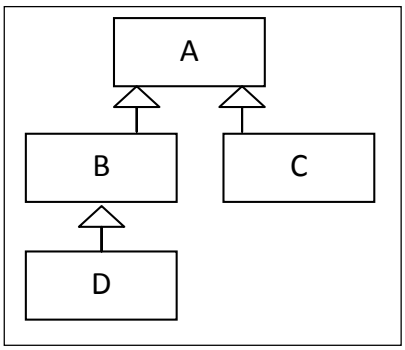
Übung

In dieser Übung sollen die Konzepte der Theorie kurz und knapp überprüft werden. Dazu brauchen wir als Ausgangslage eine Vererbungsstruktur und eine TestKlasse.

1. Vorbereitendes

U1: Schreiben Sie eine kleine Vererbungsstruktur, wie sie rechts aufgezeichnet ist. Die Klassen brauchen keine Methoden oder Attribute zu enthalten. Sie können aber auch eine Vererbungsstruktur verwenden, welche Sie schon an einem anderen Ort programmiert haben.



U2: Nun brauchen wir noch eine Klasse, welche die generischen Klassen testen kann. Schreiben Sie also eine Klasse TestGenerics im gleichen Paket wie die Klassen A, B, C und D.

2. generische Klasse Box

U3: Speichern Sie die folgende generische Klasse im gleichen Paket:

```
class Box<T>{
   private T val;

   public void setValue(T val){
      this.val = val;
   }

   public T getValue(){
      return val;
   }
}
```

Aus : http://sws.bfh.ch/~amrhein/Swing/javainsel7/javainsel_06_012.htm

2.1 Verwenden der Klasse Box

Nun wollen wir diese Klasse in der TestKlasse verwenden. Als erstes wollen wir ein Objekt erzeugen, welches, wie die Collection der letzten Aufgabe, nicht typisiert wurde. Anschliessend erstellen wir für alle Klasse der Vererbungsstruktur ein Box-Objekt. -> Box box = new Box();

U4: Erstellen Sie ein Box-Objekt und speichern Sie es in der Referenz box.

U5: Erstellen Sie ein Box-Objekt für jede Klasse der Vererbungsstruktur (aBox, bBox, cBox, dBox).

```
Box<A> aBox = new Box<A>();
```

Nun wollen wir testen, mit welchen Objekten wir die setValue-Methode aufrufen können.

U6: Erzeugen Sie dazu als erstes vier Objekte (a1, b1, c1, d1), für jede Klasse der Vererbungsstruktur.

A a1= new A();

U7: Überlegen Sie sich, welche der folgenden aufrufe von setValue möglich ist, und welche zu einem Fehler führt. Testen Sie Ihr Ergebnis ansschliessend durch Implementation.

x := "diese Implementation funktioniert"

x	aBox.setValue(a1);		bBox.setValue(a1);
x	aBox.setValue(b1);	x	bBox.setValue(b1);
x	aBox.setValue(c1);		bBox.setValue(c1);
x	aBox.setValue(d1);	x	bBox.setValue(d1);
	cBox.setValue(a1);		dBox.setValue(a1);
	cBox.setValue(b1);		dBox.setValue(b1);
x	cBox.setValue(c1);		dBox.setValue(c1);
	cBox.setValue(d1);	x	dBox.setValue(d1);

gemäss Polymorphie.

Und wie sieht es mit der Methode getValue aus ?

U8: Testen Sie die folgenden vier Zeilen. Wie können Sie sich die Fehlermeldungen erklären?

```
a1= bBox.getValue();
b1= bBox.getValue();
c1= bBox.getValue();
d1= bBox.getValue();
```

3. Zuweisungen

Nun wollen wir überprüfen, welche Zuweisungen in mit den aktuellen Referenzen box, aBox, bBox, cBox und dBox möglich sind.

U9: Testen Sie welche der folgenden Zuweisungen funktioniert:

```
box= aBox;
box= dBox;
aBox = bBox;
dBox= cBox;
```

U10: Erklären Sie, welche Zuweisungen funktionieren, und welche nicht. Lesen Sie dazu eventuell auch in der Theorie oder im Internet nach.

3.1 Wildcards

Nun wollen wir, dass alle Zuweisungen aus Aufgabe U6 möglich werden. Dazu werden die Wildcards verwendet.

U11: Ändern Sie die Deklaration von bBox so um, dass nun steht:

```
Box<?> bBox= new Box<B>();
```

U12: Testen Sie nun die folgenden Zuweisungen:

```
bBox = aBox;
bBox = cBox;
bBox = dBox;
```

Das funktioniert nun, weil Box<?> jedes typisierte Objkt von Box speichern kann. Wir möchten aber, dass dennoch die Vererbungsstruktur geprüft wird. Das heisst, dass wir einer Box<B> nur Boxen von Unterklassen zuweisen wollen, so wie das normalerweise auch beschränkt ist.

U13: Ändern Sie die Deklaration von bBox auf:

```
Box<? extends B> bBox= new Box<B>();
```

Nun wurde dasselbe Verhalten erreicht, wie wir es von Referenzen für nicht generische Objekte gewohnt sind.
Die Wildcards funktionieren aber auch in die andere Richtung. Das heisst, dass angegeben werden kann das mit Oberklassen typisierte Objekte zugewiesen werden können.

U14: Ändern Sie die Deklarationen am Anfang wie folgt ab:

```
Box<A> aBox= new Box<A>();
Box<B> bBox= new Box<B>();
Box<? super C> cBox= new Box<C>();
Box<D> dBox= new Box<D>();
```

U15: Überlegen Sie sich, welche der folgenden Zuweisungen nun nicht mehr erlaubt ist.

```
cBox= aBox;
cBox= bBox;
cBox= dBox;
```

ACHTUNG: Es kann kein Objekt mit dem Wildcard-Symbol erzeugt werden. Nur Referenzen können dieses Symbol verwenden.

U16: Testen Sie die folgende Zeile. Eclipse sollte Ihnen sagen, dass das nicht geht.

```
Box<D> dBox= new Box<?>();
```

4. generische Klassen und Methoden

Alle bisherigen Übungen behandelten ausschliesslich das Thema, wie generische Klassen von anderen Klassen verwendet werden können. Jetzt wollen wir den Fokus auf die generische Klasse Box legen.

4.1 mehrere Typen

Hier wollen wir ein kurzes Beispiel anschauen, welches mit mehreren Typen in einer generischen Klasse arbeitet:

U17: Ergänzen Sie den Klassennamen von Box wie folgt:

```
class Box<T , Q> { . . .
```

U18: Fügen Sie nun eine Variable key vom Typ Q zur Klasse hinzu und schreiben Sie auch gleich noch Getter und Setter für die Variable key.

4.2 Constraints für den Typ

Constraints können nicht nur bei den Referenzen auf generische Objekte gemacht werden (bei den Wildcards). Man kann auch den Typ der Klasse beschränken. Falls man z.B. in einer Methode unbedingt eine Methode des Typs verwenden möchte, braucht man eine zusätzliche Absicherung.

Angenommen Sie möchten eine Klasse haben, welche 5 Elemente Kapselt.

U19: Schreiben Sie die folgende Klasse:

```
public class ArrayBox <T>{
   private A[] myArray = new A[5];

   public void add(T val){
      myArray[0] = val; //Problem: Erwartet Typ A, findet aber Typ T vor. (ein cast ist nicht die optimale
   }                    Lösung)
}
```

Dieser Code führt zu einem Fehler, da nicht sicher ist, ob ein T Objekt in einem Array für A-Objekte gespeichert werden kann. Eclipse schlägt unter anderem vor, den Parameter val zu casten, aber das führt eher noch zu mehr Inkonsistenzen zur Laufzeit.

U20: Ändern Sie den Typ auf

```
<T extends A>
```

4.3 generische Methoden

Wir möchten in den Klasse Box prüfen, ob der val, bzw. key Parameter nicht null ist, bevor wir ihn abspeichern. Dazu schreiben wir eine Methode, welche auch in einer nicht generieschen Klasse stehen könnte.

U21: Schreiben Sie die Methode isNull:

```
public <P> boolean isNull(P val){
   return (val==null);
}
```

U22: Verwenden Sie diese Methode in den Methoden setValue und setKey.