Überblick

- ❏ Java-Sprache
- ❏ AWT-Klassenbibliothek (Teil 1)
- ❏ Java-Einbindung ins World Wide Web
- ❏ AWT-Klassenbibliothek (Teil 2)
- ❏ Remote Objekte (RMI)
- ❏ Zusammenfassung

Java-Sprache

Eigenschaften von Java:

- ❏ objektorientiert
- ❏ Modulkonzept
- ❏ Sicherheitskonzept
- ❏ Multithreading, Koordinierung, Ausnahmebehandlung
- ❏ Syntax ähnlich wie C++
- ❏ interpretierter Bytecode
- ❏ Einbinden von C und C++ Code möglich

Java-Sprache

❏ Primitive Datentypen in Java:

- byte, short, int, long (8/16/32/64 bit)
- float, double (32/64 bit)
- char (Unicode, 16 bit)
- boolean (true, false)
- "Normale" Operatoren: +,-,+=,<<,...

❏ Unterschiede zu C:

- kein "unsigned"
- spezieller "boolean" Datentyp
- kein "while (x--) ..." möglich
  also: "while (x-- != 0) ..."
- kein "if (ptr)" möglich
  also: "if (ptr != null) ..."

Java-Sprache

❏ Kontrollstrukturen wie bei C:

- for (... ; ... ; ...) { ... }
- while (...) { ... } oder do { ... } while (...);
- switch (...) { case .... }
- if ....
- return, break, continue wie bei C

❏ Unterschiede zu C:

- kein goto
- Statt dessen labelled break und continue:

```
outerloop: for ( .... ) {     // dies ist die Schleife "outerloop"
      while ( .... ) {
            if ( ... ) break outerloop;   // aus outerloop raus
      }
}
```

Java-Sprache

- Unterschiede zu C:
  - kein Präprozessor (#define)
  - kein typedef, union, enum, struct
  - keine Zeiger auf Variablen, etc.
  - keine Funktionen oder Prozeduren
  - keine globalen Variablen

Java-Sprache

- Array-Allokation ähnlich wie bei C++
- Prüfung von Feld-Grenzen
- Garbage-Collector (keine Freigabe nötig)

```
int iArray[] = new int[100];
for (int n=0; n<100; n++) iArray[n]=n;
int[] iArray2 = new int[100];       // gleichbedeutend wie oben


int[][] iMatrix = new int[100][100];


int[][] iMatrix2 = new int[100][];
for (int n=0; n<100; n++) iMatrix2[n]=new int[100];

// jetzt sehen iMatrix2 und iMatrix gleich aus!
```


Java-Sprache

- Klassen mit Einfachvererbung
- Baumhierarchie (kein Wald) mit der Klasse "Object" als oberste Klasse
- Davon getrennte Typ-Hierarchie mit Mehrfachvererbung und Waldhierarchie
- Basistypen (int,float,...) sind außerhalb der Klassenhierarchie
- Es gibt nur Referenzen auf Objekte (Zeiger)

Java-Sprache

- Klassendeklaration und Implementation in einem Stück
- Syntax ähnlich wie C++
- Expliziter Zugriff auf Oberklasse mit "super" (statt wie C++ mit OberKlasse::)

```
class MyPoint extends Object {
      private int x;
      private int y;
      public MyPoint(int x, int y) {
            super();          // Aufruf des Oberklassenkonstr.
            this.x = x;
            this.y = y;
      }
      public int GetX() { return x; }
      public void SetX(int px) { x = px; }
...
}
```

Java-Sprache

❑ Objekterzeugung **nur** mit new

❑ Zugriff nur mit "."

❑ Löschen automatisch mit Garbage Collection

```
MyPoint p;              // Deklaration einer Referenz auf MyPoint
p.SetX(2);              // Fehler: Null-Pointer Zugriff

p = new MyPoint(3,4);   // Neue Instanz erzeugen und zuweisen
MyPoint p2 = p;         // Zeiger auf die selbe Instanz
p.SetX(9);              // jetzt ist p == p2 == (9,4)

System.out.println(p.GetX()); // x ausdrucken

p = new MyPoint();      // Fehler: kein leerer Konstruktor da
```


Java-Sprache

❑ Zugriffsrechte: private, protected, public, "friendly"

❑ Applizierbar auf Methoden und Instanzvariablen

- private: nur innerhalb der Klasse selbst zugreifbar
- protected: in der Klasse, in Unterklassen **und innerhalb des Moduls** zugreifbar
- public: überall zugreifbar
- "friendly": default. In der Klasse und innerhalb des Moduls zugreifbar

❑ Zugriffsmodus muß vor jeder Methode / Variablen stehen

Java-Sprache

❑ Instanzvariablen können konstant sein:

- Dann muß sofort ein Wert zugewiesen werden
- Dieser kann nie wieder geändert werden
- Direkte Wertzuweisung funktioniert auch mit nicht-Konstanten

```
class ... {
      final private int t=10; // Konstante
      private int y=15;       // Default-Wert, falls im Konstruktor
                              // nichts angegeben wurde
}
```


Java-Sprache

❑ Statische Variablen / Methoden sind ähnlich wie globale Variablen / Funktionen:

❑ statische Variablen: Es gibt nur eine einzige Instanz

❑ statische Methoden: Sie können ohne Objekt aufgerufen werden

❑ Beispiel:

```
class Array {
      public static int CalculateIndex(int i) {
            return i-1;
      }
      public static String logfile="/tmp/array.log";
}


... = Array.CalculateIndex(4711);
Array.logfile="/tmp/hallo.log";
```

Java-Sprache

Pakete

- ❏ Paket: Programmeinheit mit
  - definiertem Namen (z.B. java.lang oder java.awt.image)
  - besteht aus einer oder mehreren Klassen
  - public-Klassen: sind außerhalb des Paketes verfügbar
  - private Klassen: sind nur innerhalb des Paketes verfügbar
- ❏ Quell-Dateien haben Endung ".java"
- ❏ Bytecode-Dateien haben Endung ".class"
- ❏ Dateiname der .class Datei wird vom Klassennamen bestimmt

Java-Sprache

Pakete

- ❏ Pakete werden definiert mit "package"
- ❏ Pakete können mit "import" eingebunden werden:

```
import oobp.test.*;     // alle Klassen v. oobp.test einbinden
import java.io.File;    // Klasse File aus Paket java.io einb.
```

- ❏ Klassen werden gesucht über Dateipfade
- ❏ Beispiel: Paket oobp.test, darin definierte Klasse MyClass
- ❏ Einbinden mit "import oobp.test.MyClass"
- ❏ Suche von Bytecode-Datei oobp/test/MyClass.class

Java-Sprache

Pakete

- ❏ Beispiel:

```
// Datei MyClass.java
package oobp.test;       // dies ist paket oobp.test
import java.awt.*;       // Alle Klassen aus Paket
                         // java.awt einblenden
public class MyClass {
      ....
}
class Test2 {
      .... {
            Button b=new Button();        // stammt v. java.awt
            java.awt.Button b2=...;       // funktioniert auch
      }
}
```


Java-Sprache

Pakete

- ❏ Eine public-Klasse "x" **muß** in der Datei "x.java" implementiert werden
- ❏ Am besten: für jede Klasse eigene Datei
- ❏ Oft sinnvoll, z.B. für das "Hauptprogramm": default Paket
- ❏ Wenn man kein Paket definiert, wird ohne zus. Pfad die .class-Datei gesucht

Java-Sprache

- Ein Java-Programm besteht aus einer Klasse mit statischer main-Methode
- Dokumentation zu Java und Klassenbibliotheken:
  http://www4.informatik.uni-erlangen.de/~riechman/java.html
- Beispiel: HelloWorld.java
- Wird übersetzt mit "javac", gestartet mit Java-Interpreter "java"

```
import java.io.*;
public class HelloWorld {
  public static void main(String argv[]) {
    System.out.println("Hello World");
  }
}

> set path=(/local/java/bin $path)
> javac HelloWorld.java
> java HelloWorld
```


Java-Sprache

- getrenntes Typ-Konzept
- Typen enthalten nur Schnittstellendeklaration
- Klassen können zu Typen kompatibel sein
- Diese Klassen müssen dann die komplette Schnittstelle implementieren

Java-Sprache

- Beispiel

```
public interface Printable {
      public void Print();
}

public class MyPoint extends Object implements Printable {
      ....
      public void Print() {
            System.out.println("x="+x+" y="+y);
      }
}


....
Printable p=new MyPoint(10,20);
p.Print();
```


Java-Sprache

- Typen können mehrfach vererbt werden und wirken transitiv
- Beispiel:

```
interface Streamable extends FileIO, Printable {
      // additional Methods
}

class Test implements Streamable, Testinterface {
      ...
}
```

Java-Sprache

- Exceptions sind ähnlich wie in C++: try, throw, catch
- Zusätzlich: "finally" block, der auf jeden Fall ausgeführt wird.
- Es können nur Objekte als Exception-Parameter verwendet werden, die aus Subklassen von Throwable stammen
- Exceptions, die eine Methode wirft, müssen deklariert werden

Java-Sprache


```
class TestException extends Throwable {
  public TestException(String s) {super(s);}
}

public class Test {
      public void Hello() throws TestException {
            if (...) throw (new TestException("das war nix"));
      }
      public void Testit() {
            try {
                  Hello();
                  ...
            }
            catch (TestException t) {
                  System.out.println("Exception raised...");
                  if (...) throw(....);
            }
            finally {
                  // irgendwelche Aufräumarbeiten
            }
      }
}
```