Java – Programmiersprache und Java - Technologie

1990 - erste Arbeiten bei Sun Microsystems (jetzt Oracle) unter Codenamen „Oak“ (Eiche)
Zielstellung: eine Programmierplattform für Haushaltelektronik

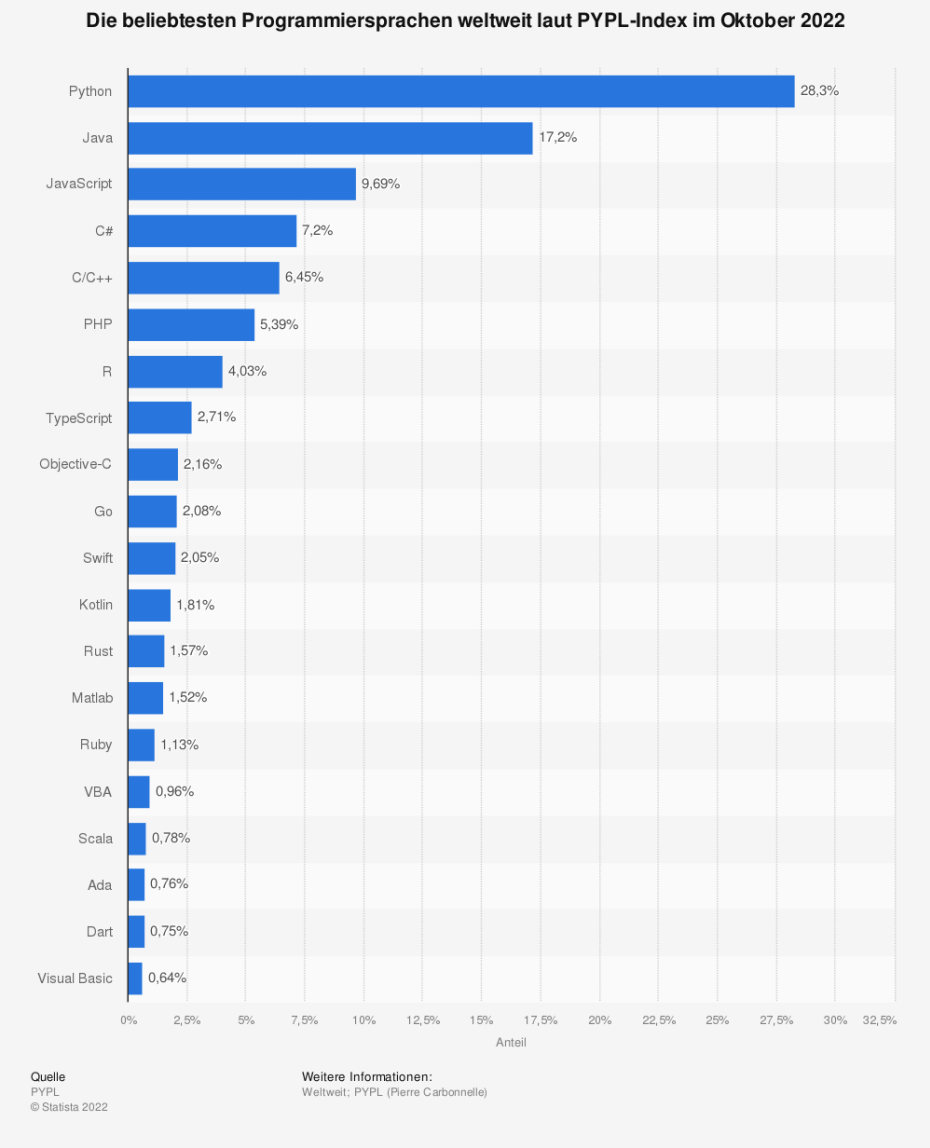
1993 – Abänderung der Zielstellung:
Entwicklung zur plattformunabhängigen, zur Programmierung von Internet-Applikationen geeigneten OOP

23. Mai 1995 – Veröffentlichung der Java-Technologie.

…

22. März 2022 – Veröffentlichung von Java 18

Verteilte Software - Java – Objektorientierte Programmierung 2



Ranking-listen, z.B.
PYPL (Popularity of Programming Language Index)

- maßgeblich ist die Häufigkeit der Google Suchanfragen nach Tutorial
- weicht ab vom tatsächlichen Nutzen der Sprache

Analyseanbieter SlashData: Umfragen von mehr als 20000 Entwickler:innen aus 166 Länder :
JavaScript, Python, Java, C/C++, C#, PHP,…

niederländische Software-Unternehmen TIOBE: berücksichtigt die Häufigkeit der Anfragen
Quelle:
https://entwickler.de/programmierung/top-10-programmiersprachen-tiobe-pypl-devoloper-nation

Java – eine Erfolgsgeschichte

aber warum?

Java - Eigenschaften

plattformenunabhängig (Java – Bytecode)

objektorientierte Sprache (Klassen, Referenzen, virtuelle Methoden)

Sicherheit (Garbage Collection, Typüberprüfung zur Laufzeit)

Threads (Nebenläufigkeit)

Netzwerkfähigkeit (Sockets, RMI, JMS, Servlets)

modularer Aufbau (mehrere Dateien)

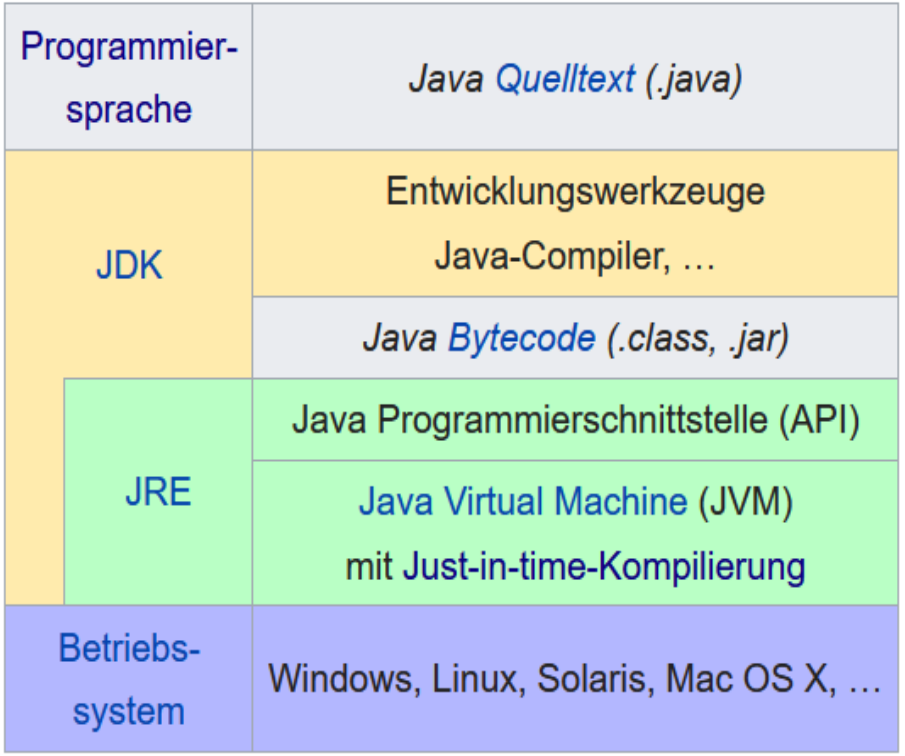
umfangreiche Klassenbibliotheken

Einbinden von Routinen anderer Programmiersprachen

Java-Technologie

- Programmiersprache
- Java Development Kit (JDK) – Entwicklungswerkzeuge …
  - Java-Compiler javac,
  - Java-Interpreter java,
  - JRE
  - Dokumentation javadoc
  - Debugger jdb
  - Bibliotheken
  - Archivierungstools
  - …
- Java-Laufzeitumgebung (JRE) – zum Ausführen der entwickelten Programme
  - Java Programmierschnittstelle – Bindung auf Quelltextebene

Aufbau der Java-Technologie



https://de.wikipedia.org/wiki/Java-Technologie

Erzeugen des Bytecodes: javac

javac name.java

Abarbeiten des Bytecodes: java, javaw (ohne Konsolenfenster)

java name

Entwicklungsumgebung - IDE (Integrated Development Environment)

- Eclipse
- NetBeans
- IntelliJ IDEA
- …

Java - Programm - Beispiel

```
package Examples;
import java.util.Scanner;
//import java.io.*

public class Echo {
	public static void main(String[] args){
		Scanner scanner=new Scanner(System.in);
		String s=scanner.next();
		System.out.println(s);
	}
}
```

//Standardstream Objekte:
//System.in – Standardeingabe
//System.out – Standardausgabe
//System.err - Standardfehlerausgabe

Syntax eines Java - Programms

```
javaprogramm     ::= [packagestatement]
                     {importstatement}
                     {typedeclaration}

packagestatement ::=   package packagename;

importstatement  ::=   import packagename.*; |
                       import classname; | import interfacename;
```

Paket: Sammlung semantisch zusammengehöriger Klassen in einem Verzeichnis

Quelle: Prof. B. Steinbach, Vorlesung „Verteilte Software“

Syntax eines Java - Programms

```
javaprogramm    ::= [packagestatement]
                        {importstatement}
                        {typedeclaration}

typedeclaration ::=   classdeclaration | interfacedeclaration | ;

classdeclaration::=   {modifier} class name [extends classname]
                        [implements interfacename {,interfacename}]
                        { ... }

interfacedeclaration ::= {modifier} interface name
                             [extends interfacename {,interfacename}]
                             { ... }
```

Klasse: Spezifikation von gleichartigen Objekten, gekennzeichnet durch gleiche Eigenschaften (Attribute) und gleiches Verhalten (Methoden)

Interface: Schnittstelle, festgelegt durch Methodendeklaration

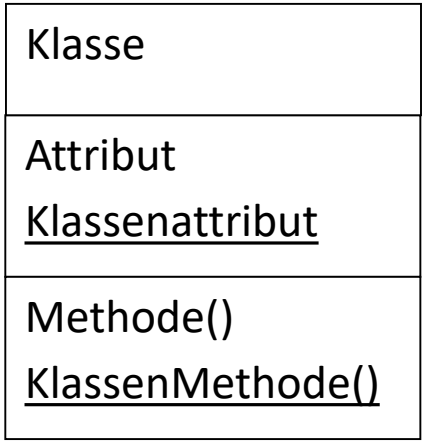
Quelle: Prof. B. Steinbach, Vorlesung „Verteilte Software“

Klasse / Interface - Beispiele

```
public class Point {
	//…
}

public interface IDrawable {
	//…
}

public class Line extends Point implements IDrawable{
	//…
}

public interface I3DDrawable extends IDrawable {
	//…
}
```

Klasse - Kapselung

```
[public]
[abstract | final]
class Bezeichner
[extends BasisKlasse]
{
   {Attribut}
   {Methode}
   {Konstruktor}
}
```



Kapselung - Verbergen oder Schützen von Daten vor dem unmittelbaren Zugriff von außen. Der Zugriff erfolgt stattdessen über definierte Schnittstellen.

Klasse – Kapselung - Beispiel

```
public class Animal {
   private String name;

   public void say(){
        System.out.println(name+": frgdrf!");
   }

   public String getName(){
        return name;
   }

   public void setName(String name){
        this.name=name;
   }
}
```

Klassen und Objekte

Klasse (*Objekttyp*): ein abstraktes Modell bzw. ein *Bauplan* für eine Reihe von ähnlichen Objekten (s. wikipedia)

Objekt: Exemplar eines bestimmen Typs, konkrete Ausprägung, wird erzeugt zur Laufzeit

```
Animal animal1=new Animal();
animal1.setName("Max"); animal1.say();
```

Klasse - Beispiel

```
public class Animal {
   private String name;
   public void say(){
        System.out.println(name+": frgdrf!");
   }
   public Animal(){
        name="unknown";
        //name = null;
   }
   public Animal(String name){
        this.name=name;
   }
   public String getName(){
        return name;
   }
   public void setName(String name){
        this.name=name;
   }
}
```

```
Animal animal1=new Animal();
Animal animal2=new Animal("Auguste");
animal1.say();
animal2.say();
```

//spezielle Referenzen:
//null
//this
//super

Variablendefinition

```
Animal animal1;
animal1 = new Animal();
int i;
```

Referenzvariablen:

new, Heap und Gabrage-Collector

Garbage Collection - GC (*Müllabfuhr)*: automatische Speicherverwaltung (Bereinigung)
Zur Laufzeit werden nicht benötigte (nicht referenzierte) Variablen automatisch identifiziert und freigegeben

Vorteil gegenüber manueller Speicherverwaltung: Vermeidung von Speicherproblemen (Speicherlecks)
Nachteil: erhöhter Ressourcenverbrauch (wegen Verzögerung)

Lokale Variablen, Methodenparameter:

Stack

Mitglieder der Klasse

Attribut

```
[public | protected | private]
[static]
[final]                          -> konstant, nur initialisierbar
[transient]                      -> nicht zu serialisieren
[volatile]                       -> ohne Codeoptimierung

Typ Variablenbezeichner [ = Initialisierung ]
{ , Variablenbezeichner [ = Initialisierung ] };
```

```
private
    String name="Bosch",
           vorname="Hieronymus";
```

Mitglieder der Klasse

Methode

```
[public | protected | private]
[static]
[abstract | final]
[native]
                -> nur Deklaration, Implementierung in einer anderen
                Sprache
[synchronized]
Resultattyp Methodenname ( [Parameterliste] )
                                    [throws Typnamenliste]{
  {Anweisungen}
}
```

```
public void say(){
   System.out.println("You say goodbye and I say hello");
}
```

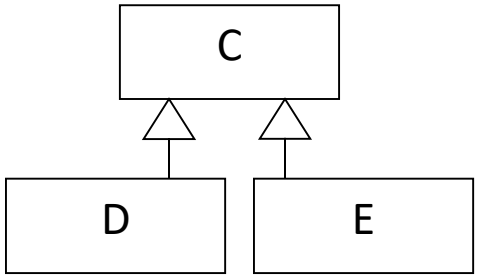
Eigenschaften

Modifizierer	Klasse	Attribut	Methode	Konstruktor
public	x	x	x	x
protected		x	x	x
private		x	x	x
static		x	x	
final	x	x	x	
abstract	x		x	
native			x	
synchronized			x	
transient		x		
volatile		x		

final – nur initialisierbar (Attribut), nicht überschreibar (Methode, Klasse)

Klasse – Vererbung (Generalisierung)

```
[public]
[abstract | final]
class Bezeichner
[extends Klassentyp]
{
   {Attribut}
   {Methode}
   {Konstruktor}
}
```



Vererbung - Beispiel

```
public class Dog extends Animal{
   private String breed;
   public Dog(){
         super(); breed="mongrel";
   }
   public Dog(String name, String breed){
         super(name);
         this.breed=breed;
   }
   @Override
   public void say(){
         super.say();
         System.out.print("I'm "+breed+", Waff waff!");
   }
}
public class Cat extends Animal{
//…
   public void say(){
                  System.out.println("Miau miau?");
}
```

```
Dog dog=new Dog("Trixi","Mops");
Cat cat=new Cat(…);
dog.say();
cat.say();
```

Polymorphie: Vielgestaltigkeit

Klassen: Referenzen der Basisklasse können Objekte der abgeleiteten Klassen zugewiesen bekommen

```
Animal dog=new Dog("Trixi","Mops");
Animal cat=new Cat(…);
dog.say();
cat.say();
```

Methoden: gleiche Namen, verschiedene Implementierungen

Vererbung und Polymorphie - Beispiel

```
abstract class Animal {
        private String name;

        public abstract String void say();

        public Animal(){
                name="unknown";
        }

        public Animal(String name){
                this.name=name;
        }
}
```

abstract?

Ist eine Methode *abstract*,
so muss die Klasse ebenfalls
abstract sein.

Ist eine Klasse *abstract*,
so kann von dieser Klasse kein
Objekt erzeugt werden.

```
Animal animal2=new Animal("August"); //geht nicht
Animal dog=new Dog("Trixi","Mops");
Animal cat=new Cat(…);
dog.say();
cat.say();
```

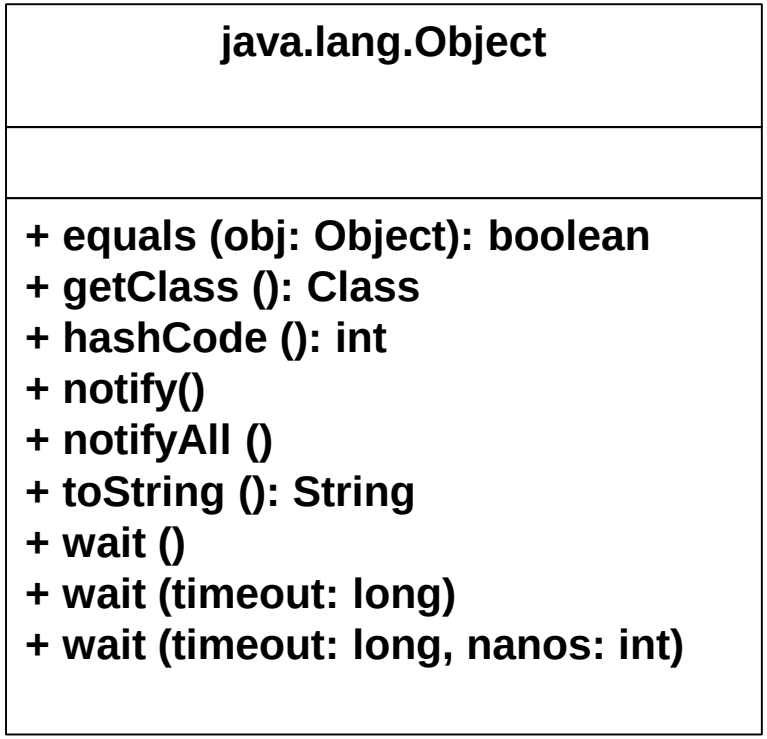
static

Klassenvariablen (Methoden)
sind nicht den Objekten zugeordnet, sondern gehören zur Klasse selbst und stehen in allen Instanzen gleichermaßen zur Verfügung

```
public final class EineKlasse {
   //...
   public static final double PI = 3.141592653589793238 46;
   //...
   }
   public static void print() {
       System.out.println(PI);
   }
}
```

```
public class AndereKlasse {

  public static void main(String[] args) {
     System.out.println(EineKlasse.PI);
     EineKlasse.print();
  }
}
```

Klasse Object




```
Scanner sc=new Scanner(System.in);
System.out.println(sc.getClass());

liefert
     class java.util.Scanner

System.out.println(sc.toString());

liefert
     java.util.Scanner[delimiters=\
p{javaWhitespace…
```

Object – Methoden überschreiben

```
public class Animal {
    String name;
    @Override
    public String toString() {
        return "Animal [name=" + name + "]";
    }
    @Override
    public int hashCode() {
       int result = 31;
       result = result + ((name == null) ? 0 : name.hashCode());
       return result;
    }
    @Override
    public boolean equals(Object obj) {
       if (this == obj) return true;
       if (obj == null) return false;
       if (this.getClass() != obj.getClass()) return false;
       Animal other = (Animal) obj;
       return name.equals(other.name);
   }
}
```

Innere Klassen

Typ	Beispiel

Statische innere Klasse	`class Aussen`
	`{ static class Innen{}`
	`}`

Mitgliedsklasse	`class Aussen`
	`{ class Innen{}`
	`}`

Lokale Klasse	`class Aussen`
	`{  void methode(){ class Innen{}`
	`  }`
	`}`

Anonyme innere Klasse	`object.methode(new Interface(){`
	`  public interfaceMethode(){`
	`  }`
	`});`

Innere Klassen

Statische Klassen – geschachtelte Top-Level-Klassen:

```
public class Aussen{
        public static class Innen{//…
        }
}

Aussen objektA=new Aussen();
Aussen.Innen objektI=new Aussen.Innen();
```

Innere Klassen

Instanzklassen – Elementklassen – echte innere Klassen:

```
public class Aussen {
        public class Innen{//…
        }
}

Aussen objektA=new Aussen();
Aussen.Innen objektI=objektA.new Innen();
```

Innere Klassen

Lokale Klassen – innerhalb eines Blocks:

```
public class Aussen{
  public void methode()
  {
        class Innen implements Runnable{
          //…
           public run(){
              //…
           }
       }
       new Innen().run();
  }
}
```

Innere Klassen

Anonyme innere Klassen – lokale Klassen ohne Namen, werden innerhalb eines Ausdrucks (Block) definiert und erzeugen automatisch ein Objekt:

```
object.methode(new Interface(){
    //…
    public interfaceMethode(){
    //…
    }
});
```

Interface

```
interface Bezeichner
[extends Schnittstellenliste]
{
  {Typ Konstantenname = Konstantenname; }
  {Typ Methodenname( [Parameterliste] ); }
}
```

- ermöglicht Mehrfachvererbung
- stellt Methode (sehr selten Konstanten) zur Verfügung, die von Klassen implementiert werden und zwar alle Methoden
- Methoden sind implizit *public* (öffentlich) und *abstract*

Interface - Beispiel

```
public interface IDrawable {
	void draw();
}

public class Line extends Point implements IDrawable{
	@Override
	public void draw(){
	/*…*/
	}
}
```

Verteilung durch objektorientierte Programmierung

- separation of concerns:
  Trennung der Zuständigkeiten
- Verteilung von Aufgaben auf Klassen (Pakete)

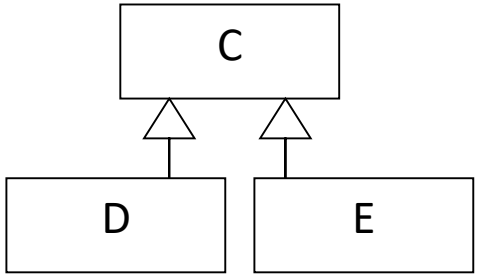
Wikipedia: separation of concerns (SoC) is a design principle for separating a computer program into distinct sections

Design Pattern

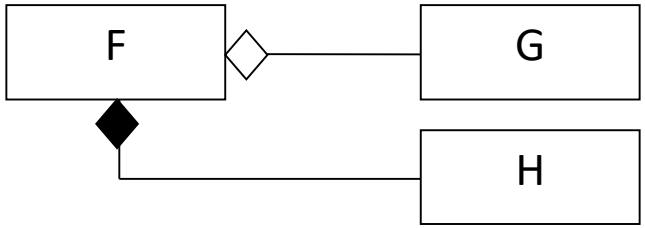
praxiserprobte Lösungsmuster (Entwürfe) für wiederkehrbare Aufgaben

Unified Modeling Language (UML)

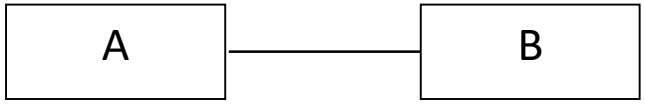
Vererbung (inheritance)



Aggregation / Komposition



Assoziation



Quelle: Prof. B.Steinbach, Vorlesung „Verteilte Software“

Literatur:

Christian Ullenboom:
Java ist auch eine Insel http://openbook.rheinwerk-verlag.de/javainsel/

Goll, Joachim, Heinisch, Cornelia: Java als erste Programmiersprache