

Rechtsbelehrung


Dieser Foliensatz ist urheberrechtlich geschützt. Änderungen an den Folien sind untersagt. Ausschließlich eine nicht-kommerzielle Nutzung ist kostenfrei. Andernfalls wird eine Gebühr fällig. Wenden Sie sich hierfür an den Autor.

Kapitel 4: Erweiterte Konzepte in Java


Beispiel

```
public class Safe {  
  
    private Object secretObject;  
  
    public Safe(Object secret) {  
        secretObject = secret;  
    }  
  
    public Object getSecret(String password) {  
        if (password.equals("sesam"))  
            return secretObject;  
        else  
            return null;  
    }  
}
```

Speichert ein
Objekt eines
beliebigen Typs.



Anhand des korrekten
Passworts kann das
Objekt wieder
ausgelesen werden.

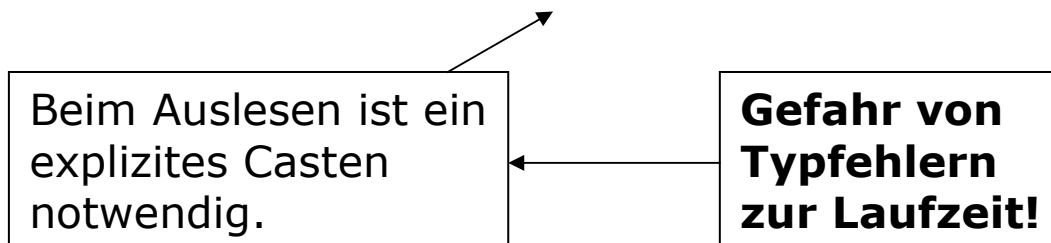


Anwendung

- Beispielsweise wird ein Rechteck übergeben...

```
Rechteck in = new Rechteck(10, 10, 50, 80);  
Safe safe = new Safe(in);
```
- Und schließlich wieder ausgelesen...

```
Rechteck out = (Rechteck)safe.getSecret("sesam");
```



- Typsichere Alternativen:
 - Spezielle Klasse `Safe` je Typ
 - **Generics**

Generics

- Parametrisierte Klasse:

```
public class Safe<T> {  
    private T secretObject;  
  
    public Safe(T secret) {  
        secretObject = secret;  
    }  
    public T getSecret(String password) {  
        if (password.equals("sesam")) return secretObject;  
        else return null;  
    }  
}
```

T ist ein generischer Typ

Konkretisierung
von T

- Anwendung:

```
Rechteck in = new Rechteck(10, 10, 50, 80);  
Safe<Rechteck> safe = new Safe<Rechteck>(in);  
...  
Rechteck out = safe.getSecret("sesam");
```

Kein Cast-
Operator
notwendig

Generics allgemein

- ❑ Generics bieten die Möglichkeit **typsichere Container-Klassen** zu programmieren.
- ❑ Container-Klassen speichern Objekte beliebigen Typs und sind dabei nicht an deren konkreten Typ interessiert, d.h. sie speichern Objekte als Ganzes und wenden nicht z.B. deren Methoden an.
- ❑ **Collections** bzw. **Sammlungen** sind spezielle Container-Klassen, die **mehrere** Objekte **gleichen** Typs verwalten (z.B. Listen und Mengen).

Die Klasse Gruppe

```
public class Gruppe extends Flaechе {  
  
    private ArrayList<Flaechе> flaechenListe;  
  
    public Gruppe(int x, int y) {  
        super(x, y);  
        flaechenListe = new ArrayList<Flaechе>();  
    }  
    public void add(Flaechе flaechе) {  
        flaechenListe.add(f);  
    }  
    public void remove(Flaechе f) {  
        flaechenListe.remove(f);  
    }  
    public void paint(Graphics g) {  
        // todo, Flaechen der Gruppe zeichnen  
    }  
}
```

Speichert eine
Liste von Flaechе-
Objekten

Flaechе der Liste
hinzufügen

Flaechе aus der
Liste löschen

Kommt noch...

Collections in Java

Grundtypen:

- Listen
 - Basisinterface: `List`
 - Geordnet
 - Duplikate erlaubt
- Mengen
 - Basisinterface: `Set`
 - I.d.R. ungeordnet
 - Duplikate nicht erlaubt
- Zuordnungen
 - Basisinterface: `Map`
 - Schlüssel-Werte-Paare
 - Schlüssel sind wie Mengen

Die Klasse ArrayList

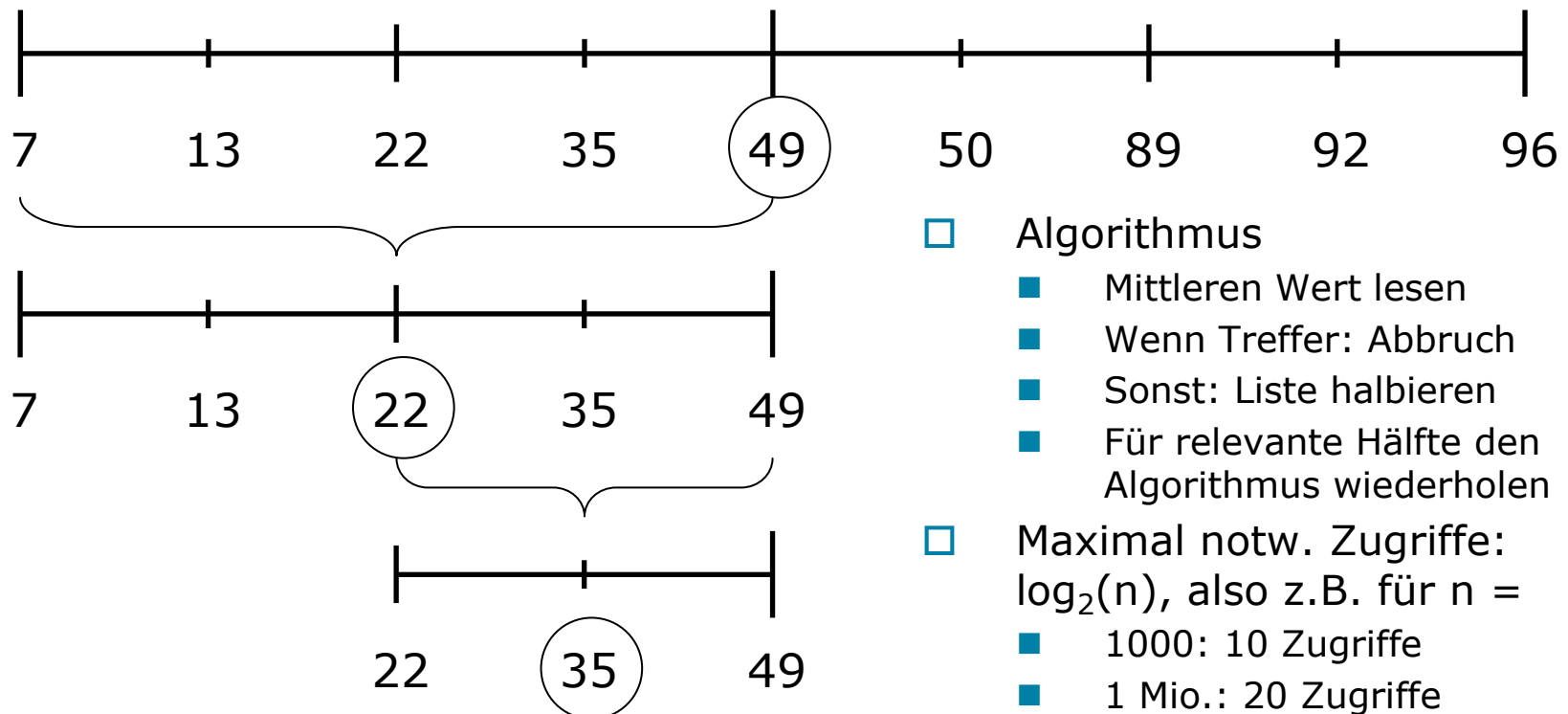
<code>boolean add(E e)</code>	Hängt das Element <code>e</code> am Ende der Liste an.
<code>void add(int index, E e)</code>	Fügt das Element <code>e</code> an einer bestimmten Position ein.
<code>void clear()</code>	Entfernt alle Element aus der Liste.
<code>Object clone()</code>	Gibt eine Kopie der Instanz zurück.
<code>boolean contains(Object o)</code>	Prüft, ob ein bestimmtes Element in der Liste enthalten ist.
<code>E get(int index)</code>	Gibt das Element an einer bestimmten Position zurück.
<code>int indexOf(Object o)</code>	Gibt den Index eines Elements zurück. Falls <code>o</code> mehrmals in der Liste enthalten ist, wird der erste Index zurückgegeben. Fall <code>o</code> nicht in der Liste enthalten ist, wird <code>-1</code> zurückgegeben.
<code>E remove(int index)</code>	Entfernt ein Element an einer bestimmten Position.
<code>boolean remove(Object o)</code>	Entfernt ein gegebenes Element aus der Liste. Falls <code>o</code> mehrmals in der Liste enthalten ist, wird nur die erste Instanz gelöscht. Falls <code>o</code> nicht in der Liste enthalten ist, wird <code>false</code> zurückgegeben, sonst <code>true</code> .
<code>E set(int index, E e)</code>	Ersetzt ein Element an einer bestimmten Position durch ein neues Element <code>e</code> . Gibt das alte Element zurück.
<code>int size()</code>	Gibt die Anzahl der Listenelemente zurück.

Zuordnungen

- Vergleichbar mit Telefonbuch:
 - Name -> Telefonnummer
 - (Key -> Value)
- Die Suche im Telefonbuch geht schnell, weil die Namen sortiert sind.
- Automatisiert ist die Suche in *sortierten* Listen z.B. mittels **Bisektion** möglich.

Bisektions- bzw. Binäre Suche

□ Gesucht: 35



□ Algorithmus

- Mittleren Wert lesen
- Wenn Treffer: Abbruch
- Sonst: Liste halbieren
- Für relevante Hälfte den Algorithmus wiederholen

□ Maximal notw. Zugriffe: $\log_2(n)$, also z.B. für $n =$

- 1000: 10 Zugriffe
- 1 Mio.: 20 Zugriffe
- 1 Mrd.: 30 Zugriffe

Hashtabelle

- ❑ ... stellt eine Map dar.
- ❑ ... wendet **keine** Bisektion an.
- ❑ ... benötigt keine sortierte Liste.
- ❑ Der Zugriff ist schneller als bei der binären Suche!
- ❑ Trick: die Position des Schlüssels wird *berechnet!*
- ❑ Die Klasse `HashMap` implementiert eine Hashtabelle.

Anwendung

```
public static void main(String[] args) {  
  
    HashMap<String, Kreis> map  
        = new HashMap<String, Kreis>();  
  
    map.put("A", new Kreis(0, 0, 10));  
    map.put("B", new Kreis(0, 0, 20));  
    map.put("C", new Kreis(0, 0, 30));  
  
    System.out.println("B mit dem Radius " +  
        map.get("B").getRadius() +  
        " wird jetzt gleich gelöscht!");  
  
    map.remove("B");  
    System.out.println(map.get("B")); ← Gibt null aus.  
}
```

Funktionsweise von HashMap

- Die Key-Klasse implementiert eine Methode `int hashCode()`.
- Für *gleiche* Instanzen muss `hashCode()` denselben Wert zurückgeben.
- Unterschiedliche Instanzen haben i.d.R. unterschiedliche Hashcodes. Das kann aber nicht immer gewährleistet werden. Sie können also auch gleich sein.
- Ob zwei Instanzen *gleich* sind, bestimmt die `boolean equals()`-Methode. Auch sie wird von der Key-Klasse implementiert.

Funktionsweise von HashMap (2)

- ❑ HashMap hat intern einen Array vom Typ `Entry`, mit den Attributen `key`, `value`, `next` und `hash`.
- ❑ Beim Setzen eines Eintrags wird der Hashcode des Keys ausgelesen und dann so modifiziert, dass sein Wert unter dem der Länge des internen Arrays liegt.
- ❑ Dadurch ist der Index im Array bestimmt.
- ❑ Falls dort bereits ein Eintrag gespeichert ist, wird er durch den neuen Eintrag ersetzt und der neue verweist dann auf den alten Eintrag (mit `next`, lineare Liste).
- ❑ Falls sich dort bereits ein *gleicher* Eintrag befindet (durch `equals()` bestimmt), wird der alte Eintrag überschrieben.

Funktionsweise von HashMap (3)

- ❑ Beim Auslesen wird der Hashcode analog berechnet.
- ❑ Bei Überschneidungen muss die lineare Liste durchlaufen werden, bis Gleichheit mit dem gesuchten Key herrscht.
- ❑ Um Überschneidungen möglichst zu verhindern, wird der interne Array nur bis zu einem bestimmten Prozentsatz gefüllt, dem sog. `LoadFactor`. Dieser ist standardmäßig 75%. Ist er überschritten, wird der Array in einen doppelt so großer Array umkopiert.
- ❑ `hashCode()` und `equals()` besitzen eine Standardimplementierung in `Object`. Der `hashCode()` ist dabei die Adresse der Instanz, `equals()` vergleicht die Adressen.
- ❑ Beachten Sie: gleiche Instanzen müssen dieselben Hashcodes erzeugen. I.d.R. berücksichtigen `equals()` und `hashCode()` diesselben Attribute des Keys.

Wichtige Methoden von HashMap

<code>HashMap(int initialCapacity, float loadFactor)</code>	Erzeugt eine <code>HashMap</code> mit einer definierten Ausgangskapazität und einem definierten Ladefaktor.
<code>HashMap()</code>	Erzeugt eine <code>HashMap</code> mit einer Kapazität von 16 und einem Ladefaktor von 0.75.
<code>void clear()</code>	Entfernt alle Einträge aus der Map.
<code>Object clone()</code>	Gibt eine Kopie der Instanz zurück.
<code>boolean containsKey(Object key)</code>	Prüft, ob ein bestimmter Schlüssel in der Map enthalten ist.
<code>V get(Object key)</code>	Gibt den Wert zu einem bestimmten Schlüssel zurück.
<code>V put(K key, V value)</code>	Setzt ein Schlüssel-Wert-Paar. Gibt den bisherigen Wert zu diesem Schlüssel zurück. Falls der Schlüssel noch nicht gesetzt war, wird <code>null</code> zurückgegeben.
<code>V remove(Object key)</code>	Entfernt ein Element mit einem bestimmten Schlüssel.
<code>int size()</code>	Gibt die Anzahl der Einträge zurück.

Erweiterte for-Schleife

- ... zum Durchlaufen von Collections
- Die fehlende `paint()`-Methode von Gruppe:

```
public void paint(Graphics g) {  
    for (Flaeche f : flaechenListe)  
        f.paint(g);  
}
```

- Aber die Elemente sollten relativ zur Gruppe positioniert werden!

Der zweite Versuch...

- Mit `translate()` lässt sich der Ursprung verschieben.

- ```
public void paint(Graphics g) {

 g.translate(getX(), getY());

 for (Flaeche f : flaechenListe)
 f.paint(g);

 g.translate(-getX(), -getY());
}
```

...

- 
- Die (bisherige) `add()`-Methode in Gruppe:

```
public void add(Flaeche flaeche) {
 flaechenListe.add(f);
}
```

- Es kann immer nur eine Flaeche hinzugefügt werden...

- Die neue `add()`-Methode:

```
public void add(Flaeche... flaechen) {
 for (Flaeche f : flaechen)
 flaechenListe.add(f);
}
```

## ... (2)

---

- Der ...-Operator erlaubt eine variable Anzahl von Methodenargumenten.
- Anwendung:

```
Rechteck r = new Rechteck(0,0,190,100);
Kreis k1 = new Kreis(10,10,40);
Kreis k2 = new Kreis(10,100,40);
Gruppe g = new Gruppe(50,50);
g.add(r, k1, k2);
```
- Der entsprechende Parameter verhält sich wie ein Array.
- Er kann mit anderen Parametern kombiniert werden, muss dann aber der letzte in der Parameterliste sein.

```
public class Gruppe extends Flaechе {

 private ArrayList<Flaechе> flaechenListe;

 public Gruppe(int x, int y) {
 super (x, y);
 flaechenListe = new ArrayList<Flaechе>();
 }
 public void add(Flaechе... flaechen) {
 for (Flaechе f : flaechen)
 flaechenListe.add(f);
 }
 public void remove(Flaechе f) {
 flaechenListe.remove(f);
 }
 public void paint(Graphics g) {
 g.translate(getX(), getY());
 for (Flaechе f : flaechenListe) f.paint(g);
 g.translate(-getX(), -getY());
 }
}
```