ADT's:

10 Grundlegende Datentypen

10.1 Spezifikation von Datentypen

Abstrakter Datentyp (ADT):
Festlegung der auf den Typ anwendbaren Operationen (Schnittstelle)
Festlegung der Wirkung der Operationen

Man legt fest:
- adt: Namen des ADTs
- sorts: Liste verwendeter Datentypen
- ops: Schnittstellen der Operationen: Name, Parameter-Typen und Ergebnis-Typ
- axs: Axiome zur Beschreibung der Semantik: so allgemein wie möglich, so ausführlich wie nötig

Dies geschieht in der Signatur (Bsp.):

adt	IntSet			
sorts	IntSet, Int, Boolean			
ops	create:			→ IntSet
	insert:	IntSet × Int		→ IntSet
	delete:	IntSet × Int		→ IntSet
	contains:	IntSet × Int		→ Boolean
	isEmpty:	IntSet		→ Boolean
axs	isEmpty(create)	= true		
	isEmpty(insert(x,i))	=false		
	insert(insert (x,i),i)	= insert(x,i)		
	contains(create, i)	= false		
	contains(insert(x,j),i)	= true	falls i=j	
		contains (x,i)	sonst	
end	IntSet			

Vorteile:
Man kann Datentyp genau soweit festlegen, wie gewünscht, ohne Implementierungsdetails vorwegzunehmen. Die Sprache, in der Axiome beschrieben werden, folgt formalen Regeln. Daher sind Entwurfswerkzeuge konstruierbar, die Spezifikation prüfen oder Prototyp erzeugen können.

Nachteile:
Bei komplexen Anwendungen wird die Zahl der Gesetze sehr groß. Es ist nicht immer leicht, anhand der Gesetze die intuitive Bedeutung eines Datentyps zu erkennen. Insbesondere ist es schwierig zu prüfen, ob die Menge der Gesetze vollständig ist

Umsetzung in Java-Code:
ops: Schnittstellen der Java-Methoden
axs: Umsetzung der Java-Methoden

Konstruktoren:	mit create und insert lassen sich alle mögl. Objekte erzeugen. Zwingend erforderlich.
Hilfskonstrukt.:	erzeugen auch Datenobjekte des Typs, aber nicht zwingend erforderlich.
Normalform:	konstruiert durch minimale Zahl von Konstruktoraufrufen.

copied from, AUD_Zusammenfassung_fsi_Forum

Erweiterter ADT *FunList*

adt FunList
sorts FunList, Pred, T
ops

nil:		→ FunList
cons:	T × FunList	→ FunList
head:	FunList	→ T
tail:	FunList	→ FunList

axs

head(nil())	= null
head(cons(t, l))	= t
tail(nil())	= nil()
tail(cons(t, l))	= l

end FunList

Übersetzung in Java-code:

```
ops: // beschreiben nur Methoden-Parameter und Rückgabewerte
    public FunList nil() //public ist nicht gegeben, nur zur Vollständigkeit
    public FunList cons(T t, Funlist list)
    public T head(FunList list)
    public FunList tail(Funlist list)
axs: //Wie genau das innerhalb der Methode aussieht ist egal, es wird nur
     definiert, was die Methode machen soll, nicht wie.
    public T head(FunList list){
     if(list == nil()) return null;
     return list.irgendwas;
    }

    public FunList tail(Funlist list){
     if(list == nil()) return nil();
     return list.irgendwas_anderes;
    }
```

Beispiel an der Klausuraufgabe SS16:
(https://www2.cs.fau.de/teaching/SS2016/AuD/organisation/oldexams/secure/16-07-28_klausur.pdf)
Die Beschreibung zur Aufgabe kann man der Klausur entnehmen.

a) Ergänzen Sie den ADT *LA* um Axiome für die Operation `getCol`:

$$getCol(New(d),\ x,\ y) = false$$

$$getCol(Step(l),\ x,\ y) = \_\_\_\_\_$$

$$falls\ \_\_\_\_\_$$

$$getCol(Step(l),\ x,\ y) = \_\_\_\_\_\ sonst$$

Um die Farbe zu bestimmen, sollte man sich erst einmal Gedanken darüber machen, wie der Step aussieht.
Aus der Aufgabenstellung:

1) Auf einem weißen Feld drehe 90 Grad nach rechts, d.h. d => (d + 3)%4 bzw.
auf einem schwarzen Feld drehe 90 Grad nach links, d.h. d => (d + 1)%4.
2) Wechsle die Farbe des Feldes (weiß nach schwarz oder schwarz nach weiß).
3) Schreite ein Feld in der aktuellen Blickrichtung fort

Man hat nun noch zu beachten, dass x und y beliebig sind.
Im 1.Fall können wir jedoch nur eine Aussage treffen, wenn x und y dem Feld entsprechen auf dem die Ameise l vor dem Step war. D.h.:

```
GetCol(...)=... ,falls (x == l.getX() && y == l.getY())
```

aber, was wissen wir? Wir wissen, wenn das Feld schwarz ist, dann ist es danach weiß, wenn es weiß ist, dann ist es dannach schwarz.
Also:

```
GetCol(Step(l), x, y ) = true, falls ((x==l.getX()&& y==l.getY() &&!getCol(l,x,y))
      //true für schwarz nur wenn x,y gleich Position von l und Feld weiß vor Step
                                    OR (getCol(l,x,y)&& (x!=l.getX() OR y!=l.getY()))
          //Oder das Feld hat die selbe Farbe wie vor dem Step und ist nicht das
                                                    selbe wie die Position von l
      //irgendwo fehlt noch:
GetCol(Step(l), x, y ) = false, falls ((x==l.getX()&& y==l.getY() && getCol(l,x,y)
```

Gedankengang:
Annahme: x=l.getX(), y=l.getY()

1. Fall:
   Das Feld x, y ist zu Beginn weiß(false), dann wird ein Step gemacht, Feld ist schwarz(true) => gib true zurück.
2. Fall:
   Das Feld x, y ist zu Beginn schwarz(true), dann wird ein Step gemacht, Feld ist weiß(false) => gib false zurück.

Die sonst-Ausgabe ist dann:

```
GetCol(Step(l), x, y ) = getCol(l, x, y)  // weil das Feld wurde durch den Step nicht verändert, und hat also die selbe Farbe wie zuvor.
```

Das ist jetzt an die Klausuraufgabe angepasst, so frei raus würde es bei mir eher so aussehen:

```
                                  { True  //if((x==l.getX()&& y==l.getY() && !getCol(l,x,y)
GetCol(Step(l), x, y ) =          { False  //if((x==l.getX()&& y==l.getY() && getCol(l,x,y)
                                  { getCol(l, x, y)  //else
```

(Dies ist meine Lösung, die ich mithilfe der Diskussion im fsi-Forum gemacht habe, also KEINE Garantie auf Richtigkeit!)