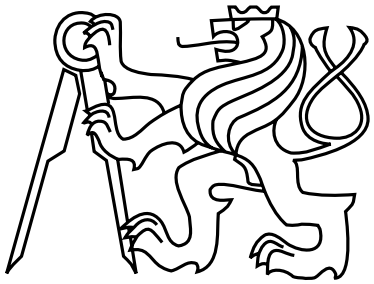
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

# Gramatika podmnožiny jazyka Java v nástroji Fika

*Tomáš Linhart*

Vedoucí práce: Mgr. Michal Píše

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Softwarové inženýrství

27. května 2011

# Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu práce Mgr. Michalovi Píšemu za pomoc při psání bakalářské práce a za nástroj Fika. Rovněž za to, že díky němu jsem se dozvěděl o existenci tohoto velice zajímavého tématu a mohl vypracovat tuto práci. Také bych rád poděkoval kolegům Tomášovi Jukinovi a Jakubovi Stejskalovi za sdílení užitečných rad ohledně nástroje Fika. Rovněž bych chtěl moc poděkovat rodičům za to, že mi umožnili studium na vysoké škole a podporovali mě finančně i psychicky při mém studiu a dali mi prostor k seberealizaci.

# Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 27. 5. 2011 .....

# Abstract

This work describes implementation of Java's grammar subset using Fika tool. Two iterations of Java's subsets are implemented focusing on readability and modularity. Each subset is always described by grammar, type system and implementation in Fika tool. Second iteration extends grammar of Java's subset with additional Java functionality and documents how hard it was to implement these features using Fika tool.

# Abstrakt

Tato práce se zabývá implementaci gramatiky podmnožiny jazyka Java v nástroji Fika. V této práci jsou implementovány dvě iterace podmnožiny jazyka Java, kde byl hlavně kladen důraz na čitelnost a modulárnost. Podmnožiny jazyka Java jsou vždy popsány gramatikou, typovým systémem a implementací v nástroji Fika. V druhé iteraci byla gramatika podmnožiny jazyka Java rozšířena o další funkcionalitu z jazyka Java a bylo zdokumentováno jak obtížné bylo danou funkcionalitu implementovat pomocí nástroje Fika.

# Obsah

# Část I

# Úvod

# Kapitola 1

# Úvod

## 1.1 Úvod do problematiky

Implementace a modulárnost gramatik je téma, kde je stále ještě dost prostoru k objevování. Když někdo dnes potřebuje gramatiku pro programovací jazyk či jiný úkol, tak ji napíše zcela na zelené louce. Možná sáhne po lexeru a parseru, ale i přesto bude muset celou gramatiku vymyslet sám. Bylo by tedy skvělé, kdyby některé části už mohl znovu použít z již existující jazyků jako je například jazyk Java. Když se člověk zamyslí, tak většina programovací jazyků je ve své podstatě stejná. Většina jazyků dnes má moduly, třídy, proměnné, aritmetiku. Občas mají i něco unikátního, ale existuje zde určitý průnik funkcionality. Nebylo by tedy skvělé, kdyby člověk při vytváření mohl sáhnout po existujících modulech, které dělají to co on potřebuje a pokud ne, tak si danou věc přidat či upravit? Cílem této práce bude právě vytvořit modulární podmnožinu gramatiky jazyka Java, kde bude cíl vytvořit moduly, které implementují podmnožinu jazyka Java. Uživatel, který tedy bude chtít vytvořit jazyk podobný jazyku Java bude tedy moct využít již existující funkcionality a jen si jí upravit dle svých potřeb. A tohle vše bude možné díky nástroji Fika jehož cíl je umožnit vytvářet modulární gramatiky.

## 1.2 Použité konvence

### 1.2.1 Gramatika

Pro popis gramatik podmnožin jazyka Java byla použita BNF[1]. Jedná se o notaci, která slouží pro zapsání gramatik ve formě pravidel. Pravidlo je ve formě

neterminál → ... terminály a neterminálny ...

Je možnost zapsat i možnost volby pomocí vertikální čáry |. Lze tedy např. napsat.

*Osloveni* → Paní | Pán
*CeleJmeno* → *Osloveni* Jméno Přijmení

Neterminál je tedy kurzívou a terminál je sans-serif písmem.

[1] Backus–Naur Form

### 1.2.2 Inferenční pravidla

Dále se v práci vyskytují inferenční pravidla, která mají vždy předpoklad a závěr. Jsou zapsány v rovnici.

$$\frac{\text{Předpoklad}}{\text{Závěr}} \tag{1.1}$$

### 1.2.3 Funkce

Dále jsou v práci funkce, které se používají v inferenčních pravidlech k získání informace z kontextu či neterminálu. Je vždy napsán název funkce a následně z jaké množiny do jaké množiny provádí převod. Na dalším řádku je vždy samotná implementace funkce. Může být i na více řádcích.

Název : Z čeho $\rightarrow$ Do čeho (1.2)
Název( Parametery ) { Vnitřek funkce }

# Část II

# Specifikace

# Kapitola 2

# Featherweight Java

## 2.1 Úvod

První iterace podmnožiny jazyka Java vychází z článku[1], který pojednává o jazyku Featherweight Java. Tento jazyk je velmi zjednodušená verze jazyka Java. Je okleštěn téměř o všechny věci, které v jazyce Java existují - jako je reflexe, bloky, vlákna a mnoho dalšího. Dokonce i přiřazení je vynecháno. Tyto vlastnosti byly z jazyka vynechány, protože pro účely jazyka Featherweight Java byly zbytečně.

Cílem tohoto jazyka bylo umožnit provést důkazy, které by na složitějších jazycích byly nemožné udělat. Také bylo důležité, aby tyto důkazy byly lehce proveditelné. To je důvod, proč byl tento jazyk tak okleštěn. Jazyk Featherweight Java je tedy podobný jazyku Java jako je lambda kalkulus podobný jazyku ML.

## 2.2 Neformální nastínění

V následující části jsou rozebrány všechny aspekty jazyka Featherweight Java a jsou podrobně popsány.

### 2.2.1 Třídy

Program v jazyce Java je tvořen třídami. Program může obsahovat libovolné množství tříd, kde každá třída musí dědit od jiné třídy. V programu již existuje třída `Object`. U každé třídy se musí psát `extends` i když dědí z třídy `Object`. Každá třída potom v sobě obsahuje pole, konstruktor a metody. Krom tříd na konci programu je jeden výraz, který má stejný smysl jako metoda `main` v jazyku Java. Viz následující ukázka jednoduchých tříd. Výraz na konci programu se neukončuje středníkem.

`class A extends Object {` *Pole Konstruktor Metody* `}`
`class B extends A {` *Pole Konstruktor Metody* `}`

### 2.2.2 Pole

Každá třída v sobě má pole. Může jich mít libovolný počet. Klidně i nula. Pouze musí být dodržena unikátnost jmen a musí existovat třídy jejíž typu jsou. Viz následující ukázka polí. Rovněž jména nesmí kolidovat s jmény polí, které byly definovány v rodičovské třídě.

Jelikož v jazyku Featherweight Java nejde přiřadit, tak by pole měly být konstantní. Mělo by se správně tedy u polí uvádět klíčové slovo `final`, ale to je vynecháno pro jednoduchost jazyka.

```
Object first;
Object second;
```

### 2.2.3 Konstruktor

Každá třída v sobě musí mít konstruktor a to i v případě, že by v něm nic nebylo. Také maximální možné množství konstruktoru je jedna. Každý konstruktor musí jako parametry mít pole z rodičovské třídy ve stejném pořadí v jakém byly definovaný a následované poli ve třídě, kde je konstruktor, ve stejném pořadí jako byly definovány. Parametry jsou odděleny čárkou. Parametry převzaté z rodičovské třídy jsou předány volání konstruktoru rodičovské třídy. Ten se volá pomocí `super`. V případě, že by rodičovská třída neměla žádné pole, tak se volá bez parametrů.

V těle konstruktoru je tedy hned na prvním řádku volání konstruktoru rodičovské třídy následovaný seznamem deklarací. Každá deklarace odpovídá jednomu poli ve třídě a opět je dodrženo pořadí v jakém byly definovány ve třídě. Rovněž musí sedět názvy parametrů na názvy polí. Viz ukázka konstruktoru.

```
A(Object first, Object second) {
  super();
  this.first = first;
  this.second = second;
}
```

### 2.2.4 Metody

Třída v jazyce Featherweight Java může obsahovat metody, ale nemusí. Každá metoda musí obsahovat návratový typ následovaný unikátním jménem. V jazyce Featherweight Java není možno přetěžovat metody. Může tedy existovat pouze jedna metoda s jedním názvem. Po názvu metody následují parametry. Jsou odděleny čárkou a může jich být libovolné množství. Může být tedy také metoda bez parametrů. Jména parametrů nejsou předepsaná pouze se nesmí opakovat a typy, které mají musí být z tříd, které v programu existují. V těle metody může být pouze jeden příkaz a to `return` následovaným výrazem. Viz ukázka metody.

```
 A setFirst(Object first) {
  return new A(first, this.second);
}
```

### 2.2.5 Výrazy

Jediná místa, kde se výrazy mohou objevit, je v těle metod a jako samostatný výraz v programu, který slouží jako vstupní bod[1]. Výrazu je v jazyku Featherweight Java několik druhů.

První z nich je buď parametr nebo proměnná `this`. Obě tyto možnosti byli použity v předcházející metodě 2.2.4. Znovu ukázka pro oživení.

`this.first` nebo `first`

Druhý výraz je přístup k poli. K poli lze přistupovat pouze nad nějakým výrazem. Lze toto vidět u předchozí příkladu, kde nad výrazem `this` přistupujeme k poli `first`. Takto lze přistupovat ke všem polím, které daný typ výrazu má. Viz následující výraz.

```
this.second
```

Třetí výraz je volání metody. Pokud chceme volat metodu, tak opět musíme mít výraz, nad kterým jí voláme. Metoda opět vrací výraz, takže lze takto výrazy snadno řetězit. Viz následující výraz.

```
this.setFirst(new Object()).first
```

Čtvrtý výraz je volání konstruktoru. Konstruktor se volá klíčovým slovem `new` a názvem třídy, u které chceme konstruktor volat a následně předáme seznam výrazu, které chceme použít jako parametry konstruktoru. Dané výrazy musí mít stejný typ jako mají parametry konstruktoru. Viz následující výraz.

```
new A(new Object(), new Object())
```

Pátý a poslední výraz je přetypování výrazu na jiný výraz. Musíme si dát pozor, aby výraz, který chceme přetypovávat byl podtypem nebo nadtypem typu nebo stejného typu, na který chceme přetypovávat. Viz následující výraz.

```
(Object) new A(new Object(), new Object())
```

Toto je korektní, protože třída `A` dědí ze třídy `Object`.

[1] Metoda `main` v jazyce Java.

### 2.2.6 Ukázka celého programu

Pro shrnutí možností jazyka Featherweight Java se podívejte na následující kód, který je přejat z článku[1]. Jedná se o implementaci páru.

```
class A extends Object {
  A() { super(); }
}
class B extends Object {
  B() { super(); }
}
class Pair extends Object {
  Object fst;
  Object snd;
  Pair(Object fst, Object snd) {
    super(); this.fst=fst; this.snd=snd;
  }
  Pair setfst(Object newfst) {
    return new Pair(newfst, this.snd);
  }
}
```

Následuje výraz, který je součástí programu. Tento výraz vytváří nový pár z třídy `A` a z třídy `B` a volá se metoda, která vytvoří nový pár z předaného výrazu.

```
new Pair(new A(), new B()).setfst(new B())
```

Celý výraz lze tedy přepsat jako:

```
new Pair(new B(), new B())
```

To lze proto kvůli tomu, že metoda `setfst` vrací novou instanci třídy `Pair`, kde jako první parametr je předaný parametr do metody a druhý parametr je použít obsah pole `snd`.

## 2.3 Formální popis jazyka

V této části je rozebrán jazyk Featherweight Java formálně. Je vždy vysvětlena část jazyka, kde je část gramatiky a následně k tomu typový systém, který se stará o otypování dané části a zajištění, že daný program je správně zformulovaný.

### 2.3.1 Množiny

Následující množiny slouží jako základní prvky pro stavbu programu a jsou používány pro definice kontextů a funkcí v inferenčních pravidlech. Tyto množiny jsou potřeba, aby bylo

možné popsat jaké množiny je daný kontext či z jaké množiny do jaké množiny funkce provádí převod.

$\mathsf{C}$ = Množina všech možných jmen tříd, které existují.
$\mathsf{F}$ = Množina všech možných jmen polí, které existují.
$\mathsf{P}$ = Množina všech možných jmen parametrů, které existují.
$\mathsf{M}$ = Množina všech možných jmen metod, které existují.

### 2.3.2 Kontexty

Následující kontexty jsou využívané v inferenční pravidlech definovaných níže. Jsou do nich ukládány informace, které jsou potřebné k tomu, aby byl celý program správně otypován. Jedná se o třídy, pole, metody atd. Slouží například k ověření toho, že daná třída v programu existuje jen jednou. Tím, že při prvním výskytu se přidají do kontextu lze u dalších tříd ověřit, že už tam nejsou a tím je zajištěna unikátnost jmen.

| | | |
|---|---|---|
| $\Gamma$ | $\subseteq (\mathsf{C}, \mathsf{C}, \mathsf{F} \times \mathsf{C}, (\mathsf{M}, \mathsf{P} \times \mathsf{C}))$ | (globální kontext programu) |
| $\Gamma_\mathsf{L}$ | $\subseteq (\{\mathsf{this}\} \cup \mathsf{P}) \times \mathsf{C}$ | (lokalní kontext výrazu) |
| $\Gamma_\mathsf{P}$ | $\subseteq \mathsf{P}$ | (kontext jmen parametrů) |
| $\Gamma_\mathsf{M}$ | $\subseteq \mathsf{M}$ | (kontext jmen metod) |
| $\gamma_\mathsf{M}$ | $\subseteq \mathsf{M}$ | (kontext jmen metod rodiče) |
| $\Gamma_\mathsf{D}$ | $\subseteq \mathsf{F}$ | (kontext jmen deklarací) |
| $\Gamma_\mathsf{F}$ | $\subseteq \mathsf{F}$ | (kontext jmen polí rodiče) |
| $\gamma_\mathsf{F}$ | $\subseteq \mathsf{F}$ | (kontext jmen polí rodiče) |
| $\Gamma_\mathsf{K}$ | $\in \mathsf{C}$ | (kontext jména třídy) |
| $\Gamma_\mathsf{C}$ | $\subseteq \mathsf{C}$ | (kontext jmen tříd) |

### 2.3.3 Třídy

Program v jazyku Featherweight Java je tvořen seznamem tříd a výrazem. Třída je tvořena klíčovým slovem `class` následovaný identifikátorem, který označuje název třídy a klíčovým slovem `extends`, po kterém je opět uveden identifikátor, který označuje název třídy, z které tato třída dědí. Ve třídě jsou následně pole, konstruktor a metody.

| | | |
|---|---|---|
| *Program* | $\rightarrow$ | *ClassList Expression* |
| *ClassList* | $\rightarrow$ | *Class ClassList* |
| | | $\mid \epsilon$ |
| *Class* | $\rightarrow$ | class Identifier extends Identifier |
| | | { *FieldList Constructor MethodList* } |

Pro správné otypování programu je třeba ověřit, že zadané třídy jsou správně otypovány, a že i výraz je správně otypován. Je třeba také ověřit, že v grafu nejsou cykly. Tento graf se vytváří z tříd. Cykly by totiž vedly k programu, který by byl nepřeložitelný. Cyklem se myslí, kdy třída `A` dědí od třídy `B` a ta dědí od třídy `A`. V prvním inferenčním pravidle je také vytvořen globální kontext a to pomocí funkce `CLASSES`, které je předán `ClassList`. Tento

globální kontext slouží v dalších pravidlech k zajištění správného otypování programu. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{\begin{array}{c}\{(\mathsf{CLASSES}(\mathit{ClassList}), \mathit{Object})\} \vdash \mathit{ClassList} : \diamond \\ (\mathsf{CLASSES}(\mathit{ClassList}), \emptyset) \vdash \mathit{Expression} : \diamond \\ \mathsf{GRAPH}(\mathsf{CLASSES}(\mathit{ClassList}))\ \mathit{neobsahuje\ cykly}\end{array}}{\vdash \mathit{ClassList}\ \mathit{Expression} : \diamond} \tag{2.1}$$

Funkce CLASSES pro vytvoření globálního kontextu přijímá ClassList, který je následně dále rozebírán až do hloubky. Dokud globální kontext neobsahuje všechny informace.

$$\mathsf{CLASSES} : \mathsf{ClassList} \to \mathcal{P}(\mathsf{C}, \mathsf{C}, \mathsf{FIELDS}, \mathsf{METHODS}) \tag{2.2}$$
$$\mathsf{CLASSES}(\ \epsilon\ ) = \{\}$$
$$\mathsf{CLASSES} : \mathsf{ClassList} \to \mathcal{P}(\mathsf{C}, \mathsf{C}, \mathsf{FIELDS}, \mathsf{METHODS}) \tag{2.3}$$
$$\mathsf{CLASSES}(\mathsf{class\ C\ extends\ BC}\ \{\mathit{FieldList}\ \mathit{Constructor}\ \mathit{MethodList}\} \cdot \mathit{ClassList}) =$$
$$(\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}(\mathit{FieldList}), \mathsf{METHODS}(\mathit{MethodList})) \cup \mathsf{CLASSES}(\mathit{ClassList})$$

Jakmile jsou jednotlivé prvky třídy (název třídy, název rodiče, pole a metody) rozebrány, tak následuje vytváření kontextu pro pole a to pomocí funkce FIELDS, která přijímá FieldList a následně rozebírá jednotlivé pole na jméno pole a jeho typ.

$$\mathsf{FIELDS} : \mathsf{FieldList} \to \mathsf{Seq}\ (\mathsf{N} : \mathsf{T}) \tag{2.4}$$
$$\mathsf{FIELDS}(\ \epsilon\ ) = \mathit{Seq}\ \{\}$$
$$\mathsf{FIELDS} : \mathsf{FieldList} \to \mathsf{Seq}\ (\mathsf{N} : \mathsf{T}) \tag{2.5}$$
$$\mathsf{FIELDS}(\ \mathsf{T\ N}\ ; \cdot \mathit{FieldList}\ ) = \mathsf{N} : \mathsf{T} \cdot \mathsf{FIELDS}(\mathit{FieldList})$$

Po polích následuje rozebírání metod a to pomocí metody METHODS, která přijímá MethodList, který následně převede na množinu n-tic, která se skládá z jména metody a z jejího typu a z parametrů, které metoda má.

$$\mathsf{METHODS} : \mathsf{MethodList} \to \mathcal{P}(\mathsf{M} \times \mathsf{C}, \mathsf{PARAMS}) \tag{2.6}$$
$$\mathsf{METHODS}(\ \epsilon\ ) = \{\}$$
$$\mathsf{METHODS} : \mathsf{MethodList} \to \mathcal{P}(\mathsf{M} \times \mathsf{C}, \mathsf{PARAMS}) \tag{2.7}$$
$$\mathsf{METHODS}(\ \mathsf{T\ M\ (}\ \mathit{ParameterList}\ \mathsf{)\{\ return}\ \mathit{Expression}\ \mathsf{;\}} \cdot \mathit{MethodList}\ ) =$$
$$(\mathsf{M} : \mathsf{T}, \mathsf{PARAMS}(\mathit{ParameterList})) \cup \mathsf{METHODS}(\mathit{MethodList})$$

Poslední část vytváření globální kontextu se vytváří z parametrů, které jsou součástí metod. Funkce PARAMS přijímá ParameterList, který následně převede na sekvenci jméno parametru a jeho typ.

$$\mathsf{PARAMS} : \mathsf{ParameterList} \to \mathsf{Seq}\ (\mathsf{P} \times \mathsf{C}) \tag{2.8}$$
$$\mathsf{PARAMS}(\ \epsilon\ ) = \mathit{Seq}\ \{\}$$
$$\mathsf{PARAMS} : \mathsf{ParameterList} \to \mathsf{Seq}\ (\mathsf{P} \times \mathsf{C}) \tag{2.9}$$
$$\mathsf{PARAMS}(\ \mathsf{T\ N} \cdot \mathit{ParameterList}\ ) = \mathsf{N} : \mathsf{T} \cdot \mathsf{PARAMS}(\mathit{ParameterList})$$

Jak si lze povšimnout, tak do globálního kontextu jsou přidány informace téměř o všem až na konstruktor. Konstruktor není v globálním kontextu jelikož nepřináší žádnou novou informaci. Konstruktor pouze nastavuje polím hodnoty. Informace, kterou by přinesl konstruktor je již tedy obsažen v polích.

Pro lepší názornost si můžeme ukázat jak funkce pro vytvoření globálního kontextu funguje tím, že vytvoří globálního kontext pro kód uvedený v předchozí kapitole 2.2.6.

```
CLASSES(class A extends Object  FieldList Constructor MethodList ,
         class B extends Object  FieldList Constructor MethodList ,
         class Pair extends Object  FieldList Constructor MethodList  ) =
  (Pair, Object, FIELDS(FieldList), METHODS(MethodList)) ∪ CLASSES(ClassList)
FIELDS((Object fst;), (Object snd;)) = fst : Object ∪ FIELDS(FieldList)
METHODS(Pair setfst(Object newfst) return new Pair(newfst, this.snd) ) =
  (setfst : Pair, PARAMS(ParameterList)) ∪ METHODS(MethodList)
PARAMS(Object newfst) = newfst : Object ∪ PARAMS(ParameterList)
```

Jedná se o hrubý průchod. Nejsou tam vzaty v potaz případy, když v listu nic není a vrátí se $\epsilon$. Zároveň, když se prochází zbytek listu, tak není rozepsáno, co v listu je a je použit zápis xxxList. Každopádně by to mělo poskytnout zběžný náhled na to jak tyto funkce fungují.

Jelikož globální kontext je dost složitý a jeho celý přepis při předávání by byl rozsáhlý, tak jsou pro něj definovány následující aliasy.

CLASSES : (C, C, FIELDS, METHODS)
FIELDS : (F $\times$ C)
METHODS : (M $\times$ C, PARAMS)
PARAMS : (P $\times$ C)

Jakmile je vytvořen globální kontext, tak je možno z něho vytvořit graf, který je potřeba pro kontrolu existence cyklů. Tento graf má vrcholy z názvu tříd a hrany z dvojice názvu třídy a názvu jeho rodiče. Graf je tedy tvořen z vrcholů a hran.

$$\text{GRAPH} : \text{CLASSES} \rightarrow (\mathcal{P}(\text{C}), \mathcal{P}(C, C)) \tag{2.10}$$
$$\text{GRAPH}(\ \text{CLASSES}\ ) = (\text{NODES}(\text{CLASSES}), \text{EDGES}(\text{CLASSES}))$$

Vrcholy jsou tvořeny získáním názvu třídy z globálního kontextu.

$$\text{NODES} : \text{CLASSES} \rightarrow \mathcal{P}(\text{C}) \tag{2.11}$$
$$\text{NODES}(\ \epsilon\ ) = \{\}$$
$$\text{NODES} : \text{CLASSES} \rightarrow \mathcal{P}(\text{C}) \tag{2.12}$$
$$\text{NODES}(\ (\text{C, BC, FIELDS, METHODS}) \cdot Set\ ) = \text{C} \cup \text{NODES}(Set)$$

Hrany jsou tvořeny získáním dvojice názvu třídy a názvu třídy rodiče z globálního kontextu.

$$\mathsf{EDGES} : \mathsf{CLASSES} \to \mathcal{P}(\mathsf{C}, \mathsf{C}) \tag{2.13}$$
$$\mathsf{EDGES}(\ \epsilon\ ) = \{\}$$
$$\mathsf{EDGES} : \mathsf{CLASSES} \to \mathcal{P}(\mathsf{C}, \mathsf{C}) \tag{2.14}$$
$$\mathsf{EDGES}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}) \cdot \mathit{Set}\ ) = (\mathsf{C}, \mathsf{BC}) \cup \mathsf{EDGES}(\mathit{Set})$$

Jakmile je otypován nejvyšší vrchol programu - jeho počátek, tak následuje otypování každé třídy, kde ClassList je rozebrán na jednotlivé třídy, které jsou následně otypovány. Vždy když je třída správně otypována, tak je přidán její název do kontextu tříd, který slouží pro ověření zda již třída nebyla předtím definována. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{(\Gamma, \Gamma_\mathsf{C}) \vdash \mathit{Class} : \diamond \quad (\Gamma, \Gamma_\mathsf{C} \cup \{\mathsf{CNAME}(\mathit{Class})\}) \vdash \mathit{ClassList} : \diamond}{(\Gamma, \Gamma_\mathsf{C}) \vdash \mathit{Class}\ \cdot\ \mathit{ClassList} : \diamond} \tag{2.15}$$

Název třídy je z deklarace třídy získán pomocí funkce CNAME. Tato funkce přijímá deklaraci třídy a vrací její jméno.

$$\mathsf{CNAME} : \mathsf{Class} \to \mathsf{C} \tag{2.16}$$
$$\mathsf{CNAME}(\ \mathsf{class\ C\ extends\ BC}\ \{\ \mathit{FieldList}\ \mathit{Constructor}\ \mathit{MethodList}\ \}\ ) = \mathsf{C}$$

Jakmile je ClassList rozebrán na jednotlivé třídy je potřeba otypovat třídu samotnou. Je třeba ověřit, že třída již nebyla definována a následně je třeba ověřit, že třída, z které je děděno existuje někde v programu. Následně je otypován FieldList, Constructor a MethodList. Toto inferenční pravidlo má ekvivalent v [1]. Odpovídá pravidlu T-CLASS ve Fig. 2. Rovněž některá jeho funkcionalita už byla obsažena v předchozích inferenčních pravidlech.

$$\frac{\begin{array}{c}\mathsf{C} \notin \Gamma_\mathsf{C} \quad \mathsf{BC} \in \mathsf{CNAMES}(\Gamma) \\ (\Gamma, \emptyset, \mathsf{FNAMES}(\Gamma, \mathsf{BC})) \vdash \mathit{FieldList} : \diamond \\ (\Gamma, \mathsf{C}, \mathsf{FNAMES}(\Gamma, \mathsf{BC})) \vdash \mathit{Constructor} : \diamond \\ (\Gamma, \emptyset, \mathsf{C}, \mathsf{MNAMES}(\Gamma, \mathsf{BC})) \vdash \mathit{MethodList} : \diamond\end{array}}{(\Gamma, \Gamma_\mathsf{C}) \vdash \mathsf{class\ C\ extends\ BC}\ \{\ \mathit{FieldList}\ \mathit{Constructor}\ \mathit{MethodList}\ \} : \diamond} \tag{2.17}$$

To jestli rodičovská třída existuje je zjišťováno, že se z globálního kontextu pomocí funkce CNAMES vytáhnout názvy všech tříd.

$$\mathsf{CNAMES} : \mathsf{CLASSES} \to \mathsf{Seq\ C} \tag{2.18}$$
$$\mathsf{CNAMES}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}) \cdot \mathit{Set}\ ) = \mathsf{C} \cup \mathsf{CNAMES}(\mathit{Set})$$

Pro otypování FieldList a Constructor je rovněž potřeba získat seznam polí v rodičích. Na to slouží funkce FNAMES, která potřebuje globální kontext a název třídy, pro které má získat

pole.

$$\mathsf{FNAMES} : (\mathsf{CLASSES}, \mathsf{C}) \to \mathcal{P}(\mathsf{F}) \tag{2.19}$$
$$\mathsf{FNAMES}(\ \epsilon\ ,\ \mathsf{T}\ ) = \{\}$$
$$\mathsf{FNAMES} : (\mathsf{CLASSES}, \mathsf{C}) \to \mathcal{P}(\mathsf{F}) \tag{2.20}$$
$$\mathsf{FNAMES}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}) \cdot Set\ ,\ \mathsf{C}) = \mathsf{FNAMES}(\mathsf{FIELDS})$$
$$\mathsf{FNAMES} : (\mathsf{CLASSES}, \mathsf{C}) \to \mathcal{P}(\mathsf{F}) \tag{2.21}$$
$$\mathsf{FNAMES}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}) \cdot Set\ ,\ \mathsf{T}) = \mathsf{FNAMES}(Set, \mathsf{T})$$
$$\quad pokud\ \mathsf{C}\ a\ \mathsf{T}\ jsou\ rozdilne$$
$$\mathsf{FNAMES} : \mathsf{FIELDS} \to \mathcal{P}(\mathsf{F}) \tag{2.22}$$
$$\mathsf{FNAMES}(\ \epsilon\ ) = \{\}$$
$$\mathsf{FNAMES} : \mathsf{FIELDS} \to \mathcal{P}(\mathsf{F}) \tag{2.23}$$
$$\mathsf{FNAMES}(\ \mathsf{N} : \mathsf{T} \cdot Seq\ ) = \mathsf{N} \cup \mathsf{FNAMES}(Seq)$$

Pro otypování MethodList je rovněž potřeba zjistit názvy metod tříd v rodičovské třídě. K tomu slouží metoda MNAMES, která z globálního kontextu pro danou třídu vrátí seznam jmen všech metod.

$$\mathsf{MNAMES} : (\mathsf{CLASSES}, \mathsf{C}) \to \mathcal{P}(\mathsf{M}) \tag{2.24}$$
$$\mathsf{MNAMES}(\ \epsilon\ ,\ \mathsf{N}\ ) = \{\}$$
$$\mathsf{MNAMES} : (\mathsf{CLASSES}, \mathsf{C}) \to \mathcal{P}(\mathsf{M}) \tag{2.25}$$
$$\mathsf{MNAMES}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}) \cdot Set\ ,\ \mathsf{C}) = \mathsf{MNAMES}(\mathsf{METHODS})$$
$$\mathsf{MNAMES} : (\mathsf{CLASSES}, \mathsf{C}) \to \mathcal{P}(\mathsf{M}) \tag{2.26}$$
$$\mathsf{MNAMES}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}) \cdot Set\ ,\ \mathsf{T}) = \mathsf{MNAMES}(Set, \mathsf{T})$$
$$\quad pokud\ \mathsf{C}\ a\ \mathsf{T}\ jsou\ rozdilne$$
$$\mathsf{MNAMES} : \mathsf{METHODS} \to \mathcal{P}(\mathsf{M}) \tag{2.27}$$
$$\mathsf{MNAMES}(\ \epsilon\ ) = \{\}$$
$$\mathsf{MNAMES} : \mathsf{METHODS} \to \mathcal{P}(\mathsf{M}) \tag{2.28}$$
$$\mathsf{MNAMES}(\ (\mathsf{M} : \mathsf{T}, \mathsf{PARAMS}) \cdot Set\ ) = \mathsf{M} \cup \mathsf{MNAMES}(Set)$$

### 2.3.4 Pole

Pole je definováno svým identifikátorem, což je název třídy, který musí existovat v programu. Následovaný názvem pole. Tento název musí být unikátní v rámci třídy. Jazyk Featherweight Java nepodporuje překrývání polí, které zdědí z rodičovské třídy. Nesmí tedy existovat pole se stejným názvem jako bylo definované v rodičovské třídě.

| | | |
|---|---|---|
| *FieldList* | → | *FieldList Field* |
| | | \| $\epsilon$ |
| *Field* | → | Identifier Identifier ; |

Nejprve je třeba rozebrat FieldList na jednotlivé Field. Je nutné ověřit, že Field a FieldList jsou správně otypovány. Jestliže Field je správně otypován, tak je do kontextu polí přidáno jeho jméno, aby mohlo být ověřeno, že již pole není deklarováno. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{(\Gamma, \Gamma_{\mathsf{F}}, \gamma_{\mathsf{F}}) \vdash \mathit{Field} : \diamond \quad (\Gamma, \Gamma_{\mathsf{F}} \cup \{\mathsf{FNAME}(\mathit{Field})\}, \gamma_{\mathsf{F}}) \vdash \mathit{FieldList} : \diamond}{(\Gamma, \Gamma_{\mathsf{F}}, \gamma_{\mathsf{F}}) \vdash \mathit{Field} \ \cdot \ \mathit{FieldList} : \diamond} \tag{2.29}$$

Jméno pole je zjišťováno pomocí funkce FNAME, která z deklarace pole vrátí jeho jméno.

$$\begin{aligned} &\mathsf{FNAME} : \mathsf{Field} \to \mathsf{F} \\ &\mathsf{FNAME}(\ \mathsf{T}\ \mathsf{N}\ ;\ ) = \mathsf{N} \end{aligned} \tag{2.30}$$

Následně je třeba otypovat konkrétní pole. Je ověřováno, že již neexistuje v kontextu polí a rovněž je ověřeno, že typ pole existuje v globálním kontextu. K tomu slouží metoda CNAMES, která již byla popsána u tříd 2.18. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{\mathsf{N} \notin \Gamma_{\mathsf{F}} \quad \mathsf{N} \notin \gamma_{\mathsf{F}} \quad \mathsf{T} \in \mathsf{CNAMES}(\Gamma)}{(\Gamma, \Gamma_{\mathsf{F}}, \gamma_{\mathsf{F}}) \vdash \mathsf{T}\ \mathsf{N}\ ; : \diamond} \tag{2.31}$$

### 2.3.5 Konstruktor

V jazyku Featherweight Java je povolen pouze jeden konstruktor. Identifikátor musí mít stejný název jako třída, v které je tento konstruktor definován. Následně po identifikátoru jsou parametry, které musí obsahovat parametry pro pole předka a následně parametry pro pole definované v této třídě. Tělo konstruktoru je složeno z volání nadřazeného konstruktoru, kterému jsou předány parametry předka a následně po tomto volání následuje seznam deklaraci jednotlivých polí a parametrů, které byly definovány v této třídě.

| | | |
|---|---|---|
| *Constructor* | → | Identifier ( *ParameterList* ) |
| | | { super ( *ParameterList* ) ; *DeclarationList* } |
| *DeclarationList* | → | *Declaration DeclarationList* |
| | | \| $\epsilon$ |
| *Declaration* | → | this . Identifier = Identifier ; |

V inferenčním pravidle je třeba ověřit, že jméno konstruktoru je stejné jako název třídy. Následně je potřeba otypovat $\mathsf{ParameterList}_A$. Pro jeho otypování je mu do kontextu polí dáno sjednocení polí předka a polí definované v právě otypované třídě. Je potřeba zajistit, že předané parametry odpovídají definovaným polím. Následně je potřeba zajistit, že je dobře otypován $\mathsf{ParameterList}_B$, a to že obsahuje pole z rodiče. Následně je třeba zkontrolovat, že předané parametry přesně sedí, včetně pořadí, definovaným polím. Jako poslední část je třeba otypovat DeclarationList a ověřit, že obsahuje všechny definované pole a sedí parametry názvů. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{\begin{array}{c}\Gamma_{\mathsf{K}} = \mathsf{C} \\ (\Gamma, \emptyset, \gamma_{\mathsf{F}} \cup \mathsf{FNAMES}(\Gamma, \mathsf{C})) \vdash \mathit{ParameterList}_A : \diamond \\ \mathsf{PNAMES}(\mathit{ParameterList}_A) = \gamma_{\mathsf{F}} \cup \mathsf{FNAMES}(\Gamma, \mathsf{C}) \\ (\Gamma, \emptyset, \gamma_{\mathsf{F}}) \vdash \mathit{ParameterList}_B : \diamond \\ \mathsf{PNAMES}(\mathit{ParameterList}_B) \cup \mathsf{FNAMES}(\Gamma, \mathsf{C}) = \gamma_{\mathsf{F}} \cup \mathsf{FNAMES}(\Gamma, \mathsf{C}) \\ (\Gamma, \emptyset, \mathsf{FNAMES}(\Gamma, \mathsf{C})) \vdash \mathit{DeclarationList} : \diamond \quad \mathsf{DNAMES}(\mathit{DeclarationList}) = \mathsf{FNAMES}(\Gamma, \mathsf{C})\end{array}}{(\Gamma, \Gamma_{\mathsf{K}}, \gamma_{\mathsf{F}}) \vdash \mathsf{C}\ (\ \mathit{ParameterList}_A\ )\ \{\ \mathsf{super}\ (\ \mathit{ParameterList}_B\ )\ \mathit{DeclarationList}\ \} : \diamond} \tag{2.32}$$

Ke zjištění jmen polí se používá funkce FNAMES, která již byla popsána u tříd 2.19. Tato funkce z globální kontextu a danou třídu vrátí seznam jmen všech polí.

Ke zjištění názvů parametrů slouží funkce PNAMES. Tato funkce přijímá jako parametr ParameterList, který následně prochází po jednotlivých prvcích a vrací sjednocení jednotlivých jmen parametrů.

$$\mathsf{PNAMES} : \mathsf{ParameterList} \rightarrow \mathcal{P}(\mathsf{P}) \tag{2.33}$$
$$\mathsf{PNAMES}(\ \epsilon\ ) = \{\}$$
$$\mathsf{PNAMES} : \mathsf{ParameterList} \rightarrow \mathcal{P}(\mathsf{P}) \tag{2.34}$$
$$\mathsf{PNAMES}(\ \mathsf{T}\ \mathsf{N}\ \cdot\ \mathit{ParameterList}\ ) = \mathsf{N} \cup \mathsf{PNAMES}(\mathit{ParameterList})$$

Ke zjištění názvů deklarace slouží funkce DNAMES. Tato funkce přijímá jako parametr DeclarationList, který následně prochází po jednotlivých prvcích a vrací sjednocení jednotlivých jmen deklarací. Za jméno deklarace je považováno jméno pole.

$$\mathsf{DNAMES} : \mathsf{DeclarationList} \rightarrow \mathcal{P}(\mathsf{F}) \tag{2.35}$$
$$\mathsf{DNAMES}(\ \epsilon\ ) = \{\}$$
$$\mathsf{DNAMES} : \mathsf{DeclarationList} \rightarrow \mathcal{P}(\mathsf{F}) \tag{2.36}$$
$$\mathsf{DNAMES}(\ \mathsf{this}\ .\ \mathsf{F} = \mathsf{D}\ \cdot\ \mathit{DeclarationList}\ ) = \mathsf{D} \cup \mathsf{DNAMES}(\mathit{DeclarationList})$$

Pro správné otypování ParameterList je potřeba otypovat všechny prvky toho listu - Parameter. Pokud je Parameter správně otypován je přidáno jeho jméno do kontextu parametrů a tím je zajištěno, že neexistuje víc parametrů stejného názvu. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{\begin{array}{c}(\Gamma, \Gamma_{\mathsf{P}}, \Gamma_{\mathsf{F}}) \vdash \mathit{Parameter} : \diamond \\ (\Gamma, \Gamma_{\mathsf{P}} \cup \{\mathsf{PNAME}(\mathit{Parameter})\}, \Gamma_{\mathsf{F}}) \vdash \mathit{ParameterList} : \diamond\end{array}}{(\Gamma, \Gamma_{\mathsf{P}}, \Gamma_{\mathsf{F}}) \vdash \mathit{Parameter}\ \cdot\ \mathit{ParameterList} : \diamond} \tag{2.37}$$

Ke zjištění názvu parametru slouží funkce PNAME ta přijímá Parameter, z kterého je získán název parametru a ten je vrácen.

$$\mathsf{PNAME} : \mathsf{Parameter} \rightarrow \mathsf{P} \tag{2.38}$$
$$\mathsf{PNAME}(\ \mathsf{T}\ \mathsf{N}\ ) = \mathsf{N}$$

Následně je třeba otypovat přímo Parameter. Je třeba ověřit, že název parametru již neexistuje v kontextu parametrů. Tím se ověří, že parametr je unikátní. Následně je potřeba ověřit, že předaný parametr má i pole, do kterého bude přiřazen. Nakonec je potřeba ověřit, že typ parametru skutečně existuje. K tomu slouží funkce CNAMES, která z globálního kontextu získá jména tříd. A zkontroluje, že typ parametru existuje ve jménech tříd. Tato funkce byla popsána v 2.18. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{\mathsf{N} \notin \Gamma_\mathsf{P} \quad \mathsf{N} \in \Gamma_\mathsf{F} \quad \mathsf{T} \in \mathsf{CNAMES}(\Gamma)}{(\Gamma, \Gamma_\mathsf{P}, \Gamma_\mathsf{F}) \vdash \mathsf{T}\ \mathsf{N} : \diamond} \tag{2.39}$$

Dále je potřeba otypovat jednotlivé deklarace definované v konstruktoru. K otypování DeclarationList je potřeba zajistit, že všechny Declaration jsou správně otypovány. Pokud je Declaration správně otypován, tak je jeho název přidán do kontextu deklarací.

$$\frac{(\Gamma, \Gamma_\mathsf{D}, \Gamma_\mathsf{F}) \vdash \mathit{Declaration} : \diamond \qquad (\Gamma, \Gamma_\mathsf{D} \cup \mathsf{DNAME}(\mathit{Declaration}), \Gamma_\mathsf{F}) \vdash \mathit{DeclarationList} : \diamond}{(\Gamma, \Gamma_\mathsf{D}, \Gamma_\mathsf{F}) \vdash \mathit{Declaration}\ \cdot\ \mathit{DeclarationList} : \diamond} \tag{2.40}$$

Ke zjištění názvu deklarace slouží funkce DNAME. Tato funkce přijímá deklaraci a následně z ní vrací jméno. Za jméno deklarace je považován název pole.

$$\begin{aligned} &\mathsf{DNAME} : \mathsf{Declaration} \to \mathsf{F} \\ &\mathsf{DNAME}(\ \mathsf{this}\ .\ \mathsf{F}\ =\ \mathsf{D}\ ) = \mathsf{D} \end{aligned} \tag{2.41}$$

Nakonec je třeba otypovat samotnou deklaraci. Ta je správně otypována pokud název pole se rovná názvu parametru. Název pole existuje v definovaných polích. Také je potřeba ověřit, že název deklarace již není v kontextu deklarací.

$$\frac{\mathsf{F} = \mathsf{P} \quad \mathsf{F} \in \Gamma_\mathsf{F} \quad \mathsf{P} \in \Gamma_\mathsf{F} \quad \mathsf{F} \notin \Gamma_\mathsf{D} \quad \mathsf{P} \notin \Gamma_\mathsf{D}}{(\Gamma, \Gamma_\mathsf{D}, \Gamma_\mathsf{F}) \vdash \mathsf{this}\ .\ \mathsf{F}\ =\ \mathsf{P}\ ;\ : \diamond} \tag{2.42}$$

### 2.3.6 Metody

Metody jsou v jazyce Featherweight Java definovaný jako seznam metod, kde metoda je definována návratovým typem následovaným názvem metody a jejími parametry. V těle metody je klíčové slovo `return` následované výrazem.

| | | |
|---|---|---|
| *MethodList* | → | *Method MethodList* |
| | | \| $\epsilon$ |
| *Method* | → | Identifier Identifier ( *ParameterList* ) |
| | | { return *Expression* ; } |

Nejprve je třeba otypovat MethodList. K jeho otypování je třeba otypovat všechny Method, které obsahuje. Pokud je Method správně otypována, tak její název je přidán do kontextu metod. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{(\Gamma, \Gamma_{\mathsf{M}}, \Gamma_{\mathsf{K}}, \gamma_{\mathsf{M}}) \vdash \mathit{Method} : \diamond \quad (\Gamma, \Gamma_{\mathsf{M}} \cup \{\mathsf{MNAME}(\mathit{Method})\}, \Gamma_{\mathsf{K}}, \gamma_{\mathsf{M}}) \vdash \mathit{MethodList} : \diamond}{(\Gamma, \Gamma_{\mathsf{M}}, \Gamma_{\mathsf{K}}, \gamma_{\mathsf{M}}) \vdash \mathit{Method}\ \cdot\ \mathit{MethodList} : \diamond} \tag{2.43}$$

K zjištění názvu slouží funkce MNAME. Tato funkce přijímá Method a následně vrací název metody.

$$\mathsf{MNAME} : \mathsf{Method} \to \mathsf{M} \tag{2.44}$$
$$\mathsf{MNAME}(\ \mathsf{T}\ \mathsf{M}\ (\ \mathit{ParameterList}\ )\ \{\ \mathsf{return}\ \mathit{Expression}\ ;\}\ ) = \mathsf{M}$$

K otypování metody je potřeba ověřit, že název metody neexistuje v kontextu metod, což slouží k zajištění unikátnosti. Za další je potřeba ověřit, že již metoda není definovaná v rodičovské třídě a to tím, že její název není v kontextu jmen metod rodičovské třídy[2]. Následně je potřeba otypovat ParameterList a zajistit, že v parametrech není parametr s názvem `this`. Nakonec je potřeba otypovat výraz, který je v metodě. Tomuto výrazu jsou do lokálního kontextu předány všechny parametry a proměnná `this`. Rovněž je potřeba ověřit, že daný výraz je buď stejného typu nebo podtypem, návratového typu metody. Toto inferenční pravidlo má ekvivalent v[1]. Je jím T-METHOD ve Fig.2. Některé jeho části byly obsaženy v předchozích inferenčních pravidel.

$$\frac{\begin{array}{c}\mathsf{M} \notin \Gamma_{\mathsf{M}} \quad \mathsf{M} \notin \gamma_{\mathsf{M}} \quad \mathsf{T} \in \mathsf{CNAMES}(\Gamma) \\ (\Gamma, \emptyset) \vdash \mathit{ParameterList} : \diamond \quad \mathsf{this} : \mathsf{T} \notin \mathsf{PLTYPED}(\mathit{ParameterList}) \\ (\Gamma, \mathsf{PLTYPED}(\mathit{ParameterList}) \cup \{\mathsf{this} : \Gamma_{\mathsf{K}}\}) \vdash \mathit{Expression} : \mathsf{T} \\ \mathsf{ETYPE}(\mathit{Expression}) <: \mathsf{T}\end{array}}{(\Gamma, \Gamma_{\mathsf{M}}, \Gamma_{\mathsf{K}}, \gamma_{\mathsf{M}}) \vdash \mathsf{T}\ \mathsf{M}\ (\ \mathit{ParameterList}\ )\ \{\ \mathsf{return}\ \mathit{Expression}\ ;\} : \diamond} \tag{2.45}$$

Ke zjištění jestli daný typ dané metody existuje slouží funkce CNAMES, která přijímá globální kontext. Ta byla popsána v 2.18.

Aby parametry funkce mohly být předány do lokálního kontextu výrazu, tak je potřeba zajistit, že parametry budou ve tvaru Název : Typ a k tomu slouží funkce PLTYPED, která přijímá ParameterList a ta každý Parameter v něm převede do správného stavu.

$$\mathsf{PLTYPED} : \mathsf{ParameterList} \to \mathsf{Seq}\ (\mathsf{P} \times \mathsf{C}) \tag{2.46}$$
$$\mathsf{PLTYPED}(\ \epsilon\ ) = \mathit{Seq}\ \{\}$$
$$\mathsf{PLTYPED} : \mathsf{ParameterList} \to \mathsf{Seq}\ (\mathsf{P} \times \mathsf{C}) \tag{2.47}$$
$$\mathsf{PLTYPED}(\ \mathsf{T}\ \mathsf{N}\ \cdot\ \mathit{ParameterList}\ ) = \mathsf{N} : \mathsf{T} \cdot \mathsf{PLTYPED}(\mathit{ParameterList})$$

### 2.3.7 Parametry

Parametry jsou v jazyce Featherweight Java definované jako list parametrů, kde každý parametr je definovaný jeho typem a následně jeho názvem.

[2]Featherweight Java nepodporuje překrývání metod.

$$
\begin{array}{lcl}
\mathit{ParameterList} & \rightarrow & \mathit{Parameter}\ \mathit{ParameterList} \\
 & & \mid \epsilon \\
\mathit{Parameter} & \rightarrow & \mathsf{Identifier}\ \mathsf{Identifier}
\end{array}
$$

Ke správnému otypování parametrů je třeba otypovat ParameterList. K docílení toho je třeba otypovat každý Parameter, který je v něm obsažen. Jestliže je Parameter správně otypován, tak jeho název je následně předán do kontextu parametrů, aby se zajistila unikátnost parametru. Toto inferenční pravidlo nemá ekvivalent v[1].

$$
\frac{(\Gamma, \Gamma_{\mathsf{P}}) \vdash \mathit{Parameter} : \diamond \quad (\Gamma, \Gamma_{\mathsf{P}} \cup \{\mathsf{PNAME}(\mathit{Parameter})\}) \vdash \mathit{ParameterList} : \diamond}{(\Gamma, \Gamma_{\mathsf{P}}) \vdash \mathit{Parameter}\ \cdot\ \mathit{ParameterList} : \diamond} \tag{2.48}
$$

Ke zjištění názvu parametru slouží funkce PNAME, která přijímá Parameter a byla popsaná v 2.38.

Následně je potřeba otypovat Parameter. Ten je správně otypován pokud název parametru neexistuje v kontextu parametrů a typ parametru existuje v názvu tříd. Toto inferenční pravidlo nemá ekvivalent v[1].

$$
\frac{\mathsf{N} \notin \Gamma_{\mathsf{P}} \quad \mathsf{T} \in \mathsf{CNAMES}(\Gamma)}{(\Gamma, \Gamma_{\mathsf{P}}) \vdash \mathsf{T}\ \mathsf{N} : \diamond} \tag{2.49}
$$

Ke zjištění názvu tříd slouží funkce CNAMES, která přijímá globální kontext. Tato funkce byla popsaná v 2.18.

### 2.3.8 Výrazy

Výrazů v jazyku Featherweight Java je několik druhů.

- Samotný identifikátor, což znamená použití lokální proměnné[3].
- Použití pole nad určitým výrazem.
- Volání metody nad určitým výrazem.
- Volání konstruktoru.
- Přetypování výrazu.

[3]Featherweight Java nepodporuje lokální proměnné. Lokálními proměnnými jsou zde myšleny parametry a výraz this

$$
\begin{array}{lcl}
\mathit{ExpressionList} & \rightarrow & \mathit{Expression}\ \mathit{ExpressionList} \\
 & & \mid \epsilon \\
\mathit{Expression} & \rightarrow & \mathsf{Identifier} \\
 & & \mid \mathit{Expression}\ .\ \mathsf{Identifier} \\
 & & \mid \mathit{Expression}\ .\ \mathsf{Identifier}\ (\ \mathit{ExpressionList}\ ) \\
 & & \mid \mathsf{new}\ \mathsf{Identifier}\ (\ \mathit{ExpressionList}\ ) \\
 & & \mid (\ \mathsf{Identifier}\ )\ \mathit{Expression}
\end{array}
$$

První z výrazu je použití lokální proměnné. Tento výraz je správně otypovaný pokud identifikátor existuje v lokálním kontextu výrazu. Toto inferenční pravidlo má ekvivalent v[1]. Je to pravidlo T-VAR ve Fig. 2.

$$\frac{\mathsf{I} : \mathsf{T} \in \Gamma_{\mathsf{L}}}{(\Gamma, \Gamma_{\mathsf{L}}) \vdash \mathsf{I} : \mathsf{T}} \tag{2.50}$$

Druhý výraz je použití pole nad výrazem. Pro správné otypování tohoto výrazu je potřeba zajistit, že výraz, nad kterým k poli přistupujeme je správně otypovaný, a že daná třída, které je daný výraz obsahuje pole s tímto názvem. Toto inferenční pravidlo má ekvivalent v[1]. Je to pravidlo T-FIELD ve Fig. 2.

$$\frac{\begin{array}{c}(\Gamma, \Gamma_{\mathsf{L}}) \vdash \mathit{Expression} : \mathsf{T}_2 \\ \mathsf{I} \in \mathsf{FNAMES}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_{\mathsf{L}}, \mathit{Expression}))\end{array}}{(\Gamma, \Gamma_{\mathsf{L}}) \vdash \mathit{Expression}\ .\ \mathsf{I} : \mathsf{T}_1} \tag{2.51}$$

Ke zjištění typu výrazu slouží funkce ETYPE. Tato funkce z globálního kontextu a lokálního kontextu pro daný výraz najde typ jakého je. Funkce má několik variant pro všechny druhy výrazů. Tato funkce vždy prochází výraz až k samému začátku až už nejde jít více na začátek.

Ve funkcích je používán speciální kontext - globálně-lokální $\Gamma_{\mathsf{GL}}$. Jedná se o sloučení $\Gamma$ a $\Gamma_{\mathsf{L}}$. To je z důvodu větší přehlednosti. Toto je pouze u funkcí, kde se ani jeden z kontextu nevyužívá.

$$\mathsf{ETYPE} : ((\mathsf{CLASSES}, (\mathsf{P} \cup \mathsf{this}) \times \mathsf{T}, \mathsf{Expression}) \rightarrow \mathsf{C} \tag{2.52}$$
$$\mathsf{ETYPE}(\ \Gamma_{\mathsf{GL}}\ ,\ (\ \mathsf{C}\ )\ \mathit{Expression}\ ) = \mathsf{C}$$
$$\mathsf{ETYPE} : ((\mathsf{CLASSES}, (\mathsf{P} \cup \mathsf{this}) \times \mathsf{T}, \mathsf{Expression}) \rightarrow \mathsf{C} \tag{2.53}$$
$$\mathsf{ETYPE}(\ \Gamma_{\mathsf{GL}}\ ,\ \mathsf{new}\ \mathsf{C}\ (\ \mathit{ExpressionList}\ )\ ) = \mathsf{C}$$
$$\mathsf{ETYPE} : ((\mathsf{CLASSES}, (\mathsf{P} \cup \mathsf{this}) \times \mathsf{T}, \mathsf{Expression}) \rightarrow \mathsf{C} \tag{2.54}$$
$$\mathsf{ETYPE}(\ \Gamma\ ,\ \Gamma_{\mathsf{L}}\ ,\ \mathit{Expression}\ .\ \mathsf{M}\ (\ \mathit{ExpressionList}\ )\ ) =$$
$$\quad \mathsf{MTYPE}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_{\mathsf{L}}, \mathit{Expression}), \mathsf{M})$$
$$\mathsf{ETYPE} : ((\mathsf{CLASSES}, (\mathsf{P} \cup \mathsf{this}) \times \mathsf{T}, \mathsf{Expression}) \rightarrow \mathsf{C} \tag{2.55}$$
$$\mathsf{ETYPE}(\ \Gamma\ ,\ \Gamma_{\mathsf{L}}\ ,\ \mathit{Expression}\ .\ \mathsf{I}) =$$
$$\quad \mathsf{FTYPE}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_{\mathsf{L}}, \mathit{Expression}), \mathsf{I})$$
$$\mathsf{ETYPE} : ((\mathsf{CLASSES}, (\mathsf{P} \cup \mathsf{this}) \times \mathsf{T}, \mathsf{Expression}) \rightarrow \mathsf{C} \tag{2.56}$$
$$\mathsf{ETYPE}(\ \Gamma\ ,\ \Gamma_{\mathsf{L}}\ ,\ \mathsf{I}\ ) = \mathsf{T}$$
$$\quad \textit{kde}\ \mathsf{I}\ \textit{je v}\ \Gamma_{\mathsf{L}}\ \textit{a ma typ}\ \mathsf{T}$$

Funkce ETYPE potřebuje znát typ metody, která je volána nad výrazem. K tomuto slouží funkce MTYPE, která z globálního kontextu pro danou třídu a danou metodu vrátí její typ.

$$\mathsf{MTYPE} : (\mathsf{CLASSES}, \mathsf{C}, \mathsf{M}) \to \mathsf{C} \tag{2.57}$$
$$\mathsf{MTYPE}(\ \epsilon\ ,\ \mathsf{T}\ ,\ \mathsf{M}\ ) = \{\}$$
$$\mathsf{MTYPE} : (\mathsf{CLASSES}, \mathsf{C}, \mathsf{M}) \to \mathsf{C} \tag{2.58}$$
$$\mathsf{MTYPE}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}) \cdot Set\ ,\ \mathsf{C}, \mathsf{M}\ ) = \mathsf{MTYPE}(\mathsf{METHODS}, \mathsf{M})$$
$$\mathsf{MTYPE} : (\mathsf{CLASSES}, \mathsf{C}, \mathsf{M}) \to \mathsf{C} \tag{2.59}$$
$$\mathsf{MTYPE}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}) \cdot Set\ ,\ \mathsf{T}, \mathsf{M}\ ) = \mathsf{MTYPE}(Set, \mathsf{T}, \mathsf{M})$$
*pokud* C *a* T *jsou rozdilne*
$$\mathsf{MTYPE} : (\mathsf{METHODS}, \mathsf{M}) \to \mathsf{C} \tag{2.60}$$
$$\mathsf{MTYPE}(\ \epsilon\ , \mathsf{N}\ ) = \{\}$$
$$\mathsf{MTYPE} : (\mathsf{METHODS}, \mathsf{M}) \to \mathsf{C} \tag{2.61}$$
$$\mathsf{MTYPE}(\ (\mathsf{M} : \mathsf{T}, \mathsf{PARAMS})\ \cdot\ Set\ ,\ M\ ) = \mathsf{T}$$
$$\mathsf{MTYPE} : (\mathsf{METHODS}, \mathsf{M}) \to \mathsf{C} \tag{2.62}$$
$$\mathsf{MTYPE}(\ (\mathsf{M} : \mathsf{T}, \mathsf{PARAMS})\ \cdot\ Set\ ,\ N\ ) = \mathsf{MTYPE}(Set, \mathsf{N})$$
*pokud* M *a* N *jsou rozdilne*

Funkce ETYPE potřebuje znát také typ pole, ke které je přistupováno nad výrazem. K tomuto slouží funkce FTYPE, která z globálního kontextu pro danou třídu a dané pole vrátí jeho typ.

$$\mathsf{FTYPE} : (\mathsf{CLASSES}, \mathsf{C}, \mathsf{F}) \to \mathsf{C} \tag{2.63}$$
$$\mathsf{FTYPE}(\ \epsilon\ ,\ \mathsf{T}\ ,\ \mathsf{F}\ ) = \{\}$$
$$\mathsf{FTYPE} : (\mathsf{CLASSES}, \mathsf{C}, \mathsf{F}) \to \mathsf{C} \tag{2.64}$$
$$\mathsf{FTYPE}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}) \cdot Set, \mathsf{C}, \mathsf{F}\ ) = \mathsf{FTYPE}(\mathsf{FIELDS}, \mathsf{F})$$
$$\mathsf{FTYPE} : (\mathsf{CLASSES}, \mathsf{C}, \mathsf{F}) \to \mathsf{C} \tag{2.65}$$
$$\mathsf{FTYPE}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}) \cdot Set, \mathsf{T}, \mathsf{F}\ ) = \mathsf{FTYPE}(Set, \mathsf{T}, \mathsf{F})$$
*pokud* C *a* T *jsou rozdilne*
$$\mathsf{FTYPE} : (\mathsf{CLASSES}, \mathsf{F}) \to \mathsf{C} \tag{2.66}$$
$$\mathsf{FTYPE}(\ \epsilon\ , \mathsf{N}\ ) = \{\}$$
$$\mathsf{FTYPE} : (\mathsf{CLASSES}, \mathsf{F}) \to \mathsf{C} \tag{2.67}$$
$$\mathsf{FTYPE}(\ \mathsf{F} : \mathsf{T}\ \cdot\ Seq\ ,\ \mathsf{F}\ ) = \mathsf{T}$$
$$\mathsf{FTYPE} : (\mathsf{CLASSES}, \mathsf{F}) \to \mathsf{C} \tag{2.68}$$
$$\mathsf{FTYPE}(\ \mathsf{F} : \mathsf{T}\ \cdot\ Seq\ ,\ \mathsf{N}\ ) = \mathsf{FTYPE}(Seq, \mathsf{N})$$
*pokud* C *a* N *jsou rozdilne*

Funkce FNAMES na získání jmen polí ze třídy už byla popsána v 2.19.

Třetí výraz je volání metody nad výrazem. Pro správné otypovaní tohoto výrazu je potřeba zajistit, že výraz, nad kterým se volá metoda je správně otypovaný, a že daná třída, které

je daný typ výrazu tuto metodu obsahuje s tímto názvem. Je třeba rovněž ověřit, že sedí výrazy předané jako parametry a to, že jsou buď stejného typu nebo jsou podtypem. Je rovněž třeba otypovat ExpressionList a zajistit, že každý výraz předávaný jako parametr je správně otypován. Toto inferenční pravidlo má ekvivalent v[1]. Je to pravidlo T-INVK ve Fig. 2.

$$\frac{\begin{array}{c}(\Gamma, \Gamma_{\mathsf{L}}) \vdash \mathit{Expression} : \mathsf{T}_2 \\ (\Gamma, \Gamma_{\mathsf{L}}) \vdash \mathit{ExpressionList} : \diamond \quad \mathsf{M} \in \mathsf{MNAMES}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_{\mathsf{L}}, \mathit{Expression})) \\ \mathsf{ETYPES}(\Gamma, \Gamma_{\mathsf{L}}, \mathit{ExpressionList}) <: \mathsf{MPLTYPES}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_{\mathsf{L}}, \mathit{Expression}), \mathsf{M})\end{array}}{(\Gamma, \Gamma_{\mathsf{L}}) \vdash \mathit{Expression}\ .\ \mathsf{M}\ (\ \mathit{ExpressionList}\ ) : \mathsf{T}_1} \tag{2.69}$$

Na získání typu výrazu slouží funkce ETYPE, která již byla popsáná u jiného výrazu 2.52.

Ke zjištění názvu metod slouží funkce MNAMES, která již byla také popsána dříve 2.24.

Ke zjištění typu výrazů předaného jako parametry volání metody. Slouží funkce ETYPES. Tato metoda zjistí typ každého výrazu v listu a vrátí sekvenci typů. Tato funkce funguje velice podobně jako popisovaná funkce ETYPE a tu jí i tato funkce využívá.

$$\mathsf{ETYPES} : ((\mathsf{CLASSES}, (\mathsf{P} \cup \mathsf{this}) \times \mathsf{T}, \mathsf{ExpressionList}) \to \mathsf{Seq}\ \mathsf{C} \tag{2.70}$$

$$\mathsf{ETYPES}(\ \Gamma_{\mathsf{GL}},\ \epsilon\ ) = \mathit{Seq}\ \{\}$$

$$\mathsf{ETYPES} : ((\mathsf{CLASSES}, (\mathsf{P} \cup \mathsf{this}) \times \mathsf{T}, \mathsf{ExpressionList}) \to \mathsf{Seq}\ \mathsf{C} \tag{2.71}$$

$$\mathsf{ETYPES}(\ \Gamma_{\mathsf{GL}}\ \ \mathit{Expression} \cdot \mathit{ExpressionList}\ ) = \mathsf{ETYPE}(\Gamma_{\mathsf{GL}}, \mathit{Expression}) \cdot \mathsf{ETYPES}(\Gamma_{\mathsf{GL}}, \mathit{ExpressionList})$$

Ke zjištění typů parametru metody slouží funkce MPLTYPES. Tato funkce z globálního kontextu pro danou třídu a metodu vrátí typy její parametrů.

$$\mathsf{MPLTYPES} : (\mathsf{CLASSES}, \mathsf{C}, \mathsf{M}) \to \mathsf{Seq\ C} \tag{2.72}$$
$$\mathsf{MPLTYPES}(\ \epsilon\ ,\ \mathsf{T}\ ,\ \mathsf{M}\ ) = \{\}$$
$$\mathsf{MPLTYPES} : (\mathsf{CLASSES}, \mathsf{C}, \mathsf{M}) \to \mathsf{Seq\ C} \tag{2.73}$$
$$\mathsf{MPLTYPES}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}) \cdot Set\ ,\ \mathsf{C}\ ,\ \mathsf{M}\ ) =$$
$$\quad \mathsf{MPLTYPES}(\mathsf{METHODS}, \mathsf{M})$$
$$\mathsf{MPLTYPES} : (\mathsf{CLASSES}, \mathsf{C}, \mathsf{M}) \to \mathsf{Seq\ C}$$
$$\mathsf{MPLTYPES}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}) \cdot Set\ ,\ \mathsf{T}\ ,\ \mathsf{M}\ ) =$$
$$\quad \mathsf{MPLTYPES}(Set, \mathsf{T}, \mathsf{M})$$
$$\quad pokud\ \mathsf{C}\ a\ \mathsf{T}\ jsou\ rozdilne$$
$$\mathsf{MPLTYPES} : (\mathsf{METHODS}, \mathsf{M}) \to \mathsf{Seq\ C} \tag{2.74}$$
$$\mathsf{MPLTYPES}(\ \epsilon\ , \mathsf{N}\ ) = \{\}$$
$$\mathsf{MPLTYPES} : (\mathsf{METHODS}, \mathsf{M}) \to \mathsf{Seq\ C} \tag{2.75}$$
$$\mathsf{MPLTYPES}(\ (\mathsf{M} : \mathsf{T}, \mathsf{PARAMS})\ \cdot\ Set\ ,\ \mathsf{M}\ ) = \mathsf{TYPES}(\mathsf{PARAMS})$$
$$\mathsf{MPLTYPES} : (\mathsf{METHODS}, \mathsf{M}) \to \mathsf{Seq\ C} \tag{2.76}$$
$$\mathsf{MPLTYPES}(\ (\mathsf{M} : \mathsf{T}, \mathsf{PARAMS})\ \cdot\ Set\ ,\ \mathsf{N}\ ) = \mathsf{MPLTYPES}(Set, \mathsf{N})$$
$$\quad pokud\ \mathsf{M}\ a\ \mathsf{N}\ jsou\ rozdilne$$

Tato funkce používá další funkcí TYPES, která z listu, který má každý prvek ve formátu Název : Typ. Vrátí sekvenci názvu typů.

$$\mathsf{TYPES} : \mathsf{P} \times \mathsf{C} \to \mathsf{Seq\ C} \tag{2.77}$$
$$\mathsf{TYPES}(\ \epsilon\ ) = Seq\ \{\}$$
$$\mathsf{TYPES} : \mathsf{P} \times \mathsf{C} \to \mathsf{Seq\ C} \tag{2.78}$$
$$\mathsf{TYPES}(\ \mathsf{N} : \mathsf{T}\ \cdot\ Seq\ ) = \mathsf{T} \cdot \mathsf{TYPES}(Seq)$$

Čtvrtý výraz je volání konstruktoru. Výraz je správně otypován pokud název konstruktoru existuje v názvech tříd a pokud předané výrazy jako parametry jsou stejného typu jako má konstruktor či jsou podtypem. Toto je zajištěno, že jsou porovnány typy polí, které musí odpovídat parametrům konstruktoru.[4] Dále je potřeba zajistit, že výrazy předávané do konstruktoru jsou správně otypovány. Toto inferenční pravidlo má ekvivalent v[1]. Je to pravidlo T-NEW ve Fig. 2.

$$\frac{\mathsf{C} \in \mathsf{CNAMES}(\Gamma) \quad (\Gamma, \Gamma_\mathsf{L}) \vdash ExpressionList : \diamond \quad \mathsf{ETYPES}(ExpressionList) <: \mathsf{FTYPES}(\Gamma, \mathsf{C})}{(\Gamma, \Gamma_\mathsf{L}) \vdash \mathsf{new\ C}\ (\ ExpressionList\ ) : \mathsf{T}} \tag{2.79}$$

Ke zjištění názvu tříd slouží funkce CNAMES. Ta již byla popsána dříve v 2.18.

Ke zjištění typů z ExpressionList slouží ETYPES, která již byla popsána dříve v 2.70.

[4] Porovnání s typem polí je z důvodu, že potom v globálním kontextu nemusí být informace o konstruktoru.

Ke zjištění typů polí slouží funkce FTYPES. Tato funkce přijímá globální kontext a název třídy a využívá již definovanou funkci TYPES 2.77.

$$\mathsf{FTYPES} : (\mathsf{CLASSES}, \mathsf{C}) \rightarrow \mathsf{Seq\ C} \tag{2.80}$$
$$\mathsf{FTYPES}(\ \epsilon\ ,\ \mathsf{T}\ ) = Seq\ \{\}$$
$$\mathsf{FTYPES} : (\mathsf{CLASSES}, \mathsf{C}) \rightarrow \mathsf{Seq\ C} \tag{2.81}$$
$$\mathsf{FTYPES}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}) \cdot Set\ ,\ \mathsf{C}\ ) = \mathsf{TYPES}(\mathsf{FIELDS})$$
$$\mathsf{FTYPES} : (\mathsf{CLASSES}, \mathsf{C}) \rightarrow \mathsf{Seq\ C} \tag{2.82}$$
$$\mathsf{FTYPES}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}) \cdot Set\ ,\ \mathsf{T}\ ) = \mathsf{FTYPES}(Set, \mathsf{T})$$
*pokud* C *a* T *jsou rozdilne*

Páty a poslední výraz je přetypování výrazu na jiný typ. Je třeba ověřit, že typ, na který je přetypováváno existuje v názvu tříd a dále je třeba ověřit, že daný výraz je podtypem či nadtypem požadovaného typu. Nakonec je třeba zajistit, že přetypovaný výraz je správně otypovaný. Toto inferenční pravidlo má ekvivalent v[1]. Jsou to pravidla T-UCAST a T-DCAST a T-SCAST ve Fig. 2.

$$\frac{\mathsf{C} \in \mathsf{CNAMES}(\Gamma) \quad \mathsf{ETYPE}(Expression) <: \mathsf{C} \quad (\Gamma, \Gamma_\mathsf{L}) \vdash Expression : \mathsf{T}_2}{(\Gamma, \Gamma_\mathsf{L}) \vdash (\ \mathsf{C}\ )\ Expression : \mathsf{T}_1} \tag{2.83}$$

Funkce na zjištění názvu tříd, která přijímá globální kontext byla již popsána dříve v 2.18.

Funkce na zjištění typy výrazu, která přijímá globální kontext a lokální kontext a výraz již byla také popsaná dříve 2.52.

Pokud jsou předávány výrazy jako parametr, tak jsou ve formě ExpressionList. Je tedy potřeba zajistit, že každý Expression v tomto listě je správně otypován. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{(\Gamma, \Gamma_\mathsf{L}) \vdash Expression : \mathsf{T} \quad (\Gamma, \Gamma_\mathsf{L}) \vdash ExpressionList : \diamond}{(\Gamma, \Gamma_\mathsf{L}) \vdash Expression\ \cdot\ ExpressionList : \diamond} \tag{2.84}$$

# Kapitola 3

# Featherweight Generic Java

### 3.0.9 Úvod

Jazyk Featherweight Generic Java je jazyk, který rozšiřuje původní jazyk Featherweight Java o generiku. Jedná se tedy o stejný případ jako když jazyk Java byl rozšířen o generickou Javu. Jazyk zůstává stejně jednoduchý jako předtím, s tím, že teď při vytváření třídy můžeme definovat generické parametry, které následně můžeme použít jako typ pro pole, parametry a metody. Metody rovněž mají možnost si definovat vlastní generické parametry, které jsou inicializovány při volání metody. Generické parametry třídy jsou inicializovány při volání konstruktoru.

Jazyk Featherweight Generic Java je zpětně kompatibilní s jazykem Featherweight Java jako je tomu stejně s jazykem Java a generikou. Program napsaný tedy pro jazyk Featherweight Java půjde otypovat a přeložit i jako program v jazyce Featherweight Generic Java. Naopak to pochopitelně nejde.

## 3.1 Neformální nastínění

V následující části jsou rozebrány všechny aspekty jazyka Featherweight Generic Java[1]. Jazyk je rovněž porovnán se svým předchůdcem Featherweight Java[2].

### 3.1.1 Typy

Novinkou na rozdíl od jazyku FJ jsou typy. Ve FJ typy byly pouze třídy v programu. Tady jsou nově i generické parametry, které jsou definovány pro každou třídu zvlášť. Generické parametry mohou být také definovány pro metody.

### 3.1.2 Třídy

Třídy jsou definovány stejně jako ve FJ. Pouze s tím rozdílem, že pro třídu mohou být definovány generické parametry. Třídy tedy mohou být také bez generických parametrů.

[1] Dále jako FGJ.
[2] Dále jako FJ.

Správný zápis bez parametrů by byl `class C <>`, ale pro zpětnou kompatibilitu špičaté závorky zanedbáváme pokud třída nemá žádné generické parametry.

```
class A <X extends Object, Y extends Object> extends Object { ... }
class B <X extends Object> extends A<X, X> { ... }
class C extends Object { ... }
```

### 3.1.3 Pole

Pole jsou také definovány stejně jako v FJ. Pouze je zde změna, že místo typu, který mohl být pouze název třídy, která existovala v programu. Zde teď může být i generický typ. Viz následující pole.

```
X first;
Y second;
Object third;
B<A<Object,Object>> four;
```

### 3.1.4 Konstruktor

U konstruktoru nastala také pouze drobná změna jako u polí. Parametry dřív měly typ pouze název třídy, která existovala v programu. Teď se již mohou objevit generické parametry. Jinak pro konstruktor platí stejná pravidla, která byla popsána pro FJ konstruktor 2.2.3. Viz následující kód konstruktoru.

```
A(X first, Y second, Object third, B<A<Object,Object>> four) {
  super();
  this.first = first;
  this.second = second;
  this.third = third;
  this.four = four;
}
```

### 3.1.5 Metody

Metody doznaly ve FGJ několik větších změn. Stejně jako třídy teď mohou mít u sebe definice generických parametrů. Lze tedy definovat nové generické parametry, které lze potom třeba použít v návratovém typu. Tyto generické parametry budou nastaveny na konkrétní typ až při volání metody, kdy bude nezbytné definovat typ do špičatých závorek. Zbytek je již potom stejný jako u metody ve FJ 2.2.3. Viz následující kód metod.

```
<Z extends Object> A<Z,Y> setFirst(Z first) {
  return new A(first, this.second);}
}
A<X,X> getSamePair() {
  return new A(this.first, this.first);
}
```

### 3.1.6 Výrazy

Ve FGJ je stejný počet výrazu jako ve FJ. Pouze opět doznaly změn, které se týkají přidání podpory generiky do jazyka. Jinak výrazy se používají na stále stejných místech. To znamená jako samostatný výraz v programu[3] a jako výrazy v metodách.

První výraz je stejný jako ve FJ. Je to použití parametru nebo proměnné `this`. Není zde tedy žádný prostor, kde by se mohla použít generika.

`this.first` nebo `first`

Druhý výraz je přístup k poli. V tomto výraze se také nic nezměnilo a je stejný jako ve FJ. Více o něm v popise o výrazech ve FJ 2.2.5.

```
this.second
```

Třetí výraz je volání metody. V tomto výraze nastala změna v tom, že při volání metody je potřeba nastavit generické parametry na korektní typy. Viz následující volání metody definované dříve 3.1.5. Nastavením parametru na `C`. Říkáme, že všude, kde se předtím vyskytoval generický parametr `Z`. Nyní chceme typ `C`. Viz následující kód.

```
this.setFirst<C>(new C())
```

Čtvrtý výraz je volání konstruktoru. U tohoto výrazu nastala stejná změna jako u volání metody. Musí se při volání konstruktoru nastavit generické parametry na správné typy. Opět dojde k nahrazení generického parametru `X` za `Object` a `Y` za `C`. Všechny metody a pole, které používali generické parametry jsou nyní konkrétního typu. Viz následující kód volání konstruktoru.

```
new A<Object,C>(new Object(), new C())
```

Páty a poslední výraz je přetypování výrazu na jiný výraz. Tento výraz byl také ovlivněn, protože můžeme teď přetypovávat třídy na generické třídy. Je zde tedy třeba dát větší důraz na to, že typy jsou přetypovatelné.

```
(A<C,C>) new B<C>(new C(), new C())
```

Tento výraz je korektní, protože třída `B<X>` dědí od třídy `A<X,Y>`.

[3] Ta má stejný účel jako metoda `main` v jazyku Java.

### 3.1.7 Ukázka celého programu

Pro shrnutí možností jazyka Featherweight Generic Java se podívejte na následující kód, který je přejat z článku[1]. Jedná se o stejnou implementaci páru jako v 2.2.6. Pouze jako typy nejsou použity `Object`, ale generické typy.

```
class A extends Object {
  A() { super(); }
}
class B extends Object {
  B() { super(); }
}
class Pair<X extends Object, Y extends Object> extends Object {
  X fst;
  Y snd;
  Pair(X fst, Y snd) {
    super();
    this.fst = fst;
    this.snd = snd;
  }
  <Z extends Object> Pair<Z,Y> setfst(Z newfst) {
    return new Pair<Z,Y>(newfst, this.snd);
  }
}
```

Tento program má následující výraz, který vytvoří nový pár z `A` a `B` a následně zavolá metodu `setfst` s generickým parametrem `B`. Toto vrátí novou instanci `Pair<B,B>`.

```
new Pair<A,B>(new A(), new B()).setfst<B>(new B())
```

Lze tedy celý výraz zjednodušit a zapsat ho jako:

```
new Pair<B,B>(new B(), new B())
```

Jedná se o totožný výraz. Je to způsobeno tím, že metoda `setfst` vrací `Pair<Z, Y>`. Generický parametr `Z` byl nastaven na `B` a parametr `Y` také.

## 3.2 Formální popis

V následující části jsou rozebrány jednotlivé aspekty FGJ více podrobněji. Je rozebrána gramatika a typový systém jazyka.

### 3.2.1 Množiny

Množiny jsou opět stejné jako ve FJ 2.3.1. Pouze přibyla nová množina pro generické parametry. Ty mohou být jakéhokoliv jména, ale používá se konvence, která říká použití písmenek X, Y, Z či T s čísly. Například T1, T2, T3 atd. Množiny jsou potřeba, aby bylo možné popsat jakého typu jsou množiny v kontextech. Také je to potřeba pro funkce, kde to slouží k zapsání z jaké množiny do jaké množiny se provádí převod.

$\mathsf{C}$ = Množina všech možných jmen tříd, které existují.
$\mathsf{F}$ = Množina všech možných jmen polí, které existují.
$\mathsf{P}$ = Množina všech možných jmen parametrů, které existují.
$\mathsf{M}$ = Množina všech možných jmen metod, které existují.
$\mathsf{G}$ = Množina všech možných jmen generický parametrů, které existují.

### 3.2.2 Kontexty

Kontexty jsou téměř stejné jako ve FJ a plní i stejnou činnost. Více o nich je zmíněno ve FJ 2.3.2. Ve FGJ jsou přidány dva nové kontexty. Jeden nový kontext, který je globální v rámci třídy slouží k ukládá informací o generických parametrech a jejich hranicích [4] a druhý kontext slouží pro ukládání generických parametrů a k jejich správnému otypování.

| | |
|---|---|
| $\Gamma \subseteq (\mathsf{C}, \mathsf{C}, \mathsf{F} \times \mathsf{C}, (\mathsf{M}, \mathsf{P} \times \mathsf{C}))$ | (globální kontext programu) |
| $\Gamma_\mathsf{L} \subseteq (\{\mathsf{this}\} \cup \mathsf{P}) \times \mathsf{C}$ | (lokalní kontext výrazu) |
| $\Gamma_\mathsf{P} \subseteq \mathsf{P}$ | (kontext jmen parametrů) |
| $\Gamma_\mathsf{M} \subseteq \mathsf{M}$ | (kontext jmen metod) |
| $\gamma_\mathsf{M} \subseteq \mathsf{M}$ | (kontext jmen metod rodiče) |
| $\Gamma_\mathsf{D} \subseteq \mathsf{F}$ | (kontext jmen deklarací) |
| $\Gamma_\mathsf{F} \subseteq \mathsf{F}$ | (kontext jmen polí rodiče) |
| $\gamma_\mathsf{F} \subseteq \mathsf{F}$ | (kontext jmen polí rodiče) |
| $\Gamma_\mathsf{K} \in \mathsf{C}$ | (kontext jména třídy) |
| $\Gamma_\mathsf{C} \subseteq \mathsf{C}$ | (kontext jmen tříd) |
| $\Delta \subseteq \mathsf{G} \times \mathsf{C}$ | (třídní kontext generických parametrů) |
| $\Gamma_\mathsf{G} \subseteq \mathsf{G}$ | (kontext generických parametrů) |

### 3.2.3 Třídy

Program v jazyku FGJ je stejný jako program ve FJ. Je tvořen třídami a výrazem. Každá třída je tvořena klíčovým slovem `class`. Po něm následuje identifikátor třídy. Dále je zde seznam generických parametrů. To je novinka o proti FJ. Tyto generické parametry slouží jako zástupné typy, které jsou nastaveny až při inicializaci třídy. Správně by se měly psát hranaté závorky, když třída nemá žádné generické parametry, ale pro jednoduchost toto zanedbáváme. Zbytečně by to zkomplikovalo gramatiku a typový systém. Po generických parametrech je v definici třídy dále rodičovská třída. Tato třída může být generická. Zbytek už je tělo třídy - pole, konstruktor a metody.

[4] Každý generický parametr má své hranice uvozené klíčovým slovem `extends` to označuje jakého podtypu musí být daný typ.

$$
\begin{array}{lcl}
\mathit{Program} & \rightarrow & \mathit{ClassList}\ \mathit{Expression} \\
\mathit{ClassList} & \rightarrow & \mathit{Class}\ \mathit{ClassList} \\
 & & \mid \epsilon \\
\mathit{Class} & \rightarrow & \mathsf{class\ Identifier} < \mathit{GenericParameterList} > \mathsf{extends} \\
 & & \mathit{TypeExpression}\ \{\ \mathit{FieldList}\ \mathit{Constructor}\ \mathit{MethodList}\ \}
\end{array}
$$

Pro otypování programu je nutno otypovat všechny třídy a samostatný výraz. Také je nutno ověřit neexistenci cyklů v programu. Toto je stejné jak ve FJ, tak i v FGJ. Toto inferenční pravidlo nemá ekvivalent v[1].

$$
\frac{\begin{array}{c}\{(\mathsf{CLASSES}(\mathit{ClassList}), \mathit{Object})\} \vdash \mathit{ClassList} : \diamond \\ (\mathsf{CLASSES}(\mathit{ClassList}), \emptyset, \emptyset) \vdash \mathit{Expression} : \diamond \\ \mathsf{GRAPH}(\mathsf{CLASSES}(\mathit{ClassList}))\ \mathit{neobsahuje\ cykly}\end{array}}{\vdash \mathit{ClassList}\ \mathit{Expression} : \diamond} \tag{3.1}
$$

Funkce CLASSES pro vytvoření globálního kontextu se liší od té, kterou používá FJ. Konkrétně do kontextu ted' byly přidány generické parametry. Je tedy třeba funkci upravit, aby tuto skutečnost brala v potaz. V kontextu jsou u tříd generické parametry.

$$
\mathsf{CLASSES} : \mathsf{ClassList} \rightarrow \mathcal{P}(\mathsf{C},\ \mathsf{C},\ \mathsf{FIELDS},\ \mathsf{METHODS}) \tag{3.2}
$$

$$
\mathsf{CLASSES}(\ \epsilon\ ) = \{\}
$$

$$
\mathsf{CLASSES} : \mathsf{ClassList} \rightarrow \mathcal{P}(\mathsf{C},\ \mathsf{C},\ \mathsf{FIELDS},\ \mathsf{METHODS}) \tag{3.3}
$$

$$
\begin{array}{l}
\mathsf{CLASSES}(\mathsf{class\ C} < \mathit{GenericParameterList} > \mathsf{extends}\ \mathit{TypeExpression} \\
\quad (\{\mathit{FieldList}\ \mathit{Constructor}\ \mathit{MethodList}\} \cdot \mathit{ClassList}) = \\
\quad (\mathsf{C}, \mathsf{TTYPE}(\mathit{TypeExpression}), \mathsf{FIELDS}(\mathit{FieldList}), \mathsf{METHODS}(\mathit{MethodList}), \\
\quad \mathsf{GPARAMS}(\mathit{GenericParameterList})) \cup \mathsf{CLASSES}(\mathit{ClassList})
\end{array}
$$

Jakmile jsou jednotlivé prvky třídy (název třídy, název rodiče, pole a metody) rozebrány následuje vytváření kontextu pro pole a to pomocí funkce FIELDS, která přijímá FieldList a následně jej rozebírá na jednotlivá pole. Na jméno pole a jeho typ.

$$
\mathsf{FIELDS} : \mathsf{FieldList} \rightarrow \mathsf{Seq}\ (\mathsf{N} : \mathsf{Type}) \tag{3.4}
$$

$$
\mathsf{FIELDS}(\ \epsilon\ ) = \mathit{Seq}\ \{\}
$$

$$
\mathsf{FIELDS} : \mathsf{FieldList} \rightarrow \mathsf{Seq}\ (\mathsf{N} : \mathsf{Type}) \tag{3.5}
$$

$$
\mathsf{FIELDS}(\ \mathit{Type}\ \mathsf{N}\ ; \cdot \mathit{FieldList}\ ) = \mathsf{N} : \mathit{Type} \cdot \mathsf{FIELDS}(\mathit{FieldList})
$$

Po polích následuje rozebírání metod a to pomocí metody METHODS, která přijímá MethodList, který následně převede na n-tici, která obsahuje jméno metody a její typ a její

parametry.

$$\mathsf{METHODS} : \mathsf{MethodList} \rightarrow \mathcal{P}(\mathsf{M} \times \mathsf{Type}, \mathsf{PARAMS}, \mathsf{GPARAMS}) \tag{3.6}$$
$$\mathsf{METHODS}(\ \epsilon\ ) = \{\}$$
$$\mathsf{METHODS} : \mathsf{MethodList} \rightarrow \mathcal{P}(\mathsf{M} \times \mathsf{Type}, \mathsf{PARAMS}, \mathsf{GPARAMS}) \tag{3.7}$$
$$\mathsf{METHODS}(\ <\ GenericParameterList\ >\ Type\ \mathsf{M}\ (\ ParameterList\ )$$
$$\{\ \mathsf{return}\ Expression\ ;\} \cdot MethodList\ ) =$$
$$(\mathsf{M} : Type, \mathsf{PARAMS}(ParameterList)$$
$$(\mathsf{GPARAMS}(GenericParameterList)) \cup \mathsf{METHODS}(MethodList)$$

Další část vytváření globální kontextu se vytváří z parametrů, které jsou součástí metod. Funkce PARAMS přijímá ParameterList, který následně převede na sekvenci jméno parametru a jeho typ a rovněž k němu nově ve FGJ přidá generické parametry.

$$\mathsf{PARAMS} : \mathsf{ParameterList} \rightarrow \mathsf{Seq}\ (\mathsf{P} \times \mathsf{Type}) \tag{3.8}$$
$$\mathsf{PARAMS}(\ \epsilon\ ) = Seq\ \{\}$$
$$\mathsf{PARAMS} : \mathsf{ParameterList} \rightarrow \mathsf{Seq}\ (\mathsf{P} \times \mathsf{Type}) \tag{3.9}$$
$$\mathsf{PARAMS}(\ Type\ \mathsf{N} \cdot ParameterList\ ) = \mathsf{N} : \mathsf{Type} \cdot \mathsf{PARAMS}(ParameterList)$$

Pro třídy a metody je potřeba do kontextu přidat také generické parametry na to slouží funkce GPARAMS, která vrátí generické parametry ve tvaru G : TypeExpression.

$$\mathsf{GPARAMS} : \mathsf{GenericParameterList} \rightarrow \mathsf{Seq}\ (\mathsf{G} \times \mathsf{TypeExpression}) \tag{3.10}$$
$$\mathsf{GPARAMS}(\ \epsilon\ ) =\ Seq\{\}$$
$$\mathsf{GPARAMS} : \mathsf{GenericParameterList} \rightarrow \mathsf{Seq}\ (\mathsf{G} \times \mathsf{TypeExpression}) \tag{3.11}$$
$$\mathsf{GPARAMS}(\ \mathsf{C}\ \mathsf{extends}\ TypeExpression \cdot GenericParameterList\ ) =$$
$$\mathsf{G} : \mathsf{TypeExpression} \cdot \mathsf{GPARAMS}(GenericParameterList)$$

Funkce TTYPE slouží k získání jména typy z generického typu. V kontextu není potřeba ukládat generický typ, protože pro účely vytvoření grafu a jiných účelů je potřeba pouze název třídy.

$$\mathsf{TTYPE} : \mathsf{TypeExpression} \rightarrow \mathsf{C} \tag{3.12}$$
$$\mathsf{TTYPE}(\ \mathsf{C}\ <\ TypeList\ >\ ) =\ \mathsf{C}$$
$$\mathsf{TTYPE} : \mathsf{G} \rightarrow \mathsf{G} \tag{3.13}$$
$$\mathsf{TTYPE}(\ \mathsf{X}\ ) =\ \mathsf{X}$$

Pro potřeby FJ bylo definováno spousta funkcí, které používají globální kontext FJ či část gramatiky FJ. Funkce by byly téměř stejné, jen by v nich byly navíc generické parametry, které tyto funkce nevyužívají. Budou tedy používány funkce z FJ s tím, že generické parametry budou ignorovat. Neměl by být problém pro čtenáře si domyslet jak by to vypadalo kdyby tam generické parametry byly.

Také jsou definovány pomocné aliasy, které zjednodušují práci při odkazovaní se pouze na určitou část kontextu.

CLASSES : (C, C, FIELDS, METHODS, GPARAMS)
FIELDS : (F $\times$ C)
METHODS : (M $\times$ C, PARAMS, GPARAMS)
PARAMS : (P $\times$ Type)
GPARAMS : (G $\times$ TypeExpression)

Funkce GRAPH, která vytvoří graf, který je nutný pro zkontrolování cyklický závislostí. Tato funkce byla definována v 2.10.

Pro správné otypování je potřeba otypovat všechny třídy. Všechny třídy jsou správně otypovány pokud každá třída je správně otypována. Pokud je třída správně otypována, tak je přidána do kontextu tříd. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{(\Gamma, \Gamma_{\mathsf{C}}) \vdash Class : \diamond \quad (\Gamma, \Gamma_{\mathsf{C}} \cup \{\mathsf{CNAME}(Class)\}) \vdash ClassList : \diamond}{(\Gamma, \Gamma_{\mathsf{C}}) \vdash Class \ \cdot \ ClassList : \diamond} \tag{3.14}$$

Funkce CNAME pro zjištění názvu třídy, která přijímá Class již byla definována v FJ. Viz 2.16.

Nakonec je třeba ověřit, že samotná třída je správně otypována. Je potřeba zajistit, že název třídy ještě neexistuje, a že generický typ, z kterého třída dědí je správně otypován. Také je potřeba ověřit, že generické parametry třídy jsou správně otypovány. Dále je potřeba zajistit, že FieldList, Constructor a MethodList jsou správně otypovány. Toto inferenční pravidlo má ekvivalent v[1]. Jeho ekvivalentem je GT-CLASS ve Fig. 7.

$$\frac{\begin{array}{c}\mathsf{C} \notin \Gamma_{\mathsf{C}} \quad (\Gamma, \mathsf{GPARAMS}(GPL)) \vdash TypeExpression : \diamond \\ (\Gamma, \mathsf{GPARAMS}(GPL), \emptyset) \vdash GenericParameterList : \diamond \\ (\Gamma, \mathsf{GPARAMS}(GPL), \emptyset, \mathsf{FNAMES}(\Gamma, \mathsf{TTYPE}(TypeExpression)) \vdash FieldList : \diamond \\ (\Gamma, \mathsf{GPARAMS}(GPL), \mathsf{C}, \mathsf{FNAMES}(\Gamma, \mathsf{TTYPE}(TypeExpression)) \vdash Constructor : \diamond \\ (\Gamma, \mathsf{GPARAMS}(GPL), \emptyset, \mathsf{C}, \mathsf{MNAMES}(\Gamma, \mathsf{TTYPE}(TypeExpression)) \vdash MethodList : \diamond\end{array}}{(\Gamma, \Gamma_{\mathsf{C}}) \vdash \mathsf{class\ C} \ < \ GPL \ > \ \mathsf{extends}\ TypeExpression \ \{ \ FL \ C \ ML \ \} : \diamond} \tag{3.15}$$

Pro zmenšení velikost inferenčního pravidla byl GenericParameterList zapsán jako GPL. FieldList jako FL, Constructor jako C a MethodList jako ML.

Funkce GPARAMS pro vytvoření listu generických parametrů, která přijímá GenericParameterList byla již popsána dříve při vytváření globálního kontextu v 3.10.

Funkce FNAMES pro získání jmen polí ve třídě, která přijímá globální kontext a název třídy již byla definována ve FJ. Viz 2.19.

Funkce MNAMES pro získání jmen metod ve třídě, která přijímá globální kontext a název třídy již byla definována ve FJ. Viz 2.24.

### 3.2.4 Generické parametry

Generické parametry se vyskytují u tříd a metod. Generický parametr je definován pomocí generického názvu následovaný klíčovým slovem `extends`, za kterým je generický typ, který udává typovou hranici pro výraz. To znamená, že namístě generické názvu může být pouze typ, který je podtypem generického názvu nebo je stejného typu.

$$\begin{array}{lcl} \mathit{GenericParameterList} & \rightarrow & \mathit{GenericParameterList}\ \mathit{GenericParameter} \\ & & \mid \epsilon \\ \mathit{GenericParameter} & \rightarrow & \mathsf{Identifier}\ \mathsf{extends}\ \mathit{TypeExpression} \end{array}$$

Generické parametry jsou správně otypovány pokud každý generický parametr v listu je správně otypován. Jestliže je správně otypován, tak jeho název je přidán do kontextu generických parametrů. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{(\Gamma, \Delta, \Gamma_{\mathsf{G}}) \vdash \mathit{GenericParameter} : \diamond \quad (\Gamma, \Delta, \Gamma_{\mathsf{G}} \cup \{\mathsf{GNAME}(\mathit{GenericParameter})\}) \vdash \mathit{GenericParameterList} : \diamond}{(\Gamma, \Delta, \Gamma_{\mathsf{G}}) \vdash \mathit{GenericParameter}\ \cdot\ \mathit{GenericParameterList} : \diamond} \tag{3.16}$$

Funkce GNAME na zjištění názvu generického parametru přijímá GenericParameter.

$$\begin{array}{l} \mathsf{GNAME} : \mathsf{GenericParameter} \rightarrow \mathsf{G} \\ \mathsf{GNAME}(\ \mathsf{X}\ \mathsf{extends}\ \mathit{TypeExpression}\ ) = \mathsf{C} \end{array} \tag{3.17}$$

Následně je potřeba otypovat GenericParameter. Ten je správně otypován pokud jeho název neexistuje v kontextu generických parametrů a rovněž neexistuje jako třída v programu a pokud je správně otypován TypeExpression. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{\mathsf{X} \notin \Gamma_{\mathsf{G}} \quad \mathsf{X} \notin \mathsf{CNAMES}(\Gamma) \quad (\Gamma, \Delta) \vdash \mathit{TypeExpression} : \diamond}{(\Gamma, \Delta, \Gamma_{\mathsf{G}}) \vdash \mathsf{X}\ \mathsf{extends}\ \mathit{TypeExpression}\ : \diamond} \tag{3.18}$$

Třída CNAMES na získání jmen všech tříd, která přijímá globální kontext již byla definována. Viz 2.18.

### 3.2.5 Typy

O proti FJ, kde typy byly pouze názvy tříd, ve FGJ typy mohou být generické parametry nebo konkrétní třídy at' už s generickými parametry nebo bez nich. Jsou tedy dva druhy typů.

$$\begin{array}{lcl} \mathit{TypeList} & \rightarrow & \mathit{Type}\ \mathit{TypeList} \\ & & \mid \epsilon \\ \mathit{Type} & \rightarrow & \mathsf{Identifier} \\ & & \mid \mathit{TypeExpression} \\ \mathit{TypeExpression} & \rightarrow & \mathsf{Identifier} < \mathit{TypeList} > \end{array}$$

Typy jsou správně otypovány pokud každý typ v listu je správně otypován.

$$\frac{(\Gamma, \Delta) \vdash Type : \diamond \quad (\Gamma, \Gamma_{\mathsf{G}}) \vdash TypeList : \diamond}{(\Gamma, \Delta) \vdash Type \ \cdot \ TypeList : \diamond} \tag{3.19}$$

První typ je použití generického parametru. Generický parametr je správně otypován pokud existuje v názvech generických parametrů.

$$\frac{\mathsf{X} \in \mathsf{GNAMES}(\Delta)}{(\Gamma, \Delta) \vdash \mathsf{X} : \diamond} \tag{3.20}$$

Funkce GNAMES na zjištění názvu generických parametrů, která přijímá třídní kontext generických parametrů.

$$\mathsf{GNAMES} : (\mathsf{G} \times \mathsf{TypeExpression}) \rightarrow \mathcal{P}(\mathsf{G}) \tag{3.21}$$
$$\mathsf{GNAMES}(\ \epsilon\ ) = \ \{\}$$
$$\mathsf{GNAMES} : (\mathsf{G} \times \mathsf{TypeExpression}) \rightarrow \mathcal{P}(\mathsf{G}) \tag{3.22}$$
$$\mathsf{GNAMES}(\ \mathsf{G} : TypeExpression \cdot Seq\ ) = \ \mathsf{G} \cup \mathsf{GNAMES}(Seq)$$

Druhý typ je konkrétní třída s nastavenými generickými parametry. Tento typ je správně otypován pokud třída existuje v názvech tříd a seznam typů v generických parametrech je správně otypován. Také typy v seznamu typů musí být podtypem hraničních typů v generických parametrech.

$$\frac{\begin{array}{c}(\Gamma, \Delta) \vdash TypeList : \diamond \quad \mathsf{C} \in \mathsf{CNAMES}(\Gamma) \\ \mathsf{TPARAMS}(TypeList) <: \mathsf{GBOUNDS}(\Gamma, \mathsf{C})\end{array}}{(\Gamma, \Delta) \vdash \mathsf{C} \ < \ TypeList \ > : \diamond} \tag{3.23}$$

Funkce CNAMES vrací názvy tříd z globálního kontextu. Byla již definována ve FJ. Viz 2.18.

Funkce TPARAMS vrací generické typy z globálního kontextu a názvu třídy.

$$\mathsf{TPARAMS} : \mathsf{TypeList} \rightarrow \mathsf{Seq}\ (\mathsf{C}) \tag{3.24}$$
$$\mathsf{TPARAMS}(\ \epsilon\ ) = \ Seq\{\}$$
$$\mathsf{TPARAMS} : \mathsf{TypeList} \rightarrow \mathsf{Seq}\ (\mathsf{C})$$
$$\mathsf{TPARAMS}(\ Type \ \cdot \ TypeList\ ) = \mathsf{TTYPE}(Type) \cdot \mathsf{TPARAMS}(ParameterList)$$

Funkce GBOUNDS vrací generické typy z globálního kontextu a názvu třídy.

$$\mathsf{GBOUNDS} : (\mathsf{CLASSES}, \mathsf{C}) \rightarrow \mathsf{Seq}\ (\mathsf{TypeExpression}) \tag{3.25}$$
$$\mathsf{GBOUNDS}(\ \epsilon\ ,\ \mathsf{T}\ ) = \{\}$$
$$\mathsf{GBOUNDS} : (\mathsf{CLASSES}, \mathsf{C}) \rightarrow \mathsf{Seq}\ (\mathsf{TypeExpression}) \tag{3.26}$$
$$\mathsf{GBOUNDS}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}, \mathsf{GPARAMS}) \cdot Set\ ,\ \mathsf{C}) = \mathsf{GBOUNDS}(\mathsf{GPARAMS})$$
$$\mathsf{GBOUNDS} : (\mathsf{CLASSES}, \mathsf{C}) \rightarrow \mathsf{Seq}(\mathsf{TypeExpression}) \tag{3.27}$$
$$\mathsf{GBOUNDS}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}) \cdot Set\ ,\ \mathsf{T}) = \mathsf{GBOUNDS}(Set, \mathsf{T})$$
$$\quad pokud\ \mathsf{C}\ a\ \mathsf{T}\ jsou\ rozdilne$$
$$\mathsf{GBOUNDS} : \mathsf{GPARAMS} \rightarrow \mathsf{Seq}(\mathsf{TypeExpression}) \tag{3.28}$$
$$\mathsf{GBOUNDS}(\ \epsilon\ ) = \{\}$$
$$\mathsf{GBOUNDS} : \mathsf{GPARAMS} \rightarrow \mathsf{Seq}(\mathsf{TypeExpression}) \tag{3.29}$$
$$\mathsf{GBOUNDS}(\ \mathsf{G} : TypeExpression \cdot Seq\ ) = \ TypeExpression \cdot \mathsf{GBOUNDS}(Seq)$$

### 3.2.6 Pole

Pole jsou ve FGJ definovány stejně jako v FJ. Opět je zde změna, že typ pole může být i generický parametr.

$$\begin{array}{lcl} FieldList & \rightarrow & FieldList\ Field \\ & & \mid \epsilon \\ Field & \rightarrow & Type\ \mathsf{Identifier}\ ; \end{array}$$

Pole jsou správně otypovány pokud každé pole v listu je správně otypováno. Pokud je pole otypováno správně, tak je jeho název přidán do kontextu polí. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{(\Gamma, \Delta, \Gamma_{\mathsf{F}}, \gamma_{\mathsf{F}}) \vdash Field : \diamond \quad (\Gamma, \Delta, \Gamma_{\mathsf{F}} \cup \{\mathsf{FNAME}(Field)\}, \gamma_{\mathsf{F}}) \vdash FieldList : \diamond}{(\Gamma, \Delta, \Gamma_{\mathsf{F}}, \gamma_{\mathsf{F}}) \vdash Field\ \cdot\ FieldList : \diamond} \tag{3.30}$$

Funkce FNAME vracející název pole, která přijímá Field byla definování již dřív v FJ. Viz 2.30

Konkrétní pole je správně otypováno pokud název pole neexistuje v kontextu polí rodičovské třídy a také již neexistuje v kontextu polí. Také je potřeba ověřit, že typ je správně otypovaný. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{\mathsf{N} \notin \Gamma_{\mathsf{F}} \quad \mathsf{N} \notin \gamma_{\mathsf{F}} \quad (\Gamma, \Delta) \vdash \mathsf{Type} : \diamond}{(\Gamma, \Delta, \Gamma_{\mathsf{F}}, \gamma_{\mathsf{F}}) \vdash \mathsf{Type\ N\ ;} : \diamond} \tag{3.31}$$

### 3.2.7 Konstruktor

Konstruktoru se podpora generiky nedotkla téměř vůbec. Platí tedy pro něj to samé, co o něm bylo napsáno ve FJ. Viz 2.3.5.

| | | |
|---|---|---|
| *Constructor* | → | Identifier ( *ParameterList* ) |
| | | { super ( *ParameterList* ) ; *DeclarationList* } |
| *DeclarationList* | → | *Declaration DeclarationList* |
| | | \| $\epsilon$ |
| *Declaration* | → | this . Identifier = Identifier ; |

Konstruktor je správně otypovaný pokud název konstruktoru je stejný jako název třídy, v které je. Pokud $\mathsf{ParameterList}_A$ a $\mathsf{ParameterList}_A$ jsou správně otypovány a také pokud parametry odpovídají definovaným polím. Rovněž musí být správně otypovaný i DeclarationList a seznam jmen deklarací musí odpovídat seznamu jmen polí.Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{\begin{array}{c}\Gamma_{\mathsf{K}} = \mathsf{C} \\ (\Gamma, \Delta, \emptyset, \gamma_{\mathsf{F}} \cup \mathsf{FNAMES}(\Gamma, \mathsf{C})) \vdash \mathit{ParameterList}_A : \diamond \\ \mathsf{PNAMES}(\mathit{ParameterList}_A) = \gamma_{\mathsf{F}} \cup \mathsf{FNAMES}(\Gamma, \mathsf{C}) \\ (\Gamma, \Delta, \emptyset, \gamma_{\mathsf{F}}) \vdash \mathit{ParameterList}_B : \diamond \\ \mathsf{PNAMES}(\mathit{ParameterList}_B) \cup \mathsf{FNAMES}(\Gamma, \mathsf{C}) = \gamma_{\mathsf{F}} \cup \mathsf{FNAMES}(\Gamma, \mathsf{C}) \\ (\Gamma, \emptyset, \mathsf{FNAMES}(\Gamma, \mathsf{C})) \vdash \mathit{DeclarationList} : \diamond \quad \mathsf{DNAMES}(\mathit{DeclarationList}) = \mathsf{FNAMES}(\Gamma, \mathsf{C})\end{array}}{(\Gamma, \Delta, \Gamma_{\mathsf{K}}, \gamma_{\mathsf{F}}) \vdash \mathsf{C}\ (\ \mathit{ParameterList}_A\ )\ \{\ \mathsf{super}\ (\ \mathit{ParameterList}_B\ )\ \mathit{DeclarationList}\ \} : \diamond} \tag{3.32}$$

Funkce FNAMES z globálního kontextu a názvu třídy vrací seznam jmen polí. Byla již definována ve FJ. Viz 2.19.

Funkce PNAMES z ParameterList vrací seznam jmen parametrů. Byla již definována ve FJ. Viz 2.33.

Funkce DNAMES z DeclarationList vrací seznam jmen parametrů. Byla již definována ve FJ. Viz 2.35.

Parametry jsou správně otypovány pokud každý parametr v listu je správně otypován. Rovněž pokud je parametr správně otypován jeho název se přidá do kontextu parametrů. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{\begin{array}{c}(\Gamma, \Delta, \Gamma_{\mathsf{P}}, \Gamma_{\mathsf{F}}) \vdash \mathit{Parameter} : \diamond \\ (\Gamma, \Delta, \Gamma_{\mathsf{P}} \cup \{\mathsf{PNAME}(\mathit{Parameter})\}, \Gamma_{\mathsf{F}}) \vdash \mathit{ParameterList} : \diamond\end{array}}{(\Gamma, \Delta, \Gamma_{\mathsf{P}}, \Gamma_{\mathsf{F}}) \vdash \mathit{Parameter}\ \cdot\ \mathit{ParameterList} : \diamond} \tag{3.33}$$

Funkce PNAME vrací jméno parametru. Tato funkce přijímá Parameter. Byla již definována ve FJ. Viz 2.38.

Parametr je správně otypován pokud název není v kontextu parametrů a existuje pole, které

má stejný název jako parametr a typ parametru je správně otypován. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{\mathsf{N} \notin \Gamma_\mathsf{P} \quad \mathsf{N} \in \Gamma_\mathsf{F} \quad (\Gamma, \Delta) \vdash \mathit{Type} : \diamond}{(\Gamma, \Delta, \Gamma_\mathsf{P}, \Gamma_\mathsf{F}) \vdash \mathit{Type}\ \mathsf{N} : \diamond} \tag{3.34}$$

Deklarace jsou správně otypovány pokud každá deklarace v listu je správně otypována. Pokud je správně otypována, tak její název je přidán do kontextu deklarací. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{\begin{array}{c}(\Gamma, \Gamma_\mathsf{D}, \Gamma_\mathsf{F}) \vdash \mathit{Declaration} : \diamond \\ (\Gamma, \Gamma_\mathsf{D} \cup \mathsf{DNAME}(\mathit{Declaration}), \Gamma_\mathsf{F}) \vdash \mathit{DeclarationList} : \diamond\end{array}}{(\Gamma, \Gamma_\mathsf{D}, \Gamma_\mathsf{F}) \vdash \mathit{Declaration}\ \cdot\ \mathit{DeclarationList} : \diamond} \tag{3.35}$$

Funkce DNAME vrací jméno deklarace. Tato funkce přijímá Declaration. Za jméno deklarace je považován název pole. Byla již definována ve FJ. Viz 2.41.

Deklarace je správně otypována. Pokud jméno pole je stejné jako jméno parametru. Pokud existuje pole daného názvu a existuje parametr daného názvu. Také pokud deklarace již nebyla deklarovaná dříve. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{\mathsf{F} = \mathsf{P} \quad \mathsf{F} \in \Gamma_\mathsf{F} \quad \mathsf{P} \in \Gamma_\mathsf{F} \quad \mathsf{F} \notin \Gamma_\mathsf{D} \quad \mathsf{P} \notin \Gamma_\mathsf{D}}{(\Gamma, \Gamma_\mathsf{D}, \Gamma_\mathsf{F}) \vdash \mathsf{this}\ .\ \mathsf{F}\ =\ \mathsf{P}\ ;\ : \diamond} \tag{3.36}$$

### 3.2.8 Metody

Metody jsou téměř stejné jako ve FJ. Změna, kterou doznaly je, že mají nově generické parametry stejně jako má třída. Rovněž změna typu měla vliv na návratovou hodnotu metody. Metoda tedy začíná generickými parametry a následuje její návratový typ. Dále název metody a nakonec její parametry. V těle metody se nachází klíčové slovo `return` následované výrazem.

| | | |
|---|---|---|
| *MethodList* | → | *Method MethodList* |
| | | \| $\epsilon$ |
| *Method* | → | $<$ *GenericParameterList* $>$ *Type* Identifier ( *ParameterList* ) |
| | | { return *Expression* ; } |

Metody jsou správně otypované jestliže, každá metoda v listu je správně otypována. Pokud je správně otypována její název je přidán do kontextu metod. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{\begin{array}{c}(\Gamma, \Delta, \Gamma_\mathsf{M}, \Gamma_\mathsf{K}, \gamma_\mathsf{M}) \vdash \mathit{Method} : \diamond \\ (\Gamma, \Delta, \Gamma_\mathsf{M} \cup \{\mathsf{MNAME}(\mathit{Method})\}, \Gamma_\mathsf{K}, \gamma_\mathsf{M}) \vdash \mathit{MethodList} : \diamond\end{array}}{(\Gamma, \Delta, \Gamma_\mathsf{M}, \Gamma_\mathsf{K}, \gamma_\mathsf{M}) \vdash \mathit{Method}\ \cdot\ \mathit{MethodList} : \diamond} \tag{3.37}$$

Funkce MNAME vrací jméno metody. Tato funkce přijímá Method. Byla již definována ve FJ. Viz 2.44.

Metoda je správně otypována pokud již neexistuje metoda se stejným názvem a rovněž neexistuje metoda se stejným názvem v rodičovské třídě. Návratový typ metody musí být správně otypován. Generické parametry a parametry metody musí být také správně otypovány. Musí se také ověřit, že v parametrech neexistuje parametr s názvem `this`. Výraz v metodě musí být také správně otypován a jeho typ musí být podtypem návratového typu nebo stejného typu. Toto inferenční pravidlo má ekvivalent v[1]. Je jím GT-METHOD ve Fig. 7.

$$\frac{\begin{array}{c}\mathsf{M} \notin \Gamma_{\mathsf{M}} \quad \mathsf{M} \notin \gamma_{\mathsf{M}} \quad (\Gamma, \Delta \cup \mathsf{GPARAMS}(\mathit{GenericParameterList})) \vdash \mathit{Type} : \diamond \\ (\Gamma, \Delta \cup \mathsf{GPARAMS}(\mathit{GenericParameterList}), \Delta) \vdash \mathit{GenericParameterList} : \diamond \\ (\Gamma, \Delta \cup \mathsf{GPARAMS}(\mathit{GenericParameterList}), \emptyset) \vdash \mathit{ParameterList} : \diamond \\ \mathsf{this} : \mathsf{T} \notin \mathsf{PLTYPED}(\mathit{ParameterList}) \\ (\Gamma, \Delta \cup \mathsf{GPARAMS}(\mathit{GenericParameterList}), \\ \mathsf{PLTYPED}(\mathit{ParameterList}) \cup \{\mathsf{this} : \Gamma_{\mathsf{K}}\}) \vdash \mathit{Expression} : \mathsf{T} \\ \mathsf{ETYPE}(\mathit{Expression}) <: \mathit{Type}\end{array}}{(\Gamma, \Delta, \Gamma_{\mathsf{M}}, \Gamma_{\mathsf{K}}, \gamma_{\mathsf{M}}) \vdash < \ \mathit{GPL} \ > \ \mathit{Type}\ \mathsf{M}\ (\ \mathit{PL}\ )\ \{\ \mathsf{return}\ \mathit{Expression}\ ;\} : \diamond} \tag{3.38}$$

Pro zkrácení zápisu v inferenčním pravidle bylo GenericParameterList zapsáno jako GPL a ParameterList jako PL.

Funkce GPARAMS na zjištění generických parametrů. Tato funkce přijímá GenericParameterList a již byla popsána při vytváření globálního kontextu. Viz 3.10.

Funkce PLTYPED na zjištění typů parametrů. Tato funkce přijímá ParameterList a byla již definována ve FJ. Viz 2.46.

Funkce ETYPE na zjištění typu výrazu. Tato funkce přijímá Expression a byla již definována dříve ve FJ. Viz 2.52.

### 3.2.9 Parametry

Parametry jsou list parametrů, kde jednotlivý parametr je definován svým typem a identifikátorem. Oproti FJ je zde změna typu. Typ zde může být už generického rázu, což nebylo u FJ možné.

| | | |
|---|---|---|
| *ParameterList* | $\rightarrow$ | *Parameter ParameterList* |
| | | $\mid \epsilon$ |
| *Parameter* | $\rightarrow$ | *Type* Identifier |

Parametry jsou správně otypovány pokud každý parametr v listu je otypovaný. Pokud je parametr správně otypovaný, tak jeho název je přidán do kontextu parametrů. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{\begin{array}{c}(\Gamma, \Delta, \Gamma_{\mathsf{P}}) \vdash \mathit{Parameter} : \diamond \\ (\Gamma, \Gamma_{\mathsf{P}} \cup \{\mathsf{PNAME}(\mathit{Parameter})\}) \vdash \mathit{ParameterList} : \diamond\end{array}}{(\Gamma, \Delta, \Gamma_{\mathsf{P}}) \vdash \mathit{Parameter}\ \cdot\ \mathit{ParameterList} : \diamond} \tag{3.39}$$

Funkce PNAME vrací jméno parametru. Tato funkce přijímá Parameter. Za jméno deklarace je považován název pole. Byla již definována ve FJ. Viz 2.38.

Samotný parametr je správně otypován pokud název parametru není v kontextu parametrů a typ je správně otypovaný. Toto inferenční pravidlo nemá ekvivalent v[1].

$$\frac{\mathsf{N} \notin \Gamma_\mathsf{P} \quad (\Gamma, \Delta) \vdash Type : \diamond}{(\Gamma, \Delta, \Gamma_\mathsf{P}) \vdash Type\ \mathsf{N} : \diamond} \tag{3.40}$$

### 3.2.10 Výrazy

Ve FGJ je stejný počet výrazu jako ve FJ. Tedy pět. Pouze byly obohaceny o generické parametry. Při volání konstruktoru je nutné nastavit generické parametry a stejně tak při volání metody. Plus při přetypování se musí vzít v potaz generické parametry. Jinak výrazy zůstaly stejné. Více se o nich lze tedy dočíst ve FJ 2.3.8.

| | | |
|---|---|---|
| *ExpressionList* | → | *Expression ExpressionList* |
| | | \| $\epsilon$ |
| *Expression* | → | Identifier |
| | | \| *Expression* . Identifier |
| | | \| *Expression* < *TypeList* > . Identifier ( *ExpressionList* ) |
| | | \| new *TypeExpression* ( *ExpressionList* ) |
| | | \| ( *TypeExpression* ) *Expression* |

První výraz zůstal stejný. Jedná se o použití lokální proměnné a ty se používají stále stejně. Toto inferenční pravidlo má ekvivalent v[1]. Je jím GT-VAR ve Fig. 7.

$$\frac{\mathsf{I} : \mathsf{T} \in \Gamma_\mathsf{L}}{(\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash \mathsf{I} : \mathsf{T}} \tag{3.41}$$

Druhý výraz je získání pole nad výrazem. Zde se opět nic nezměnilo. Stačí ověřit, že ve jménech polích existuje dané pole s daným typem. Dále je potřeba ověřit, že výraz nad, kterým se přistupuje k poli je správně otypovaný. Toto inferenční pravidlo má ekvivalent v[1]. Je jím GT-FIELD ve Fig. 7.

$$\frac{\begin{array}{c}(\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash Expression : \mathsf{T}_2 \\ \mathsf{I} \in \mathsf{FNAMES}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_\mathsf{L}, Expression))\end{array}}{(\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash Expression\ .\ \mathsf{I} : \mathsf{T}_1} \tag{3.42}$$

Ke zjištění jakého typu je výraz, nad kterým se přistupuje k poli se používá funkce ETYPE. Tato metoda přijímá globální kontext, lokální kontext a výraz a vrátí typ. Tato funkce byla již popsána ve FJ 2.52.

Ke zjištění všech jmen polí daného typu slouží funkce FNAMES. Tato funkce zjistí z globálního kontextu pro daný typ všechny jména polí. Tato funkce byla již popsána ve FJ 2.19.

Třetí výraz je volání metody nad výrazem. Zde je potřeba ověřit, že výraz, nad kterým je volání metody, je správně otypovaný, a že typy pro generické parametry jsou správně otypovány. Dále je potřeba ověřit, že list výrazu v parametru metody je správně otypován. Je potřeba také ověřit, že metoda existuje v typu daného výrazu. Dále je potřeba zkontrolovat, že parametry metody jsou správného typu a jsou dodrženy generické typy, a že typy nastavené na místo generických parametrů jsou podtypem nebo stejného typu. Toto inferenční pravidlo má ekvivalent v[1]. Je jím GT-INVK ve Fig. 7.

$$\frac{\begin{array}{c}(\Gamma, \Delta, \Gamma_{\mathsf{L}}) \vdash \mathit{Expression} : \mathsf{T}_2 \quad (\Gamma, \Delta) \vdash \mathit{TypeList} : \diamond \\ (\Gamma, \Delta, \Gamma_{\mathsf{L}}) \vdash \mathit{ExpressionList} : \diamond \quad \mathsf{M} \in \mathsf{MNAMES}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_{\mathsf{L}}, \mathit{Expression})) \\ \mathsf{R}(\mathsf{ETYPES}(\Gamma, \Gamma_{\mathsf{L}}, \mathit{ExpressionList}), \Delta) <: \mathsf{R}(\mathsf{MPLTYPES}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_{\mathsf{L}}, \mathit{Expression}), \mathsf{M}), \Delta) \\ \mathsf{TPARAMS}(\mathit{TypeList}) <: \mathsf{GBOUNDS}(\Gamma, \mathsf{MTYPE}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_{\mathsf{L}}, \mathit{Expression}), \mathsf{M}), \mathsf{M})\end{array}}{(\Gamma, \Delta, \Gamma_{\mathsf{L}}) \vdash \mathit{Expression}\ .\ \mathsf{M}\ <\ \mathit{TypeList}\ >\ (\ \mathit{ExpressionList}\ ) : \mathsf{T}_1} \tag{3.43}$$

Ke zjištění jakého typu je výraz, nad kterým se přistupuje k poli se používá funkce ETYPE. Tato metoda přijímá globální kontext, lokální kontext a výraz a vrátí typ. Tato funkce byla již popsána ve FJ. Viz 2.52.

Ke zjištění všech jmen metod daného typu slouží funkce MNAMES. Tato funkce zjistí z globálního kontextu pro daný typ všechny jména metod. Tato funkce byla již popsána ve FJ. Viz 2.24.

Na zjištění typů výrazu, které jsou předávány do metody slouží funkce ETYPES. Tato funkce přijímá ExpresionList a vrátí sekvenci typů. Tato funkce byla již popsána ve FJ. Viz 2.70.

Na zjištění typů parametrů metody slouží funkce MPLTYPES. Tato funkce přijímá globální kontext, typ třídy a název metody a vrátí sekvenci typů. Tato funkce byla již popsána ve FJ. Viz 2.70.

Na zjištění typů z TypeList slouží metoda TPARAMS. Ta již byla popsána při otypování třídy. Viz 3.24.

Ke zjištění hranic generických parametrů metody slouží funkce GBOUNDS. Ta z globálního kontextu, daného typu a názvu metody zjistí hranice generických parametrů a vrátí je jako

sekvenci typů.

$$\mathsf{GBOUNDS} : (\mathsf{CLASSES}, \mathsf{C}, \mathsf{M}) \rightarrow \mathsf{Seq\ TypeExpression} \tag{3.44}$$
$$\mathsf{GBOUNDS}(\ \epsilon\ ,\ \mathsf{T}\ ,\ \mathsf{M}\ ) = Seq\ \{\}$$
$$\mathsf{GBOUNDS} : (\mathsf{CLASSES}, \mathsf{C}, \mathsf{M}) \rightarrow \mathsf{Seq\ TypeExpression} \tag{3.45}$$
$$\mathsf{GBOUNDS}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}, \mathsf{GPARAMS}) \cdot Set\ ,\ \mathsf{C}\ ,\ \mathsf{M}\ ) = \mathsf{GBOUNDS}(\mathsf{METHODS}, \mathsf{M})$$
$$\mathsf{GBOUNDS} : (\mathsf{CLASSES}, \mathsf{C}, \mathsf{M}) \rightarrow \mathsf{Seq\ TypeExpression} \tag{3.46}$$
$$\mathsf{GBOUNDS}(\ (\mathsf{C}, \mathsf{BC}, \mathsf{FIELDS}, \mathsf{METHODS}, \mathsf{GPARAMS}) \cdot Set\ ,\ \mathsf{T}\ ,\ \mathsf{M}\ ) = \mathsf{GBOUNDS}(Set, \mathsf{T}, \mathsf{M})$$
$$pokud\ \mathsf{C}\ a\ \mathsf{T}\ jsou\ rozdilne$$
$$\mathsf{GBOUNDS} : (\mathsf{METHODS}, \mathsf{M}) \rightarrow \mathsf{Seq\ TypeExpression} \tag{3.47}$$
$$\mathsf{GBOUNDS}(\ \epsilon\ , \mathsf{N}\ ) = Seq\ \{\}$$
$$\mathsf{GBOUNDS} : (\mathsf{METHODS}, \mathsf{M}) \rightarrow \mathsf{Seq\ TypeExpression} \tag{3.48}$$
$$\mathsf{GBOUNDS}(\ (\mathsf{M} : \mathsf{T}, \mathsf{PARAMS}, \mathsf{GPARAMS})\ \cdot\ Set\ ,\ \mathsf{M}\ ) = \mathsf{GBOUNDS}(\mathsf{GPARAMS})$$
$$\mathsf{GBOUNDS} : (\mathsf{METHODS}, \mathsf{M}) \rightarrow \mathsf{Seq\ TypeExpression} \tag{3.49}$$
$$\mathsf{GBOUNDS}(\ (\mathsf{M} : \mathsf{T}, \mathsf{PARAMS}, \mathsf{GPARAMS})\ \cdot\ Set\ ,\ \mathsf{N}\ ) = \mathsf{GBOUNDS}(Set, \mathsf{N})$$
$$pokud\ \mathsf{M}\ a\ \mathsf{N}\ jsou\ rozdilne$$

Tato funkce ještě ke své funkcionalitě potřebuje předchozí GBOUNDS definovanou pro zjišťování generických hranic u tříd. Viz 3.25.

Funkce R má jednoduchý účel. Má za úkol nahradit všechny generické parametry za typy v třídním kontextu generických parametrů.

$$\mathsf{R} : (\mathsf{G}, \mathsf{G} \times \mathsf{C}) \rightarrow \mathsf{Seq\ C} \tag{3.50}$$
$$\mathsf{R}(\ \epsilon\ ,\ \Delta\ ) = Seq\ \{\}$$
$$\mathsf{R} : (\mathsf{G}, \mathsf{G} \times \mathsf{C}) \rightarrow \mathsf{Seq\ C} \tag{3.51}$$
$$\mathsf{R}(\ \mathsf{TYPES}\ ,\ \mathsf{G} : \mathsf{TypeExpression} \cdot Seq\ ) = [\mathsf{G} \backslash TypeExpression]\mathsf{TYPES} \cdot \mathsf{R}(\mathsf{TYPES}, Seq)$$

Čtvrtý výraz je volání konstruktoru. Je potřeba ověřit, že výrazy předané jako parametry konstruktoru jsou správně otypované. Dále je třeba ověřit, že volaná třída je správně otypována. Nakonec je potřeba porovnat zda sedí typy konstruktoru s předanými výrazy. Toto inferenční pravidlo má ekvivalent v[1]. Je jím GT-NEW ve Fig. 7.

$$\frac{(\Gamma, \Delta, \Gamma_{\mathsf{L}}) \vdash ExpressionList : \diamond \quad (\Gamma, \Delta) \vdash TypeExpression : \diamond \qquad \mathsf{R}(\mathsf{ETYPES}(ExpressionList), \Delta) <: \mathsf{R}(\mathsf{FTYPES}(\Gamma, \mathsf{TTYPE}(TypeExpression)), \Delta)}{(\Gamma, \Delta, \Gamma_{\mathsf{L}}) \vdash \mathsf{new}\ TypeExpression\ (\ ExpressionList\ ) : \mathsf{T}} \tag{3.52}$$

Funkce ETYPES zjišťuje typy z ExpressionList. Byla již definována v FJ. Viz 2.70.

Funkce TTYPE na zjištění názvu třídy z generického typu. Byla již definovaná při tvorbě globální kontextu. Viz 3.12.

Funkce FTYPES na zjištění typů polí z globálního kontextu a typu. Byla již definována ve FJ. Viz 2.80.

Funkce R na převod generických typu na konkrétní typy byla popsána u předchozího výrazu. Viz 3.50.

Pátý a poslední výraz je přetypování z jednoho typu výrazu na jiný typ. Je nutné výraz správně otypovat. To zajistí i to, že se jedná o převod do převoditelného typu. Toto inferenční pravidlo má ekvivalent v[1]. Jsou jimi GT-UCAST a GT-DCAST a GT-SCAST ve Fig. 7.

$$\frac{(\Gamma, \Delta) \vdash \mathit{TypeExpression} : \diamond \quad (\Gamma, \Delta, \Gamma_{\mathsf{L}}) \vdash \mathit{Expression} : \mathsf{T}_2}{(\Gamma, \Delta, \Gamma_{\mathsf{L}}) \vdash (\ \mathit{TypeExpression}\ )\ \mathit{Expression} : \mathsf{T}_1} \tag{3.53}$$

Výrazy mohou být v listu a to když jsou předávány jako parametry metodě nebo konstruktoru. Výrazy jsou správně otypovány pokud každý výraz v listu je správně otypován.

$$\frac{(\Gamma, \Delta, \Gamma_{\mathsf{L}}) \vdash \mathit{Expression} : \mathsf{T} \quad (\Gamma, \Delta, \Gamma_{\mathsf{L}}) \vdash \mathit{ExpressionList} : \diamond}{(\Gamma, \Delta, \Gamma_{\mathsf{L}}) \vdash \mathit{Expression}\ \cdot\ \mathit{ExpressionList} : \diamond} \tag{3.54}$$

# Část III

# Implementace

# Kapitola 4

# Implementace

V této kapitole je popsána implementace podmnožin jazyka Java. Je zde popsáno jakým způsobem formální popis byl převáděn do implementace ve Fika nástroji a jiných nástrojů. Není popsáno jak byla převedená celá gramatika a každé inferenční pravidlo, ale je znázorněn princip jak implementace probíhala. Pokud čtenáře zajímá kompletní implementace, tak by si měl prohlédnout přiložené CD, na kterém se nacházejí implementované obě iterace podmnožin jazyka Java. Tedy Featherweight Java a Featherweight Generic Java.

## 4.1 Implementace gramatiky ve Fika nástroji

První krok implementace byl přepsat gramatiku do Fika nástroje. Jelikož Fika vychází z BNF a přidává pár nových vlastností jako jsou možnosti definovat si vlastní moduly a ty následně importovat do jiných modulů. Vytvářet abstraktní pravidla. Přepisovat stávající pravidla. Rovněž Fika má jedno omezení oproti BNF, a to že nelze kombinovat alternace[1] a konkatenace[2]. Také je omezení, že alternace nesmí obsahovat terminály[3].

Rovněž bylo použito pár patternu ve Fika nástroji. Jedním z nich jsou listy. Ty byly definovány ve Fika nástroji následně.

```
module Lists {
  List -> ListBody | ListTail;
  ListBody -> Element List;
  ListTail -> ;
  abstract Element;
}
```

Tento list je následně importován do jiných modulů a jména pravidel jsou přejmenovány vhodně dle použití. Například následující kód, který implementuje gramatiku pro pole.

[1] Pravidla s možnostmi. Tzv. s |.
[2] Pravidla bez možností. Pouze řada pravidel.
[3] Konstantní řetězce. Například identifikátor, klíčová slova apod.

```
module Fields_1 {
  import Lists
    rename List as FieldList
    rename Element as Field
    rename ListBody as FieldListBody
    rename ListTail as FieldListTail;

  override Field -> "FINAL" "IDENTIFIER" "IDENTIFIER" "SEMICOLON";
}
```

Existují další listy. Například parametry jsou odděleny vždy nějakým separátorem. Je tedy potřeba list, který má oddělen prvky separátorem. Pro tyto účely je `DelimitedNonEmptyLists`. Ten importuje `NonEmptyLists`, který jen přepisuje `ListTail` na `Element`. Tento list pak přidá `Delimeter` mezi `Element` a `List`.

```
module DelimitedNonEmptyLists {
  import NonEmptyLists;
  override ListBody -> Element Delimiter List;
  abstract Delimiter;
}
```

Rovněž díky omezením, že alternace nesmí obsahovat terminály, bylo třeba přepsat definici výrazu, aby toto bylo kompatibilní s Fika nástrojem. Viz následující kód.

```
override Expression -> ExpressionVariable | ExpressionField |
 ExpressionMethod | ExpressionInitializer | ExpressionCast;

ExpressionVariable -> "IDENTIFIER";
ExpressionField -> Expression "DOT" "IDENTIFIER";
ExpressionMethod -> Expression "DOT" "IDENTIFIER" "LPARENT"
  OptionalExpressionList "RPARENT";
ExpressionInitializer -> "NEW" "IDENTIFIER" "LPARENT"
  OptionalExpressionList "RPARENT";
ExpressionCast -> "LPARENT" "IDENTIFIER" "RPARENT" Expression;
```

Podobně byl převeden i zbytek gramatiky pro obě iterace. Jak pro jazyk Featherweight Java tak i pro jazyk Featherweight Generic Java.

Následně je ze zápisu ve Fika nástroji. Vygenerovat lexer a parser a třídy v jazyce Java. Pro lexer je používána knihovna JFlex. Do lexeru je třeba rovněž dopsat terminální symboly jelikož ty Fika nástroj neumí vygenerovat. Pro parser je používána knihovna Beaver. Fika pro každý neterminál vytvoří třídu, která obsahuje jako členy prvky konkatenace, která byla na pravé straně tohoto neterminálu. Pokud na pravé straně byla alternace, tak levá strana je interface, z kterého pravidla v alternaci dědí. Do těch vygenerovaných tříd bude potřeba dopsat typový systém a o tom pojednává další část.

## 4.2 Implementace typového systému

Typový systém je překlopení inferenčních pravidel a funkcí do implementace v nějakém programovacím jazyce. Konkrétně typový systém byl implementován v jazyce Java, protože i sám nástroj Fika je implementovaný v programovacím jazyce Java a třídy, které generuje jsou také v jazyce Java.

Pro snazší implementaci byl také použit nástroj AspectJ. Tento nástroj přidává podporuje AOP[4] do jazyka Java. Tento přístup dovoluje snadno upravovat celé třídy. Z AspectJ byla konkrétně použita část Inter-type declarations. Ta dovoluje přidávat do tříd nové pole, metody. Nastavovat rodiče. Přidávat rozhraní a následně je v daných třídách rovnou implementovat. AspectJ a AOP toho umí mnohem více, například vytvářet tzv. pointy a potom přidávat nové volání metod do určitých částí programu, ale toto nebylo potřeba pro potřeby implementace typové systému, tak proto to není používáno.

Jelikož Fika nástroj generuje členy ve třída s jmény `a0, a1, a2` atd. je potřeba tyto členy správně pojmenovat. Přidají se tedy pomocí AspectJ do třídy gettery. Viz následující kód.

```
aspect Naming {
  public ClassList Program.getClassList() { return a1; }
  public Expression Program.getExpression() { return a2; }
}
```

Takto jsou vytvořeny gettery pro všechny členy tříd, což umožní k nim přistupovat pod čitelnými jmény.

Následně je potřeba nad touto hierarchií tříd vytvořit metody, které zajistí implementaci typové systému. Každé inferenční pravidlo je převedeno do implementace v AspectJ způsobem, že kontexty, z kterých vyplýval závěr jsou použity jako parametry funkce. To co vyplývalo z těch kontextu je třída, na které je toto inferenční pravidlo implementováno. Předpoklad je obsah funkce.

Následující kód je implementace první inferenčního pravidla v AspectJ. Nad třídou se definuje nová funkce `ok`, která vytvoří globální kontext a kontext tříd a do něj přidá třídu `Object`, která existuje standardně v programu. Následně otypuje ClassList potom Expression a vytvoří graf a ověří jestli v něm nejsou cykly.

```
aspect StaticSemantics {
  public boolean Program.ok() {
    GlobalContext context = createGlobalContext(getClassList());
    Set<String> gamma_c = new HashSet<String>();
    gamma_c.add("Object");
    return getClassList().ok(context, gamma_c) &&
```

[4] Aspect-Oriented Programming

```
      getExpression().ok(context, new HashSet<TypedVariable>())  &&
      createGraph(context).ok();
  }
}
```

Stejným způsobem jsou implementovány další inferenční pravidla pro celý typový systém.

V typovém systému jsou také funkce, které se volají v inferenčních pravidlech. Tyto funkce jsou definovány vždy nad třídou, která je v závěru inferenčního pravidla. Metoda CNAMES je tedy definována následně nad uzlem ClassList.

```
public boolean ClassList.getClassNames(GlobalContext context) {
  return context.getClasses();
}
```

Stejným principem je potom implementován zbytek funkcí. Pokud je funkce používána ve více pravidlech, tak je pro třídu vytvořen společný interface, který následně dané třídy, které funkci používají implementují.

V inferenčních pravidlech se používají kontexty. Tyto kontexty jsou vždy předávány do metod jako parametry. Jejich implementace se liší dle toho jakého rázu je daný kontext. V případě, že se jedná o jednoduchý kontext jako je například kontext tříd, tak to je množina. Na implementaci tohoto kontextu byl použit `Set` z knihoven jazyka Java. Složitější kontexty jako je třeba globální kontext jsou implementovány pomocí vlastní třídy. V tomto případě `GlobalContext`. Stejným principem jsou implementovány zbylé kontexty.

# Část IV

# Závěr

# Kapitola 5

# Závěr

V této práci byly úspěšně popsány dvě podmnožiny jazyka Java. První podmnožina byl jazyk Featherweight Java, který vychází z článku[1]. Druhá podmnožina byl jazyk Featherweight Generic Java, což je jazyk Featherweight Java obohacený o generiku. Tento jazyk byl také popsán v článku[1]. Pro tyto podmnožiny byla navrhnuta formálně gramatika a statická sémantika[1]. Následně tento formální popis byl převeden do implementace za použití nástroje Fika a jiných nástrojů. Implementace ověřila, že návrh byl správný. Rovněž formální popis dovoluje tyto podmnožiny implementovat i v jiném jazyce než Java a za použití jiných nástrojů. Také je vidět, že se dalo znovu použít spousta věci z první iterace podmnožiny jazyka Java v druhé iteraci podmnožiny jazyka Java. To ukazuje možnosti znovupoužitelnosti a člověk by klidně mohl použít formální popis a implementaci pro jiný jazyk s minimálně úpravama. Pokud by byl podobný jazyku Java.

Tato práce mě také obohatila o mnoho nových informací. V minulosti jsem nikdy neimplementoval ani nepopisoval programovací jazyk. Musel jsem se tedy naučit mnoho z problematiky gramatik, statických sémantik. Měl jsem také možnost si pohrát se zajímavými nástroji jako je nástroj Fika, JFlex, Beaver a v neposlední řadě AspectJ, který mě potěšil hodně a rozhodně znalosti nabyté jeho používáním zhodnotím v budoucích projektech.

[1] Nebo-li typový systém.

# Literatura


[1] IGARASHI, A. – PIERCE, B. C. – WADLER, P. Featherweight Java: a minimal core calculus for Java and GJ. *ACM Trans. Program. Lang. Syst.* May 2001, 23, s. 396–450. ISSN 0164-0925. doi: http://doi.acm.org/10.1145/503502.503505. Dostupné z: <`http://doi.acm.org/10.1145/503502.503505`>.

[2] PIERCE, B. C. *Types and programming languages.* Cambridge, MA, USA : MIT Press, 2002. ISBN 0-262-16209-1.

[3] PISE, M. The Fika Parser Generator. In *Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*, SCAM '10, s. 99–100, Washington, DC, USA, 2010. IEEE Computer Society. doi: http://dx.doi.org/10.1109/SCAM.2010.27. Dostupné z: <`http://dx.doi.org/10.1109/SCAM.2010.27`>. ISBN 978-0-7695-4178-5.

[4] TURBAK, F. – GIFFORD, D. – SHELDON, M. A. *Design Concepts in Programming Languages.* 1. Massachusetts Institute of Technology, Cambridge, Massachusetts 02142, London, England : The MIT Press, 1th edition, 2008. ISBN 978-0-262-20175-9.

[5] web:cstug. CSTUG — $\mathcal{CS}$TEX Users Group — hlavní stránka. `http://www.cstug.cz/`, stav ze 28. 4. 2011.

[6] web:info. K336 Info. `http://info336.felk.cvut.cz`, stav ze 28. 4. 2011.

[7] web:infobp. K336 Info — pokyny pro psaní bakalářských prací. `https://info336.felk.cvut.cz/clanek.php?id=504`, stav ze 28. 4. 2011.

[8] web:latexdocweb. LATEX — online manuál. `http://www.cstug.cz/latex/lm/frames.html`, stav ze 28. 4. 2011.

# Příloha A

# Featherweight Java

## A.1 Gramatika

| | | |
|---|---|---|
| *Program* | → | *ClassList Expression* |
| *ClassList* | → | *Class ClassList* |
| | | \| $\epsilon$ |
| *Class* | → | class Identifier extends Identifier |
| | | { *FieldList Constructor MethodList* } |
| *FieldList* | → | *FieldList Field* |
| | | \| $\epsilon$ |
| *Field* | → | Identifier Identifier ; |
| *Constructor* | → | Identifier ( *ParameterList* ) |
| | | { super ( *ParameterList* ) ; *DeclarationList* } |
| *DeclarationList* | → | *Declaration DeclarationList* |
| | | \| $\epsilon$ |
| *Declaration* | → | this . Identifier = Identifier ; |
| *ParameterList* | → | *Parameter ParameterList* |
| | | \| $\epsilon$ |
| *Parameter* | → | Identifier Identifier |
| *MethodList* | → | *Method MethodList* |
| | | \| $\epsilon$ |
| *Method* | → | Identifier Identifier ( *ParameterList* ) |
| | | { return *Expression* ; } |
| *ExpressionList* | → | *Expression ExpressionList* |
| | | \| $\epsilon$ |
| *Expression* | → | Identifier |
| | | \| *Expression* . Identifier |
| | | \| *Expression* . Identifier ( *ExpressionList* ) |
| | | \| new Identifier ( *ExpressionList* ) |
| | | \| ( Identifier ) *Expression* |

## A.2 Inferenční pravidla

### A.2.1 Typování výrazů

$$\frac{(\Gamma, \Gamma_\mathsf{L}) \vdash \mathit{Expression} : \mathsf{T} \quad (\Gamma, \Gamma_\mathsf{L}) \vdash \mathit{ExpressionList} : \diamond}{(\Gamma, \Gamma_\mathsf{L}) \vdash \mathit{Expression} \ \cdot \ \mathit{ExpressionList} : \diamond} \tag{A.1}$$

$$\frac{\mathsf{I} : \mathsf{T} \in \Gamma_\mathsf{L}}{(\Gamma, \Gamma_\mathsf{L}) \vdash \mathsf{I} : \mathsf{T}} \tag{A.2}$$

$$\frac{\begin{array}{c}(\Gamma, \Gamma_\mathsf{L}) \vdash \mathit{Expression} : \mathsf{T}_2 \\ \mathsf{I} \in \mathsf{FNAMES}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_\mathsf{L}, \mathit{Expression}))\end{array}}{(\Gamma, \Gamma_\mathsf{L}) \vdash \mathit{Expression} \ . \ \mathsf{I} : \mathsf{T}_1} \tag{A.3}$$

$$\frac{\begin{array}{c}(\Gamma, \Gamma_\mathsf{L}) \vdash \mathit{Expression} : \mathsf{T}_2 \\ (\Gamma, \Gamma_\mathsf{L}) \vdash \mathit{ExpressionList} : \diamond \quad \mathsf{M} \in \mathsf{MNAMES}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_\mathsf{L}, \mathit{Expression})) \\ \mathsf{ETYPES}(\Gamma, \Gamma_\mathsf{L}, \mathit{ExpressionList}) <: \mathsf{MPLTYPES}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_\mathsf{L}, \mathit{Expression}), \mathsf{M})\end{array}}{(\Gamma, \Gamma_\mathsf{L}) \vdash \mathit{Expression} \ . \ \mathsf{M} \ (\ \mathit{ExpressionList} \ ) : \mathsf{T}_1} \tag{A.4}$$

$$\frac{\begin{array}{c}\mathsf{C} \in \mathsf{CNAMES}(\Gamma) \quad (\Gamma, \Gamma_\mathsf{L}) \vdash \mathit{ExpressionList} : \diamond \\ \mathsf{ETYPES}(\mathit{ExpressionList}) <: \mathsf{FTYPES}(\Gamma, \mathsf{C})\end{array}}{(\Gamma, \Gamma_\mathsf{L}) \vdash \mathsf{new} \ \mathsf{C} \ (\ \mathit{ExpressionList} \ ) : \mathsf{T}} \tag{A.5}$$

$$\frac{\mathsf{C} \in \mathsf{CNAMES}(\Gamma) \quad \mathsf{ETYPE}(\mathit{Expression}) <: \mathsf{C} \quad (\Gamma, \Gamma_\mathsf{L}) \vdash \mathit{Expression} : \mathsf{T}_2}{(\Gamma, \Gamma_\mathsf{L}) \vdash (\ \mathsf{C} \ ) \ \mathit{Expression} : \mathsf{T}_1} \tag{A.6}$$

### A.2.2 Typování parametrů

$$\frac{\mathsf{N} \notin \Gamma_\mathsf{P} \quad \mathsf{T} \in \mathsf{CNAMES}(\Gamma)}{(\Gamma, \Gamma_\mathsf{P}) \vdash \mathsf{T} \ \mathsf{N} : \diamond} \tag{A.7}$$

$$\frac{(\Gamma, \Gamma_\mathsf{P}) \vdash \mathit{Parameter} : \diamond \quad (\Gamma, \Gamma_\mathsf{P} \cup \{\mathsf{PNAME}(\mathit{Parameter})\}) \vdash \mathit{ParameterList} : \diamond}{(\Gamma, \Gamma_\mathsf{P}) \vdash \mathit{Parameter} \ \cdot \ \mathit{ParameterList} : \diamond} \tag{A.8}$$

### A.2.3 Typování metod

$$\frac{\begin{array}{c}\mathsf{M} \notin \Gamma_\mathsf{M} \quad \mathsf{M} \notin \gamma_\mathsf{M} \quad \mathsf{T} \in \mathsf{CNAMES}(\Gamma) \\ (\Gamma, \emptyset) \vdash \mathit{ParameterList} : \diamond \quad \mathsf{this} : \mathsf{T} \notin \mathsf{PLTYPED}(\mathit{ParameterList}) \\ (\Gamma, \mathsf{PLTYPED}(\mathit{ParameterList}) \cup \{\mathsf{this} : \Gamma_\mathsf{K}\}) \vdash \mathit{Expression} : \mathsf{T} \\ \mathsf{ETYPE}(\mathit{Expression}) <: \mathsf{T}\end{array}}{(\Gamma, \Gamma_\mathsf{M}, \Gamma_\mathsf{K}, \gamma_\mathsf{M}) \vdash \mathsf{T} \ \mathsf{M} \ (\ \mathit{ParameterList} \ ) \ \{ \ \mathsf{return} \ \mathit{Expression} \ ; \} : \diamond} \tag{A.9}$$

$$\frac{\begin{array}{c}(\Gamma, \Gamma_\mathsf{M}, \Gamma_\mathsf{K}, \gamma_\mathsf{M}) \vdash \mathit{Method} : \diamond \\ (\Gamma, \Gamma_\mathsf{M} \cup \{\mathsf{MNAME}(\mathit{Method})\}, \Gamma_\mathsf{K}, \gamma_\mathsf{M}) \vdash \mathit{MethodList} : \diamond\end{array}}{(\Gamma, \Gamma_\mathsf{M}, \Gamma_\mathsf{K}, \gamma_\mathsf{M}) \vdash \mathit{Method} \ \cdot \ \mathit{MethodList} : \diamond} \tag{A.10}$$

### A.2.4 Typování konstruktoru

$$\frac{\mathsf{F} = \mathsf{P} \quad \mathsf{F} \in \Gamma_\mathsf{F} \quad \mathsf{P} \in \Gamma_\mathsf{F} \quad \mathsf{F} \notin \Gamma_\mathsf{D} \quad \mathsf{P} \notin \Gamma_\mathsf{D}}{(\Gamma, \Gamma_\mathsf{D}, \Gamma_\mathsf{F}) \vdash \mathsf{this}\ .\ \mathsf{F}\ =\ \mathsf{P}\ ;\ : \diamond} \tag{A.11}$$

$$\frac{\begin{array}{c}(\Gamma, \Gamma_\mathsf{D}, \Gamma_\mathsf{F}) \vdash \mathit{Declaration} : \diamond \\ (\Gamma, \Gamma_\mathsf{D} \cup \mathsf{DNAME}(\mathit{Declaration}), \Gamma_\mathsf{F}) \vdash \mathit{DeclarationList} : \diamond\end{array}}{(\Gamma, \Gamma_\mathsf{D}, \Gamma_\mathsf{F}) \vdash \mathit{Declaration}\ \cdot\ \mathit{DeclarationList} : \diamond} \tag{A.12}$$

$$\frac{\mathsf{N} \notin \Gamma_\mathsf{P} \quad \mathsf{N} \in \Gamma_\mathsf{F} \quad \mathsf{T} \in \mathsf{CNAMES}(\Gamma)}{(\Gamma, \Gamma_\mathsf{P}, \Gamma_\mathsf{F}) \vdash \mathsf{T}\ \mathsf{N} : \diamond} \tag{A.13}$$

$$\frac{\begin{array}{c}(\Gamma, \Gamma_\mathsf{P}, \Gamma_\mathsf{F}) \vdash \mathit{Parameter} : \diamond \\ (\Gamma, \Gamma_\mathsf{P} \cup \{\mathsf{PNAME}(\mathit{Parameter})\}, \Gamma_\mathsf{F}) \vdash \mathit{ParameterList} : \diamond\end{array}}{(\Gamma, \Gamma_\mathsf{P}, \Gamma_\mathsf{F}) \vdash \mathit{Parameter}\ \cdot\ \mathit{ParameterList} : \diamond} \tag{A.14}$$

$$\frac{\begin{array}{c}\Gamma_\mathsf{K} = \mathsf{C} \\ (\Gamma, \emptyset, \gamma_\mathsf{F} \cup \mathsf{FNAMES}(\Gamma, \mathsf{C})) \vdash \mathit{ParameterList}_A : \diamond \\ \mathsf{PNAMES}(\mathit{ParameterList}_A) = \gamma_\mathsf{F} \cup \mathsf{FNAMES}(\Gamma, \mathsf{C}) \\ (\Gamma, \emptyset, \gamma_\mathsf{F}) \vdash \mathit{ParameterList}_B : \diamond \\ \mathsf{PNAMES}(\mathit{ParameterList}_B) \cup \mathsf{FNAMES}(\Gamma, \mathsf{C}) = \gamma_\mathsf{F} \cup \mathsf{FNAMES}(\Gamma, \mathsf{C}) \\ (\Gamma, \emptyset, \mathsf{FNAMES}(\Gamma, \mathsf{C})) \vdash \mathit{DeclarationList} : \diamond \quad \mathsf{DNAMES}(\mathit{DeclarationList}) = \mathsf{FNAMES}(\Gamma, \mathsf{C})\end{array}}{(\Gamma, \Gamma_\mathsf{K}, \gamma_\mathsf{F}) \vdash \mathsf{C}\ (\ \mathit{ParameterList}_A\ )\ \{\ \mathsf{super}\ (\ \mathit{ParameterList}_B\ )\ \mathit{DeclarationList}\ \} : \diamond} \tag{A.15}$$

### A.2.5 Typování polí

$$\frac{\mathsf{N} \notin \Gamma_\mathsf{F} \quad \mathsf{N} \notin \gamma_\mathsf{F} \quad \mathsf{T} \in \mathsf{CNAMES}(\Gamma)}{(\Gamma, \Gamma_\mathsf{F}, \gamma_\mathsf{F}) \vdash \mathsf{T}\ \mathsf{N}\ ;\ : \diamond} \tag{A.16}$$

$$\frac{(\Gamma, \Gamma_\mathsf{F}, \gamma_\mathsf{F}) \vdash \mathit{Field} : \diamond \quad (\Gamma, \Gamma_\mathsf{F} \cup \{\mathsf{FNAME}(\mathit{Field})\}, \gamma_\mathsf{F}) \vdash \mathit{FieldList} : \diamond}{(\Gamma, \Gamma_\mathsf{F}, \gamma_\mathsf{F}) \vdash \mathit{Field}\ \cdot\ \mathit{FieldList} : \diamond} \tag{A.17}$$

### A.2.6 Typování třídy

$$\frac{\begin{array}{c}\mathsf{C} \notin \Gamma_\mathsf{C} \quad \mathsf{BC} \in \mathsf{CNAMES}(\Gamma) \\ (\Gamma, \emptyset, \mathsf{FNAMES}(\Gamma, \mathsf{BC})) \vdash \mathit{FieldList} : \diamond \\ (\Gamma, \mathsf{C}, \mathsf{FNAMES}(\Gamma, \mathsf{BC})) \vdash \mathit{Constructor} : \diamond \\ (\Gamma, \emptyset, \mathsf{C}, \mathsf{MNAMES}(\Gamma, \mathsf{BC})) \vdash \mathit{MethodList} : \diamond\end{array}}{(\Gamma, \Gamma_\mathsf{C}) \vdash \mathsf{class}\ \mathsf{C}\ \mathsf{extends}\ \mathsf{BC}\ \{\ \mathit{FieldList}\ \mathit{Constructor}\ \mathit{MethodList}\ \} : \diamond} \tag{A.18}$$

$$\frac{(\Gamma, \Gamma_\mathsf{C}) \vdash \mathit{Class} : \diamond \quad (\Gamma, \Gamma_\mathsf{C} \cup \{\mathsf{CNAME}(\mathit{Class})\}) \vdash \mathit{ClassList} : \diamond}{(\Gamma, \Gamma_\mathsf{C}) \vdash \mathit{Class}\ \cdot\ \mathit{ClassList} : \diamond} \tag{A.19}$$

$$\frac{\begin{array}{c}\{(\mathsf{CLASSES}(\mathit{ClassList}), \mathit{Object})\} \vdash \mathit{ClassList} : \diamond \\ (\mathsf{CLASSES}(\mathit{ClassList}), \emptyset) \vdash \mathit{Expression} : \diamond \\ \mathsf{GRAPH}(\mathsf{CLASSES}(\mathit{ClassList}))\ \mathit{neobsahuje\ cykly}\end{array}}{\vdash \mathit{ClassList}\ \mathit{Expression} : \diamond} \tag{A.20}$$

# Příloha B

# Featherweight Generic Java

## B.1 Gramatika

| | | |
|---|---|---|
| *TypeList* | → | *Type TypeList* |
| | | \| $\epsilon$ |
| *Type* | → | Identifier |
| | | \| *TypeExpression* |
| *TypeExpression* | → | Identifier < *TypeList* > |
| *Program* | → | *ClassList Expression* |
| *ClassList* | → | *Class ClassList* |
| | | \| $\epsilon$ |
| *Class* | → | class Identifier < *GenericParameterList* > extends |
| | | *TypeExpression* { *FieldList Constructor MethodList* } |
| *GenericParameterList* | → | *GenericParameterList GenericParameter* |
| | | \| $\epsilon$ |
| *GenericParameter* | → | Identifier extends *TypeExpression* |
| *FieldList* | → | *FieldList Field* |
| | | \| $\epsilon$ |
| *Field* | → | *Type* Identifier ; |
| *Constructor* | → | Identifier ( *ParameterList* ) |
| | | { super ( *ParameterList* ) ; *DeclarationList* } |
| *DeclarationList* | → | *Declaration DeclarationList* |
| | | \| $\epsilon$ |
| *Declaration* | → | this . Identifier = Identifier ; |
| *ParameterList* | → | *Parameter ParameterList* |
| | | \| $\epsilon$ |
| *Parameter* | → | *Type* Identifier |
| *MethodList* | → | *Method MethodList* |
| | | \| $\epsilon$ |
| *Method* | → | < *GenericParameterList* > *Type* Identifier ( *ParameterList* ) |
| | | { return *Expression* ; } |
| *ExpressionList* | → | *Expression ExpressionList* |

$$
\begin{array}{lcl}
 & & \mid \epsilon \\
\mathit{Expression} & \rightarrow & \mathsf{Identifier} \\
 & & \mid \mathit{Expression}\ .\ \mathsf{Identifier} \\
 & & \mid \mathit{Expression} < \mathit{TypeList} > .\ \mathsf{Identifier}\ (\ \mathit{ExpressionList}\ ) \\
 & & \mid \mathsf{new}\ \mathit{TypeExpression}\ (\ \mathit{ExpressionList}\ ) \\
 & & \mid (\ \mathit{TypeExpression}\ )\ \mathit{Expression}
\end{array}
$$

## B.2 Inferenční pravidla

### B.2.1 Typování výrazů

$$\frac{(\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash \mathit{Expression} : \mathsf{T} \quad (\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash \mathit{ExpressionList} : \diamond}{(\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash \mathit{Expression}\ \cdot\ \mathit{ExpressionList} : \diamond} \tag{B.1}$$

$$\frac{\mathsf{I} : \mathsf{T} \in \Gamma_\mathsf{L}}{(\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash \mathsf{I} : \mathsf{T}} \tag{B.2}$$

$$\frac{\begin{array}{c}(\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash \mathit{Expression} : \mathsf{T}_2 \\ \mathsf{I} \in \mathsf{FNAMES}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_\mathsf{L}, \mathit{Expression}))\end{array}}{(\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash \mathit{Expression}\ .\ \mathsf{I} : \mathsf{T}_1} \tag{B.3}$$

$$\frac{\begin{array}{c}(\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash \mathit{Expression} : \mathsf{T}_2 \quad (\Gamma, \Delta) \vdash \mathit{TypeList} : \diamond \\ (\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash \mathit{ExpressionList} : \diamond \quad \mathsf{M} \in \mathsf{MNAMES}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_\mathsf{L}, \mathit{Expression})) \\ \mathsf{R}(\mathsf{ETYPES}(\Gamma, \Gamma_\mathsf{L}, \mathit{ExpressionList}), \Delta) <: \mathsf{R}(\mathsf{MPLTYPES}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_\mathsf{L}, \mathit{Expression}), \mathsf{M}), \Delta) \\ \mathsf{TPARAMS}(\mathit{TypeList}) <: \mathsf{GBOUNDS}(\Gamma, \mathsf{MTYPE}(\Gamma, \mathsf{ETYPE}(\Gamma, \Gamma_\mathsf{L}, \mathit{Expression}), \mathsf{M}), \mathsf{M})\end{array}}{(\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash \mathit{Expression}\ .\ \mathsf{M}\ <\ \mathit{TypeList}\ >\ (\ \mathit{ExpressionList}\ ) : \mathsf{T}_1} \tag{B.4}$$

$$\frac{\begin{array}{c}(\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash \mathit{ExpressionList} : \diamond \quad (\Gamma, \Delta) \vdash \mathit{TypeExpression} : \diamond \\ \mathsf{R}(\mathsf{ETYPES}(\mathit{ExpressionList}), \Delta) <: \mathsf{R}(\mathsf{FTYPES}(\Gamma, \mathsf{TTYPE}(\mathit{TypeExpression})), \Delta)\end{array}}{(\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash \mathsf{new}\ \mathit{TypeExpression}\ (\ \mathit{ExpressionList}\ ) : \mathsf{T}} \tag{B.5}$$

$$\frac{(\Gamma, \Delta) \vdash \mathit{TypeExpression} : \diamond \quad (\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash \mathit{Expression} : \mathsf{T}_2}{(\Gamma, \Delta, \Gamma_\mathsf{L}) \vdash (\ \mathit{TypeExpression}\ )\ \mathit{Expression} : \mathsf{T}_1} \tag{B.6}$$

### B.2.2 Typování parametrů

$$\frac{\mathsf{N} \notin \Gamma_\mathsf{P} \quad (\Gamma, \Delta) \vdash \mathit{Type} : \diamond}{(\Gamma, \Delta, \Gamma_\mathsf{P}) \vdash \mathit{Type}\ \mathsf{N} : \diamond} \tag{B.7}$$

$$\frac{(\Gamma, \Delta, \Gamma_\mathsf{P}) \vdash \mathit{Parameter} : \diamond \quad (\Gamma, \Gamma_\mathsf{P} \cup \{\mathsf{PNAME}(\mathit{Parameter})\}) \vdash \mathit{ParameterList} : \diamond}{(\Gamma, \Delta, \Gamma_\mathsf{P}) \vdash \mathit{Parameter}\ \cdot\ \mathit{ParameterList} : \diamond} \tag{B.8}$$

### B.2.3 Typování metod

$$\frac{\begin{array}{c}\mathsf{M} \notin \Gamma_\mathsf{M} \quad \mathsf{M} \notin \gamma_\mathsf{M} \quad (\Gamma, \Delta \cup \mathsf{GPARAMS}(\mathit{GenericParameterList})) \vdash \mathit{Type} : \diamond \\ (\Gamma, \Delta \cup \mathsf{GPARAMS}(\mathit{GenericParameterList}), \Delta) \vdash \mathit{GenericParameterList} : \diamond \\ (\Gamma, \Delta \cup \mathsf{GPARAMS}(\mathit{GenericParameterList}), \emptyset) \vdash \mathit{ParameterList} : \diamond \\ \mathsf{this} : \mathsf{T} \notin \mathsf{PLTYPED}(\mathit{ParameterList}) \\ (\Gamma, \Delta \cup \mathsf{GPARAMS}(\mathit{GenericParameterList}), \\ \mathsf{PLTYPED}(\mathit{ParameterList}) \cup \{\mathsf{this} : \Gamma_\mathsf{K}\}) \vdash \mathit{Expression} : \mathsf{T} \\ \mathsf{ETYPE}(\mathit{Expression}) <: \mathit{Type}\end{array}}{(\Gamma, \Delta, \Gamma_\mathsf{M}, \Gamma_\mathsf{K}, \gamma_\mathsf{M}) \vdash <\ \mathit{GPL}\ >\ \mathit{Type}\ \mathsf{M}\ (\ \mathit{PL}\ )\ \{\ \mathsf{return}\ \mathit{Expression}\ ;\} : \diamond} \tag{B.9}$$

$$\frac{\begin{array}{c}(\Gamma, \Delta, \Gamma_\mathsf{M}, \Gamma_\mathsf{K}, \gamma_\mathsf{M}) \vdash \mathit{Method} : \diamond \\ (\Gamma, \Delta, \Gamma_\mathsf{M} \cup \{\mathsf{MNAME}(\mathit{Method})\}, \Gamma_\mathsf{K}, \gamma_\mathsf{M}) \vdash \mathit{MethodList} : \diamond\end{array}}{(\Gamma, \Delta, \Gamma_\mathsf{M}, \Gamma_\mathsf{K}, \gamma_\mathsf{M}) \vdash \mathit{Method}\ \cdot\ \mathit{MethodList} : \diamond} \tag{B.10}$$

### B.2.4 Typování konstruktoru

$$\frac{\mathsf{F} = \mathsf{P} \quad \mathsf{F} \in \Gamma_\mathsf{F} \quad \mathsf{P} \in \Gamma_\mathsf{F} \quad \mathsf{F} \notin \Gamma_\mathsf{D} \quad \mathsf{P} \notin \Gamma_\mathsf{D}}{(\Gamma, \Gamma_\mathsf{D}, \Gamma_\mathsf{F}) \vdash \mathsf{this}\ .\ \mathsf{F}\ =\ \mathsf{P}\ ;\ : \diamond} \tag{B.11}$$

$$\frac{\begin{array}{c}(\Gamma, \Gamma_\mathsf{D}, \Gamma_\mathsf{F}) \vdash \mathit{Declaration} : \diamond \\ (\Gamma, \Gamma_\mathsf{D} \cup \mathsf{DNAME}(\mathit{Declaration}), \Gamma_\mathsf{F}) \vdash \mathit{DeclarationList} : \diamond\end{array}}{(\Gamma, \Gamma_\mathsf{D}, \Gamma_\mathsf{F}) \vdash \mathit{Declaration}\ \cdot\ \mathit{DeclarationList} : \diamond} \tag{B.12}$$

$$\frac{\mathsf{N} \notin \Gamma_\mathsf{P} \quad \mathsf{N} \in \Gamma_\mathsf{F} \quad (\Gamma, \Delta) \vdash \mathit{Type} : \diamond}{(\Gamma, \Delta, \Gamma_\mathsf{P}, \Gamma_\mathsf{F}) \vdash \mathit{Type}\ \mathsf{N} : \diamond} \tag{B.13}$$

$$\frac{\begin{array}{c}(\Gamma, \Delta, \Gamma_\mathsf{P}, \Gamma_\mathsf{F}) \vdash \mathit{Parameter} : \diamond \\ (\Gamma, \Delta, \Gamma_\mathsf{P} \cup \{\mathsf{PNAME}(\mathit{Parameter})\}, \Gamma_\mathsf{F}) \vdash \mathit{ParameterList} : \diamond\end{array}}{(\Gamma, \Delta, \Gamma_\mathsf{P}, \Gamma_\mathsf{F}) \vdash \mathit{Parameter}\ \cdot\ \mathit{ParameterList} : \diamond} \tag{B.14}$$

$$\frac{\begin{array}{c}\Gamma_\mathsf{K} = \mathsf{C} \\ (\Gamma, \Delta, \emptyset, \gamma_\mathsf{F} \cup \mathsf{FNAMES}(\Gamma, \mathsf{C})) \vdash \mathit{ParameterList}_A : \diamond \\ \mathsf{PNAMES}(\mathit{ParameterList}_A) = \gamma_\mathsf{F} \cup \mathsf{FNAMES}(\Gamma, \mathsf{C}) \\ (\Gamma, \Delta, \emptyset, \gamma_\mathsf{F}) \vdash \mathit{ParameterList}_B : \diamond \\ \mathsf{PNAMES}(\mathit{ParameterList}_B) \cup \mathsf{FNAMES}(\Gamma, \mathsf{C}) = \gamma_\mathsf{F} \cup \mathsf{FNAMES}(\Gamma, \mathsf{C}) \\ (\Gamma, \emptyset, \mathsf{FNAMES}(\Gamma, \mathsf{C})) \vdash \mathit{DeclarationList} : \diamond \quad \mathsf{DNAMES}(\mathit{DeclarationList}) = \mathsf{FNAMES}(\Gamma, \mathsf{C})\end{array}}{(\Gamma, \Delta, \Gamma_\mathsf{K}, \gamma_\mathsf{F}) \vdash \mathsf{C}\ (\ \mathit{ParameterList}_A\ )\ \{\ \mathsf{super}\ (\ \mathit{ParameterList}_B\ )\ \mathit{DeclarationList}\ \} : \diamond} \tag{B.15}$$

### B.2.5 Typování polí

$$\frac{\mathsf{N} \notin \Gamma_\mathsf{F} \quad \mathsf{N} \notin \gamma_\mathsf{F} \quad (\Gamma, \Delta) \vdash \mathsf{Type} : \diamond}{(\Gamma, \Delta, \Gamma_\mathsf{F}, \gamma_\mathsf{F}) \vdash \mathsf{Type}\ \mathsf{N}\ ;\ : \diamond} \tag{B.16}$$

$$\frac{(\Gamma, \Delta, \Gamma_\mathsf{F}, \gamma_\mathsf{F}) \vdash \mathit{Field} : \diamond \quad (\Gamma, \Delta, \Gamma_\mathsf{F} \cup \{\mathsf{FNAME}(\mathit{Field})\}, \gamma_\mathsf{F}) \vdash \mathit{FieldList} : \diamond}{(\Gamma, \Delta, \Gamma_\mathsf{F}, \gamma_\mathsf{F}) \vdash \mathit{Field}\ \cdot\ \mathit{FieldList} : \diamond} \tag{B.17}$$

### B.2.6 Typování typů

$$\frac{\mathsf{X} \in \mathsf{GNAMES}(\Delta)}{(\Gamma, \Delta) \vdash \mathsf{X} : \diamond} \tag{B.18}$$

$$\frac{(\Gamma, \Delta) \vdash \mathit{Type} : \diamond \quad (\Gamma, \Gamma_\mathsf{G}) \vdash \mathit{TypeList} : \diamond}{(\Gamma, \Delta) \vdash \mathit{Type} \ \cdot \ \mathit{TypeList} : \diamond} \tag{B.19}$$

$$\frac{\begin{array}{c}(\Gamma, \Delta) \vdash \mathit{TypeList} : \diamond \quad \mathsf{C} \in \mathsf{CNAMES}(\Gamma) \\ \mathsf{TPARAMS}(\mathit{TypeList}) <: \mathsf{GBOUNDS}(\Gamma, \mathsf{C})\end{array}}{(\Gamma, \Delta) \vdash \mathsf{C} \ < \ \mathit{TypeList} \ > : \diamond} \tag{B.20}$$

### B.2.7 Typování generických parametrů

$$\frac{\mathsf{X} \notin \Gamma_\mathsf{G} \quad \mathsf{X} \notin \mathsf{CNAMES}(\Gamma) \quad (\Gamma, \Delta) \vdash \mathit{TypeExpression} : \diamond}{(\Gamma, \Delta, \Gamma_\mathsf{G}) \vdash \mathsf{X} \ \mathsf{extends} \ \mathit{TypeExpression} \ : \diamond} \tag{B.21}$$

$$\frac{\begin{array}{c}(\Gamma, \Delta, \Gamma_\mathsf{G}) \vdash \mathit{GenericParameter} : \diamond \\ (\Gamma, \Delta, \Gamma_\mathsf{G} \cup \{\mathsf{GNAME}(\mathit{GenericParameter})\}) \vdash \mathit{GenericParameterList} : \diamond\end{array}}{(\Gamma, \Delta, \Gamma_\mathsf{G}) \vdash \mathit{GenericParameter} \ \cdot \ \mathit{GenericParameterList} : \diamond} \tag{B.22}$$

### B.2.8 Typování třídy

$$\frac{\begin{array}{c}\mathsf{C} \notin \Gamma_\mathsf{C} \quad (\Gamma, \mathsf{GPARAMS}(\mathit{GPL})) \vdash \mathit{TypeExpression} : \diamond \\ (\Gamma, \mathsf{GPARAMS}(\mathit{GPL}), \emptyset) \vdash \mathit{GenericParameterList} : \diamond \\ (\Gamma, \mathsf{GPARAMS}(\mathit{GPL}), \emptyset, \mathsf{FNAMES}(\Gamma, \mathsf{TTYPE}(\mathit{TypeExpression})) \vdash \mathit{FieldList} : \diamond \\ (\Gamma, \mathsf{GPARAMS}(\mathit{GPL}), \mathsf{C}, \mathsf{FNAMES}(\Gamma, \mathsf{TTYPE}(\mathit{TypeExpression})) \vdash \mathit{Constructor} : \diamond \\ (\Gamma, \mathsf{GPARAMS}(\mathit{GPL}), \emptyset, \mathsf{C}, \mathsf{MNAMES}(\Gamma, \mathsf{TTYPE}(\mathit{TypeExpression})) \vdash \mathit{MethodList} : \diamond\end{array}}{(\Gamma, \Gamma_\mathsf{C}) \vdash \mathsf{class} \ \mathsf{C} \ < \ \mathit{GPL} \ > \ \mathsf{extends} \ \mathit{TypeExpression} \ \{ \ \mathit{FL} \ \mathit{C} \ \mathit{ML} \ \} : \diamond} \tag{B.23}$$

$$\frac{(\Gamma, \Gamma_\mathsf{C}) \vdash \mathit{Class} : \diamond \quad (\Gamma, \Gamma_\mathsf{C} \cup \{\mathsf{CNAME}(\mathit{Class})\}) \vdash \mathit{ClassList} : \diamond}{(\Gamma, \Gamma_\mathsf{C}) \vdash \mathit{Class} \ \cdot \ \mathit{ClassList} : \diamond} \tag{B.24}$$

$$\frac{\begin{array}{c}\{(\mathsf{CLASSES}(\mathit{ClassList}), \mathit{Object})\} \vdash \mathit{ClassList} : \diamond \\ (\mathsf{CLASSES}(\mathit{ClassList}), \emptyset, \emptyset) \vdash \mathit{Expression} : \diamond \\ \mathsf{GRAPH}(\mathsf{CLASSES}(\mathit{ClassList})) \ \mathit{neobsahuje\ cykly}\end{array}}{\vdash \mathit{ClassList} \ \mathit{Expression} : \diamond} \tag{B.25}$$

# Příloha C

# Seznam použitých zkratek

**FJ** Featherweight Java

**FGJ** Featherweight Generic Java

**BNF** Backus–Naur Form

**AOP** Aspect-oriented programming

**JRE** Java Runtime Environment

⋮

# Příloha D

# Instalační a uživatelská příručka

Pro běh implementace podmnožiny jazyka Java je potřeba JRE[1] a to ve verzi minimálně 1.5. Rovněž spouštěcí skripty byly napsány pro Bash, což na Windows vyžaduje instalaci Cygwin či přepsání spouštěcího skriptu do .bat či spouštět přímo přes proces java.

Příklad užití v Bash bude následující. Předpokládejme, že otevřený adresář je kořen CD.

```
$ cd implementace/FeatherweightJava
$ chmod +x run.sh # nastaveni prav pro spusteni
$ ./run.sh examples/priklad_1.fj
```

To spustí implementaci nad předaným kódem. Pokud v něm nejsou žádné chyby, tak se kód zparsuje a statická sémantika ověří jestli v něm nejsou chyby.

[1] Java Runtime Environment

# Příloha E

# Obsah přiloženého CD

**Struktura přiloženého CD je následující.**

```
.
|-- html - prezentace
|-- implementace
|   |-- FeatherweightGenericJava
|   |   |-- FJParser.parser - parser jazyka
|   |   |-- FJScanner.flex - lexer jazyka
|   |   |-- aspects - aspekty pro implementaci statické sémantiky
|   |   |-- build.xml - buildovací skript
|   |   |-- classes - zkompilované zdrojové kódy
|   |   |-- examples - ukázkové příklady v jazyce FeatherweightGenericJava
|   |   |-- generated - vygenerované třídy pomocí JFlex a Beaver
|   |   |-- lib - knihovny potřebné pro program
|   |   |-- run.sh - spouštěcí skript
|   |   |-- src - složka obsahující zdrojové kódy
|   |   `-- tools - nástroje nezbytné pro chod aplikace
|   `-- FeatherweightJava
|       |-- FJParser.parser - parser jazyka
|       |-- FJScanner.flex - lexer jazyka
|       |-- aspects - aspekty pro implementaci statické sémantiky
|       |-- build.xml - buildovací skript
|       |-- classes - zkompilované zdrojové kódy
|       |-- examples - ukázkové příklady v jazyce FeatherWeightJava
|       |-- generated - vygenerované třídy pomocí JFlex a Beaver
|       |-- lib - knihovny potřebné pro program
|       |-- run.sh - spouštěcí skript
|       |-- src - složka obsahující zdrojové kódy
|       `-- tools - nástroje nezbytné pro chod aplikace
|-- index.html - prezentace
|-- install.txt - instalační manuál
|-- readme.txt - popis balíčku
`-- text - PDF se zdrojovými kódy LaTeXu
```