Cheatsheet Fortran OOP

Derived Types:
Ähnlich zu Structs in C. Datencontainer, aus denen aber keine echten Objekte erstellt werden können - sie haben keine Methoden, nur Attribute. Beispiel:

```
type shape
   integer :: color
   logical :: filled
   integer :: x
   integer :: y
end type shape

type(shape) :: shp1
```

Modules:
Packages, die aus einem Daten- und einem Methodenbereich bestehen. Werden mit “use <<package_name>>” importiert. Beispiel:

```
module mod_bsp
   real, private :: x = 1.2
   real :: y = 9.9

   contains

   real function addX(a)
      real, intent (in) :: a
      addX = x + a
   end function addX
end module mod_bsp
```

Klasse:
Wird aus Derived Types und Modules zusammengebaut. Die Derived Types bilden die Attribute, das Module die Methoden der Klasse ab. Die Type Bound Procedures geben im Derived Type an, welche Methoden aus dem Module genutzt werden können und somit zu Instanzmethoden werden. Diese müssen explizit angegeben werden.

```
module class_Circle
   type, public :: Circle
      real :: radius
   contains (Type Bound Procedures)
      procedure :: area => circle_area
   end type Circle
contains
   function circle_area(this) result(area)
      class(Circle), intent(in) :: this
      real :: area
      area = 3.14 * this%radius**2
   end function circle_area
end module class_Circle
```

Tipps zur OOP:

- Fortran kennt keinen Konstruktur. Um nichtsdestotrotz Redundanzen zu vermeiden, bietet es sich an, Init-Methoden zu schreiben, die ein Objekt mit Attributen belegen und intern die Konstruktoren der Superklassen aufrufen (also bekanntes Vorgehen wie aus Java, Python oder anderen Sprachen)
- Vererbung wird im Derived Type angegeben: `type, extends(shape) :: Circle`
    - Circle erbt also von shape
- Überschreiben von Methoden innerhalb einer Vererbungshierarchie geschieht, indem man denselben binding-name wählt. Der binding-name ist der Aufrufname der Methode, mit der die Methode von außen aufgerufen werden kann (als Instanzmethode). Beispiel:

  ```
  type shape
  contains
     procedure :: draw => drawShape
  end type shape
  ```

    - draw ist hier der binding-name. Man würde auf einem Objekt des Typs Shape `shp1%draw()` aufrufen, wenn das Objekt shp1 heißt
    - drawShape ist der Name der tatsächlichen Methode im Module.
- Polymorphismus: Objekt wird mit dem class-Keyword erstellt. Anschließend muss der dynamische Typ mit allocate() zugewiesen werden. Beispiel:

  ```
  class(dackel), allocatable:: irgendeinHund
  allocate(irgendeinHund)
  allocate(kampfdackel :: irgendeinHund)
  ```

    - es wird nur eins der beiden allocate Statements genutzt
        - das erste bedeutet: statischer Typ = dynamischer Typ
        - das zweite bedeutet: statischer Typ != dynamischer Typ. In dem Fall ist Kampfdackel eine Subklasse von Dackel
- Information Hiding: es gibt public und private, die nur innerhalb von Modules genutzt werden können. Folgendes kann mit Zugriffsmodifikatoren belegt werden:
    - Attribute des Derived Types
    - Type Bound Procedures
    - Module-Methoden