Generičnost

Z FAV wiki

Genericita je možnost programovaciho jazyka definovat místo typů jen „vzory typů“, kde typy proměnných, použité v definici typu (rozuměj v typu jako ADT), jsou vyvedeny vně definice jako parametry a jsou určeny později klientskou aplikací. Základnim užitím genericity jsou třidy kontejnerů, které jsou určené k udržování skupin objektů určitého typu, například vzor třidy Seznam je definovan vlastně jako Seznam<G>, kde G je typ objektů, ktere mohou být do seznamu vloženy (kouzlo genericity vynikne pak v kombinací s dědičností, kdy do seznamu mohou být vloženy nejen objekty typu G, ale i objekty všech možných dědiců třídy (typu) G). Konkrétní typ, použitelný v textu programovacího jazyka pak vzniká, když G nahradíme skutečným existujícím typem.

Příklad:

Máme List<T> kde T je libovolný typ. Když List alokujeme, použijeme List<int> intList = new List<int>(), a tento list poté akceptuje jen čísla typu int.

Extrémní případy dovolují mnohé šílenosti, jako třeba

```
List < List < List < HodnotaTabulky > > >
```

je seznam tabulek :D

Pokud definujeme vzor (template, proto se používá T), používáme místo int prostě zástupce T

Např.

```
public interface List<T> {      // vytváříme interface, který říká, že nad typem <T> bude list
  void add(T x);                // Operace přidání prvku (typu <T>)
  Iterator<T> iterator();       // Struktura má iterátor, který definujeme níže
}
```

```
public interface Iterator<T> {  // Iterátor pohybující se po prvcích typu <T>
  T next();                     // Metoda vrací další prvek v pořadí
  boolean hasNext();            // Jen aby bylo vidět, že nemusíme jen vracet <T>
}
```

Což je takhle k ničemu, protože typ T kromě hodnoty nemá nic, co bychom mohli využít. Použijeme tedy obalení do třídy, např

```
public class Wrapper {
  T value;
  Wrapper next();
}
```

a s tou pak pracujeme.

V některých jazycích lze T omezit, třeba jen na typy implementující Iterable.