Programozási nyelvek Java

Kozsik Tamás előadása alapján
Készítette: Nagy Krisztián

11. előadás

Léteznek olyan kifejezések, melyek nem jól definiáltak. Nem egyértelmű a jelentésük. A programozó felelőssége elkerülni az ilyen eseteket.

Szemantika folytatása:
Lustaság és mohóság kérdése:

Lusta: &&, ||
Mohó: +,*, ==

boolean: true, false, exception, végtelen számítás

&&	**i**	**h**	**e**	∞	**&**	**i**	**h**	**e**	∞
i	i	h	e	∞	**i**	i	h	e	∞
h	h	h	h	h	**h**	h	h	e	∞
e	e	e	e	e	**e**	e	e	e	e
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Javaban balról jobbra értékelődnek ki a paraméterek (nem minden nyelvben van így), operátorokat, metódusokat
Szemantikai hatás – kifejezés jelentése egyértelmű

Elemi műveletek közül: egyik sem kommutatív.
- biased towards A

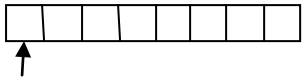
Mellékhatásos kifejezések: mellékhatásos operátorok, mellékhatásos függvények

A + B : kiértékelem A-t, kiértékelem B-t, majd elvégzem az összeadást.

(i++)+t[i]: vegyük i-t(érték), növeljük i-t (mellékhatás), tömb eleme a megnövelt helyen(érték), értékek összege (érték)

Argumentumok/ operandusok kiértékelési sorrendje hatással van a kiszámított értékre és a mellékhatásokra, valamint azok sorrendjére.

Matematikában: $\mathbb{Z}$
Programozásban például: byte $\mathbb{Z}|_{[-128,127]}$



előjel bit (0 – nem negatív, 1 – negatív)

$11111111 \equiv -1$
$00000000 \equiv 0$
$00000001 \equiv 1$
$00000010 \equiv 2$

Legelterjettebb a kettes komplemens ábrázolás. Ez JAVA-ban is így van.
A matematikai művveletek sem felelnek meg a programozásban használt műveletekkel.
Például: -(-128) = -128

Matematikában : $\mathbb{Q}, \mathbb{R}$

Programozásban: double, float. Értékkészletük szintén véges.
Továbbá: $\exists v \in int \ \wedge v \notin double$
aritmetikája: nagy a hiba

$0{,}1101 \cdot 2^{101}$
bináris

<u>Generic</u>

$Ha\ A <: B\ , akkor\ A[\ \ ] <: B[\ \ ]$
$Ha\ A <: B, akkor\ ArrayList < A > \not<: ArrayList < B >$ és $A \neq B$ és $B \neq ?$
Példa:
$ArrayList < Integer > <: List < Integer >$
$List < Integer > \not<: List < Object >$

Java generic: invariáns generic paraméter (altípusossághoz nem változtathatom meg a típusparamétert)
$class\ ArrayList < A > extends\ ArrayList < A > implements\ List < A >$
$\forall A: ArrayList < A > <: AbstractList < A >$ és $ArrayList < A > <: List < A >$
Ez a parametrikus és altípusos polimorfizmus viszonyát is bemutatta.

Bounded parametrikus polimorfizmus:
Korlátozás a típusparaméterre.

```
class SearchTree <T extends Comparable1>{
        void insert(T item){
            …                                   upper bound:    Comparable1
            item.less(other)                                        ▲
            …                                                       |
        }                                                           T
}

interface Comparable1{
        boolean less(Comparable1 other);
}

class C implements Comparable1{
        …
        public boolean less(Comparable1 other){…}
}

class A {…}
```

„Olyan típus, amelynek értékei összehasonlíthatók)

```
interface Comparable2{
        int compareTo(Comparable2 other); // nem less, hanem compareTo
}
```

SearchTree<C> => OK SearchTree<A> => fordítási hiba

„Olyan típus, aminek az értékei összehasonlíthatók T értékkel”

```
interface Comparable3<T>{
        int compareTo(T other);
}

class C implements Comparable3<C>{
        …
        public int compareTo(C other){…}
}


class SearchTree<T extends Comparable3<T>>…
```

class D extends C ~~implements Comparable3<D>~~{…}

Egy osztály nem valósíthatja meg ugyan azt az interface-t különböző típusparaméterekkel.
(az implementáció következménye)

D <: C <: Comparable<C>
D ≮:Comparable<D> feloldhatatlan Javaban

List<?> list of-unknown

List<?> list = new ArrayList<Integer>();
list = new LinkedList<String>();

list.add(”pityu”); // ILLEGÁLIS!

Lekorlátozza a lehetőségeket.

class SearchTree<T extends Comparable3<? super T>>{…}

lower bound

SearchTree<C> és SearchTree<D> -re is jó lesz.