Klassen, Typen, Objekte

Adrian Schüpbach
adrian_laurent.schuepbach@alumni.ethz.ch

Worum geht es?

- Klassen
- Typen
- Objekte
- Scope

Klassen, Typen und Objekte

Klassen, Typen und Objekte

- Bis jetzt alles in einer Klasse programmiert
- Was ist aber eine Klasse?

Klassen, Typen und Objekte

Klassen

- Eine Klasse ist eine Einheit, die Daten und Funktionalität beinhaltet
  - Daten sind Variablen eines bestimmten Typs
  - Variablen, die zu einer Klasse gehören, heissen Felder (Fields)
  - Funktionalität wird als Methoden programmiert

Klassen, Typen und Objekte

Klassen

- Eine Klasse ist eine Einheit, die Daten und Funktionalität beinhaltet
  - Daten sind Variablen eines bestimmten Typs
  - Variablen, die zu einer Klasse gehören, heissen Felder (Fields)
  - Funktionalität wird als Methoden programmiert
- Klasse beschreibt also komplexeren Zusammenhang von Daten, die zusammengehören
- Klasse ist Datentyp

Klassen, Typen und Objekte

Klassen

Klasse mit zwei Felder

```
class KomplexeZahl {
  int realTeil;
  int imaginaerTeil;
}

public class Programm25 {
  public static void main(String[] args) {
    KomplexeZahl c1;
  }
}
```

- Deklaration der Variablen `c1` des Typs `KomplexeZahl`

Klassen, Typen und Objekte

Klassen erzeugen → Objekte

- Klasse ist “Vorlage” für Objekt
- Variable einer Klasse/eines Typs ist nur *Zeiger* auf Objekt (wie bei Array)
- Objekt *zuerst erstellen*, dann benutzen
  - `Klasse var = new Klasse();`

Klassen, Typen und Objekte

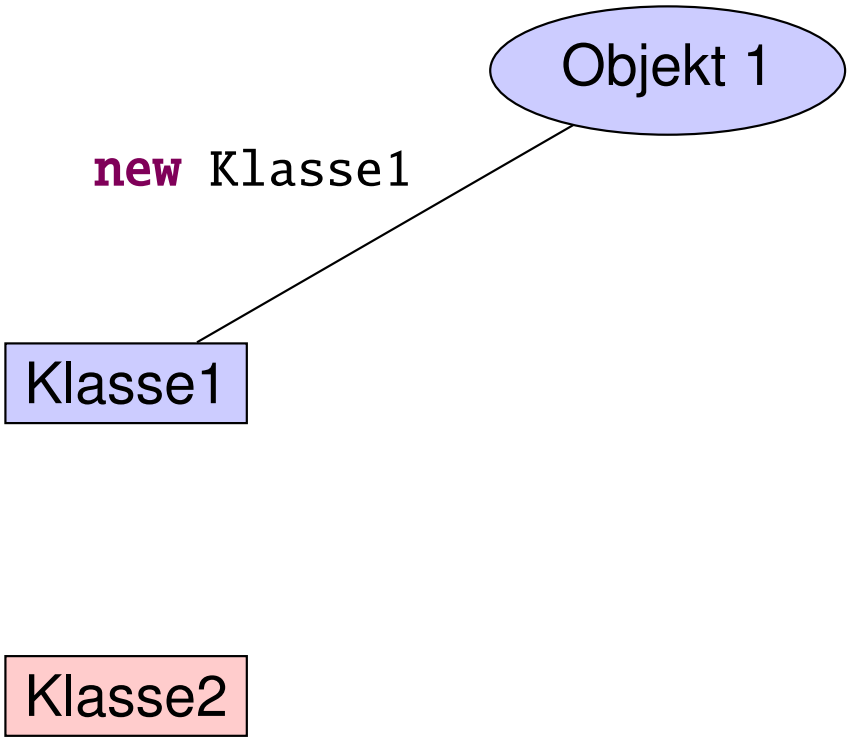
Klassen erzeugen → Objekte

Klasse1

Klasse2

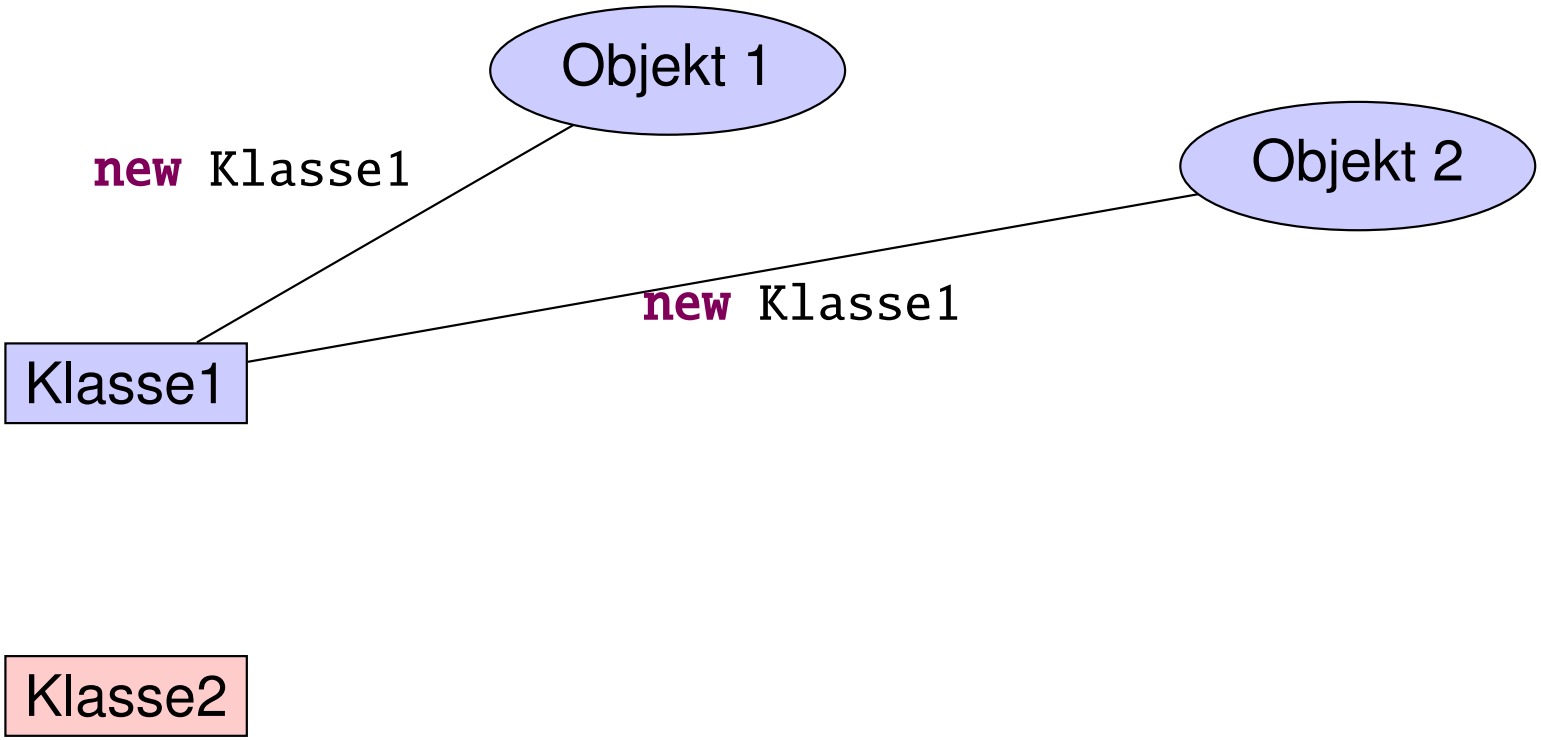
Klassen, Typen und Objekte

Klassen erzeugen → Objekte



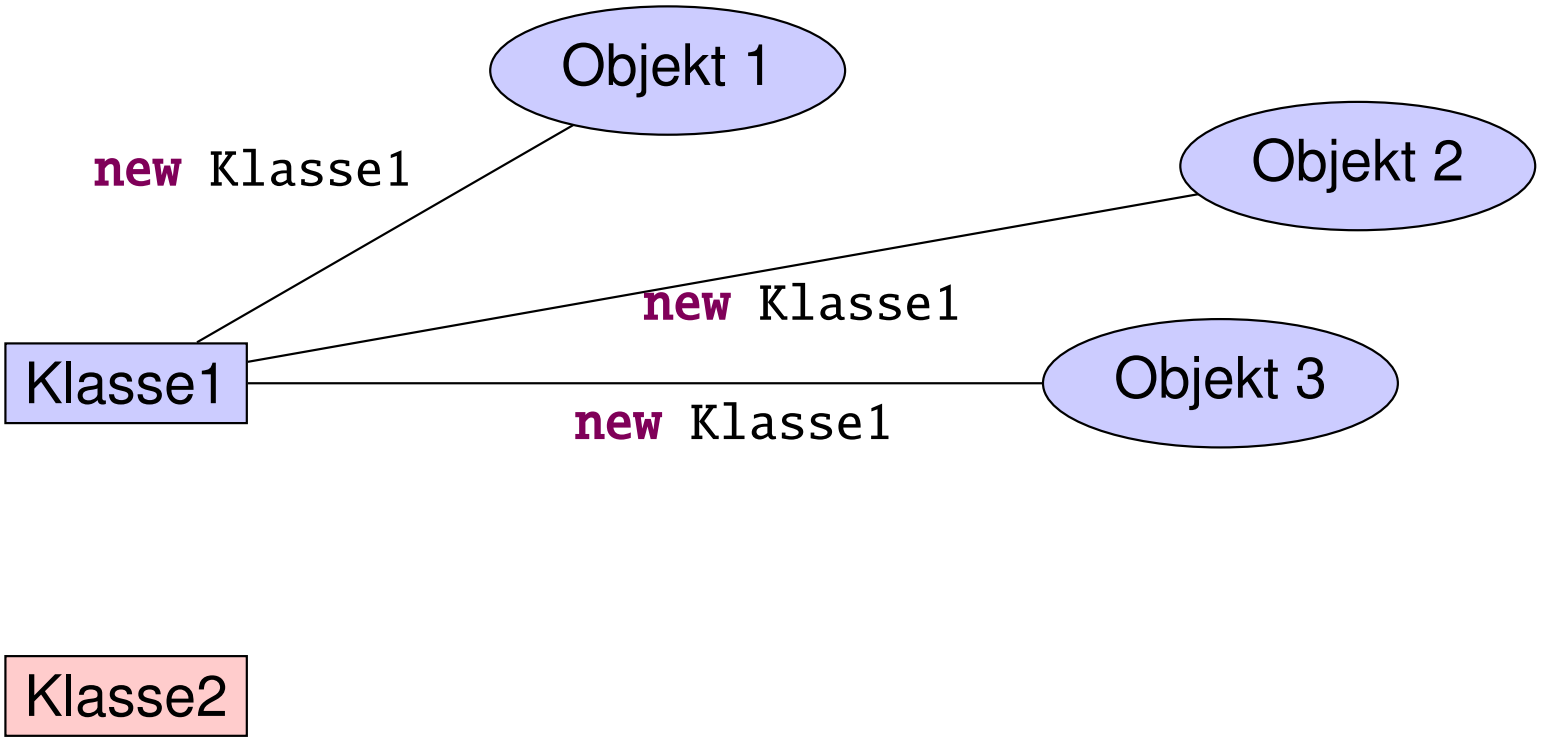
Klassen, Typen und Objekte

Klassen erzeugen → Objekte



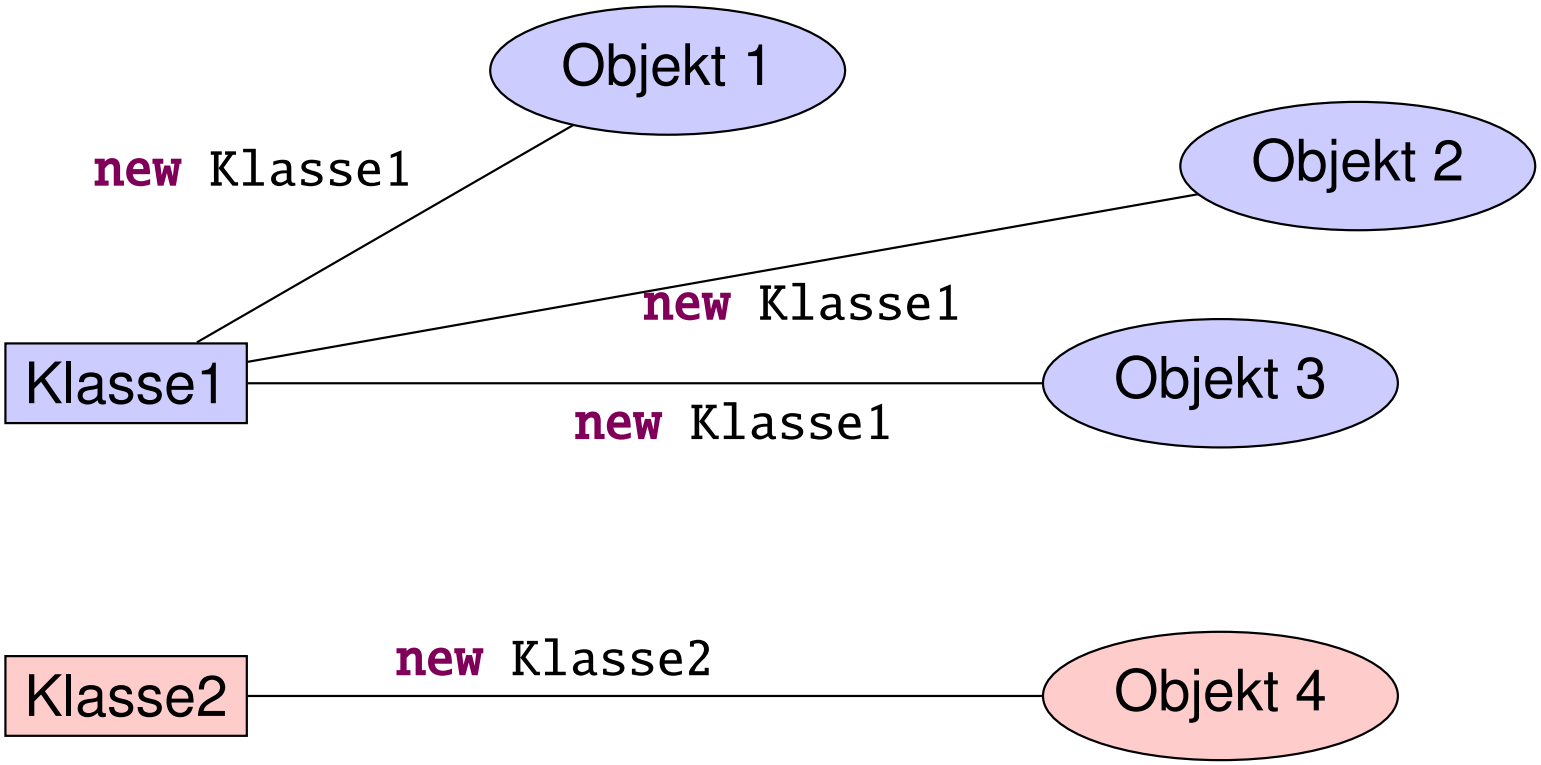
Klassen, Typen und Objekte

Klassen erzeugen → Objekte



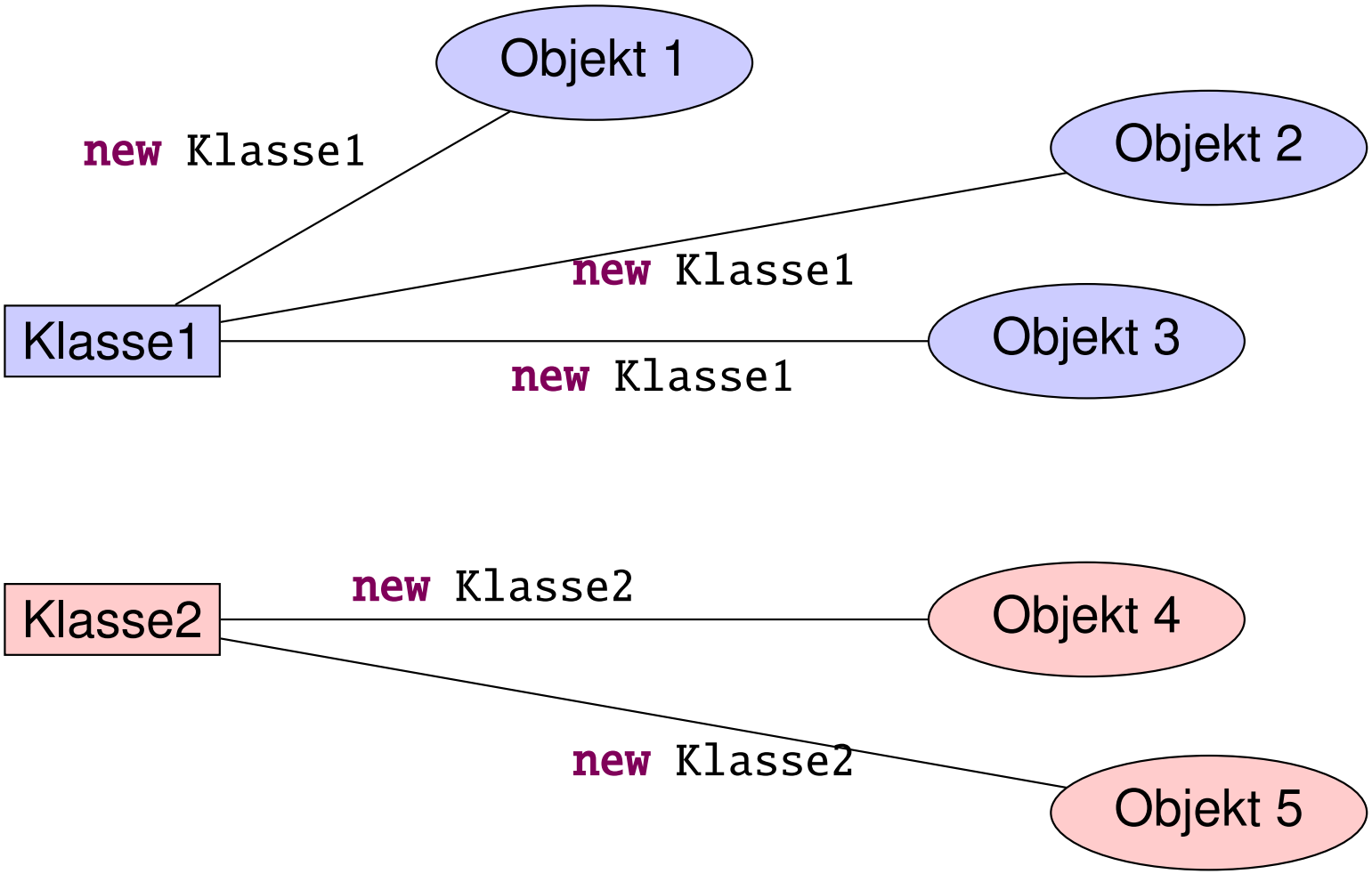
Klassen, Typen und Objekte

Klassen erzeugen → Objekte



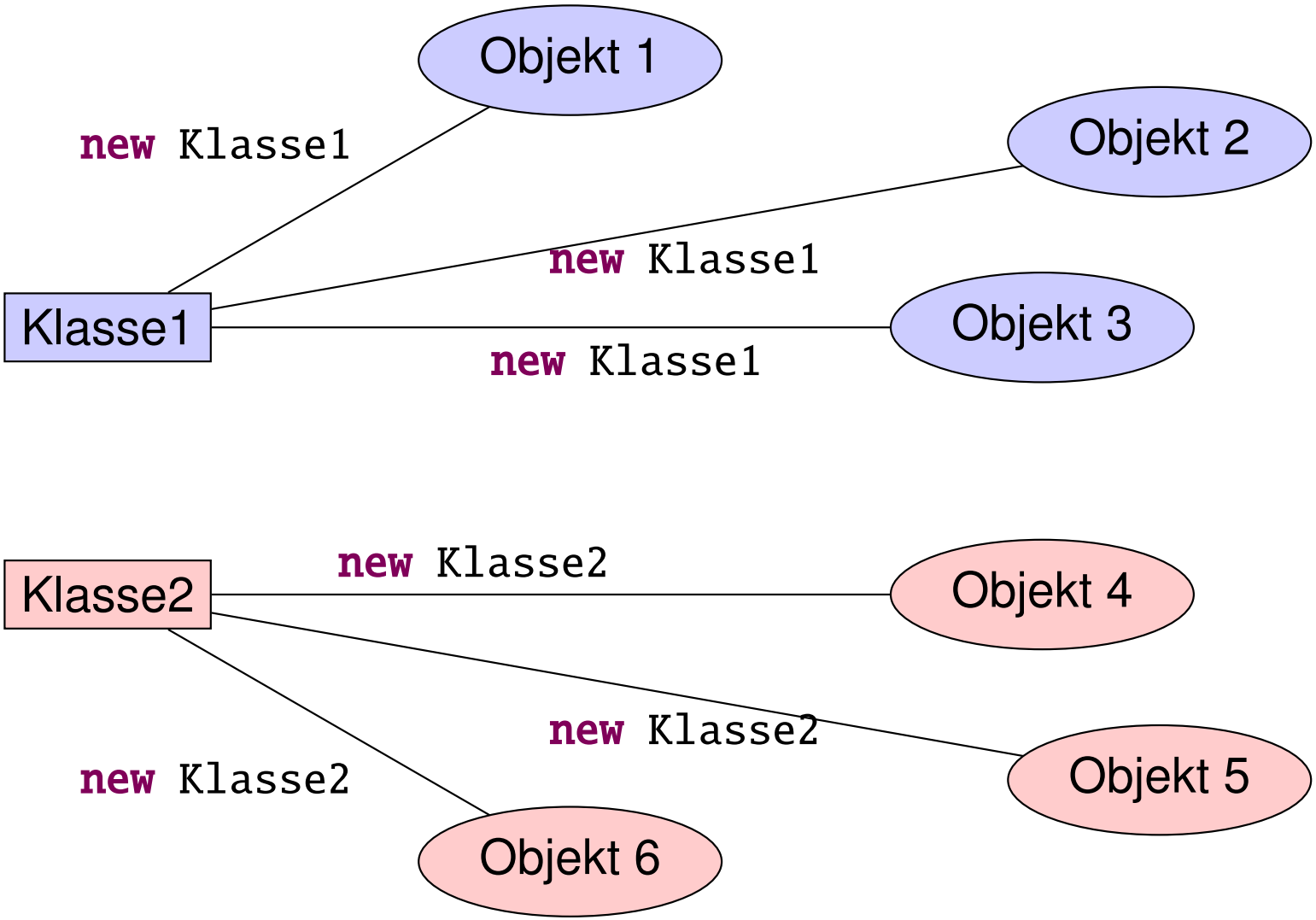
Klassen, Typen und Objekte

Klassen erzeugen → Objekte



Klassen, Typen und Objekte

Klassen erzeugen → Objekte



Klassen, Typen und Objekte

Klassen

- Zugriff auf Inhalt der Klasse
  - “Variablenname”.“Inhaltsname”
  - Gilt für Felder und Methoden

Klassen, Typen und Objekte

Objekte

Klasse mit zwei Felder

```
class KomplexeZahl {
  int realTeil;
  int imaginaerTeil;
}

public class Programm26 {
  public static void main(String[] args) {
    //c1 zeigt auf Objekt des Typs KomplexeZahl
    KomplexeZahl c1 = new KomplexeZahl();
    c1.realTeil = 3; //realTeil zuweisen
    c1.imaginaerTeil = 17;
    System.out.println("c:(" + c1.realTeil + ", "
                       + c1.imaginaerTeil + ")");
  }
}
```

Klassen, Typen und Objekte

Klasse mit Funktionalität

Klasse mit zwei Felder

```
class KomplexeZahl {
  int realTeil;
  int imaginaerTeil;

  void addToReal(int wert) {
    realTeil += wert;
  }
}
```

- Methode `addtoRealTeil()` addiert einen Wert zum Realteil

Klassen, Typen und Objekte

Klasse mit Methode

```
class KomplexeZahl {
  int realTeil;
  int imaginaerTeil;

  void addToReal(int wert) {
    realTeil += wert;
  }
}
public class Programm27 {
  public static void main(String[] args) {
    KomplexeZahl c1 = new KomplexeZahl();
    c1.realTeil = 3;     c1.imaginaerTeil = 17;
    c1.addToReal(5); //Methodenaufruf auf Objekt c1
    System.out.println("c:(" + c1.realTeil + ", "
                       + c1.imaginaerTeil + ")");
  }
```

Klassen, Typen und Objekte

Klasse mit Methoden

```
class KomplexeZahl {
  int realTeil;  int imaginaerTeil;

  void addToReal(int wert) {
    realTeil += wert;
  }
  void addToImag(int wert) {
    imaginaerTeil += wert;
  }
  // direkter Zugriff auf realTeil und
  // imaginaerTeil, da hier IM Objekt
  void ausgabe() {
    System.out.println("c:(" + realTeil + ", "
                       + imaginaerTeil + ")");
  }
}
```

Klassen, Typen und Objekte

Hauptprogramm

```
public class Programm28 {
  public static void main(String[] args) {
    KomplexeZahl c1 = new KomplexeZahl();
    c1.realTeil = 3;     c1.imaginaerTeil = 17;
    c1.addToReal(5); //Methodenaufruf auf Objekt c1
    c1.ausgabe();
  }
}
```

Klassen, Typen und Objekte

Rückgabewerte

- Methoden in Objekten können Wert zurückgeben
  - Wie gewöhnt: `return()`

Klassen, Typen und Objekte

Rückgabewerte

```
class KomplexeZahl {
  int realTeil;  int imaginaerTeil;

  void addToReal(int wert) { realTeil += wert; }
  void addToImag(int wert) { imaginaerTeil+=wert; }
  void ausgabe() {
    System.out.println("c:(" + realTeil + ", "
                       + imaginaerTeil + ")");
  }
  double berechneBetrag() {
    return (Math.sqrt((realTeil*realTeil +
                       imaginaerTeil*imaginaerTeil)));
  }
}
```

Klassen, Typen und Objekte

Hauptprogramm: Berechne Betrag

```
public class Programm29 {
  public static void main(String[] args) {
    KomplexeZahl c1 = new KomplexeZahl();
    c1.realTeil = 6;     c1.imaginaerTeil = 8;
    System.out.println("Betrag = " +
                       c1.berechneBetrag());
  }
}
```

Klassen, Typen und Objekte

Konstruktor

Klassen, Typen und Objekte

Konstruktor

- Sinnvoll, ein erzeugtes Objekt direkt zu initialisieren
  - Anfangswerte zuweisen
  - Interne Werte auf bestimmten Anfangswert setzen

Klassen, Typen und Objekte

Konstruktor

- Sinnvoll, ein erzeugtes Objekt direkt zu initialisieren
  - Anfangswerte zuweisen
  - Interne Werte auf bestimmten Anfangswert setzen

- Konstruktor
  - Ähnlich, wie “normale” Methode
  - Hat keinen Rückgabetyp
  - Muss gleich heissen, wie Klasse
  - Kann Parameterliste haben

Klassen, Typen und Objekte

Konstruktor

```
class KomplexeZahl {
  int realTeil;  int imaginaerTeil;

  public KomplexeZahl(int real, int imag) {
    realTeil = real;
    imaginaerTeil = imag;
  }
  void ausgabe() {
    System.out.println("c:(" + realTeil + ", "
                       + imaginaerTeil + ")"); }
}
public class Programm30 {
  public static void main(String[] args) {
    KomplexeZahl c1 = new KomplexeZahl(3, 9);
    c1.ausgabe();
  }
```

Klassen, Typen und Objekte

Rückgabe neuer Objekte

- Beispiel:
  - Zwei komplexe Zahlen addieren
  - Resultat einer neuen Variablen zuweisen

Klassen, Typen und Objekte

Rückgabe neuer Objekte

- Beispiel:
  - Zwei komplexe Zahlen addieren
  - Resultat einer neuen Variablen zuweisen
- Algorithmus: Additionsmethode implementieren
  1. Gehört zur Klasse der komplexen Zahlen
  2. Nimmt eine komplexe Zahl als Argument
  3. Addiert diese zu sich
  4. Erzeugt neues Objekt mit Resultat
  5. Gibt dieses Objekt zurück

Klassen, Typen und Objekte

Rückgabe neuer Objekte

Rückgabe eines neuen Objektes

```
class KomplexeZahl {
  int realTeil;  int imaginaerTeil;

  public KomplexeZahl(int real, int imag) {
    realTeil = real;
    imaginaerTeil = imag;
  }
  KomplexeZahl add(KomplexeZahl z) {
    return(new KomplexeZahl(realTeil + z.realTeil,
                    imaginaerTeil + z.imaginaerTeil));
  }
}
```

Klassen, Typen und Objekte

Rückgabe neuer Objekte

Hauptprogramm

```
public class Programm31 {
  public static void main(String[] args) {
    KomplexeZahl c1 = new KomplexeZahl(3, 9);
    KomplexeZahl c2 = new KomplexeZahl(2, 11);

    KomplexeZahl c3 = c1.add(c2);
    c3.ausgabe();
  }
}
```

Klassen, Typen und Objekte

Dateinamen

Klassen, Typen und Objekte

Dateinamen

- Konventionen für Dateinamen

Klassen, Typen und Objekte

Dateinamen

- Unterschied `public class Klasse` und `class Klasse`
- Dateiname `public class Klasse`
  - Klasse kann von jeder anderen Klasse benutzt werden
  - Dateiname *muss* gleich sein, wie Klassenname
  - → `Klasse.java`
- Dateiname `class Klasse`
  - Klasse kann nur von Klassen im selben Paket benutzt werden
    - → mehr dazu später
  - Datei kann beliebigen Namen haben

Klassen, Typen und Objekte

Programm-Organisation

- Eine Hauptklasse in Datei mit dem selben Namen
  - Beinhaltet `public static void main(String[] args)`
  - Bsp.: `public class MeinProgramm` → `MeinProgramm.java`
- Mehrere zusätzliche Klassen in der Datei möglich
  - `class C1`
  - `class Datenstruktur`
  - ...
- → Compiler erzeugt für jede Klasse separate `class`-Datei

Klassen, Typen und Objekte

Programm-Organisation

- Weitere Möglichkeite: Statische innere Klasse
  - Klasse innerhalb einer Klasse
  - → Compiler erzeugt für jede Klasse separate `class`-Datei

Innere Klasse

```
public class Programm52 {
  public static class InnereKlasse {
    int f;
  }
  public static void main(String[] args) {
    InnereKlasse ik = new InnereKlasse();
  }
}
```

Klassen, Typen und Objekte

Demo

Demo

- Mehrere Personen
- Jede Person hat mehrere Telefonnummern
- $\rightarrow$ Alles in einer Datei

Sichtbarkeit/Scope

Sichtbarkeit/Scope

- Variablen sind in einem bestimmten Bereich gültig
- Bereich heisst Scope
- Ausserhalb des Bereichs sind Variablen *nicht* sichtbar

Sichtbarkeit/Scope

- Beispiel: Variable `i` nur in **`for`**-Schleife sichtbar

Scope: `for`-Schleife

```
public class Programm32 {
  public static void main(String[] args) {

    //hier nicht

    for (int i = 0; i < 10; i++) {
      // hier ist i gueltig
    }

    //hier auch nicht

  }
}
```

Sichtbarkeit/Scope

- Benutzt wird Variable des *innersten* Scopes
- Mehrfachdeklaration des *gleichen* Variablennamens in *verschiedenen* Scopes möglich

Sichtbarkeit/Scope

Verschachtelte Scopes

```
public class Programm33 {
    int i; //nr1
    public static void main(String[] args) {
        Programm33 p33 = new Programm33();
        p33.f();
    }
    public void f() {
        //hier nr1 gueltig
        for (int i = 0 /* nr2 */; i < 10; i++) {
            // hier nr2 gueltig
        }
        //hier nr1 gueltig
    }
    public void mach() {
        //hier nr1 gueltig
    }
```

Sichtbarkeit/Scope

- Spezialfall: Kann immer auf Felder zugreifen
  - `this` zeigt auf “sich selber” (Objekt)
  - Kann “Zeiger”.“Feld” benutzen

Sichtbarkeit/Scope

Verschachtelte Scopes

```
public class Programm34 {
    int i; //nr1
    public static void main(String[] args) {
        Programm34 p34 = new Programm34();
    }
    public void f() {
        //hier nr1 gueltig
        for (int i = 0 /* nr2 */; i < 10; i++) {
            this.i = i; //nr1 mit nr2 zuweisen
        }
        //hier nr1 gueltig
    }
    public void mach() {
        //hier nr1 gueltig
    }
}
```

Sichtbarkeit/Scope

Konstruktor und this

```
class KomplexeZahl {
  int realTeil;  int imaginaerTeil;

  public KomplexeZahl(int realTeil,
                      int imaginaerTeil) {
    this.realTeil = realTeil;
    this.imaginaerTeil = imaginaerTeil;
  }
}
```

Text – Zeichen – Zahlen – Strings

Text – Zeichen – Zahlen – Strings

- Text ist eine Anreihung von Zeichen
- Text wird in Anführungs- und Schlusszeichen gesetzt
  - “Guten Tag!”
  - “Es ist 16:40 Uhr.”

Text – Zeichen – Zahlen – Strings

- Einzelnes Zeichen ist **`char`**
  - Erinnerung: Wertebereich **`char`**: 00000 – 65535
- $\Rightarrow$ Zeichen wird als Wert gespeichert

Text – Zeichen – Zahlen – Strings

Beispiele

Wert	Zeichen/Darstellung
97	'a'
65	'A'
105	'i'
73	'I'
32	' '
46	'.'
33	'!'

Text – Zeichen – Zahlen – Strings

- Wie sieht es mit den Zahlen aus?

Text – Zeichen – Zahlen – Strings

- Wie sieht es mit den Zahlen aus?
- “Es ist 16:40 Uhr.”
  - Zahlen gehören zum Text

- Speichern und darstellen von Zahlen?

Text – Zeichen – Zahlen – Strings

- Wie sieht es mit den Zahlen aus?
- “Es ist 16:40 Uhr.”
  - Zahlen gehören zum Text

- Speichern und darstellen von Zahlen?
  - Im Text ist Zahl auch nur ein Zeichen
  - Zahl hat bestimmten Wert

Text – Zeichen – Zahlen – Strings

Beispiele

Wert	Zeichen/Darstellung
48	’0’
49	’1’
50	’2’
..	...
55	’7’
56	’8’
57	’9’

Text – Zeichen – Zahlen – Strings

- Achtung:
  - Tastatur liefert immer **Zeichen**, keine Zahlen!
  - Bildschirm stellt **Zeichen** anhand eines Zahlenwertes dar

Text – Zeichen – Zahlen – Strings

Wenn man mit...

- ... Zahlen und Tastatur ...
- ... Zahlen und Bildschrim ...
- ... Zahlen und Drucker ...
- ... Zahlen und Dateien ...

zu tun hat, muss man wissen, ob es ...

- ... richtige Zahlen ...
- ... Zahlenzeichen ...

sind.

Text – Zeichen – Zahlen – Strings

Übersetzung Zeichen ↔ Zeichenwert

- Zeichenwerte müssen interpretiert werden
- Für Interpretation gibt es Tabellen
  - Zeichenwert → Zeichen
  - Zeichen → Zeichenwert

Text – Zeichen – Zahlen – Strings

Übersetzung Zeichen ↔ Zeichenwert

- Zeichenwerte müssen interpretiert werden
- Für Interpretation gibt es Tabellen
  - Zeichenwert → Zeichen
  - Zeichen → Zeichenwert
- ASCII-Tabelle
  - Übersetzt 8Bit (7Bit in USA...) Zeichenwerte
  - Andere Tabelle, je nach Land
    - CH: Tabelle 850
    - Wir brauchen z.Bsp. Umlaute und Akzente
- UTF16-Tabelle
  - Einheitlich, international
  - Beinhaltet ASCII-Tabelle (7Bit)

Text – Zeichen – Zahlen – Strings

Übersetzung Zeichen ↔ Zeichenwert

- Zeichenwerte müssen interpretiert werden
- Für Interpretation gibt es Tabellen
  - Zeichenwert → Zeichen
  - Zeichen → Zeichenwert
- ASCII-Tabelle
  - Übersetzt 8Bit (7Bit in USA...) Zeichenwerte
  - Andere Tabelle, je nach Land
    - CH: Tabelle 850
    - Wir brauchen z.Bsp. Umlaute und Akzente
- UTF16-Tabelle
  - Einheitlich, international
  - Beinhaltet ASCII-Tabelle (7Bit)

- Java benutzt UTF16-Tabelle

Text – Zeichen – Zahlen – Strings

Zeichenketten

- Text besteht aus mehreren Zeichen
  - Einzelne Zeichen als Kette zusammenfassen

Text – Zeichen – Zahlen – Strings

Zeichenketten

- Text besteht aus mehreren Zeichen
  - Einzelne Zeichen als Kette zusammenfassen
- Möglichkeit 1:

Text – Zeichen – Zahlen – Strings

Zeichenketten

- Text besteht aus mehreren Zeichen
  - Einzelne Zeichen als Kette zusammenfassen
- Möglichkeit 1:
  - Array aus einzelnen Zeichen

Text – Zeichen – Zahlen – Strings

Zeichenketten

- Text besteht aus mehreren Zeichen
  - Einzelne Zeichen als Kette zusammenfassen
- Möglichkeit 1:
  - Array aus einzelnen Zeichen
- Möglichkeit 2:
  - Klasse, die Text beinhaltet

Text – Zeichen – Zahlen – Strings

Zeichenketten

- Text besteht aus mehreren Zeichen
  - Einzelne Zeichen als Kette zusammenfassen
- Möglichkeit 1:
  - Array aus einzelnen Zeichen
- Möglichkeit 2:
  - Klasse, die Text beinhaltet

- Java hat Klasse *String*

Text – Zeichen – Zahlen – Strings

Zeichenketten

- `String` ist Klasse und Datentyp
- Deklaration: `String text="Hallo!";`
- Ausgabe: `System.out.println(text);`

- `String` kümmert sich um das **`char`**-Array

Text – Zeichen – Zahlen – Strings

Zeichenketten

Strings

```
public class Programm35 {
  public static void main(String[] args) {
    String text = "Hallo";
    int zahl = 5;
    String alles = text + " KOMMA " + zahl;
    System.out.println("Das ist alles: " + alles);
  }
}
```

Text – Zeichen – Zahlen – Strings

Zeichenketten

Nützliche Methoden:

- `length()`: Gibt die Länge des Strings zurück (Anzahl Zeichen)
- `substring()`: Git einen Teil des Strings zurück
- `charAt()`: Gibt ein einzelnes Zeichen zurück
- → Java API nachschauen + Vorschläge in Eclipse

Text – Zeichen – Zahlen – Strings

Text in Zahlen umwandeln

- Erinnerung: Tastatur liefert Text/Zeichen, also “5”, nicht 5
- Wenn es Zahl sein soll → umwandeln
- Klassen *Integer, Double, Float* und *Boolean* können das

Wichtige Klassen

“Zahlenklassen” und Boolean

- Gelernt: `int`, `double`, `float`, `boolean` Basistypen
- Integer, Double, Float und Boolean sind *Klassen*
  - Beinhalten Feld (Variable) des entsprechenden Basistyps
  - Beinhalten Funktionalität
- Wichtige Funktionalität: Text in Zahlen umwandeln

Wichtige Klassen

“Zahlenklassen” und Boolean

Text in Zahlen umwandeln

```
public class Programm36 {
  public static void main(String[] args) {
    int zahl = Integer.parseInt("5");
    float zahlf = Float.parseFloat("4.5");
    double zahld = Double.parseDouble("227.13");
    boolean b = Boolean.parseBoolean("true");

    String s = new Integer(zahl).toString();
  }
}
```

Demo

Demo

- Mehrere Personen
- Jede Person hat mehrere Telefonnummern
- **Person hat einen Namen!** → `String`
- → Mehrere Dateien

- Kompilieren auf der Kommandozeile: `javac personen2/*.java`
- Starten von der Kommandozeile: `java personen2.PersonenHaupt`