# Datentypen

|  |  |  |
| --- | --- | --- |
| **Datentyp** | **Grösse** | **Beschreibung** |
| boolean | - | Boolescher Wahrheitswert |
| char | 16bit | Unicode Zeichen |
| byte | 8bit | Zweierkomplement-Wert |
| short | 16bit | Zweierkomplement-Wert |
| int | 32bit | Zweierkomplement-Wert |
| long | 64bit | Zweierkomplement-Wert |
| float | 32bit | Gleitkommazahl |
| double | 64bit | Gleitkommazahl |

# Zugriffsmodifikatoren

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Modifikator** | **Klasse** | **Package** | **Sub-Klasse** | **Other** | **UML** |
| public | Ja | Ja | Ja | Ja | + |
| protected | Ja | Ja | Ja | Nein | # |
| Keiner | Ja | Ja | Nein | Nein | ~ |
| private | Ja | Nein | Nein | Nein | - |

# KeyListener

public class KeyAdapterTest extends JFrame {  
  
 public static void main(String[] args) {  
 KeyAdapterTest program = new KeyAdapterTest();  
 }  
  
 private KeyAdapterTest() {  
 super("Key Test");  
 addKeyListener(new MyKeyListener()); // Key Listener hinzufügen  
 setSize(200, 200);  
 setVisible(true);  
 }  
  
 // Private Klasse als KeyListener  
 private class MyKeyListener extends KeyAdapter {  
 public void keyPressed(KeyEvent e) {  
 System.out.println("Key: " + e.getKeyChar());  
 }  
 }  
}

# Strings vergleichen

String example = "Hello";

String test = "Hello";

example == test; // false;

example.equals(test) ; // true ;

# Klassen

public class Klasse {

public static final double PI = 3.141592; // Konstante, (Klassenvariable)

public static int ID = 42; // Klassenvariable

private int age; // Instanzvariable

private String name = "Klasse"; // Instanzvariable

public Klasse(int alter) { // Konstruktor

this.name = name ; // Assignment

}

public int getDoubleAge() { // Methode

int factor = 2 ; // Lokale variable

return factor \* age ;

}

}

# Listen (Alternative zu Vector())

List<String> data = new ArrayList<>(); // Create list

data.add("String 1");  
data.add("String 2");

data.remove(1); // Removes "String 2"

data.get(0); // Returns "String 1"

for (String s : data) {  
 System.out.println(s);  
}

# Testing

* Unit-Tests (Den Code prüfen)
  + Mit JUnit
  + Grenwerte, -fälle testen (Edge-cases)
* TDD: Test driven developement
  + Erst die Unit-Tests schreiben, danach den Code dazu

# Projektmethoden

* Spiralmodell
  + Mehrere Iterationen
  + Jeweils Analyse, Design, Implementation und Test-Phasen
  + Eine Iteration ist also ein kleiner Wasserfall
  + Anforderungen können in jeder Iteration berücksichtigt werden
* Wasserfall-Modell
  + Anforderungen können später nur schwer berücksichtigt werden
  + Jede Phase einmal, Linear

# UML Klassendiagramm

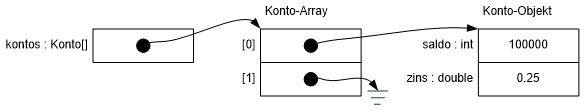
* Können helfen, während der Analyse die Anforderungen zu verstehen
* Zeigen den statischen Aufbau eines Systems
* Beziehungen zwischen Klassen (Assoziationen und Vererbung)
* Abstrakte Klassen und Methoden sind Kursiv

# UML Sequenzdiagramm

* Zeigen einen dynamischen Teil des Systems
* Zeigen nur einen Bruchteil des grossen Systems
* Horizontal werden die Objekte dargestellt, dazwischen die Meldungen
* Die vertikale Achse zeigt die Lebenslinien (Life lines) der Objektee

# Speicherdiagramm

* Zeigen den aktuellen Zustand im Speicher
* In Java kann man nur per Referenz auf ein Objekt zeigen



Dazugehöriger Codeausschnitt:

class Konto {

int saldo = 100000;

double zins = 0.25;

}

...

Konto[] kontos = new Konto[2];

kontos[0] = new Konto();