

Objektově orientovaný Fortran

Objektově orientované programování (OOP) je od 90. let běžně dostupné v programovacích jazycích C++, Python, Java, C# ad. a osvědčilo se při práci na rozsáhlejších projektech (př. grafická uživatelská rozhraní). Poskytuje možnost obalit existující kód (do tříd a objektů) tak, že doplnění i změny lze realizovat přidáváním nového kódu, nikoliv úpravami starého. Podpora OOP ve Fortranu byla definována standardem **Fortran 2003** z roku 2004. Pro platformu Intel/AMD realizoval použitelně OOP nejdříve překladač **NAG**, cíli se nyní (2010) blíží i překladače **gfortran** a **Intel**, dosud však ne g95, Absoft, Lahey či Portland. Některé překladače pro jiný hardware (Cray, IBM) prý OOP zahrnují již celé. Následující příklady, s několika výjimkami, jsou ověřeny překladačem gfortran 4.5.0 20100304.

Objektová terminologie (za rovnítky naznačena realizace ve Fortranu)

- **třída** (class) s **daty**-vlastnostmi-atributy (fields-attributes) a **metodami** (methods)
= odvozený typ (derived type) s položkami (components) a typově vázanými procedurami (type-bound procedures)
- **objekt**, instance objektu (object, instance)
= proměnná odvozeného typu (structure)
- **posílání zpráv** (message passing), tj. volání metod formou object%method(other-arguments)
= volání typově vázaných procedur s atributem PASS
- **zapouzdření** (encapsulation), svázání dat a metod, přístup k datům pomocí metod
= odvozený typ s procedurami, nastavení dostupnosti položek pomocí atributů PUBLIC a PRIVATE
- **dědičnost** (inheritance), tj. přejímání vlastností a metod rodičovské třídy (superclass) při definování třídy (subclass)
= rozšíření odvozeného typu (type extension)
- **polymorfismus** (polymorphism), tj. schopnost objektů dynamicky nabývat typu příbuzných tříd
= polymorfní proměnné (polymorphic variables) s dynamickým typem, překrývání procedur (procedure overriding)
- **konstruktory** a **destruktory** (constructors, destructors), tj. metody pro inicializaci a likvidaci objektu
= konstruktor struktury (structure constructor), finalizační metoda (finalizer)

Vytvoření třídy

Třídy je vhodné definovat v modulech. Do definice odvozeného typu (**TYPE ; END TYPE**) se vnoří (**CONTAINS**) seznam metod (**PROCEDURE**), odkazující na modulové procedury. Definice tak obsahuje sekci dat, obvykle zapouzdřených popisem **PRIVATE** (default je **PUBLIC**), a sekci metod, obvykle **PUBLIC** (rovněž default). Popisování formálních argumentů metod popisem **CLASS()** místo **TYPE()** umožňuje polymorfismus, konkrétně předávání skutečných argumentů příbuzného typu. Default atribut **PASS** umožňuje předat metodě první (nebo explicitně uvedený) argument formou object%f(other-arguments) místo voláním f(object,other-arguments), atribut **NOPASS** to potlačuje (např. pro metody bez argumentu).

```
př.  MODULE m
      TYPE t ; REAL x ; CONTAINS ; PROCEDURE get ; END TYPE
      TYPE t2 ; PRIVATE ; REAL :: x=0 ; CONTAINS ; PROCEDURE,PASS,PUBLIC :: get=>get2 ; END TYPE
      CONTAINS
      REAL FUNCTION get(this) ; CLASS(t) this ; get=this%x ; END FUNCTION
      FUNCTION get2(this) RESULT (r) ; CLASS(t2) this ; REAL r ; r=this%x ; END FUNCTION
      END MODULE
```

Vytvoření objektu, polymorfní proměnné

Popis proměnné odvozeného typu **TYPE()** vytvoří objekt daného typu. K inicializaci **PUBLIC** položek objektu lze užít **konstruktor struktury**, položky s default inicializací v něm lze vynechat. Popis **CLASS()** vytvoří polymorfní proměnnou odvozeného typu, jež může být pouze ukazatelem, alokovatelnou proměnnou nebo formálním argumentem. Polymorfní proměnná má kromě deklarovaného typu i **dynamický typ**, který musí být příbuzný (tj. odvozen rozšířením) deklarovaného typu a který získá ukazatel od cíle, alokovaný objekt z příkazu **ALLOCATE** a formální argument od skutečného argumentu. Podle dynamického typu lze větvit příkazem **SELECT TYPE** a lze jej testovat funkcemi **EXTENDS_TYPE_OF** a **SAME_TYPE_AS**. Polymorfní proměnná se může vyskytnout na pravé straně přiřazovacího příkazu s nepolymorfní proměnnou téhož deklarovaného typu na levé straně. Lze vytvářet neomezeně polymorfní ukazatele, **CLASS(*)**, které mohou ukazovat i na cíle standardního typu, jejich použitelnost je však značně omezena.

```
př.  TYPE(t),TARGET :: a=t(0.) ; CLASS(t),POINTER :: p
      p=>a      nebo  ALLOCATE (t :: p)      nebo  ALLOCATE (p,SOURCE=a)
      print *,EXTENDS_TYPE_OF(p,a),SAME_TYPE_AS(p,a)
př.  CLASS(*),POINTER :: up ; REAL,TARGET :: r ; up=>r
      SELECT TYPE (up) ; TYPE IS (t) ; .. ; CLASS IS (t) ; .. ; TYPE IS (REAL) ; .. ; CASE DEFAULT ; .. ; END SELECT
```

Dědičnost, překrývání metod, abstraktní typy

Od odvozeného typu lze jeho rozšířením (**EXTENDS**) odvozovat příbuzné typy přidáváním položek i metod a **překrývání metod**. Zděděné položky mohou být souhrnně adresovány jménem rodičovského typu. Překrývající metoda musí mít, s výjimkou **PASS** argumentu, totožné rozhraní jako překrytá metoda a nelze překrýt nepřekrytelné metody (**NON_OVERRIDABLE**). Typy lze odvozovat od abstraktních typů (**ABSTRACT**), k nimž nelze vytvářet samostatné objekty a které vlastně jen poskytují seznam minimální výbavy příbuzných typů. Metody abstraktních typů mohou obsahovat pouze rozhraní bez implementace, musí pak mít atribut **DEFERRED** a odkazovat na blok abstraktního rozhraní (**ABSTRACT INTERFACE**) nebo na **INTERFACE** blok vhodné procedury.

```
př.  MODULE m
      PRIVATE ; PUBLIC t2d,t3d,dist
      TYPE t2d ; REAL x,y ; CONTAINS ; PROCEDURE :: dist=>dist2 ; END TYPE
      TYPE,EXTENDS(t2d) :: t3d ; REAL z ; CONTAINS ; PROCEDURE :: dist=>dist3 ; END TYPE
      CONTAINS
      REAL FUNCTION dist2(this) ; CLASS(t2d) this ; dist2=sqrt(this%x**2+this%y**2) ; END FUNCTION
      REAL FUNCTION dist3(this) ; CLASS(t3d) this ; dist3=sqrt(dist2(this%t2d)**2+this%z**2) ; END FUNCTION
      END MODULE
      TYPE(t2d) :: a2=t2d(x=1.,y=1.)
      TYPE(t3d) :: a3=t3d(t2d(1.,1.),1.)
      print *,a2,a2%dist(),a3%t2d%dist() ; print *,a3,a3%dist()

př.  MODULE m
      TYPE,ABSTRACT :: t0d ; CONTAINS ; PROCEDURE(adist),DEFERRED,PASS :: dist ; END TYPE
      ABSTRACT INTERFACE
      REAL FUNCTION adist(this) ; IMPORT t0d ; CLASS(t0d) :: this ; END FUNCTION
      END INTERFACE
      END MODULE
      TYPE,EXTENDS(t0d) :: t2d ; REAL x,y ; CONTAINS ; PROCEDURE :: dist=>dist2 ; END TYPE atd.
      TYPE(t2d) :: a2=t2d(x=1.,y=1.) ; TYPE(t3d) :: a3=t3d(x=1.,y=1.,z=1.) atd.
```

Generické metody a operátory, zobecněné konstruktory, finalizační metody

Stejnomené (**přetížené**) metody s různým rozhraním, s výjimkou **PASS** argumentu, se uvádějí v popisu **GENERIC**. Tentýž popis slouží i pro definici nových a přetížení existujících **operátorů**. Konstruktor struktury lze pomocí **INTERFACE** bloku přetížit generickou funkcí. Nepovinná finalizační procedura (**FINAL**) se volá automaticky při destrukci objektu.

```
př.  MODULE m
      TYPE t ; INTEGER i ; REAL r
      CONTAINS
      PROCEDURE,PRIVATE :: seti ; PROCEDURE,PRIVATE :: setr ; PROCEDURE,PRIVATE :: plus
      GENERIC :: set=>seti,setr
      GENERIC :: OPERATOR(+) => plus
      FINAL : write_message
      END TYPE
      INTERFACE t ; TYPE(t) FUNCTION init(i,j) ; INTEGER,INTENT(IN) :: i,j ; END FUNCTION ; END INTERFACE
      CONTAINS
      TYPE(t) FUNCTION init(i,j) ; INTEGER,INTENT(IN) :: i,j ; init%i=i ; init%r=j ; END FUNCTION
      SUBROUTINE seti(this,i) ; CLASS(t) this ; INTEGER i ; this%i=i ; END SUBROUTINE
      SUBROUTINE setr(this,r) ; CLASS(t) this ; REAL r ; this%r=r ; END SUBROUTINE
      TYPE(t) FUNCTION plus(this,that) ; CLASS(t),INTENT(IN) :: this,that
      plus%i=this%i+that%i ; plus%r=this%r+that%r ; END FUNCTION
      SUBROUTINE write_message(this) ; CLASS(t) this ; print *, 'object finalized' ; END SUBROUTINE
      END MODULE
      TYPE(t),POINTER :: a ; ALLOCATE (a) ; CALL a%set(1) ; CALL a%set(2.) ; print *,a,a+a ; DEALLOCATE (a)
```

V kontextu OOP ve Fortranu zbývá uvést volitelné parametrizování odvozených typů (mohou být parametrizovány proměnnými s atributy **KIND** a **LEN**). Parametrizované odvozené typy, generické konstruktory, finalizační metody a neomezeně polymorfní ukazatele nejsou však aktuálně implementovány v žádném z kraje zmíněných překladačů pro Intel/AMD.