [ : error] ;`

Objektorientierte Programmierung

Definition

- Datenkapselung (encapsulation)
- Vererbung (inheritance)
- Polymorphismus (polymorphism)

Datenkapselung

- "Behälter" für zusammengehörende Methoden (Funktionen) und Eigenschaften (Daten)
- geregelter Zugriff auf Inhalt des Behälters (`private`, `public`)

Objektorientierte Programmierung

Definition

- Datenkapselung (encapsulation)
- Vererbung (inheritance)
- Polymorphismus (polymorphism)

Vererbung

- Weitergabe von Methoden und Eigenschaften
- Verwendung von bestehendem Programmcode
- saubere und vereinfachte Programmentwicklung

Objektorientierte Programmierung

Definition

- Datenkapselung (encapsulation)
- Vererbung (inheritance)
- Polymorphismus (polymorphism)

Polymorphismus

- Methoden mit gleicher Signatur aber unterschiedlicher Implementierung
- Überladen, Überschreiben und dynamisches/statisches Einbinden von Methoden

Klassen

Definition

- Sammlung von **Datenwerten** und **Methoden**(Funktionen)
- Datentyp ist eine **Referenz**

Beispiel

```
/** Represents a Cartesian (x,y) point */
public class Point {
     public double x, y;                    // The coordinates of the point
     public Point(double x, double y) {     // A constructor that
         this.x = x; this.y = y;            // initializes the fields
     }

     public double distanceFromOrigin(  ) {   // A method that operates on
         return Math.sqrt(x*x + y*y);       // the x and y fields
     }
}
```

Objekte

Definition

- repräsentiert den (Daten)-Wert einer **Klasse**,
  z.B. den spezifischen Punkt (X, Y) vom Datentyp, der durch Klasse `Point` beschrieben wird

Beispiel

```
// Create a Point object representing (2,-3.5).
Point p = new Point(2.0, -3.5);      //Create an object

//Using an object
double x = p.x;                      // Read a field of the object
p.y = p.x * p.x;                     // Set the value of a field
double d = p.distanceFromOrigin(  ); // Access a method of the object

//null reference
Point p = null;
```

Vererbung

- neue Klasse (Subclass) erweitert existierende Klasse (Superclass)
- Subclass erbt Mitglieder seiner Superclass
- Subclass kann neue Mitglieder deklarieren
- Subclass kann bestehende Mitglieder überschreiben (Override)

Vererbung (Forts.)

Beispiel (aus *JAVA in a Nutshell*)

```
public class Circle {
  // A class field
  public static final double PI= 3.14159;     // A useful constant

  // A class method: just compute a value based on the arguments
  public static double radiansToDegrees(double rads) {
    return rads * 180 / PI;
  }

  // An instance field
  public double r;                  // The radius of the circle

  // A constructor method to initialize variables
  public Circle(double r) {
    this.r=r;                       // Initialize the instance field r
  }

  // Instance method operates on the instance fields of an object
  public double area() {            // Compute the area of the circle
    return PI * r * r;
  }
}
```

Vererbung (Forts.)

```
public class PlaneCircle extends Circle {
  // We automatically inherit the fields and methods of Circle,
  // so we only have to put the new stuff here.
  // New instance fields that store the center point of the circle
  public double cx, cy;

  // A new constructor method to initialize the new fields
  // It uses a special syntax to invoke the Circle() constructor
  public PlaneCircle(double r, double x, double y) {
    super(r);       // Invoke the constructor of the superclass, Circle()
    this.cx = x;    // Initialize the instance field cx
    this.cy = y;    // Initialize the instance field cy
  }

  // The area() method is inherited from Circle
  // A new instance method that checks whether a point is inside the circle
  // Note that it uses the inherited instance field r
  public boolean isInside(double x, double y) {
    double dx = x - cx, dy = y - cy;            // Distance from center
    double distance = Math.sqrt(dx*dx + dy*dy);  // Pythagorean theorem
    return (distance < r);                      // Returns true or false
  }
}
```

Verstecken und Überschreiben

Beispiel (aus *JAVA in a Nutshell*)

```
class A {
  int i = 1;                // An instance field hidden by subclass B
  int f() { return i; }     // An instance method overridden by subclass B
}

class B extends A {
  int i;                    // This field hides i in A
  int f() {                 // This method overrides f() in A
    i = super.i + 1;        // It can retrieve A.i like this
    return super.f() + i;   // It can invoke A.f() like this
  }
}
```

Interfaces

Definition

- Referenztyp wie `class`
- alle Methoden `abstract` also keine Implementation, nur Programmierschnittstelle (API)
- Interfacemethoden müssen in Klasse implementiert werden
- erlauben Mehrfachvererbung, da Vererbung nur von einer Klasse aber mehreren Interfaces möglich ist

Interfaces (Forts.)

Beispiel (aus *JAVA in a Nutshell*)

```
public interface Centered {
    void setCenter(double x, double y);
    double getCenterX();
    double getCenterY();
}

public class CenteredRectangle extends Rectangle  implements Centered {
  // New instance fields
  private double cx, cy;
  // A constructor
  public CenteredRectangle(double cx, double cy, double w, double h) {
    super(w, h);
    this.cx = cx;
    this.cy = cy;
  }
  // We inherit all the methods of Rectangle but must
  // provide implementations of all the Centered methods.
  public void setCenter(double x, double y) { cx = x; cy = y; }
  public double getCenterX() { return cx; }
  public double getCenterY() { return cy; }
}
```

Keywords

`abstract`	Klassen: Methoden nicht implementiert, keine Instanzen
	Methoden: Methode erst in Subklasse implementiert
`final`	Klassen: keine Subklassen
	Methoden: nicht überschreibbar
	"Variablen": Konstanten (`final static`)
`private`	Zugriff nur innerhalb der Klasse
`protected`	Zugriff nur innerhalb Paket und Subklassen
`public`	freier Zugriff
`static`	Methoden: Klassenmethode, nicht instanziert
	"Variablen": Klassenvariable, nur eine Instanz für alle Subklassen

Pakete (Packages)

Definition

- Sammlung von Klassen, Interfaces und anderer Referenztypen
- Gruppieren von zusammengehörigen Klassen
- Gemeinsamer Namensraum
- JAVA Basispakete: `java.`, z.B. `java.lang`,`java.util`, Erweiterungen unter `javax`, z.B. `javax.swing`

Beispiel

```
//Declare a class to be part of a package
package my.testpackages.examples;

//Use packages
import java.io.File; //import a single type
import java.io.*;    //import "on demand"
```

Namensraum (Namespace)

- Aufteilung des JAVA Namensraumes durch global einmalige Paketnamen
- Verhindern der Namenskollision von Klassen, z.B. `java.util.List` und `java.awt.List`
- vollständiger Name erforderlich, einfacher Namen nur wenn:
  - automatisch importiert (`java.lang`)
  - Paket mit `import` Deklaration eingebunden (gilt nicht für Subpakete)

Beispiel

```
import java.util.*;  //Prevent naming conficts
import java.awt.*;
import java.awt.List;
```

JAVA Plattform Pakete

Auswahl

Paket	Beschreibung
`java.io`	Ein- und Ausgabe (Datei,Streams,...)
`java.lang`	Basisklassen, z.B. `String`,`Math`,`System`,`Exception`
`java.net`	Netzwerkunterstützung
`java.util`	Werkzeugklassen
`java.util.jar`	Lesen und Schreiben von JAR Dateien
`java.util.zip`	Lesen und Schreiben von ZIP Dateien
`java.awt`	Grafische Darstellung
`java.swing`	Grafische Darstellung
`javax.crypto`	Ver- und Entschlüsselung von Daten

JAVA Plattform Beispiele (aus *JAVA in a Nutshell*)

java.lang.String

```
String s = "Now";               // String objects have a special literal syntax
String t = s + " is " + 23.4;   // Concatenate strings with +
t = object.toString();          // Convert objects to strings with toString()
int len = t.length();           // Number of characters in the string: 16
String sub = t.substring(4);    // Extract substrings
boolean b = t.equals("hello"); // Compare strings
```

java.lang.Math

```
double d = Math.toRadians(27);      // Convert 27 degrees to radians
d = Math.cos(d);                    // Take the cosine
d = Math.sqrt(d);                   // Take the square root
d = Math.log(d);                    // Take the natural logarithm
double up = Math.ceil(d);           // Round to ceiling
double down = Math.floor(d);        // Round to floor
long nearest = Math.round(d);       // Round to nearest
```

JAVA Plattform Beispiele (Forts.)

java.lang.Number

```
String s = "-42";
byte b = Byte.parseByte(s);             // s as a byte
int i = Integer.parseInt(s);            // s as an int
double d = Double.parseDouble(s);       // s as a double

// The valueOf() method can handle arbitrary bases between 2 and 36
int i = Integer.valueOf("egg", 17).intValue();   // Base 17!

// Integer class can convert numbers to strings
String decimal = Integer.toString(42);
```

Formatierte Ausgabe (ab JAVA 5.0)

```
System.out.printf("%s logged in after %d attempts. Last login at: %tc%n",
                  username, numattempts, lastLoginDate);
double x = 1.234E9; // (1.234 billion)
// returns "1234000000.000000 1.234000e+09 1.234000e+09 1234.000000"
s = String.format("%f %e %g %g", x, x, x, x/1e6);
```

JAVA Plattform Beispiele (Forts.)

java.util.Array

```
import java.util.Arrays;

int[] intarray = new int[] { 10, 5, 7, -3 }; // An array of integers
Arrays.sort(intarray);                       // Sort it in place
int pos = Arrays.binarySearch(intarray, 7);  // Value 7 is found at index 2
pos = Arrays.binarySearch(intarray, 12);     // Not found: negative return value

// Arrays of objects can be sorted and searched too
String[] strarray = new String[] { "now", "is", "the", "time" };
Arrays.sort(strarray);   // sorted to: { "is", "now", "the", "time" }

// Arrays.fill() initializes array elements
byte[] data = new byte[100];          // An empty array; elements set to 0
Arrays.fill(data, (byte) -1);         // Set them all to -1
```

JAVA Plattform Beispiele (Forts.)

java.io

```
import java.io.*;

//Reading Text from a File filename
try {
  BufferedReader in = new BufferedReader(new FileReader(filename));
  String line;
  while((line = in.readLine()) != null) {  // Read line, check for end-of-file
    System.out.println(line);              // Print the line
  }
  in.close();    // Always close a stream when you are done with it
}
catch (IOException e) { { /* Handle exceptions */ }

//Writing Text to a File
try {
  File f = new File(homedir, ".config");
  PrintWriter out = new PrintWriter(new FileWriter(f));
  out.println("## Automatically generated config file. DO NOT EDIT!");
  out.close();  // We're done writing
}
catch (IOException e) { /* Handle exceptions */ }
```

JAVA Plattform Beispiele (Forts.)

JAVA Examples in a Nutshell
by David Flanagan,
3rd Edition, 2004, O'Reilly
`http://proquest.safaribooksonline.com/0596006209`
`http://examples.oreilly.com/jenut/`

Literatur

JAVA in a Nutshell
by David Flanagan,
6th Edition, 2014, O'Reilly
`http://proquest.tech.safaribooksonline.de/book/programming/java/9781449371296`

The Java Language Specification
`https://docs.oracle.com/javase/specs/`

Teil II: Open Source Physics Bibliothek

Zielstellung

Open Source Physics (OSP)

"Java applications and a code library with examples for teaching computational physics and doing simulations in the natural sciences and mathematics."

- einfache Implementierung und Visualisierung von physikalischen Berechnungen und Simulationen
- freie Verfügbarkeit (GNU GPL)
  `http://www.compadre.org/osp/`

Inhalt

JAVA Klassenbibliotheken für:

- Benutzerschnittstelle für Ein- und Ausgabe (inkl. Tooltips, Zoom, Mousesteuerung)
- Framework für grafische Darstellung/Visualisierung (2D,3D, inkl. Vielzahl von Diagrammtypen)
- Gitter, Skalar- und Vektorfelder, komplexe Felder
- Löser für Gewöhnliche Differentialgleichungen
- Aufzeichnen einer Simulation und Export als Filmdatei
- Erstellen von Applets
- Speichern von Simulationenzuständen in XML

OSP packages

... basieren auf JAVA 1.5 und Swing (Grafik)

org.opensourcephysics.

controls	Benutzerschnittstelle, OSP XML Framework
display	Darstellung 2D Objekte und Graphen
display2d	Darstellung von 2D Daten als Kontur- und Oberflächen grafik
display3d	Darstellung von 3D Objekten
ejs	Benutzerdefinierte Bedienschnittstelle
numerics	Numerische Analyseroutinen
tools	Dienstprogramme (Launcher, etc.)

Model-View-Control (MVC)

Model

- beschreibt die Physik und beinhaltet Daten über den Zustand des Systems sowie Methoden zu dessen Änderung

Control

- behandelt Benutzereingaben und leitet diese an andere Objekte weiter

View

- repräsentiert Daten in Tabellen, Diagrammen oder anderen grafischen Darstellungen

MVC Beispiel

Freier Fall - Model: FallingBall.java (1/2)

```
public class FallingBall {
  double y, v, t;              // instance variables
  double dt;                   // default package protection
  final static double g = 9.8; // constant (note non-use of Java convention)

  // Constructs a FallingBall at x=0 with v=0;
  public FallingBall() { // constructor
    System.out.println("A new FallingBall object is created.");
  }

  // Steps (advances) the postion of the ball using the Euler algorithm.
  public void step() {
    y = y+v*dt; // Euler algorithm for numerical solution
    v = v-g*dt;
    t = t+dt;
  }
  ...
```

MVC Beispiel

Freier Fall - Model: FallingBall.java (2/2)

```
  ...

  // Computes the position of the ball using the analytic solution of the equation of motion.
  // @param y0 double, v0 double
  // @return double
  public double analyticPosition(double y0, double v0) {
    return y0+v0*t-0.5*g*t*t;
  }

  //Computes the velocity of the ball using the analytic solution of the equation of motion.
  // @param v0 double
  // @return double
  public double analyticVelocity(double v0) {
    return v0-g*t;
  }
}
```

MVC Beispiel

Freier Fall - View(+Control): FallingBallApp.java (1/2)

```
public class FallingBallApp { // beginning of class definition

  // Starts the Java application.
  // @param args  command line parameters
  public static void main(String[] args) { // beginning of method definition
    FallingBall ball = new FallingBall(); // declaration and instantiation
    double y0 = 10;                       // example of declaration and assignment statement
    double v0 = 0;
    ball.t = 0; // note use of dot operator to access instance variable
    ball.dt = 0.01;
    ball.y = y0;
    ball.v = v0;
    while(ball.y>0) {
      ball.step();
    }
    ...
```

MVC Beispiel

Freier Fall - View(+Control): FallingBallApp.java (2/2)

```
    ...
    System.out.println("Results");
    System.out.println("final time = "+ball.t);
    // displays numerical results
    System.out.println("y = "+ball.y+" v = "+ball.v);
    // displays analytic results
    System.out.println("analytic y = "+ball.analyticPosition(y0, v0));
    System.out.println("analytic v = "+ball.analyticVelocity(v0));
    System.out.println("acceleration = "+FallingBall.g);
  } // end of method definition
} // end of class definition
```

MVC+OSP Beispiel

Freier Fall - Model: `Particle.java`

```
abstract public class Particle {
  double y, v, t; // instance variables
  double dt;      // time step

  // Constructs a Particle.
  public Particle() { // constructor
    System.out.println("A new Particle is created.");
  }

  //Steps (advances) the dynamical variables using a numeric method.
  abstract protected void step();

  //Computes the position at the current time using the analytic solution.
  // @return double
  abstract protected double analyticPosition();

  //Computes the velocity at the current time using the analytic solution.
  // @return double
  abstract protected double analyticVelocity();
}
```

MVC+OSP Beispiel

Freier Fall - Model: FallingParticle.java (1/2)

```
public class FallingParticle extends Particle {
  final static double g = 9.8;   // constant
  private double y0 = 0, v0 = 0; // initial position and velocity

  // Constructs a Falling Particle with the given initial conditions.
  // @param y double, v double
  public FallingParticle(double y, double v) { // constructor
    System.out.println("A new FallingParticle object is created.");
    this.y = y; // instance value set equal to passed value
    this.v = v; // instance value set equal to passed value
    y0 = y;     // no need to use "this" because there is only one y0
    v0 = v;
  }

  // Steps (advances) the dynamical variables using the Euler method.
  public void step() {
    y = y+v*dt; // Euler algorithm
    v = v-g*dt;
    t = t+dt;
  }
  ...
```

MVC+OSP Beispiel

Freier Fall - Model: `FallingParticle.java` (2/2)

```
  ...
  // Computes the position of the ball at the current time using the analytic solution
  // of the equation of motion.
  // @param y0 double, v0 double
  // @return double
  public double analyticPosition() {
    return y0+v0*t-(g*t*t)/2.0;
  }

  // Computes the velocity of the ball at the current time using the analytic solution
  // of the equation of motion.
  // @param y0 double, v0 double
  // @return double
  public double analyticVelocity() {
    return v0-g*t;
  }
}
```

MVC+OSP Beispiel

Freier Fall - Model: FallingParticlePlotApp.java (1/2)

```
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.*;
public class FallingParticlePlotApp extends AbstractCalculation {
  PlotFrame plotFrame = new PlotFrame("t", "y", "Falling Ball");

  // Calculates the trajectory of a falling particle and plots the position as a function of time.
  public void calculate() {
    plotFrame.setAutoclear(false); // data not cleared at beginning of each calculation
    // gets initial conditions
    double y0 = control.getDouble("Initial y");
    double v0 = control.getDouble("Initial v");
    // sets initial conditions
    Particle ball = new FallingParticle(y0, v0);
    // gets parameters
    ball.dt = control.getDouble("dt");
    double t = ball.t; // gets value of time from ball object
    ...
```

MVC+OSP Beispiel

Freier Fall - Model: FallingParticlePlotApp.java (2/2)

```
    ...
    while(ball.y>0) {
      ball.step();
      plotFrame.append(0, ball.t, ball.y);
      plotFrame.append(1, ball.t, ball.analyticPosition());
    }
  }

  // Resets the program to its initial state.
  public void reset() {
    control.setValue("Initial y", 10);
    control.setValue("Initial v", 0);
    control.setValue("dt", 0.01);
  }

  // Starts the Java application.
  // @param args  command line parameters
  public static void main(String[] args) {
    // sets up calculation control structure using this class
    CalculationControl.createApp(new FallingParticlePlotApp());
  }
}
```

AbstractCalculation

- einfache Berechungen ohne Benutzerinteraktion

```
import org.opensourcephysics.controls.*;

public class MyApp extends AbstractCalculation {
...
  // Hier muss die Berechnung implementiert werden!
  public void calculate() {
    double d=control.getDouble("Parametername"); //Auslesen von Parametern aus dem control
    ...}

  // Hier sollten die Parameter und Variablen initialisiert/zurueckgesetzt werden.
  public void reset() {
    control.setValue("Parametername",0.1); //Erzeugt und initialisiert neuen Parameter;
    ...}

  // Erzeugen des controls
  public static void main( String[] args) {
    CalculationControl.createApp(new MyApp());
  }
}
```

AbstractSimulation

- Berechungen mit Benutzerinteraktion

```
import org.opensourcephysics.controls.*;

public class MyApp extends AbstractSimulation {
...
  // Hier muss die Berechnung initialisiert werden!
  public void initialize() {
  double d=control.getDouble("Parametername"); //Auslesen von Parametern aus dem control
    ...}
  // Fuehrt einen Berechnungsschritt aus (wird jede 1/10s automatisch aufgerufen)
  public void doStep() {
    ...}
  // Hier sollten die Parameter und Variablen initialisiert/zurueckgesetzt werden.
  public void reset() {
    control.setValue("Parametername",0.1); //Erzeugt und initialisiert einen
    ...}                                   //Parameter
  // Erzeugen des controls
  public static void main( String[] args) {
    SimulationControl.createApp(new MyApp());
  }
}
```

AbstractSimulation (Forts.)

- Veränderbare Parameter und Steuerung der Simulation

```
import org.opensourcephysics.controls.*;

public class MyApp extends AbstractSimulation {
...
  public void doStep() {
    ...
    if (...) control.calculationDone("Finished").  // Abbruch der Simulation
    ...}

  public void reset() {
    enableStepsPerDisplay(true);                 //Erlaubt mehrere Schritte bevor
    setStepsPerDisplay(10);                      //Anzeige aktualisiert wird.
    control.setAdjustableValue("dt",0.1);        //Erzeugt und initialisiert einen
    ...}                                         //veraenderbaren Parameter.

  // Diese Methode wird bei jedem Start der Simulation ausgefuehrt
  public void startRunning(){
       dt=control.getDouble("dt");
    ...}
}
```

PlotFrame

- Darstellung von 2D Daten und Funktionen

```
import org.opensourcephysics.frames.*;

 // Erzeugen eines neuen Diagramms
 PlotFrame frame = new PlotFrame("x-Achse","y-Achse","Diagrammtitel");

 frame.append(idx,x,y);          // Neuen Punkt (x,y) an Datensatz mit Index idx anhaengen

 frame.setConnected(true);       //Datenpunkte aller Datensaetze durch Linie verbinden
 frame.setConnected(idx,true); //Datenpunkte von Datensatz idx durch Linie verbinden

 frame.setAutoclear(false);      //Datensaetze bei Initialisierung nicht loeschen

 frame.setMessage("Zeit:"+t);  //Text in Infofenster setzen

 frame.clearData();              //Alle Datensaetze und Zeichenobjekte loeschen
```

DisplayFrame

- Darstellung von 2D Plots und Objekten ohne Datensatzverwaltung (sonst wie `PlotFrame`)

```
import org.opensourcephysics.frames.*;    //Package mit DisplayFrame
import org.opensourcephysics.display.*;   //Package mit Drawable Interface und vordef. Objekten
import java.awt.*;                        //Java Abstract Window Toolkit

 // Erzeugen eines neuen Diagramms
 DisplayFrame frame = new DisplayFrame("x-Achse","y-Achse","Diagrammtitel");

 frame.addDrawable(object);               //Neues Zeichenobjekt (Typ Drawable) hinzufuegen;
 frame.addDrawable(new Circle(ix,iy,ir)); //Kreis mit Radius ir an Pos. (ix,iy) (!Pixelkoord.!)

 frame.setPreferredMinMax(xmin,xmax,ymin,ymax); //Setzen des Darstellungsbereiches

 frame.clearDrawables();                  //Loescht alle hinzugefuegten Zeichenobjekte
```

Drawable Interface

- Implementierung der grafischen Darstellung in einer Klasse

```
import org.opensourcephysics.display.*;   //Package mit Drawable Interface und vordef. Objekten
import java.awt.*;                         //Java Abstract Window Toolkit

class MyDrawable implements Drawable {   //Klasse fuer eigenes Zeichenobjekt
    double x,y,w;                         //muss draw() Methode implementieren

    // Konstruktor
    MyDrawable(double x,double y,double w) {
      this.x=x; this.y=y; this.w=w;
    }
    // Zeichen-Methode
    public void draw(DrawingPanel drawingPanel, Graphics g) {
      int ix=drawingPanel.xToPix(x);      //Umrechnung Welt in Pixel Koord.
      int iy=drawingPanel.yToPix(y);
      int iwx=(int)(drawingPanel.getXPixPerUnit()*w);
      int iwy=(int)(drawingPanel.getYPixPerUnit()*w);
      g.setColor(Color.green);            //Zeichenfarbe setzen
      g.fillRect(ix, iy, iwx, iwy);       //Zeichenroutine fuer Rechteck aus java.awt
    }
}
```

Display3DFrame

- Darstellung von 3D Objekten

```
import java.awt.*;
import org.opensourcephysics.frames.Display3DFrame;
import org.opensourcephysics.display3d.simple3d.*;

// Erzeugen einer neuen 3D Darstellung
Display3DFrame frame = new Display3DFrame("Titel");

Element ball = new ElementEllipsoid();  //Erzeugen eines neuen vordef. 3D-Objekts;

ball.setXYZ(ix,iy,iz);                  //Setzen der Position (ix,iy,iz) eines Objekts

ball.setSizeXYZ(irx,iry,irz);           //Setzen der Radien eines ElementEllipsoid-Objekts

iz=ball.getZ();                         //Auslesen der z-Koordinate eines Objekts

frame.addElement(ball);                 //Objekt zur Darstellung hinzufuegen
```

Scalar2DFrame

- Darstellung von 2D Dichtefunktionen
- andere Darstellungsarten: Oberflächenplot, Konturlinien, interpolierter Dichteplot

```
import org.opensourcephysics.frames.Scalar2DFrame;

// Erzeugen eines neuen Dichtediagramms
Scalar2DFrame frame = new Scalar2DFrame("x","y","Titel");

double data[][];
frame.setAll(data, xmin, xmax,ymin, ymax); //Setzen von Daten und x-y-Bereich
frame.setPreferredMinMax(xmin, xmax, ymin, ymax);  // Setzen x-y-Bereich
frame.setAll(data);                        //Setzen der Daten

frame.setZRange(false, 0.0, 1.0);          //Auswahl automat. Skalierung und z-Bereich

double x = frame.indexToX(i);              //Bestimmung von x- und y-Werten aus
double y = frame.indexToY(j);              //Feldindizes data[i][j]
```

HistogramFrame

- Darstellung von Histogrammen

```
import org.opensourcephysics.frames.HistogramFrame;

// Erzeugen eines neuen Histogramms
HistogramFrame frame = new HistogramFrame("x","P(x)","Histogramm");

frame.setBinWidth(0.1);                  //Setzen der Bingroesse

frame.append(x);                         //Wert hinzufuegen

frame.getBinWidth();                     //Bingroesse bestimmen

double[][] data = frame.getPoints();     //Daten aus Histrogramm auslesen
                                         // data[0][..]- x-Werte, data[1][..] P(x)-Werte
```

Function Interface

- Beschreibung von Funktionen mit einer Variablen

```
import org.opensourcephysics.numerics.*;

public class QuadraticPolynomial implements Function {  //Interface-> Methode evaluate()

 double a,b,c;

 //Konstruktor zu Initialisierung der Parameter
 public QuadraticPolynomial(double a, double b, double c) {
   this.a=a;
   this.b=b;
   this.c=c;
 }

 //Berechnung des Funktionswertes an der Stelle x
 public double evaluate(double x) {
   return a*x*x+b*x+c;
 }
}
```

FunctionDrawer

- Darstellung von Function Objekten

```
import org.opensourcephysics.numerics.*;
import org.opensourcephysics.frames.*;

PlotFrame frame = new PlotFrame("x", "f(x)", "Function Plot");

// Erzeuge Function-Objekt
Function f = new QuadraticPolynomial(0.1,-2,5.1);

// Erzeuge Objekt fuer grafische Darstellung einer Function
FunctionDrawer fDrawer = new FunctionDrawer(f);

// Setze Darstellungsparameter (Intervall, Anzahl Punkte, Fuellung bis y=0)
fDrawer.initialize(xmin, xmax , iPoints, bFilled);

// Haenge FunctionDrawer an PlotFrame
frame.addDrawable(fDrawer);
```

ParsedFunction

- Übersetzen eines Strings in eine Funktion

```
import org.opensourcephysics.numerics.*;

String strFunc="sin(x)";

Function f  = null;
   try {
     f = new ParsedFunction(strFunc, "x");
   }
   // Fehler abfangen
   catch(ParserException ex) {
     control.println("Error parsing function string: "+strFunc);
     return;
   }
```

ODE Interface

- Definition des Differentialgleichungssystems

```
import org.opensourcephysics.numerics.*;

class MyODE implements ODE {
  double [] state = new double [3];      //Anlegen des Zustandsvektors

  public MyODE(double y, double v); {
     state[0]=y; state[1]=v; state[2]=0; //Anfangszustand initialisieren
  }
  public double[] getState() {           //Liefert Zustandsvektor, muss implementiert werden
     return state;
  }

  //Liefert Raten (Ableitungen des Zustandsvektors), muss implementiert werden
  public void getRate(double[] state, double[] rate) {
     rate[0] = state[1];                 //Ableitung dy/dt
     rate[1] = -9.81;                    //Ableitung dv/dt
     rate[2] = 1;                        //Ableitung dt/dt
  }
}
```

ODE Solver

- Numerische Lösung des Differentialgleichungssystems mit unterschiedlichen Verfahren (Euler, Runge-Kutta etc.)

```
import org.opensourcephysics.numerics.*;

MyODE myODEmodel = new MyODE(y,v);          //ODE Objekt erzeugen

//Erzeugen eines EulerRichardson Solver fuer myODEmodel
ODESolver mySolver = new EulerRichardson(myODEmodel);

mySolver.step();                            //Ausfuehren eines Solver-Schrittes

mySolver.setStepSize(dt);                   //Schrittweite des Solvers setzen
```