Iterator

Cursor, Iterátor

Poskytuje spôsob, akým je možné sekvenčne pristupovať k prvkom zloženého objektu bez toho, aby bola odhalená jeho vnútorná reprezentácia.

Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Motivácia

- Zapúzdrenie a skrytie spôsobu prechodu cez komplikované štruktúry:
  - typicky kolekcie (zoznamy, tabuľky, ..),
  - vlastné štruktúry (vlastná hierarchická organizácia dát).
- Umožnenie filtrovania – niekedy nemusí byť žiadúce, aby bol v prehliadke navštívený každý objekt.
- Umožňuje navrhovať algoritmy, ktoré pracujú s iterátormi namiesto kolekcií, čím je možné takéto algoritmy aplikovať na rôzne kolekcie bez nutnosti zmeny kódu algoritmu a bez znalosti vnútornej reprezentácie kolekcie.

Iterátory – C++

Kategória				Vlastnosti	Platné výrazy	
Všetky kategórie				kopírovací konštruktor, priradenie, deštruktor	`X b(a);`	`b = a;`
				Môžu byť inkrementované	`++a`	`a++`
S náhodným prístupom	**Obojsmerné**	**Dopredné**	**Vstupné**	Podporuje operácie rovný/nerovný	`a == b`	`a != b`
				môže byť dereferencovaný ako rvalue	`*a`	`a->m`
			Výstupné	môže byť dereferencovaný ako lvalue (musí byť mutable)	`*a = t`	`*a++ = t`
				default konštruktor	`X a;`	`X()`
				ani dereferencovanie ani inkrementácia neovplyvňujú dereferencovateľnosť	`{b = a; *a++; *b;}`	
				môže byť dekrementovaný	`--a` `a--`	`*a--`
				podporuje aritmetické + a -	`a + n` `n + a`	`a - n` `a - b`
				podporuje porovnávania <, >, <= a >= medzi iterátormi navzájom	`a < b` `a > b`	`a <= b` `a >= b`
				podporuje skrátené operácie priradenia += a -=	`a += n`	`a -= n`
				podporuje operátor []	`a[n]`	

Knižnica <algorithm>

- A range is any sequence of objects that can be accessed through iterators or pointers, such as an array or an instance of some of the STL containers. Notice though, that algorithms operate through iterators directly on the values, not affecting in any way the structure of any possible container (it never affects the size or storage allocation of the container).

```
template < class ForwardIterator, class UnaryPredicate, class T >
  void replace_if (ForwardIterator first, ForwardIterator last,
                   UnaryPredicate pred, const T& new_value)
{
  while (first!=last) {
    if (pred(*first)) *first=new_value;
    ++first;
  }
}
```

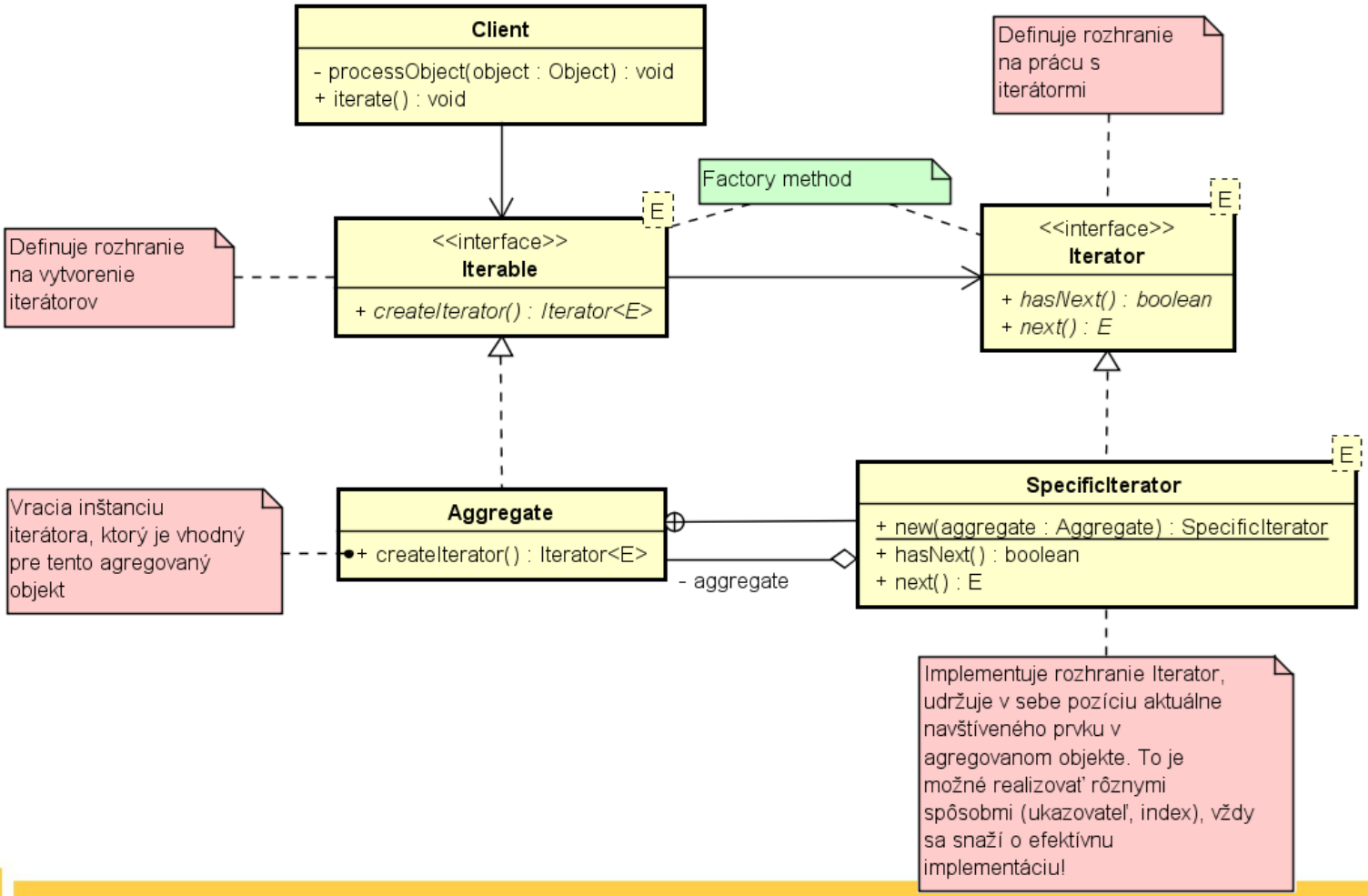
```
template <class ForwardIterator>
  bool is_sorted (ForwardIterator first, ForwardIterator last)
{
  if (first==last) return true;
  ForwardIterator next = first;
  while (++next!=last) {
    if (*next<*first)     // or, if (comp(*next,*first)) for version (2)
      return false;
    ++first;
  }
  return true;
}
```

```
template <class InputIterator1, class InputIterator2>
  bool includes (InputIterator1 first1, InputIterator1 last1,
                 InputIterator2 first2, InputIterator2 last2)
{
  while (first2!=last2) {
    if ( (first1==last1) || (*first2<*first1) ) return false;
    if (!(*first1<*first2)) ++first2;
    ++first1;
  }
  return true;
}
```

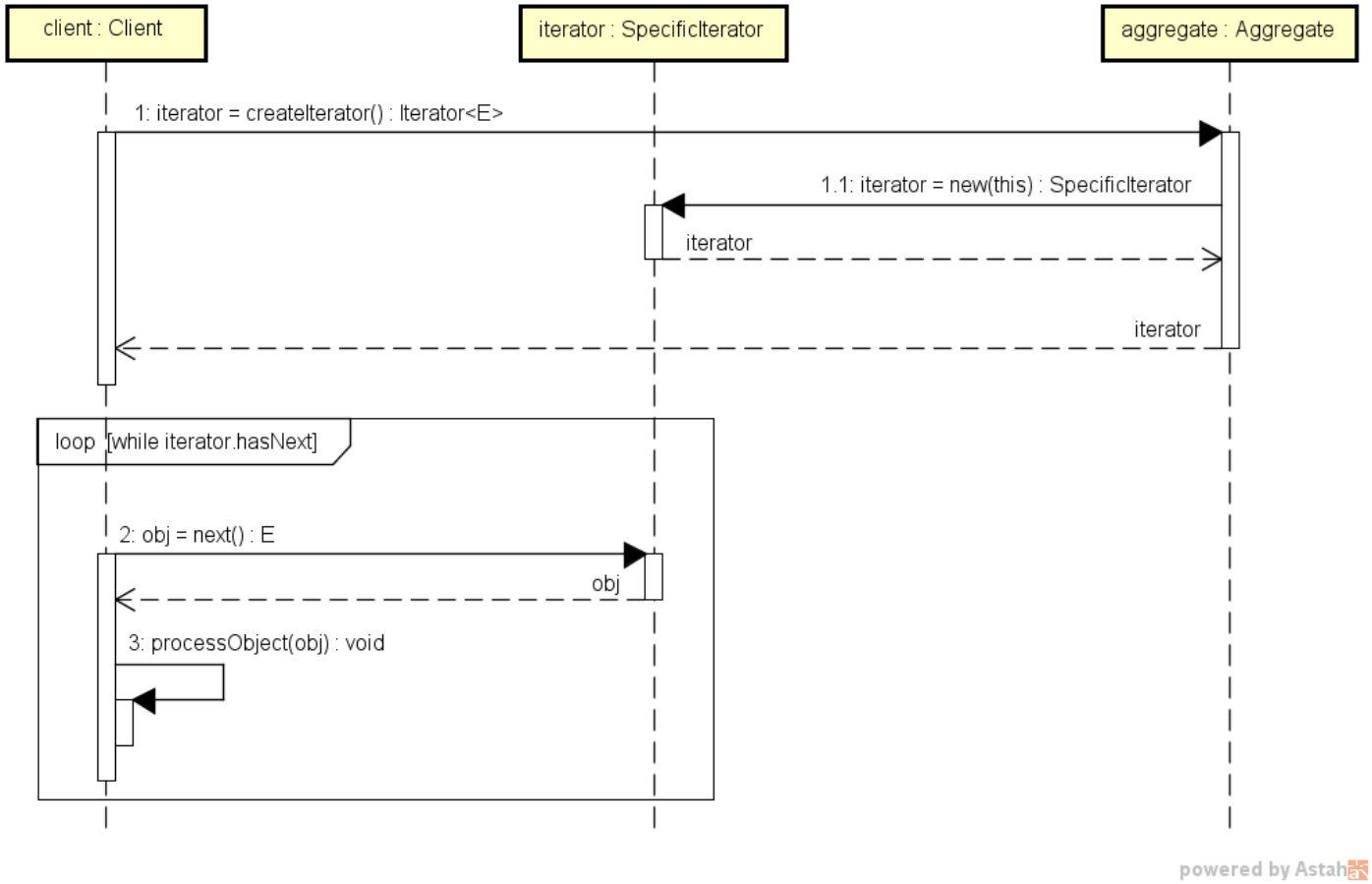
Aplikovateľnosť

- Iterátor je vhodné použiť, keď:
  - chcete pristupovať na prvky zloženého objektu bez toho, aby ste odhalili jeho vnútornú štruktúru.
  - chcete umožniť viac prechodov (naraz) cez zložený objekt.
  - chcete poskytnúť jednotné rozhranie pre prechod rôznych zložených štruktúr (využitie polymorfizmu pri prechode štruktúrami).

Implementácia



Spolupráca


client : Client
iterator : SpecificIterator
aggregate : Aggregate
1: iterator = createIterator() : Iterator<E>
1.1: iterator = new(this) : SpecificIterator
iterator
iterator
loop [while iterator.hasNext]
2: obj = next() : E
obj
3: processObject(obj) : void
powered by Astah

Dôsledky

- Umožňuje jednoducho zmeniť algoritmus prechodu štruktúrou. Typickým príkladom je strom, ktorý je možné prejsť rôznymi spôsobmi (preorder, postorder, levelorder), pričom iterátor implementuje konkrétny spôsob prechodu. Jednoduchou zmenou iterátora je možné zmeniť spôsob prechodu, pričom navonok nie je potrebné nič meniť.
- Zjednodušuje rozhranie zloženého objektu, nakoľko ten nemusí definovať metódy spojené s prechodom štruktúry.
- Keďže si iterátor udržuje vlastný stav (kde v štruktúre sa aktuálne nachádza), je možné, aby bolo cez štruktúru naraz realizovaných niekoľko prehliadok.

Príbuzné vzory

- *Composite* – Iterátory často prechádzajú zložené objekty.
- *Factory method* – Iterátory sú často vytvárané pomocou Továrenskej metódy. Typickým príkladom v jazyku Java sú kolekcie, ktoré implementujú Iterable s metódou iterator(), ktorá vracia objekt implementujúci rozhranie Iterator. To, aký iterátor sa vytvorí závisí na konkrétnej triede kolekcie.
- *Memento* – často ukladá aktuálny stav Iterátora. Memento býva v Iterátore definované neverejne.
- *Proxy* – Iterátor je špeciálnym typom Zástupcu.

Upozornenie

- Tieto študijné materiály sú určené výhradne pre študentov predmetu 5II132 Návrhové vzory (Design Patterns) na Fakulte riadenia a informatiky Žilinskej univerzity v Žiline.
- Reprodukovanie, šírenie (i častí) materiálov bez písomného súhlasu autora nie je dovolené.

Ing. Michal Varga, PhD.
Katedra informatiky
Fakulta riadenia a informatiky
Žilinská univerzita v Žiline
Michal.Varga@fri.uniza.sk