C++

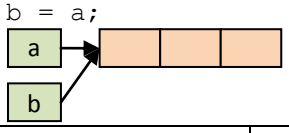
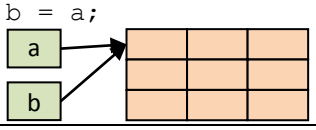
Dateihandhabung	.cpp	Sourcefile (auscodiert)
	.h	Headerfile (Deklarationen und Funktionsprototypen)
Kommentar	`// Kommentar` `/* Kommentar */`	einzeiliger Kommentar mehrzeiliger Kommentar
Garbage Collection	nein	Automatische Speicherbereinigung
Case-Sensitiv	ja	Achtung auf Gross-/Kleinschreibung
Arithmetische Operatoren	`a = b` `a + b` `a - b` `a * b`	Zuweisung; Grundoperationen (Punkt vor Strich)
	`a%b` `a/b`	Modulo (Rest einer Division); Ganzzahldivision bei Ganzzahl, normale Division bei Fliesszahl
	`++a` `a++` `--a` `a--`	Pre-/Postinkrement; Pre-/Postdekrement
Vergleichende Operatoren	`a == b` `a != b`	Gleich; Ungleich
	`a < b` `a > b` `a <= b` `a >= b`	kleiner; grösser; kleiner gleich; grösser gleich
Logische Operatoren (! vor && vor ||)	`!a` / `not a` `a && b` / `a and b` `a || b` / `a or b`	NOT; AND; OR
Bitweise Operatoren	`~a` `a & b` `a | b`	NOT; AND; OR
	`a ^ b` `a << b` `a >> b`	XOR; left shift; right shift
Kombinierte Operatoren	`+=` `-=` `*=` `/=` `%=`	Operation mit direkter Zuweisung
Eingabe	`#include <iostream>` `std::cin << "Hello\n" << varA;`	über Tastatur in Console
Ausgabe	`#include <iostream>` `std::cout << "Hello World!";`	Auf Console

Datentypen				
	Ganzzahl	`(signed) int`	16 / 32 Bit	$-2Mio\ bis\ 2Mio\ /\ -32.768\ bis\ 32.767$
		`unsigned int`	16 / 32 Bit	$-2Mio\ bis\ 2Mio\ /\ 0\ bis\ 65.535$
		`short int`	16 Bit	$-32.768\ bis\ 32.767$
		`(signed) long`	32 Bit	$-2Mio\ bis\ 2Mio$
		`unsigned long`	32 Bit	$0\ bis\ 4Mio$
	Zeichen	`(signed) char`	8 Bit	$-128\ bis\ 127$
		`unsigned char`	8 Bit	$0\ bis\ 255$
	Wahrheitswert	`bool`	8 Bit	$true, false$
	Gleitkomma	`float`	32 Bit	$3.4e^{-38}\ bis\ 3.4e^{38}$
		`double`	64 Bit	$1.7e^{-308}\ bis\ 1.7e^{308}$
		`long double`	80 Bit	$3.4e^{-4932}\ bis\ 1.1e^{4932}$
	Unverwendet	`null`	-	-

Umwandlungen / Casts	`l = (long) i;` `l = long(i);`	int to long

Variablen / Arrays	**Variablen**	**1D Array**	**2D Array**
Überschreitungen sind möglich	a 1 b 2	a[0] 2 a a[1] 3 a[2] 5	a[][0] a[][1] a[][2] a a[0] 2 3 5 a[1] 4 8 1
Grösse	-	`sizeof(d);`	-

Deklaration	`int a, b;`
	`int a = 1;`
	`int b = 2;`
		`int *a;`
		`a = new int[3];`
		`int a[4];`
			`int b [2][3];`

Zuweisung	`b = 2;`	`a[1] = 3;`	`b[1][2] = 1;`
Abfrage	`b;  //2`	`a[2];  //5`	`a[0][2];  //5`
Zuweisung	`a = b;`	`b = a;`	`b = a;`
bedeutet	a 2		

char	`char c1 = 'a';`	Deklarieren

	`char c2 = c1 + 1;  //b`
	`(int) 'A';  //65`
	`(char) 65;  //a`
		Konvertieren

	`char a[] = {'B','3','ü','\0'};`	Array (\0 als Abschluss)
String ➔ char Array	`char a[] = "Hallo";`	Deklaration direkt
	`char a[] = {'H', 'a', 'l', 'l', 'o', '\0'};`	Deklaration über char-Array
	`std:string s1 = „Hallo";`	Deklaration über std-Klasse
Stringbuffer	-	

if-then-else	`if (a == 2)`
	`{`
	`  //Anweisung wenn a=2`
	`}else if(a == 3){ //optional`
	`  //Anweisung wenn a=3`
	`}else{ //optional`
	`  //Anweisung sonst`
	`}`
			If Anweisung, muss vom Typ bool sein
			
			Durchläuft eine der 3 Anweisungen

while	`while (i<100)`
	`{`
	`  //Anweisung solange i<100`
	`{`
			Durchlauf solange die Bedingung wahr ist

do-while	`do {`
	`  //Anweisung min. 1 mal`
	`} while (i<100);`
			Anweisung wird min. einmal durchgeführt
			Wiederholung solange wahr

for	`for (int i=1;i<=10;i++){`
	`  //Anweisung solange i<=10`
	`}`
			Zählschleife, Deklariert die lokale Variable i
			Durchläuft solange die Bedingung wahr ist
			und erhöht dann 'i' um 1

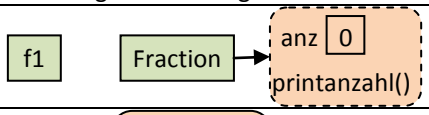
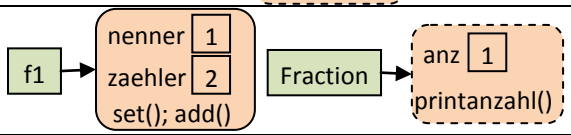
switch	`switch (m) {`
	`case 1: case 2:`
	`  //Anweisung`
	`  break;`
	`case 3:`
	`  //Anweisung`
	`  break;`
	`default:`
	`  Out.print("error");`
	`}`
			Unterscheidung eines Wertes, String nicht erlaubt
			Fall wenn m=1 oder m=2
			break beendet case
			
			Fall wenn m=3
			
			default wird genommen, falls nichts anderes zutrifft

Jump statements	`break;`		Aus Schleife herauskommen
	`continue;`		überspringe den Rest der Schleife
Klassen	`class Rectangle{` `}`		Für Objekte Vererbung

Methode ohne Parameter	`void printHeader(){`
	`  //Anweisung`
	`}`
			kleingeschrieben, void = kein Rückgabewert
			Aufruf: `printHeader();`

Methode mit Parameter	`void sub(int x, int y) {`
	`  //Anweisung`
	`}`
			x,y = Übergabewerte
			Aufruf: `sub(100,200);`

Funktionen mit Rückgabewerten	`int max(int x, int y) {`
	`  return x+y;`
	`}`
			int = Rückgabetyp
			return gibt Wert zurück und bricht Funktion ab
			Anwendung: `z = max(a,b);`

Schlüsselwörter	this	`this->code = code;`	Referenz auf eigene Objekt, bei gleichen Variablennamen
		~~`this(code, ID);`~~	~~Aufruf eines anderen Konstruktors der gleichen Klasse~~
	static	`static int count;`	Klassenvariablen, unabhängig von Objekten (z.B. zählen)
		`static e(int a, int b)`	Funktionssammmlung (zur Speichererspanis)
		`static int PI = 3;`	Konstanten (zur Speichererspanis)
Sichtbarkeit für Methoden und Variablen	`private`		nur in der eigenen Klasse sichtbar (lokal)
	`public`		überall sichtbar (global)
	`protected`		nur in eigener und abgeleiteten Klassen sichtbar
Klassenaufruf mit new	`Fraction f1;`		
	`f1 = new Fraction (1,2);`		

Konstruktor	`Fraction(int n, int z){`
	`  //Anweisung`
	`}`
			werden beim Anleigen mit new … durchlaufen
			Zweck: Anfangswerte setzen
			Bedingung: Name gleich wie Klassenname

Destruktor	`f1 = null;`		Speicher wieder frei geben
Definitionen	`#define xy 5;`		Präprozessor ersetzt x durch 5

Klassifikation (Vererbung)

```
-------Article.h----------------------
class Article
{
   public:
      Article(int code, int price);
      virtual int showInfo();
   protected:
      int code, price;
      void getArticle(){...}
};
-------Book.h-------------------------
class Book : public Article
{
   public:
      Book(int price);
      int showInfo();
   private:
      char autor[];
}
-------Book.c-------------------------
Book::Book(int code, int price):
Article(code, price)
{
}
int Book::showInfo(){...}
```

Idee

Gemeinsamkeiten zu teilen
Alle Felder und Methoden werden geerbt
Erweiterbarkeit mit neuen Unterklassen

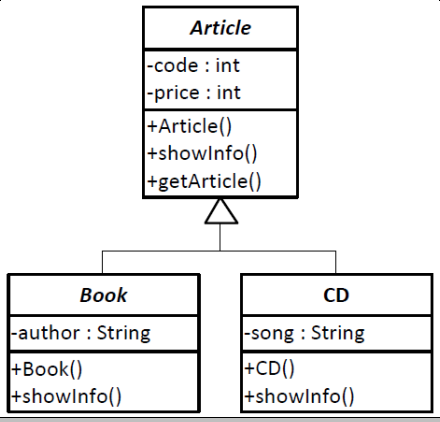
Umsetzung

Unterklasse : public Oberklasse

Methoden überschreiben

showInfo wird überschrieben
getArticle wird unverändert übernommen

Aufruf von Methoden/Konstruktor der Oberklasse



Kompatibilität-zwischen Ober- und Unterklassen

Zuweisungen

- statischer Typ -> Article
- dynamischer Typ -> Book, CD

```
Article *a = new Book();
Article *b = new CD();
a->price = "..";      //Fehler, weil a: Article
a->author="x";        //kein Fehler, a: Buch
```

Prüfung auf Referenz (Ist-Beziehung)

```
if (a instanceof Book){...}
if(dynamic_cast<Book *>(a) != null)
```

Typenumwandlung

```
Book b = (Book) a;    //sofern instanceof gemacht
```

Dynamische Bindung

```
a->showInfo();        // a von Book
```

Abstrakte Klassen (vorgegebene Methoden)

- Unimplementiert in der Oberklasse
- Methode müssen in den Unterklassen geschrieben werden
- Abstrakte Klassen können nicht instanziert werden
- Abstakte Methode -> Abstrakte Klasse

```
class Animal {
   virtual void speak() = 0;      Abstrakte Klasse
}
class Bird : public Animal
{                                 Konkrete Klasse
   void speak(){...};
}
```

~~Interfaces (Schnittstellen)~~

~~Klassen aus Methodenköpfen~~
~~spezielle abstrakte Klassen~~
~~Mehrfachverwendung möglich~~
~~Gleichbehandlung von Klassen, die nicht in Beziehung zueinander stehen~~

```
interface Writer {
   void open();
   void write(char c) }
class Textbox implements Writer, Reader{...}
```

Wrapper-Klassen und Boxing

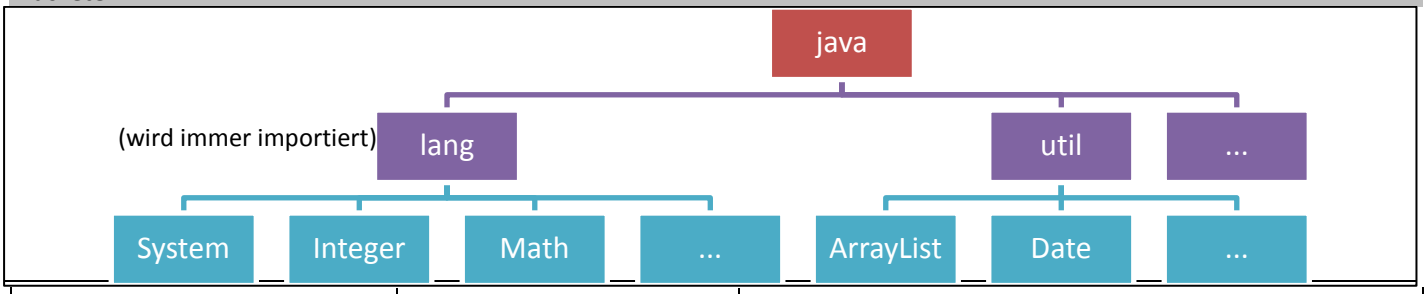
Grund	Basistypen sind keine Klassen								
Lösung	Wrapper-Typen (Basistypen kompatibel zu Object machen)								
Verwendung	`List.add(new Integer(5));`					**Autoboxing (seit Java 5)** `List.add(5);`			
	`int value = ((Integer)list.get(0)).intValue();`					**Autounboxing** `int value = list.get(0);`			
Typen	**Datentyp**	boolean	char	byte	short	int	long	float	double
	Klasse	Boolean	Character	Byte	Short	Integer	Long	Float	Double

Generizität

Prinzip: Typenplatzhalter T
 	
T wird durch noch unbekannten Datentyp ersetzt
	`class List<T> {`
	`  T[] data;`
	`  void add(T x){…}`
	`  T remove();`
	`}`
		`class SortedListe<T extends Comparable<T>>{`
		`  T[] data = (T[]) new Comparable[100];`
		`  void add(T elem) {… elem.compareTo() …}`
		`  T remove();`
		`}`

Generische Methode	`public static <T extends Comparable<T>> T max(T[] a){ ... }`	
Mehrerer Parameter	`class List<T, V>{…}`	
Arraydeklaration	`T[] data = new T[100];` //Fehler `T[] data = (T[]) new Object[100];` //kein Fehler	
Verwendung Typsicherheit Vermeiden von Laufzeitfehler Weniger Typenumwandlungen	`List<String> list = new List<String>();` `list.add("abc");`	

Packete



Packete	Import	Dokumentation
package ch.ntb.* Ziel: Ordnung schaffen	**import** ch.ntb.*	• hinkt immer etwas hinterher • /** … **/ • Doku automatisch erstellbar

Exceptions (Ausnahmen)

Fehlercodes
Jede Funktion liefert einen Resultatwert
`f() {…}`

Ausnahmebehandlung
- Ausnahmebehandlung vom Normalablauf trennen
- Führt nicht zu Systemabbruch

Systemausnahmen (vordefiniert)
- von jvm: java virtual machine

Benutzerausnahmen (vordefiniert)
- vom Benutzer mit **throw** geworfen,

Eigene Exception

Exception Handler (fangen mit catch)
Reagieren auf Ausnahmen
Der erste passende Handler wird gewählt
> deshalb: Reihenfolge wichtig

finally (wird immer ausgeführt)
z.B. schliessen von Dateien und Verbindungen

Weiterwerfen
Ausnahmen behandeln oder mit **throws** weiterwerfen

```
try{
  f();
  if(i==n) throw new OverflowException(i);
  g() throws OverflowException{
  }
}catch(IndexOutofBoundsException ex){
  Out.println("Indexfehler");
  ex.printStackTrace();
}catch (NullPointerexception ex)
  Out.print("…");
}
finally{…}

class OverflowException extends Exception {
  int info;
  OverflowException(int info) {
    this.info = info;
  }
}
```

Runtime-Exceptions

ArithmeticException	//Division durch 0
NullPointerException	//Zeiger hat Wert null
ArrayIndexOutOfBoundsException	//Arraygrösse überschritten

Threads („Parallele Prozesse“)

- Threads teilen sich den Speicher
- Programme laufen quasi-parallel

Erzeugen von Threads
Von **Thread abgeleitet**
run implementieren
mit **start**() starten
mit **sleep**(int delay), Zeit abwarten [ms]

```
public class Prozess_1 extends Thread{
  public static void main(String[] args) {
    new Prozess_1().start(); new Prozess_2().start();
  }
  public void run(){
    int n = 0, delay = 500;
    while(n<10){
      Out.print(n+" ");
      try{ sleep(delay); n++;
      }catch(InterruptedException e){};
} } }
public class Prozess_2 extends Thread{
  public void run(){...}
}
```

Synchronisation
Threads können sich in die Quere kommen
Grund: Unterbruch im kritischen Bereich
Lösung: Sperrverfahren

```
a) Objekt als Schlüssel / Schloss (lock)
  int balance; Object lock = new Object();
  void deposit(int x){
    synchronized(lock){ balance=balance+x; }
  } //dasselbe beim withdraw
b) synchronized-Methode (wenn ganze Methode kritisch)
  int balance;
  synchronized void deposit(int x){ balance=balance+x;}
  //dasselbe beim withdraw
```

wait & notify
… lösen Blockaden auf

```
notify() //weckt irgendein Thread auf
wait()   //Legt Thread in Warteschlange, gibt lock frei
notifyAll() //weckt alle Threads auf
```

Softwaretechnik

Idee: gut lesbare, sinnvolle, verständliche Programme/Klassen

Vorgehen
Statik:

1. Kandidaten für Klassen suchen — Nach Substantiven durchsuchen
2. Klassen in UML Diagramm zeichnen — Nur Klassennamen schreiben
3. Einteilen in Klassen und Attribute — einfach=Attribut, komplex=Klasse
   wer kennt wen/wer besteht aus wem
4. Beziehungen zwischen Klassen — Beziehungen in UML aufzeichnen

Dynamik:

5. Objektdiagramm — Situation zu einem gewissen Zeitpunkt
6. Sequenzdiagramm — Zeitlicher Ablauf

Streams

Ein I/O Stream repräsentiert eine Quelle oder ein Ziel
Streams unterstützen viele Datentypen

Typen

	Bytes (8-bit)	Zeichen (characters)
Eingang	InputStream	Reader
Ausgang	OutputStream	Writer

Klasse System
3 Streams werden automatisch erzeugt:
System.in; System.out; Systen.err;

```
import java.io.*;
public class FileTest {
  public static void main() {
    try {
      FileWriter out = new FileWriter("log.txt");
      BufferedWriter b = new BufferedWriter(out);
      PrintWriter p = new PrintWriter(b);
      p.println("This is the first sentence");
      p.close();
    } catch (Exception e) { }
  }
}
```

Pipes
Kommunikation zwischen verschiedenen Programmen oder Threads

```
PipedOutputStream pos = new PipedOutputStream();
PipedInputStream pis = new PipedInputStream();
pos.connect( pis ); // oder pis.connect( pos );
```

Link – Spechen sie Java

C++ TYPISCHES

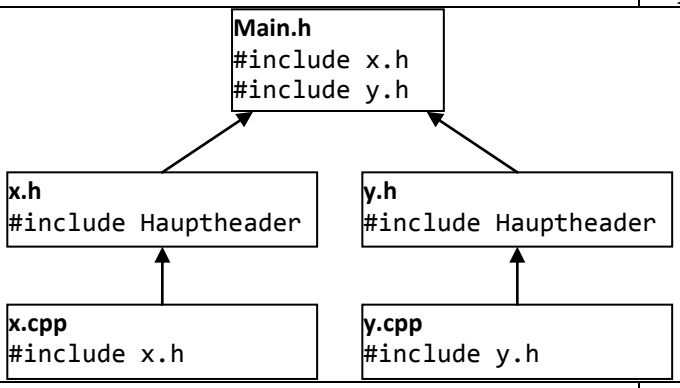
Pointer	`char arr [5];`
	`char *ptr;`
	`for (ptr = arr; ptr <= arr +4; ptr++)`
	`{`
	`  *ptr = 0;`
	`}`
		Array deklarieren
		Pointer deklarieren
		for-Schleife mit Pointer

	`&pointer`	Adresse des Pointers
Mehrfachvererbung		
Initialiserungsliste		

Defaultparameter	`Button::Button(bool b, int h, int w=200)`
	`new Button(true, 50);`
	`new Button(true, 50, 300);`
		Parameter bei der Definition bereits setzen
		letzter Parameter (beim Qt)

Polymorphismus	`Sohn1 abgeleitet von Vater`
	`Sohn2 abgeleitet von Vater`
	`Vater mit Methode meth1`
		Ausgangslage

	`Base *varA;`
	`var = new Sohn1();`
	`(static_cast<Sohn1 *>(varA))->meth1;`
		Ersatz für ‚super' / Pointer auf Pointer

super-Ersatz	`Vater::meth1();`	
const	`obj const *name;`	Pointer konstant
	`ob * const name;`	Objekt konstant
Mehrere Files		
Methodenaufruf eines Pointers	`pointer1->func();` `(*pointer1).func();`	beide Varianten gleich
inline (im headerfile)	`inline void summe(int a, int b){a+b;};`	Funktion direkt definieren

Call by	
Call by Value	**Call by Reference**

`bool ok;`
`ok = false;`
`function1(ok);`
`if(ok == false)`
`{`
`}`
	`bool *ok;`
	`*ok = false;`
	`function1(&ok);`
	`if(ok == false)`
	`{`
	`}`

`function1(bool ok)`
`{`
`  ok = true;`
`}`
	`function1(bool* ok)`
	`{`
	`  *ok = true;`
	`}`

Kompilierungs-vorgang unter Windows