Neuerungen in Java 5/6/7

Überblick

- Java 5
  - neue Sprachfeatures
  - Erweiterungen Klassenbibliothek
- Java 6
  - Erweiterungen Klassenbibliothek
- Java 7
  - Java wird Open Source
- Referenzen

Java 5 – neue Sprachfeatures

- Generics
- Autoboxing / Unboxing
- Enhanced „for“ Loop
- VarArgs
- Enums
- Static import
- Metadata (Annotations)

Java 5 – Generics I

- generische Datentypen
- ähnlich wie Templates in C++
- Typsicherheit zur Compilezeit
- Unterstützung durch Collections Framework; Legacy Code führt zu Compiler-Warnungen

Generics II
Legacy Code

```
1  import java.util.ArrayList;
2  import java.util.Iterator;
3  import java.util.List;
4
5
6  public class LegacyCollections
7  {
8     public static void main (String [] args)
9     {
10       List stringList = new ArrayList ();
11
12       stringList.add ("alpha");
13       stringList.add ("beta");
14       stringList.add ("gamma");
15       stringList.add (new Integer (33));
16
17
18
19       for (Iterator itr = stringList.iterator (); itr.hasNext (); )
20       {
21          String str = (String) itr.next ();  // ClassCastException zur Laufzeit
22          System.out.println ("str is: " + str);
23       }
24    }
25 }
```

Generics III
Typsichere Collections

```
1  import java.util.ArrayList;
2  import java.util.Iterator;
3  import java.util.List;
4
5
6  public class TypeSafeCollections
7  {
8     public static void main (String [] args)
9     {
10       List <String> stringList = new ArrayList <String> ();
11
12       stringList.add ("alpha");
13       stringList.add ("beta");
14       stringList.add ("gamma");
15       stringList.add (new Integer (33));
16
17
18
19       for (Iterator <String> itr = stringList.iterator (); itr.hasNext (); )
20       {
21          String str = itr.next ();  // kein Cast mehr notwendig
22          System.out.println ("str is: " + str);
23       }
24    }
25 }
26
```

The method add(String) in the type List<String> is not applicable for the arguments (Integer)

Press 'F2' for focus.

Generics IV
typsichere Collections (dekompiliert)

```
1// Decompiled by Jad v1.5.8g. Copyright 2001 Pavel Kouznetsov.
2// Jad home page: http://www.kpdus.com/jad.html
3// Decompiler options: packimports(3)
4// Source File Name:   TypeSafeCollections.java
5
6import java.io.PrintStream;
7import java.util.*;
8
9public class TypeSafeCollections
10{
11
12    public TypeSafeCollections()
13    {
14    }
15
16    public static void main(String args[])
17    {
18        List stringList = new ArrayList();
19        stringList.add("alpha");
20        stringList.add("beta");
21        stringList.add("gamma");
22        String str;
23        for(Iterator itr = stringList.iterator(); itr.hasNext(); System.out.println((new StringBuilder("str is: ")).append(str).toString()))
24            str = (String)itr.next();
25
26    }
27}
```

Autoboxing / Unboxing I

- automatische Umwandlung zwischen elementaren Datentypen und ihren Wrapper-Objekten

Autoboxing / Unboxing II
Legacy Code

```
1  import java.util.ArrayList;
2  import java.util.Iterator;
3  import java.util.List;
4
5
6  public class AutoboxingExample
7  {
8     public static void main (String [] args)
9     {
10       List intList = new ArrayList ();
11       intList.add (new Integer (1));
12       intList.add (new Integer (2));
13       intList.add (new Integer (4));
14       intList.add (new Integer (8));
15       intList.add (new Integer (16));
16
17       int sum = 0;
18       for (Iterator itr = intList.iterator (); itr.hasNext (); )
19       {
20          Integer nextInt = (Integer) itr.next ();
21          sum = sum + nextInt.intValue ();
22       }
23       System.out.println ("sum: " + Integer.toString (sum));
24    }
25
26 }
```

Autoboxing / Unboxing III Vereinfachung 1

```
1  import java.util.ArrayList;
2  import java.util.Iterator;
3  import java.util.List;
4
5
6  public class AutoboxingExample
7  {
8     public static void main (String [] args)
9     {
10       List intList = new ArrayList ();
11       intList.add (1);
12       intList.add (2);
13       intList.add (4);
14       intList.add (8);
15       intList.add (16);
16
17       int sum = 0;
18       for (Iterator itr = intList.iterator (); itr.hasNext (); )
19       {
20          Integer nextInt = (Integer) itr.next ();
21          sum = sum + nextInt;
22       }
23       System.out.println ("sum: " + Integer.toString (sum));
24    }
25
26 }
```

Autoboxing / Unboxing IV Vereinfachung 2

```
1  import java.util.ArrayList;
2  import java.util.List;
3
4
5  public class AutoboxingExample
6  {
7     public static void main (String [] args)
8     {
9        List <Integer> intList = new ArrayList <Integer> ();
10       intList.add (1);
11       intList.add (2);
12       intList.add (4);
13       intList.add (8);
14       intList.add (16);
15
16       int sum = 0;
17       for (Integer nextInt : intList)
18       {
19          sum = sum + nextInt;
20       }
21       System.out.println ("sum: " + Integer.toString (sum));
22    }
23
24 }
```

Autoboxing / Unboxing IV Vereinfachung 2 (dekompiliert)

```
 6import java.io.PrintStream;
 7import java.util.*;
 8
 9public class AutoboxingExample
10{
11
12    public AutoboxingExample()
13    {
14    }
15
16    public static void main(String args[])
17    {
18        List intList = new ArrayList();
19        intList.add(Integer.valueOf(1));
20        intList.add(Integer.valueOf(2));
21        intList.add(Integer.valueOf(4));
22        intList.add(Integer.valueOf(8));
23        intList.add(Integer.valueOf(16));
24        int sum = 0;
25        for(Iterator iterator = intList.iterator(); iterator.hasNext();)
26        {
27            Integer nextInt = (Integer)iterator.next();
28            sum += nextInt.intValue();
29        }
30
31        System.out.println((new StringBuilder("sum: ")).append(Integer.toString(sum)).toString());
32    }
33}
```

Enhanced „for“ Loop I

- „für jedes Element vom Typ <T> in Menge <M>“
- auf eigenen Typen verfügbar durch Implementierung des Interface java.lang.Iterable

Enhanced „for“ Loop II

```
1  import java.util.Arrays;
2  import java.util.Iterator;
3  import java.util.List;
4
5  public class ForEachExample implements Iterable <Integer>
6  {
7     List <Integer> m_List = Arrays.asList (new Integer [] {1, 2, 3, 4, 5, 6});
8
9     public Iterator <Integer> iterator ()
10    {
11       return m_List.iterator ();
12    }
13
14    public static void main (String [] args)
15    {
16       System.out.println ("List content follows: ");
17       ForEachExample instance = new ForEachExample ();
18       for (Integer item : instance)
19       {
20          System.out.println (item.toString ());
21       }
22    }
23 }
```

Enhanced „for“ Loop III dekompiliert

```
6 import java.io.PrintStream;
7 import java.util.*;
8
9 public class ForEachExample
10     implements Iterable
11 {
12
13     public ForEachExample()
14     {
15         m_List = Arrays.asList(new Integer[] {
16             Integer.valueOf(1), Integer.valueOf(2), Integer.valueOf(3), Integer.valueOf(4), Integer.valueOf(5), Integer.valueOf(6)
17         });
18     }
19
20     public Iterator iterator()
21     {
22         return m_List.iterator();
23     }
24
25     public static void main(String args[])
26     {
27         System.out.println("List content follows: ");
28         ForEachExample instance = new ForEachExample();
29         Integer item;
30         for(Iterator iterator1 = instance.iterator(); iterator1.hasNext(); System.out.println(item.toString()))
31             item = (Integer)iterator1.next();
32
33     }
34
35     List m_List;
36 }
```

VarArgs I

- variable Anzahl Argumente in Methoden / Konstruktoren
- immer letztes Argument der Methode
- immer vom selben Typ

VarArgs II

```
2  public class VarArgsExample
3  {
4     public VarArgsExample (String firstArg, Integer... varArg)
5     {
6        System.out.println (firstArg);
7        int count = 1;
8        for (Integer item : varArg)
9        {
10          System.out.println ("argument " + count++ + ": " + item);
11       }
12    }
13
14    public static void main (String [] args)
15    {
16       VarArgsExample example1 = new VarArgsExample ("huhu", 8);
17       VarArgsExample example2 = new VarArgsExample ("hihi", 9, 10, 11);
18       VarArgsExample example3 = new VarArgsExample ("hehe", new Integer [] {12, 13});
19
20    }
21 }
```

VarArgs III (dekompiliert)

```
6 import java.io.PrintStream;
7
8 public class VarArgsExample
9 {
10
11     public transient VarArgsExample(String firstArg, Integer varArg[])
12     {
13         System.out.println(firstArg);
14         int count = 1;
15         Integer ainteger[] = varArg;
16         int i = 0;
17         for(int j = ainteger.length; i < j; i++)
18         {
19             Integer item = ainteger[i];
20             System.out.println((new StringBuilder("argument ")).append(count++).append(": ").append(item).toString());
21         }
22
23     }
24
25     public static void main(String args[])
26     {
27         VarArgsExample example1 = new VarArgsExample("huhu", new Integer[] {
28             Integer.valueOf(8)
29         });
30         VarArgsExample example2 = new VarArgsExample("hihi", new Integer[] {
31             Integer.valueOf(9), Integer.valueOf(10), Integer.valueOf(11)
32         });
33         VarArgsExample example3 = new VarArgsExample("hehe", new Integer[] {
34             Integer.valueOf(12), Integer.valueOf(13)
35         });
36     }
```

Enums I

- Aufzählungstypen
- Aufzählungen sind als Klassen, ihre Werte als Objekte realisiert
- können in switch-Statements verwendet werden
- Implementieren Comparable und Serializable
- Unterstützung durch Collections-Framework (EnumSet, EnumMap)

Enums II

```
1  public class EnumExample
2  {
3     public enum Color
4     {
5        ROT (255, 0, 0), GRUEN (0, 255, 0), BLAU (0, 0, 255), GELB (255, 255, 0);
6
7        private final int r;
8        private final int g;
9        private final int b;
10
11       Color (int r, int g, int b)
12       {
13          this.r = r;
14          this.g = g;
15          this.b = b;
16       }
17
18       public String toRGB ()
19       {
20          return "(" + r + "," + g + "," + b + ")";
21       }
22    }
23
24    public static void compareColors (Color c1, Color c2)
25    {
26       System.out.print (c1 + (c1.equals (c2) ? " = " : " != ") + c2);
27    }
28
29    public static void main (String [] args)
30    {
31       System.out.println ("--");
32       System.out.println (Color.ROT);
33       System.out.println (Color.BLAU);
34       System.out.println (Color.ROT);
35       // equals funktioniert auch
36       System.out.println ("--");
37       compareColors (Color.ROT, Color.BLAU);
38       compareColors (Color.ROT, Color.ROT);
39       compareColors (Color.BLAU, Color.ROT);
40       compareColors (Color.ROT, Color.ROT);
41       System.out.println ("--");
42       for (Color f : Color.values ())
43       {
44          System.out.println (f + ":" + f.toRGB ());
45       }
46    }
47 }
```

Static import I

- erlaubt unqualifizierten Zugriff auf statische Member ohne vom Typ zu erben, der die statischen Member hält
- individuell einzelnen Member oder alle statischen Member eines Typs

Static import II

```
1  import static java.lang.Math.cos;
2
3  import java.util.ArrayList;
4
5  import static java.lang.Class.*;
6
7  public class StaticImportExample
8  {
9     public static void main (String [] args)
10    {
11       try
12       {
13          Class arrayListClass = forName (ArrayList.class.getName ());
14       }
15       catch (ClassNotFoundException e)
16       {
17          e.printStackTrace();
18       }
19
20       double cos_PI_HALF = cos (Math.PI / 2d);
21    }
22 }
```

Annotations I

- Anbringen von Zusatzinformationen, die zur Compile- und/oder Laufzeit zur Verfügung stehen
- erlaubt das Weglassen von handgeschriebenem ‚boilerplate‘ Code, der stattdessen generiert wird
- als Applikationsentwickler häufiger „verwenden“ als „definieren“
- Annotations vorgegeben durch JDK / Klassenbibliotheken (z. B. JUnit4)

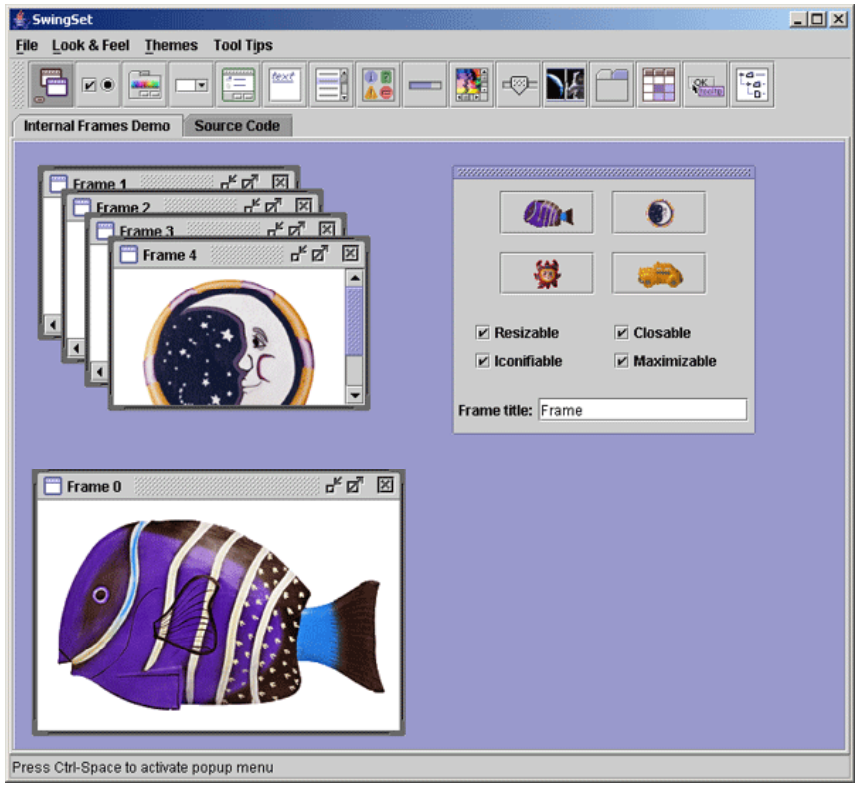
Annotations II

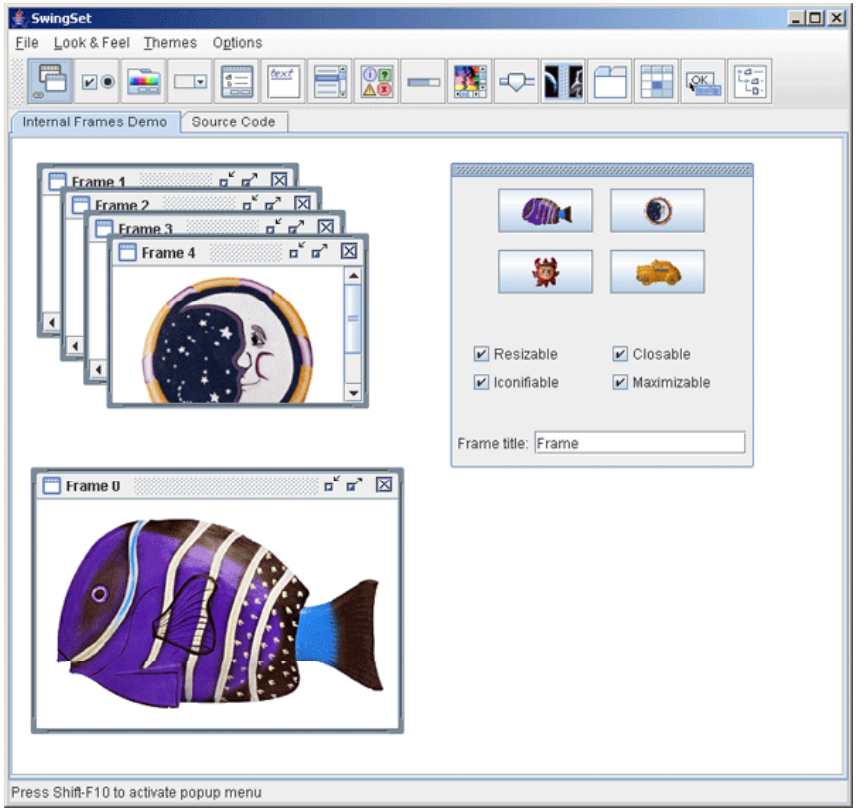
```java
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import java.lang.reflect.Method;


public class AnnotationExample
{
   /**
    * Meta-Annotation: Information steht zur Laufzeit zur Verfügung
    * (weitere Ausprägungen: CLASS / SOURCE)
    */
   @Retention (RetentionPolicy.RUNTIME)
   /**
    * Meta-Annotation: wo kann diese Annotation stehen
    * (weitere Ausprägungen: FIELD / PARAMETER / CONSTRUCTOR / LOCAL_VARIABLE / ANNOTATION_TYPE / PACKAGE
    */
   @Target (ElementType.METHOD)
   public @interface MyAnnotation
   {
      int id ();
      String description ();
   }

   @MyAnnotation (id=-1, description="sample")
   public static void main (String [] args)
   {
      Class cls = AnnotationExample.class;
      for (Method m : cls.getMethods ())
      {
         MyAnnotation annotation = m.getAnnotation (MyAnnotation.class);
         if (annotation != null)
         {
            System.out.println ("annotation id          : " + annotation.id ());
            System.out.println ("annotation description: " + annotation.description ());
         }
      }
   }
}
```

Java 5 – Erweiterungen Klassenbibliothek

- neues Look and Feel 'Ocean‘
- Java Management Extensions (JMX)
- Anpassung und Erweiterung des Collection Framework (u. a. Queue, Generics)
- Concurrency Framework
- Formatted Input / Output

Ocean Metal Look and Feel





Java 6 – Erweiterungen Klassenbibliothek

- JDBC 4.0
- Anpassung und Erweiterung des Collection Framework (u. a. Deque)
- Scripting Schnittstelle
- .NET interoperabler Webservice per Annotation
- System Tray Unterstützung
- Splash Screen Unterstützung

JDBC 4.0

```
1 public class Employee
2 {
3    private int employeeId;
4    private String firstName;
5    private String lastName;
6
7    public int getEmployeeId () { return employeeId; }
8    public void setEmployeeId (int employeeId) { this.employeeId = employeeId; }
9    public String getFirstName () { return firstName; }
10   public void setFirstName (String firstName) { this.firstName = firstName; }
11   public String getLastName () { return lastName; }
12   public void setLastName (String lastName) { this.lastName = lastName; }
13 }
14
15 //
16
17 interface EmployeeQueries extends BaseQuery
18 {
19   @Select (sql="SELECT employeeId, firstName, lastName FROM employee")
20   DataSet<Employee> getAllEmployees ();
21
22   @Update (sql="delete from employee")
23   int deleteAllEmployees ();
24 }
25
26 //
27   Connection con = ...
28   EmployeeQueries empQueries = con.createQueryObject (EmployeeQueries.class);
29   DataSet<Employee> empData = empQueries.getAllEmployees ();
30
```

Scripting Schnittstelle

```
1  import javax.script.*;
2
3  public class EvalScript
4  {
5     public static void main (String [] args) throws Exception
6     {
7        ScriptEngineManager factory = new ScriptEngineManager ();
8
9        ScriptEngine engine = factory.getEngineByName ("JavaScript");
10
11       engine.eval ("print('Hello, World')");
12    }
13 }
```

.NET interoperabler Webservice per Annotation I

```
1
2  import javax.jws.WebMethod;
3  import javax.jws.WebService;
4
5  @WebService ()
6  public class Hello
7  {
8     private String message = new String ("Hello, ");
9
10    @WebMethod ()
11    public String sayHello (String name)
12    {
13       return message + name + ".";
14    }
15 }
```

.NET interoperabler Webservice per Annotation II

```
1 import javax.xml.ws.WebServiceRef;
2
3 import helloservice.endpoint.HelloService;
4 import helloservice.endpoint.Hello;
5
6 public class HelloClient
7 {
8    @WebServiceRef (wsdlLocation = "http://localhost:8080/helloservice/hello?wsdl")
9    static HelloService service;
10   public static void main (String [] args)
11   {
12      try
13      {
14         HelloClient client = new HelloClient ();
15         Hello port = service.getHelloPort();
16         String response = port.sayHello ("Peter");
17      }
18      catch (Exception e)
19      {
20         e.printStackTrace ();
21      }
22   }
23 }
```

Java 7 – Java wird Open Source

- Freigabe aller wesentlichen JDK-Quelltexte der SE bis Mitte 2007 (GPLv2, Classpath exception)
- bereits verfügbar: javac, HotSpot VM, JavaHelp
- Marke Java und Logo bleiben geschützt

Referenzen

- Java 5 Docs http://java.sun.com/j2se/1.5/
- Java 6 RC Docs http://java.sun.com/javase/6
- Free and Open Source Java FAQ
  http://www.sun.com/software/opensource/java/faq.jsp