EINFÜHRUNG OBJEKTORIENTIERTER KONZEPTE

Übersicht

- Grundbegriffe
  - Klasse
  - Objekt
- Attribute
- Methoden
- Beispiel Wirtschaftssimulationsspiel
- Exkurs HashMap

KLASSE ALS BAUPLAN
OBJEKT ALS REALISIERUNG DES BAUPLANS

Wiederholung: Strukturtyp in C

struct - Strukturtyp

- Zusammensetzen einzelner Daten
- Erzeugen eines neuen Datentyps mit Elementen
- Elemente sind an den zusammengesetzten Strukturtyp gebunden
- Vereinfachen die Handhabung zusammengehöriger Daten im Programmcode

Anwendungsbeispiele

- Personendaten bestehen aus Name, Vorname, Geburtsdatum, Ort
- Position besteht aus x-, y-, z-Koordinate
- Farbe besteht aus Rot-, Grün-, Blauanteil
- Datum besteht aus Tag, Monat, Jahr

Beispiele:

```
struct datum
{
    int jahr;
    int monat;
    int tag;
};

struct position
{
    float latitude;
    float longitude;
};
```

Drei Elemente vom Typ int werden zu einer Struktur zusammengefasst.

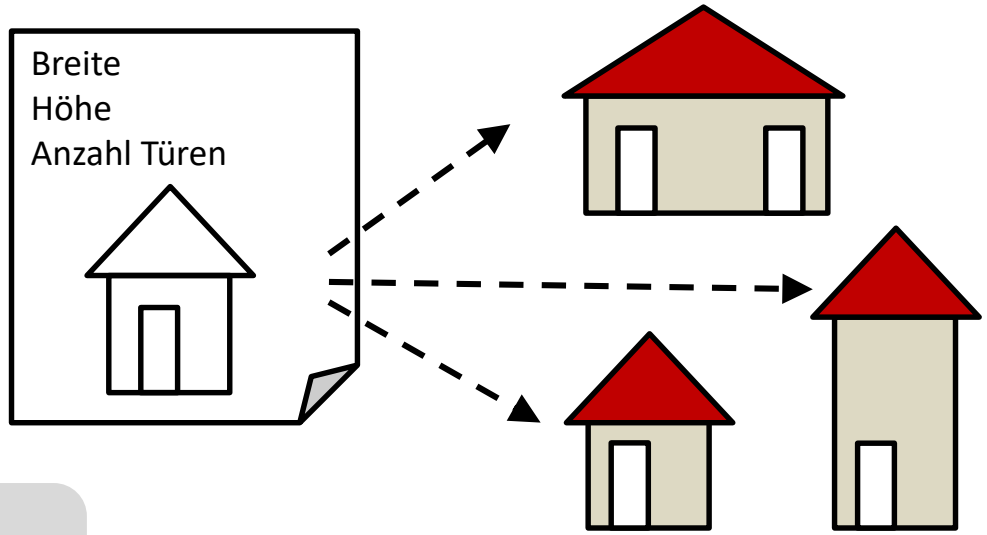
Zwei Elemente vom Typ float werden zu einer Struktur zusammengefasst.

```
struct person
{
    char first_name[30];
    char last_name[30];
    int age;
    float weight;
};
```

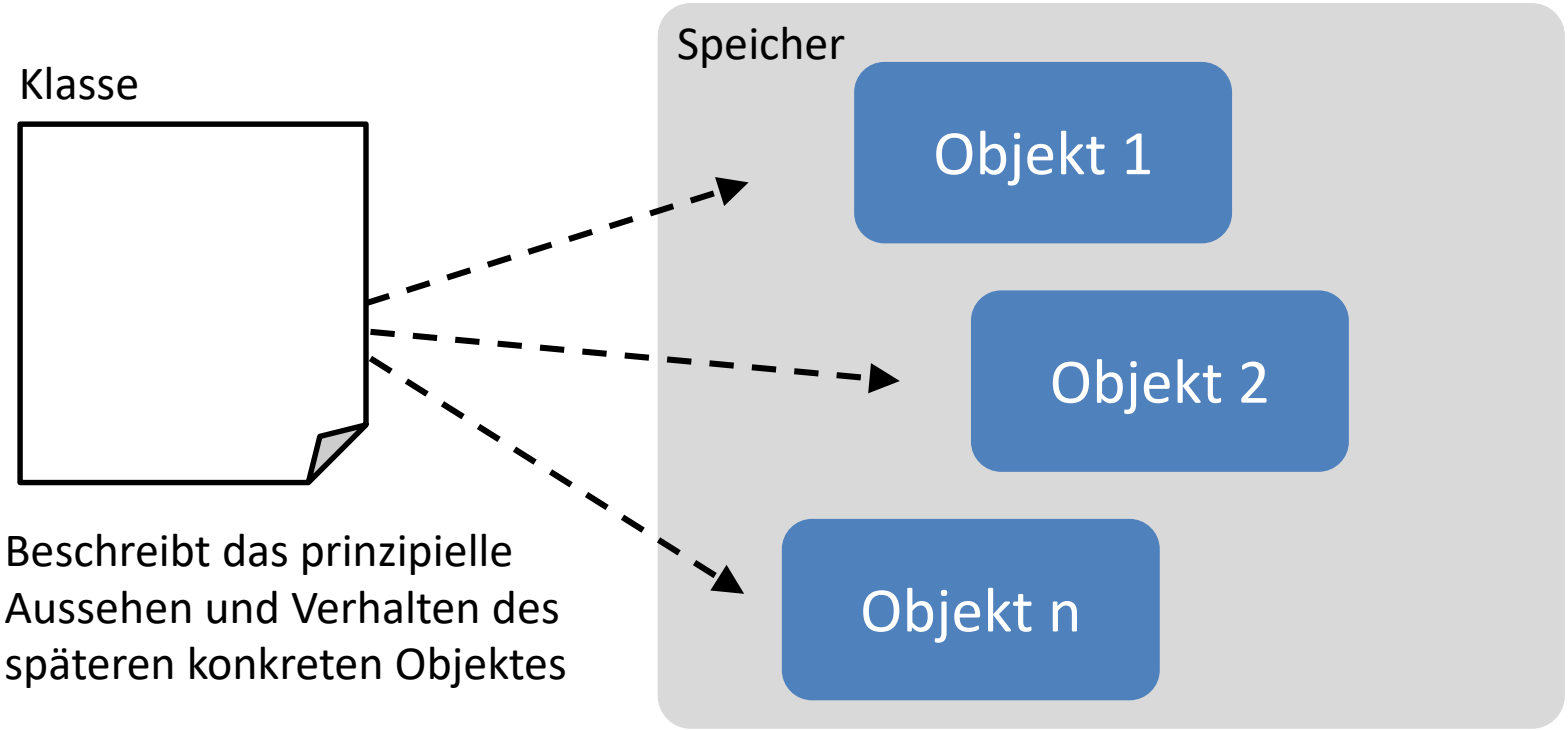
Unterschiedliche Datentypen innerhalb einer Struktur sind erlaubt.

Klasse und Objekt

- Grundbegriffe der Objektorientierten Programmierung
- Klasse ist variabler Bauplan
- Objekt ist ein realisierter Bauplan



Unterschiedliche Realisierungen des Bauplans



Beschreibt das prinzipielle Aussehen und Verhalten des späteren konkreten Objektes

Unterschiedliche Objekte im Speicher gemäß der Beschreibung in der Klasse

Bestandteile von Klassen

Attribute

- Eigenschaften
- vgl. Element innerhalb des Strukturtyps
- muss für eine Klasse eindeutig sein

Methoden

- beschreiben das Verhalten
- ermöglichen Änderungen der Eigenschaften
- Können von anderen Objekten / Programmteilen aufgerufen werden

Konstruktor

- spezielle Methode
- wird bei Erzeugung ausgeführt

BEISPIEL ARBEITSKRÄFTEVERWALTUNG

Beispiel Arbeitskräfteverwaltung

Attribute

- Eigenschaften
- vgl. Element innerhalb des Strukturtyps)
- muss für eine Klasse eindeutig sein

Methoden

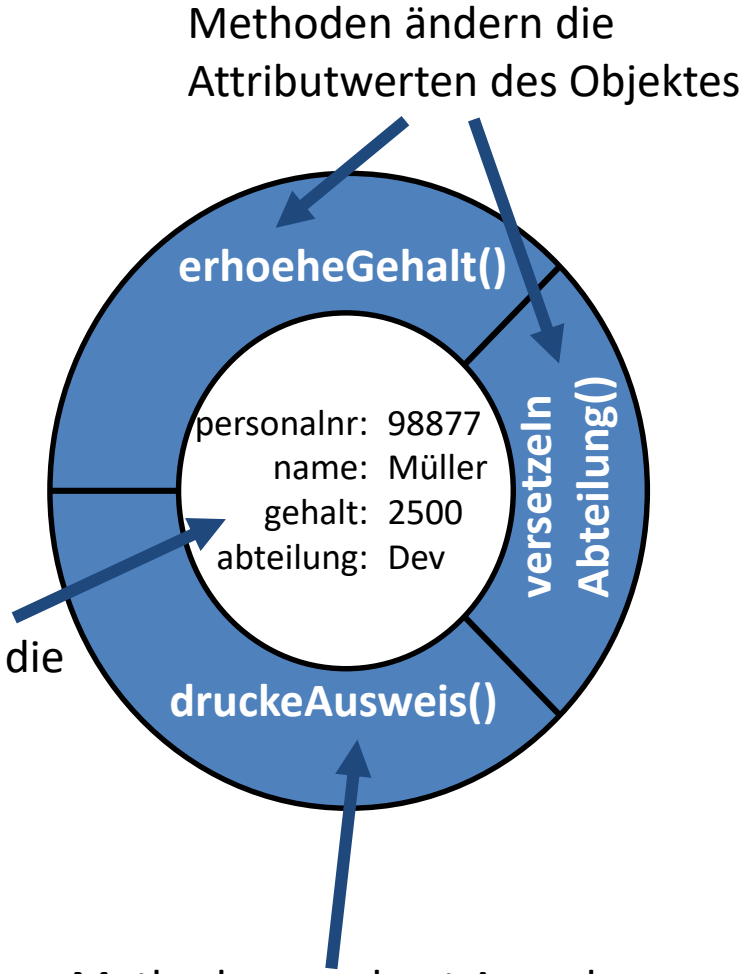
- beschreiben das Verhalten
- ermöglichen Änderungen der Eigenschaften
- Können von anderen Objekten / Programmteilen aufgerufen werden

Konstruktor

- spezielle Methode
- wird bei Erzeugung ausgeführt

Hinweis: Objektorientierung ist auch eine Denkweise, die Objekte der realen Welt in Software abzubilden!

Beispiel: mitarbeitende Arbeitskraft



Implementierung der Klasse Arbeitskraft (demo001.personalverwaltung)

```
public class Arbeitskraft {

    private String personalnr;
    private String name;
    private int gehalt;
    private String abteilung;

    public Arbeitskraft(String nr, String name, int gehalt) {
        this.personalnr=nr;
        this.name=name;
        this.gehalt=gehalt;
    }

    public void erhoeheGehalt(int gehaltserhoehung) {
        this.gehalt = this.gehalt+gehaltserhoehung;
    }

    public void versetzeInAbteilung(String neueAbteilung) {
        this.abteilung=neueAbteilung;
    }

    public String druckeAusweis() {
        String ausgabeString = this.personalnr + ", "+this.name+", "+this.abteilung;
        return ausgabeString;
    }
}
```

Attribute mit ihren Datentypen werden deklariert

Konstruktor für ersten Aufruf (beim „Erstellen“ des Objektes)

Methode für Veränderung von Gehalt

Methode für Versetzung in andere Abteilung

Methode für Ausgabe von Attributen

Nutzung der Klasse Arbeitskraft (demo001.personalverwaltung)

```
public class VerwaltungArbeitskraefte {

    public static void main(String[] args) {

        Arbeitskraft a1 = new Arbeitskraft("001", "Müller", 1000);
        Arbeitskraft a2 = new Arbeitskraft("002", "Schulz", 1000);

        a1.versetzeInAbteilung("Entwicklung");
        a2.versetzeInAbteilung("Vertrieb");

        System.out.println(a1.druckeAusweis());
        System.out.println(a2.druckeAusweis());
    }

}
```

main-Methode für Start des Programms

Klasse wird als Datentyp verwendet

Erzeugen von zwei Objekten der Klasse Arbeitskraft mit **Konstruktor**

Festlegen der Abteilung

Ausgabe von Personendaten

Speicher

personalnr: 001
name: Müller
gehalt: 1000
abteilung: Entwicklung

personalnr: 002
name: Schulz
gehalt: 1000
abteilung: Vertrieb

Hinweis: Diese Implementierung ist eine sehr vereinfachte Darstellung der Arbeitskräfteverwaltung. Es soll die einfache Arbeit mit Klassen und Objekten gezeigt werden. Viele realistische Aspekte fehlen noch!

Aufgabe

Lebensmittelprodukt als Klasse modellieren

- Welche Attribute sind erforderlich?
- Welche Methoden könnten notwendig sein?

EXKURS DATENSTRUKTUR ARRAYLIST

Konzept der ArrayList

Grundprinzip

- Klasse für Speichern von Objekten in einer Reihenfolge
- Methoden für Zugriff auf Objekte
  - Anfügen am Ende
  - Auslesen von einer bestimmten Position
  - Setzen an einer bestimmten Position
  - Löschen an einer bestimmten Position
- Datentyp für Objekt

Methoden für Zugriff (Auszug)

- add(Objekt) – am Ende der Liste das Objekt anfügen
- get(position) – das abgelegte Objekt an einer bestimmten Position aus der ArrayList holen
- set(position, Objekt) – das Objekt an der bestimmten Position einfügen
- remove(position) – das Objekt an der bestimmten Position löschen
- size() – Länge der Liste ermitteln

Beispiel 1: ArrayList mit Integern

Liste der Integer
25
15
16
63

Beispiel 2: Liste von Personen

Liste der Objekte
1, Müller, Marius
2, Schulze, Sven
3, Lehmann, Lutz
4, Schmidtzke, Sören

REFERENZTYP INTEGER IN ARRAYLIST

ArrayList Beispiel (demo001.arraylistdemo)

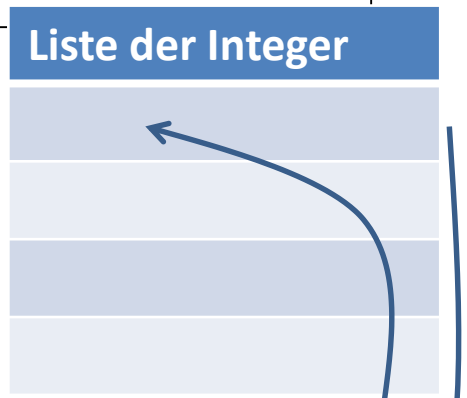
```
public class ArrayListMain1 {

    public static void main(String[] args) {

        ArrayList<Integer> integerListe = new ArrayList<Integer>();

        for(int i=0; i<10; i++) {
            Integer zufallszahl = (int)Math.round(Math.random()*100);
            integerListe.add(zufallszahl);
        }
        for(int i=0; i<10; i++) {
            System.out.println("Pos. "+i+": "+integerListe.get(i));
        }
    }
}
```

- Konstruktor der ArrayList aufrufen
- <Integer> → ArrayList soll Integer speichern
- leere Liste steht als Objekt bereit



- Integer zu ArrayList hinzufügen
- füllen in mehreren Durchläufen

Objekt an Position 0 ist 25

- Objekt an Position i aus ArrayList lesen
- über alle Positionen iterieren

Hinweis: In Java gibt es **primitive Typen** und **Referenztypen**. Für das Speichern in anderen Datenstrukturen werden Referenztypen verwendet.

ArrayList Beispiel (demo001.arraylistdemo)

```java
public class ArrayListMain1 {

    public static void main(String[] args) {

        ArrayList<Integer> integerListe = new ArrayList<Integer>();

        for(int i=0; i<10; i++) {
            Integer zufallszahl = (int)Math.round(Math.random()*100);
            integerListe.add(zufallszahl);
        }
        for(int i=0; i<10; i++) {
            System.out.println("Pos. "+i+": "+integerListe.get(i));
        }
    }
}
```

Ausgabe auf Kommandozeile:

```
Pos. 0: 25
Pos. 1: 15
Pos. 2: 16
Pos. 3: 63
```

ArrayList mit Integern

Liste der Integer
25
15
16
63

OBJEKT VON EIGENER KLASSE IN ARRAYLIST

ArrayList Beispiel (demo001.arraylistdemo)

```java
public class Person {

    private int nr;
    private String lastname;
    private String firstname;

    public Person(int nr, String lastname, String firstname) {
        this.nr=nr;
        this.lastname=lastname;
        this.firstname=firstname;
    }
}
```

```java
import java.util.ArrayList;

public class ArrayListMain2 {
    public static void main(String[] args) {
        ArrayList<Person> personenliste = new ArrayList<Person>();
        personenliste.add(new Person(1, "Müller", "Marius"));
        personenliste.add(new Person(2, "Schulze", "Sven"));
        personenliste.add(new Person(3, "Lehmann", "Lutz"));
        personenliste.add(new Person(4, "Schmitzke", "Sören"));
    }
}
```

ArrayList nach dem Einfügen

Liste der Objekte
1, Müller, Marius
2, Schulze, Sven
3, Lehmann, Lutz
4, Schmidtzke, Sören

ERWEITERUNG PERSONALVERWALTUNG

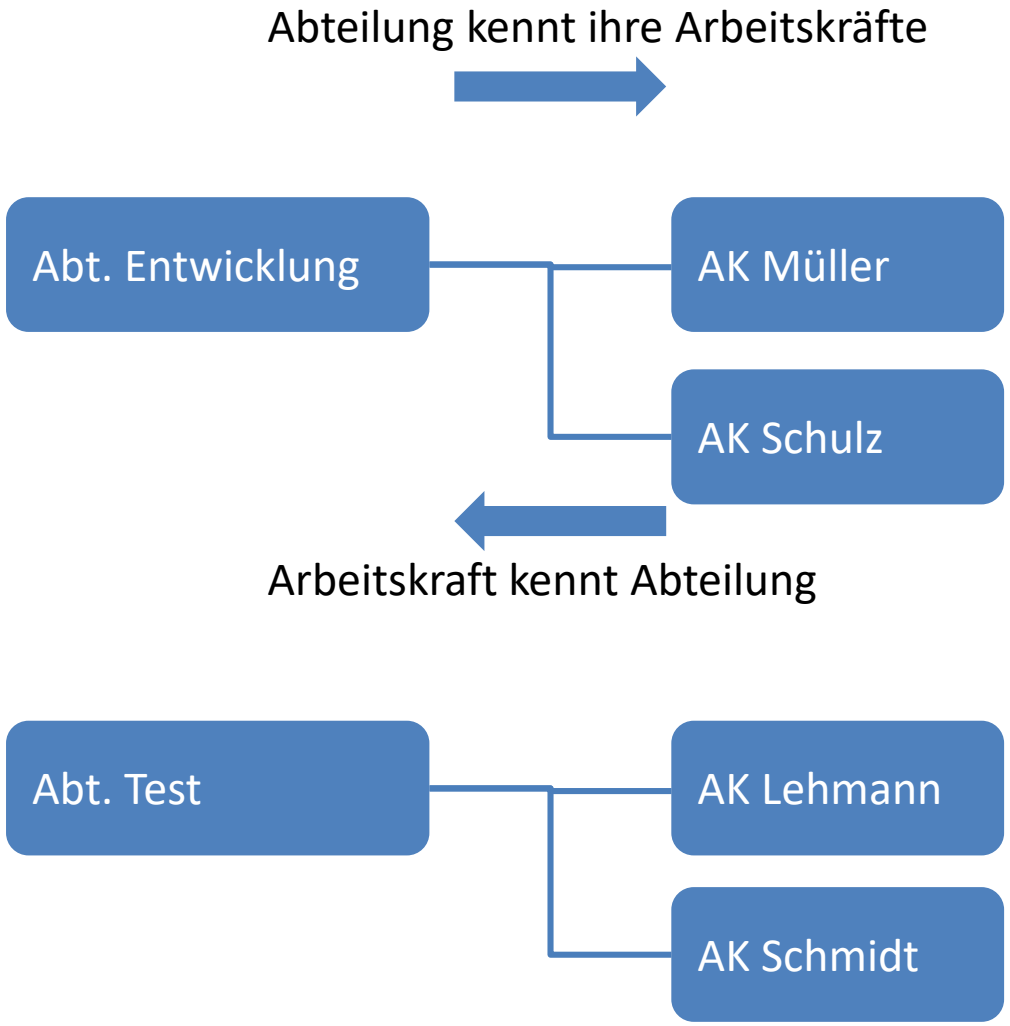
Erweiterung Arbeitskräfteverwaltung /1

Neue Anforderungen

- **Abteilung** soll als Objekt abgebildet werden
- Liste von Arbeitskräften innerhalb einer Abteilung
- Arbeitskraft hinzufügen
- Arbeitskraft anhand von Nr. entfernen
- Arbeitskräfte in Abteilung ausgeben

Lösungsansatz

- Abteilung als Klasse anlegen
- ArrayList für Arbeitskräfte in Abteilung einbetten
- Attribut für Abteilung in Arbeitskraft hinzufügen
- Methode hinzufuegen() für das Hinzufügen
- Methode loeschen() für das Löschen
- Methode für Ausgeben



demo002.personalverwaltung

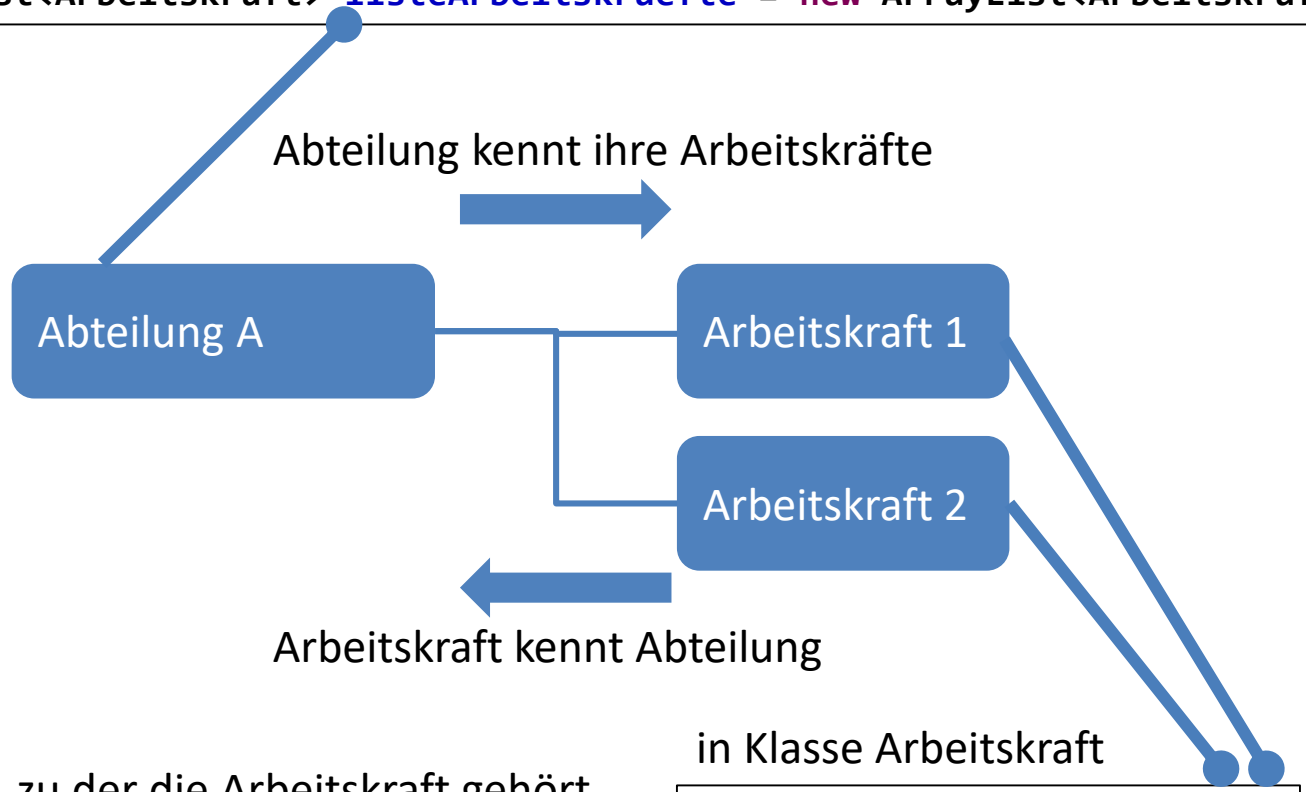
Erweiterung Arbeitskräfteverwaltung /2

demo002.personalverwaltung

Arbeitskräfte speichern, die zu einer Abteilung gehören.
Neue Arbeitskraft über Methode **hinzufuegen()** speichern.

in Klasse Abteilung

```
private ArrayList<Arbeitskraft> listeArbeitskraefte = new ArrayList<Arbeitskraft>();
```



Referenz auf Abteilung speichern, zu der die Arbeitskraft gehört.
Abteilung über Methode **setAbteilung()** speichern.

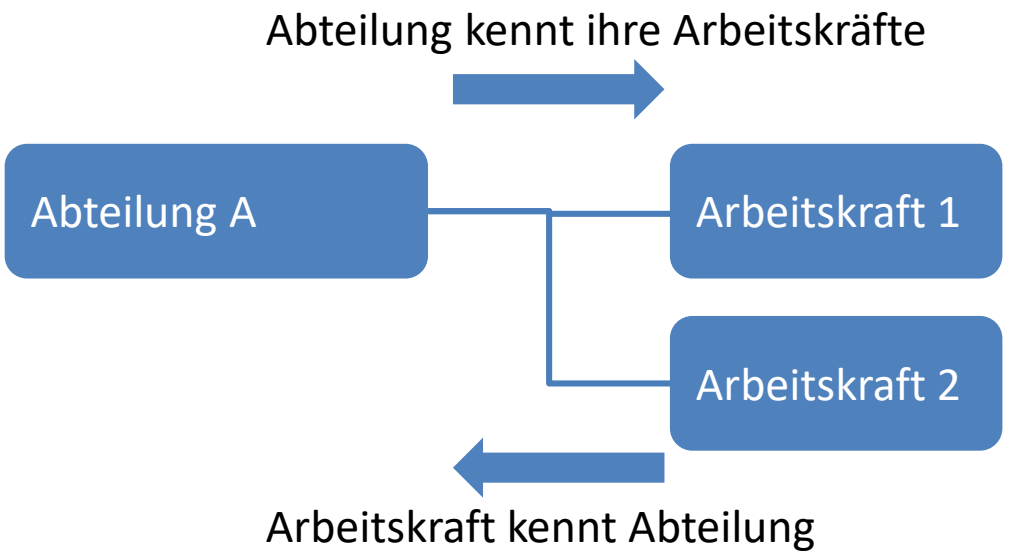
in Klasse Arbeitskraft

```
private Abteilung abteilung;
```

Erweiterung Arbeitskräfteverwaltung /3

demo002.personalverwaltung

Mit einer Datenstruktur können mehrere Arbeitskräfte zu der Abteilung in Beziehung gesetzt werden.



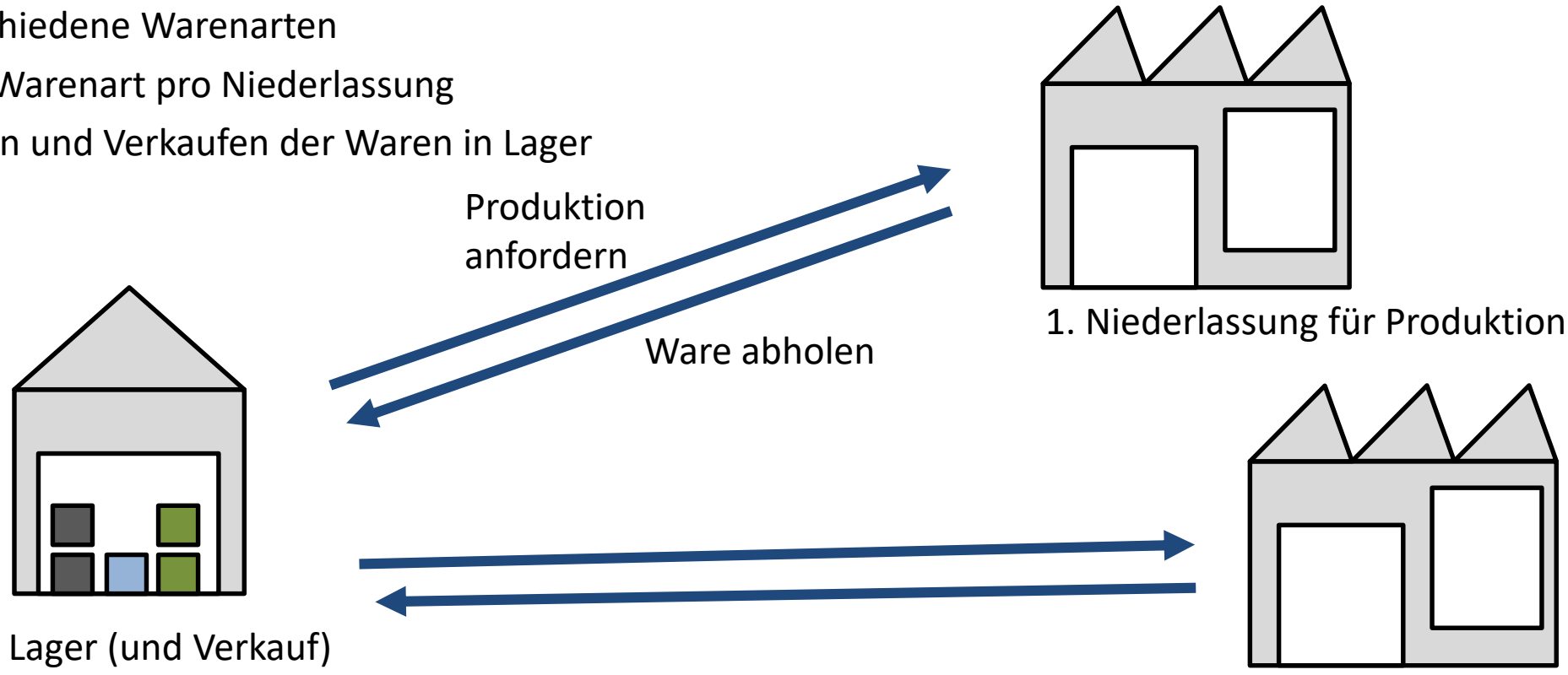
Über eine Referenz kann die jeweilige Abteilung in Beziehung gesetzt werden.

BEISPIEL WIRTSCHAFTSSIMULATION

Wirtschaftssimulation

- Spielorganisation in Durchläufen
- Waren werden
  - produziert
  - transportiert
  - verkauft
- Verschiedene Warenarten
- eine Warenart pro Niederlassung
- Lagern und Verkaufen der Waren in Lager

Hinweis: Diese Wirtschaftssimulation ist eine sehr vereinfachte Darstellung der tatsächlichen Abläufe. Es soll die einfache Arbeit mit Klassen und Objekten gezeigt werden. Viele realistische Aspekte fehlen noch!



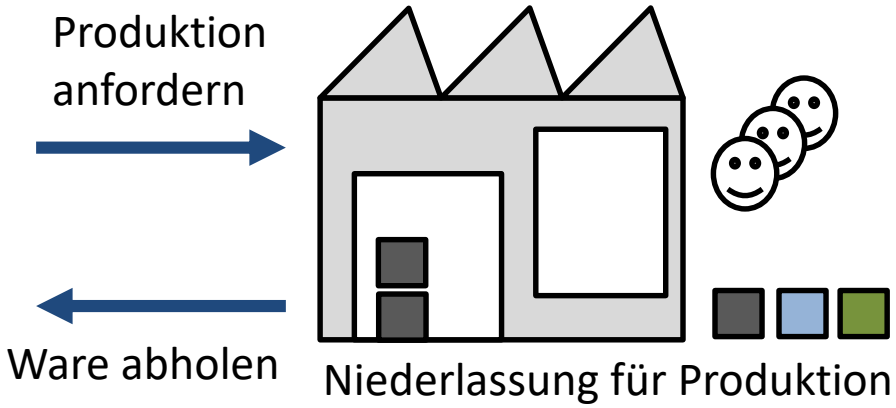
Niederlassung

Eigenschaften einer Niederlassung

- Ort der Niederlassung
- Anzahl der Arbeitenden
- Warenart, die produziert wird
- zuletzt produzierte Menge

Verhalten einer Niederlassung

- Anfordern der Produktion
- Produktion und Abholen im anschließenden Durchlauf
- Ausgabe der produzierten Waren (zufällige Schwankung)
- Arbeitende einstellen/entlassen



Niederlassung

Eigenschaften einer Niederlassung

- Ort der Niederlassung
- Anzahl der Arbeitenden
- Warenart, die produziert wird
- zuletzt produzierte Menge

Verhalten einer Niederlassung

- Anfordern der Produktion
- Produktion und Abholen im anschließenden Durchlauf
- Ausgabe der produzierten Waren (zufällige Schwankung)
- Arbeitende einstellen/entlassen

```
public class Niederlassung {
     private String ort;
     private Warenart warenartProduktion;
     private int arbeitende;
     private boolean produktionAngefordert;
     private int produzierteMenge=0;
     private int faktorArbeiterZuMenge=10;

     public Niederlassung(String ort, Warenart warenart, int arbeitende) {
          ...
     }
     public void anfordern() {
          ...
     }
     public void produzieren() {
          ...
     }
     public int abholen() {
          ...
     }
     public void einstellen(int anzahlEinzustellende) {
          ...
     }
     public void entlassen(int anzahlZuEntlassende) {
          ...
     }
}
```

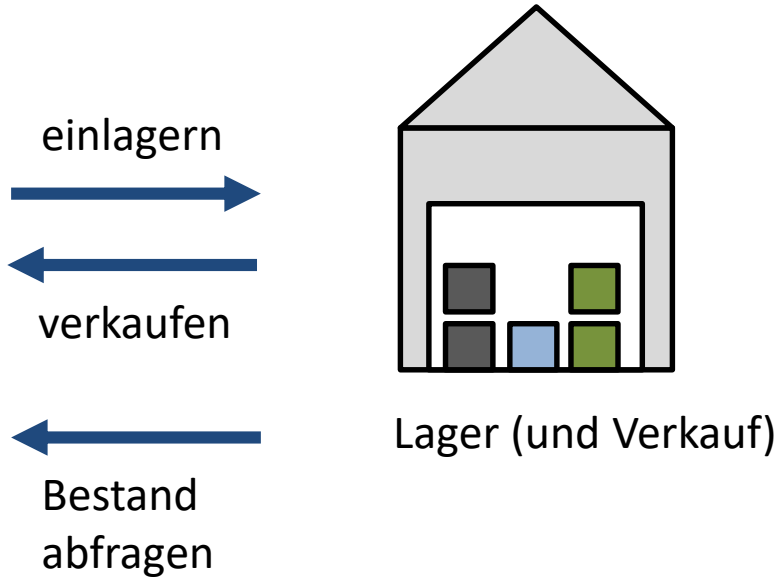
Lager

Eigenschaften eines Lagers

- Ort des Lagers
- Lagerbestand pro Warenart

Verhalten eines Lagers

- einlagern einer Warenart mit einer Menge
- verkaufen einer Menge von einer bestimmten Warenart
- Lagerbestand abfragen



Lager

Eigenschaften eines Lagers

- Ort des Lagers
- Lagerbestand pro Warenart

Verhalten eines Lagers

- einlagern einer Warenart mit einer Menge
- verkaufen einer Menge von einer bestimmten Warenart
- Lagerbestand abfragen

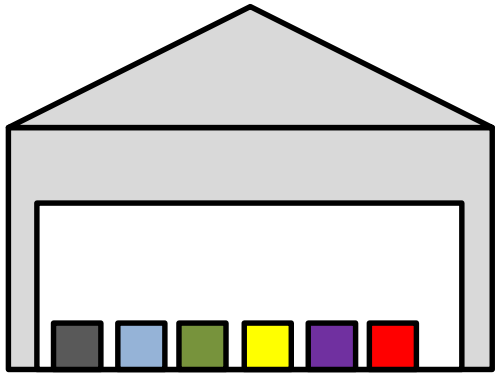
```
public class Lager {
     private String ort;
     private HashMap<Warenart, Integer> lagerbestand = new HashMap<Warenart, Integer>();
     public Lager(String ort) {
          ...
     }
     public void einlagern(Warenart warenart, int mengeEinlagern) {
          ...
     }
     public int verkaufen(Warenart warenart, int mengeVerkaufen, int preis) {
          ...
     }
     public Set<Warenart> getEingelagerteWaren() {
          return lagerbestand.keySet();
     }
     public int getBestand(Warenart warenart) {
          ...
     }

}
```

Warenart

Verschiedene Warenarten

- Enum als Datenstruktur
- ermöglicht Auswahl einer Warenart aus einer Menge von möglichen Waren
- Verwendung wie enum in C



```
public enum Warenart {
      BIER,
      WEIN,
      KORN,
      GLAS,
      TUCH,
      GOLD
}
```

EXKURS DATENSTRUKTUR HASHMAP

Konzept der HashMap

Grundprinzip

- Klasse für Speichern von Schlüssel-Wert-Paaren
- Methoden für Zugriff auf Schlüssel-Wert-Paare
  - Einfügen
  - Löschen
  - Auslesen
  - Verändern
- Datentyp für Schlüssel und Wert festlegen

Schlüssel (Integer)	Wert (Name)
1	Müller
3	Schulze
23	Lehmann
44	Schmidtzke

Methoden für Zugriff (Auszug)

- put(Schlüssel, Wert) – zu einem bestimmen Schlüssel einen Wert ablegen
- get(Schlüssel) – den abgelegten Wert zu einem bestimmten Schlüssel aus der HashMap holen
- remove(Schlüssel) – den Schlüssel und den zugehörigen Wert löschen
- clear() – gesamte HashMap mit allen Schlüssel-Wert-Paaren löschen

HashMap Beispiel

```
import java.util.HashMap;

public class HashMapMain {
     public static void main(String[] args) {
          // Objekt der Klasse HashMap erzeugen
          HashMap<String, String> landHauptstadt = new HashMap<String, String>();

          // Schluessel-Wert-Paare hinzufuegen
          landHauptstadt.put("England", "London");
          landHauptstadt.put("Deutschland", "Berlin");
          landHauptstadt.put("Norwegen", "Oslo");
          landHauptstadt.put("USA", "Washington DC");
          landHauptstadt.put("Frankreich", "Paris");
          // Schluessel-Wert-Paare ausgeben
          System.out.println(landHauptstadt);
     }
}
```

Ausgabe auf Kommandozeile:

```
{USA=Washington DC, Deutschland=Berlin, Frankreich=Paris,
Norwegen=Oslo, England=London}
```

Schlüssel (Land)	Wert (Hauptstadt)
England	London
Deutschland	Berlin
Norwegen	Oslo
USA	Washington DC
Frankreich	Paris

HashMap im Lager

```
private HashMap<Warenart, Integer> lagerbestand = new HashMap<Warenart, Integer>();
```

```
public enum Warenart {
    BIER,
    WEIN,
    KORN,
    GLAS,
    TUCH,
    GOLD
}
```

Beispiel eines möglichen Bestandes:

Schlüssel (Warenart)	Wert (Integer)
Wein	5
Korn	13
Tuch	18
Gold	11

Hinweis: Hier im Beispiel sind Bier und Glas nicht vorhanden, das heißt, es ist keine Ware dieser Warenart im Lager. In der Programmlogik muss entsprechend mit fehlenden Einträgen umgegangen werden.

ANWENDUNG HASHMAP IM LAGER

HashMap beim Einlagern in Lager

Einlagern

- falls schon Bestand der Warenart vorhanden
  - Einzulagernde Menge hinzufügen
- Falls noch nicht im Bestand
  - Schlüssel-Wert-Paar hinzufügen

```
public void einlagern(Warenart warenart, int mengeEinlagern) {
    if(lagerbestand.containsKey(warenart)) {
        int bisherigerBestand = lagerbestand.get(warenart).intValue();
        int neuerBestand = bisherigerBestand+mengeEinlagern;
        lagerbestand.put(warenart, neuerBestand);
    } else {
        lagerbestand.put(warenart, mengeEinlagern);
    }
}
```

```
einlagern(KORN, 3)
```

Korn ist schon im Bestand vorhanden. Der bisherige Bestand von 13 wird um 3 auf 16 erhöht.

Schlüssel (Warenart)	Wert (Integer)
Wein	5
Korn	**13**
Tuch	18
Gold	11

Schlüssel (Warenart)	Wert (Integer)
Wein	5
Korn	**16**
Tuch	18
Gold	11

HashMap beim Einlagern in Lager

Einlagern

- falls schon Bestand der Warenart vorhanden
  - Einzulagernde Menge hinzufügen
- Falls noch nicht im Bestand
  - Schlüssel-Wert-Paar hinzufügen

```
public void einlagern(Warenart warenart, int mengeEinlagern) {
    if(lagerbestand.containsKey(warenart)) {
        int bisherigerBestand = lagerbestand.get(warenart).intValue();
        int neuerBestand = bisherigerBestand+mengeEinlagern;
        lagerbestand.put(warenart, neuerBestand);
    } else {
        lagerbestand.put(warenart, mengeEinlagern);
    }
}
```

```
einlagern(GLAS, 10)
```

Glas ist noch nicht im Bestand vorhanden. Der Bestand von Glas wird eingefügt und auf 10 gesetzt.

Schlüssel (Warenart)	Wert (Integer)
Wein	5
Korn	13
Tuch	18
Gold	11

Schlüssel (Warenart)	Wert (Integer)
Wein	5
Korn	16
Tuch	18
Gold	11
Glas	**10**

Zusammenfassung

- Klasse ist Bauplan
- Objekt ist konkrete Realisierung
- Attribute beschreiben den Zustand
- Methoden erlauben den Zugriff auf das Objekt
- Datenstruktur ArrayList als Feld für das Speichern von Objekten
- Datenstruktur HashMap für Schlüssel-Wert-Paare
- Beispiel Personalverwaltung
- Beispiel Wirtschaftssimulation