Van C++ naar Java

Van boekhouder naar ontwerper

THRijswijk

van C++ naar Java

studiematriaal

! Deze sheets

! Zoek zelf een boek:
" **Java tutorial**. Ook online zie http://java.sun.com/docs/books/tutorial/index.html
" diverse Java tutorials op WWW
" Thinking in Java. Ook online zie http://www.mindview.net/Books/TIJ/
" Java voor studenten
" ...

! Java 2 SDK Standard Edition version 1.5 (30 september 2004)
" http://java.sun.com
! IBM J9 voor QNX
" http://www-106.ibm.com/developerworks/java/

THRijswijk

van C++ naar Java

Wat gaan we doen?

! Enkele vragen beantwoorden.
" Wat is er zo bijzonder aan Java?
" Wat zijn de verschillen tussen C++ en Java?
" ...

! Enkele voorbeelden bekijken.
" Welkom bij de Java workshop
" Breuken
" Honden
" ...

THRijswijk

Java

eigenschappen

! Java is eenvoudiger te gebruiken dan C++ maar ook minder krachtig.
! Java is eenvoudig te gebruiken voor C++ programmeurs.
! Java is (bijna) volledig object georiënteerd.
! Java is platform onafhankelijk. JVM=Java Virtual Machine
" AWT (Abstract Windows Toolkit)
" Swing (pure Java alternatief voor AWT, recommended)
" **Multi-threading**
" Networking
! Java is ideaal voor WWW applicaties.

THRijswijk

Verschillende soorten Java programma's

Java is niet alleen voor het WWW

! **Applicaties** (standalone applications)
" Tekst in- en uitvoer te vergelijken met een C++ programma met in-en uitvoer via cin en cout.
" Grafische gebruikers interface (GUI) te vergelijken met een Windows programma.

! **Applets**
" Een in een HTML document opgenomen applicatie die uitgevoerd wordt door de browser.

THRijswijk

Javascript

javascript != java

! Dit is een zeer eenvoudige programmeertaal die in browsers te gebruiken is.

! Een Javascript is in source code opgenomen in een html file.

! Heeft dezelfde syntax als Java.

! Heeft verder **NIETS** met Java te maken! Is **niet** OO, **niet** herbruikbaar (alleen copy-paste), **niet** onderhoudbaar.

THRijswijk

Java en OOP

Verschillen met C++.

! **Geen** globale variabelen en functies mogelijk.
! **Geen** multiple inheritance wel multiple interfaces.
! Interfaces. Vergelijkbaar met pure abstract base class in C++.
! **Één** class hiërarchie. Alle classes erven direct of indirect over van class Object.
! **Geen** templates. Wel generics. new in 1.5
! **Geen** operator overloading.
! Array en String zijn "echte" classes.
! Objecten zijn **alleen** via "reference" bereikbaar. Alle objecten bevinden zich op de heap.
! **Geen** scheiding van interface en implementatie binnen **één** class. **Geen** headerfiles.

THRijswijk

Java variabelen

Java kent twee soorten variabelen.

! Variabelen van een **primitive data type**.
" boolean (true of false)
" char (16 bits Unicode)
" byte (8 bits signed two's complement)
" short (16 bits signed two's complement)
" int (32 bits signed two's complement)
" long (64 bits signed two's complement)
" float (32 bits IEEE 754-1985 FP)
" double (64 bits IEEE 754-1985 FP)
! Variabelen van een **object reference type.**
" String
" int[]
" ...

Voor alle primitieve data types zijn wrapper classes beschikbaar o.a. voor coversie van/naar String b.v. Long

THRijswijk

Java references

! Java object variabelen zijn **altijd** "pointers" (in Java "references" genoemd!).

! Een object variabele refereert naar een object (op de heap) of heeft de waarde **null**.

! Objecten worden aangemaakt met **new**:
Breuk b=new Breuk(3, 7);

! **Toekennen** en **vergelijken** heeft **pointer semantics**!
Breuk c=new Breuk(3, 7);
if (b!=c)
  System.out.println("Dat is gek!")

THRijswijk

Object assignment

Hond h1=new Hond();

h1

Hond h2;

h2=null

h2=h1;

h2

Hond h3=new Hond(h1);

h3

THRijswijk

Java en Structured Programming

Verschillen met C (C++).

- **Multi threading.** Programma draait als proces, elk programma bestaat uit meerdere threads.
- Automatic garbage collection. **Géén** delete. Draait als low-priority thread.
- Dynamic linking. JVM linked de benodigde classes tijdens executie.
- **char** type is 16 bits Unicode. Internationaal.
- **Geen** pointer arithmetic en conversies.
- **Geen** address-of operator &.
- **Geen** compiler afhankelijke taaldefinities:
  - Vaste grote voor elk type. Bijvoorbeeld int is **altijd** 32 bits.
  - Evaluatievolgorde is **altijd** van links naar rechts.
  - >> is **altijd** ASR. Nieuwe operator >>> voor LSR.

THRijswijk

Java en Structured Programming

Verschillen met C (C++).

- Java kent **geen** union en struct.
- Java heeft **geen** preprocessor.
  - **Geen** #include
  - **Geen** #define
  - **Geen** #ifdef ... #endif
  - **Geen** #pragma
- Java heeft **geen** default parameters.
- Java heeft **geen** typedef.
- Java heeft **geen** const. Een variabele kan wel final gedeclareerd worden. Assignment is dan niet mogelijk.

final double π=3.14159265359;

THRijswijk

Wat maakt Java ideaal voor het Net?

- Platform independent. Compile once, run anywhere.
- Veiligheid.
  - Browser kan bepaalde bewerkingen (b.v. file I/O) verbieden.
  - Geen pointer arithmetic en conversies.
- Multi threaded.
- Dynamic linkage. Geen grote exe. Je (down)load alleen wat je gebruikt.
- Kleine executables. Doordat veel functionaliteit (b.v. AWT, Swing) lokaal al in libraries aanwezig is.

THRijswijk

Java

Static memberfuncties en static datamembers.

- Java heeft net als C++ ook static methods (= static memberfuncties) en static fields (=static datamembers).
- In Java worden static methods meer gebruikt dan in C++ omdat in Java geen "losse" functies gedefinieerd kunnen worden.
- In plaats van een "losse" functie wordt dan een static method gebruikt, zo is bijvoorbeeld de main() functie uit C en C++ in Java een static method.

THRijswijk

Welkom1.java

```
public class Welkom1 {
  public static void main(String[] args) {
    System.out.println(
      "Welkom bij van C++ naar Java.");
  }
}
```

- Naam van de file met extensie ".java" **moet** hetzelfde zijn als de classnaam!
- Er is geen headerfile.
- javac Welkom1.java zet source om in Welkom1.class (byte codes).
- QNX: j9c == javac

```
java Welkom1
Welkom bij van C++ naar Java.
```

```
j9 Welkom1
Welkom bij van C++ naar Java.
```

THRijswijk

Honden

```
public abstract class Hond {
  public final void haalKrant() {
  //...
    blaf();
  }
  public abstract void blaf();
  //...
}
```

```
public class SintBernard extends
    Hond {
  public void blaf() {
    System.out.println("Woef woef");
  }
}
```

THRijswijk

Honden

```
public class THond {
  public void doeJeWerk(Hond h) {
    h.haalKrant();
  }
  public static void main(String[] s) {
    THond test=new THond();
    Tekkel harry=new Tekkel();
    SintBernard bor=
      new SintBernard();
    for (int i=0; i<10; ++i)
      if (i%3==0)
        test.doeJeWerk(harry);
      else
        test.doeJeWerk(bor);
  }
}
```

```
Kef kef
Woef woef
Woef woef
Kef kef
Woef woef
Woef woef
Kef kef
Woef woef
Woef woef
Kef kef
```

THRijswijk

Java interfaces

- In Java bestaat de mogelijkheid om alleen de interface van een type te specificeren zonder de implementatie te definiëren. (**interface**)
- Een class kan dan deze interface overerven, dit wordt met het keyword **implements** aangegeven.
- In C++ kan dit d.m.v. een pure ABC (Abstract Base Class).
- In C++ is hier geen speciale syntax voor.

THRijswijk

Hond interface

```
interface Hond {
   void blaf();
   //...
}
```

```
public class Tekkel implements Hond
{
   public void blaf() {
      System.out.println("Kef kef");
   }
}
```

THRijswijk

Multiple inheritance

! Java ondersteund **geen** multiple implementation inheritance. Je kunt maar van **één** class overerven (extends).

! Java ondersteund **wel** multiple interface inheritance. Je kunt van **meerdere** interfaces overerven (implements).

THRijswijk

Breuk.java

```
public class Breuk {
   public Breuk() { }
   public Breuk(int tt) { t=tt; }
   public Breuk(int tt, int nn) {
      t=tt; n=nn;
      normaliseer();
   }
   public Breuk(Breuk b) {
      t=b.t; n=b.n;
   }
   public void add(Breuk b) {
      t=t*b.n+b.t*n;
      n*=b.n;
      normaliseer();
   }
//...
   private normaliseer() {
// ...
   }
   private int t=0;
   private int n=1;
}
```

Geen automatische copy constructor.

Fields kunnen geïnitialiseerd worden.

THRijswijk

TBreuk.java

```
public class TBreuk {
   public static void main(String[] args) {
      Breuk b1=new Breuk(1, 2);
      Breuk b2=new Breuk(3);
      Breuk b3=new Breuk(b1);
      b2.add(b1);
      System.out.println("b2 = "+b2);
      if (b1!=b3)
         System.out.println("Vreemd!!");
   }
}
```

```
b2 = Breuk@871738a0
Vreemd!!
```

THRijswijk

Breuk toString method

```
public class Breuk {
// idem zoals hiervoor
   public String toString()  {
      String s=new String();
      s=t+"/"+n;
      return s;
   }
}
```

```
b2 = 7/2
Vreemd!!
```

THRijswijk

Packages

== C++ namespaces.

! Java heeft ook een manier om classes (.class files) te groeperen **packages** genaamd.
! In Java is een package altijd gekoppeld aan een **directory**. Packagenaam=directorynaam.
! Elk directory heeft een eigen package. Als de .class code van een bepaalde .java file in een andere package moet komen moet je dit opgeven.

```
package Bd;
public class Breuk { /*...*/ }
```

```
package Bd;
public class String { /*...*/ }
```

THRijswijk

Import packages

3 manieren van gebruik:

```
Bd.String s1=new Bd.String("Hallo");
```

```
import Bd.String;
//...
   String s1=new String("Hallo");
   String s2=new String("Dag");
```

```
import Bd.*;
//...
   String s1=new String("Hallo");
   Breuk b=new Breuk(3,2);
```

THRijswijk

Access specifiers

! In Java is een package ook een middel tot access control.
" Public overal toegankelijk.
" Protected toegankelijk in alle afgeleide classes **én** in classes in dezelfde package.
" (Package) toegankelijk in classes in dezelfde package.
" Private alleen in deze class zelf toegankelijk.

! Class kan **alleen** Public of Package zijn.
! Java kent **geen** friends.
! Bij overerving mag access alleen **toenemen**. LSP.

THRijswijk

Java GUI

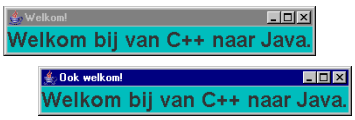
```
import javax.swing.*;
import java.awt.*;

class MyFrame extends JFrame {
   public MyFrame(String s) {
      super(s);
      JLabel label = new JLabel(
         "Welkom bij van C++ naar Java.");
      label.setFont(...);
      getContentPane().setBackground(...);
      add(label); setLocation(...);
      pack(); setVisible(true);
   }
}

public class Welkom2a {
   public static void main(String[] args) {
      JFrame frame1 = new MyFrame(
         "Welkom!");
      frame1.setDefaultCloseOperation(
         JFrame.EXIT_ON_CLOSE);
      JFrame frame2 = new MyFrame(
         "Ook welkom!");
   }
}
```

THRijswijk

Welkom2.java



Om deadlocks bij het opstarten te voorkomen is het beter om main als volgt te wijzigen:
http://java.sun.com/docs/books/tutorial/uiswing/misc/threads.html

```
public class Welkom2a {
   public static void main(String[] args) {
      javax.swing.SwingUtilities.invokeLater(
      new Runnable() {
         public void run() {
            JFrame frame1 = new MyFrame("...");
            frame1.setDefaultCloseOperation(...);
            JFrame frame2 = new MyFrame("...");
         }
      });
   }
}
```

THRijswijk

62

Mouse listening

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

class MyFrame extends JFrame implements
        MouseListener {
   public MyFrame(String s) {
      super(s);
      label = new JLabel("...");
      add(label);
      // ...
      addMouseListener(this);
   }
   public void mouseClicked(MouseEvent e) {}
   public void mousePressed(MouseEvent e) {}
   public void mouseReleased(MouseEvent e) {}
   public void mouseEntered(MouseEvent e) {
      label.setForeground(Color.red);
   }
   public void mouseExited(MouseEvent e) {
      label.setForeground(Color.white);
   }
   private JLabel label;
}
```

THRijswijk

63

Listeners

- KeyListener
- MenuKeyListener
- MenuListener
- PopupMenuListener,
- MouseListener
- MouseMotionListener
- MouseWheelListener
- WindowListener
- ...

THRijswijk

64

Inner classes

- Nested
  - Defined **static** within another class. (Just for grouping.)
  - Can be used from outside enclossing class.
- Member
  - Defined within another class.
  - Helper class.
  - Can only be used from within enclosing class.
  - **Has access to members of enclosing class.**
- Local
  - Idem as Member class +
  - Defined in a method.
- Anonymous
  - Idem as Local class +
  - Defined in an expression
  - No name (no constructor)
  - Only one (unnamed) object.

THRijswijk

65

Mouse listening with a member class

```
class MyFrame extends JFrame {
   class MyAdapter extends MouseAdapter {
      public void mouseEntered(MouseEvent e) {
         label.setForeground(Color.red);
      }
      public void mouseExited(MouseEvent e) {
         label.setForeground(Color.white);
      }
   }
   public MyFrame(String s) {
      super(s);
      // ...
      addMouseListener(new MyAdapter());
   }
   private JLabel label;
}


//  rest idem
```

THRijswijk

66

Mouse listening with an anonymous class

```
class MyFrame extends JFrame {
   public MyFrame(String s) {
      super(s);
      // ...
      addMouseListener(new MouseAdapter() {
         public void mouseEntered(MouseEvent e) {
            label.setForeground(Color.red);
         }
         public void mouseExited(MouseEvent e) {
            label.setForeground(Color.white);
         }
      });
   }
   private JLabel label;
}


//  rest idem
```

THRijswijk

67

Java via WWW

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

public class Welkom3 extends JApplet {
   public Welkom3() {
      label = new JLabel(
         "Welkom bij van C++ naar Java.",
         JLabel.CENTER);
      label.setFont(new Font(
         "Arial", Font.BOLD, 24));
      getContentPane().setBackground(
         Color.yellow);
      add(label);
      addMouseListener(new MouseAdapter() {
         public void mouseEntered(MouseEvent e) {
            play(getCodeBase(),"hi.au");
            label.setForeground(Color.red);
         }
         public void mouseExited(MouseEvent e) {
            label.setForeground(Color.black);
         }
      });
   }
   private JLabel label;
}
```

THRijswijk

68

Java via WWW

```
<HTML>
<HEAD>
  <TITLE>Welkom!</TITLE>
</HEAD>
<BODY>
<H1>
  Welkom op de "van C++ naar Java" page.
</H1>
<P>
Er wordt nu ook applet gestart om u welkom te
heten ...
<P>
<APPLET
   CODE="Welkom3.class"
   WIDTH="400"
   HEIGHT="150">
</APPLET>
<P>
En hier is de
<A HREF="welkom3.java">source</A>.
</BODY>
</HTML>
```

THRijswijk

69

Exceptions in Java

- Java ondersteund exceptions met vrijwel dezelfde syntax en semantiek als in C++.

- Verschillen met C++:
  - Na de catch blokken kan nog een **finally** blok worden toegevoerd. De code uit dit blok wordt **altijd** uitgevoerd onafhankelijk van feit hoe het try block wordt verlaten.
  - Onderscheid tussen ingebouwde (unchecked) exceptions (afgeleid van RuntimeException of Error) en toegevoegde (checked) exceptions (afgeleid van Exception).
  - Checked exceptions **moeten** in de throws clause van de functie die ze gooit opgegeven worden.

THRijswijk

70

Casting en RTTI in Java

- Java gebruikt de C syntax voor casting met de semantiek van C++ dynamic casting.
- Java heeft uitgebreidere RTTI dan C++.
  - Elke class in Java is direct of indirect afgeleid van de class Object. Deze class heeft een memberfunctie getClass() die een object van de class Class retourneerd.
  - De class Class bevat RTTI info. Methods:
    - boolean isInterface()
    - String getName()
    - Class getSuperClass()
    - Class[] getInterfaces()
    - Object newInstance()
    - static Class forName(String s) Maakt het dynamisch opzoeken en creëren van objecten mogelijk!

THRijswijk

71

Maak een object als je de naam van de class kent!

```
class Name2Object {
  public static void main(String[] args) {
    try {
      Class c=
        Class.forName("SintBernard");
      Hond h=(Hond)c.newInstance();
      h.blaf();
    }
  } catch (Exception e) {
     /* quick and dirty */
  }
}
```

THRijswijk

72

De class Object

Methods equals, hashCode en clone.

- In Java erven alle classes direct of indirect over van de class Object. Methods:
  - public String **toString**() hebben we gezien.
  - public boolean **equals**(Object obj) /*en*/ public int **hashCode**()
  - protected Object **clone**() throws CloneNotSupportedException
  - ...
  - ... Thread support.
  - ...
  - protected void **finalize**() throws Throwable;
  - public final getClass() RTTI hebben we gezien.

THRijswijk

73

De class Object

equals

- De default implementatie vergelijkt de adressen.

```
public class Breuk {
//...
  public boolean equals(Object obj) {
    if (obj instanceof Breuk) {
      Breuk c=(Breuk)obj;
      return t==b.t&&n==b.n;
    }
    return false;
  }
}
```

THRijswijk

74

TBreuk.java

```
public class TBreuk {
  public static void main(String[] args) {
    Breuk b1=new Breuk(1, 2);
    Breuk b2=new Breuk(3);
    Breuk b3=new Breuk(b1);
    b2.add(b1);
    System.out.println("b2 = "+b2);
//  if (b1!=b3)
//    System.out.println("Vreemd!!");
    if (b1.equals(b3))
      System.out.println("Ok!!");
  }
}
```

```
b2 = 7/2
Ok!!
```

THRijswijk

75

De class Object

hashCode

- De default implementatie bepaalt een hashCode aan de hand van het adres.
- Als we objecten in een HashTable willen gebruiken moeten we zorgen voor een “goede” hashCode.

```
public class Breuk {
//...
  public int hashCode() {
    return new Integer(t).hashCode()
         + new Integer(n).hashCode();
  }
}
```

THRijswijk

76

De class Object

clone

- De default implementatie (die protected is) “kijkt” eerst of de class de interface Cloneable implementeerd.
  - Als dit niet zo is: throw CloneNotSupportedException.
  - Als dit wel zo is: maak een bitwise (shallow) copy.
- Classes voor algemeen hergebruik moeten een “goede” clone definiëren.

THRijswijk

77

Cloneable Breuk

```
public class Breuk implements Cloneable {
//...
  // Implementatie van Object.clone werkt goed
  // maar is protected...
  public Object clone() {
    try {
      return super.clone();
    } catch (CloneNotSupportedException e) {
      // can't happen really!
      throw new InternalError();
    }
  }
}

//...

  Breuk b1(1, 2);
  Breuk b2=(Breuk)b1.clone();
```

THRijswijk

78

Cloneable Stack

```
public class Stack implements Cloneable {
  private Vector items;
//...
  public Object clone() {
    try {
      Stack s = (Stack)super.clone();
      s.items = (Vector)items.clone();
      return s;
    } catch (CloneNotSupportedException e) {
      // can't happen really!
      throw new InternalError();
    }
  }
}
```

THRijswijk

79

De class Object

finalize

! De finalize method wordt door de garbage collector aangeroepen als het object opgeruimd wordt of als de JVM afsluit (zonder calamiteit).

```
public class Breuk {
//...
   protected void finalize() throws
         Throwable {
      super.finalize();
      System.out.println(
         "Breuk: "+this+" destroyed.");
   }
}
```