1 Úvod

V průběhu našeho studování specifikace QVTo jsme si dělali poznámky, abychom se lépe orientovali v dané problematice a byli schopni předat naše zkušenosti mezi sebou či studentům pokračujícím v projektu. Varování - tyto poznámky obsahují náš subjektivní pohled, nezaručujeme jejich naprostou správnost. Autoři poznámek předpokládají čtenářovu základní znalost programovacího jazyka Java, protože se na něj budou odkazovat.

1.1 Deklarace proměnných

deklarace má tvar var <identifikator> : Typ
př. var a: String

1.2 Komentáře

- Komentáře se dělají jako v Javě
- existují dva typy:
  - Jednořádkové
    př. `//Toto je jednořádkový komentář`
  - Obecné (víceřádkové - řádkově neomezené)
    `př. /*Toto je`
    `dvouřádkový komentář */`
  -

1.3 Podmínka If then else

- Podmínka má tvar:
  if (condition) then /*statement1 */ else /* statement2 */ endif;
- závorky ani else větev nejsou potřebné, dají se vynechat(závorky značí jen blok, podobně jako v Javě a jiných jazycích), středník za endif ani ostatní části vynechatelné nejsou
- `př. if(0 = 0) then log("Common") else log ("Miracle") endif;`

1.4 Operátory

- Konec výrazu
  ;
- Přiřazení Objectu nebo Setu
  :=

- Přidání položky do Setu
  +=
- Operátor je rovno
  =
- Operátor není rovno
  <>
- Operátory přístupu
  . − >
-
- Odlišnými oproti běžným programovacím jazykům jsou obzvlášť operátory += a <>, které nejsou v programovacích jazycích obvyklé
- S operátorem := je možné použít tzv. conditional expression k zkrácení zápisu. Tento operátor slouží k přiřazení hodnoty proměnné
  př. var a: String var b: Integer; a:=(if (b = b) then “Logické” else “Zázrak“)
- Operátor = slouží k přiřazení reference nebo k porovnání
- Operátor přístupu . slouží k přístupu k jednotlivým proměnným
- Operátor přístupu -¿ se používá k přístupu k položkám a metodám kolekcí

1.5 klíčové slovo switch

příkaz switch se používá k řízení toku podobně jako v Javě, nicméně nepoužívá jeden výraz, ale vždy za klíčovým slovem case následuje podmínka, klíčové slovo else je použito, pro případy, kdy není splněna žádná z předchozích podmínek
př. switch case (condition1) /* Statement1 */ case (condition2) /* Statement2 */ else /* Statement3*/

1.6 klíčové slovo self

klíčové slovo self je equivalentní k this v Javě – získáváte pomocí něj přístup k objektu, nad kterým pracujete (jeho metodám, atributům)
př. self.name :=””

1.7 klíčové slovo result

Klíčové slovo result pracuje podobně jako Self s atributy a metodami, ale nepracuje s vstupním objektem, ale výstupním, lze ho použít v mappingu cyklus while

1.8 cyklus while

se používá k opakovanému provádění těla cyklu, stejně jako v Javě
př. while (condition) /*statement */

1.9 Deklarace transformace

- skládá se z jména, vstupního a výstupního metamodelu
- může být poděděna pomocí klíčového slova extends, v tom případě potomek dědí všechna mapování a dotazy, které může předefinovat

1.10 klíčové slovo access

má podobnou funkci, ale narozdíl od klíčového slova extends není možné cokoliv předefinovat, celá transformace je použita jako celek

1.11 klíčová slova new a transform

slouží k instanciaci přijaté transformace

1.12 Modeltype definition

- definice typu modelu
- reference na modeltype nebo je možné vložit celou definici (inline definice)
- může referencovat na lokální file(př. 1) nebo je definována pomocí reference na package namespace URI (př. 2) • local specific reference - v Eclipse se to dělá prefixací “platform:/resource/“, za kterou následuje relativní cesta k souboru v workspacu
- př. 1 modeltype MM1 uses “platform:/resource/MM1toMM2/transforms/MM1.ecore“
- př. 2 modeltype MM1 uses “http://mm1/1.0“

1.13 Helper

- Operace, která vykonává výpočet na jednom nebo více objektech (parametrech) a tvoří výsledek
- Tělo helperu je uspořádaný seznam výrazů, které jsou vykonány v řadě po sobě (v sekvenci). Helper může jako vedlejší efekt modifikovat parametry

Pozn: Autoři textu nevyužívali helpery a zadavatel vyslovil podezření na nefunkčnost helperů v současné verzi QVTo, předcházející definice je vytažena z QVTo Specifikace Query

.

1.14 Query

je helper bez vedlejších efektů, tzn nemění vstupní „objekt“
př. `query APP::reduced::Property::isID():Boolean{ if(self.serialization.isID = true) then { return true; }endif; return false;}`

1.15 when

- podmínka následující po klíčovém slově when je nazývána pre-condition nebo též guard
- k provedení daného mapování musí být tato precondition splněna
- `tvar: mapping MM1::Model::toModel() : MM2::Model when {self.Name.startWith(\M");} { //konkrétní mapping }`

1.16 Disjuncts

- seřazený seznam mapování
- je zavolané první mapování, jehož guard (typ a podmínka uvozená klíčovým slovem when) je platný
- pokud není platný žádný guard je vrácena hodnota null
- pomocí disjuncts lze nahradit nemožnost přetížení mapování
- př. `mapping UML::Feature::convertFeature() : JAVA::Element disjuncts convertAttribute, convertOperation{}`

```
mapping UML::Attribute::convertAttribute : JAVA::Field {
name := self.name;
}
```

```
 mapping UML::Operation::convertConstructor : JAVA::Constructor
when { self.name = self.namespace.name;} {
name := self.name;
}
 mapping UML::Operation::convertOperation : JAVA::Constructor
when { self.name <> self.namespace.name;
} { name := self.name; }
```

1.17 Main funkce

účel funkce main() je nastavit proměnné prostředí a zavolat první mapování

1.18 log("message")

- vypisuje zprávu do konzole
- má tři levely: warning, error a fatal
- Při nesplnění assertu levelu fatal transformace skončí
- Je tvaru: assert level (condition) with log("message")
- Pozn. V příkladě od zadavatele je assert bez levelu, nejspíš je implicitní level warning nebo error
- assert warning (self.x > 2) with log("Hodnota x je menší než 2 a je rovna:" + self.x)

1.19 Map vs xmap

- map - v případě, že se neprovede mapování, vrátí null
- xmap – v případě, že se nepovede mapování, vyvolá výjimku

1.20 Dictionary

- Javovská Map = Kolekce (container) uskladňující data uspořádaná podle klíče
- úplný popis operací viz 8.3.7 v specifikaci QVT
- proveditelné operace s Dictionary:
  - Dictionary(KeyT , T) :: get (k : KeyT) : T
  - Dictionary(KeyT , T) :: hasKey (k : KeyT) : Boolean
  - Dictionary(KeyT , T) :: put (k : KeyT , v : T) : Void

  - Dictionary (KeyT , T) :: size ( ) : Integer
  - Dictionary(KeyT,T)::values() : List(T)
  - Dictionary(KeyT,T)::keys() : List(KeyT)
  - Dictionary(KeyT,T)::isEmpty() : Boolean
- př. var x:Dict(String,Actor); // Dictionary Itemů typu Actor s klíčem String

1.21 ForEach

- Iterátor nad kolekcí, provede tělo pro všechny prvky, pro něž je zadaná podmínka platná
-
```
self.owningClass.owningModel.classes->forEach(cl | cl.isPrimitive
= true){
if(cl.name = self.type.name) then {
return true;
} endif;
```

1.22 ForOne

- Iterátor nad kolekcí, provede tělo pro první prvek, pro který je zadaná podmínka platná, pro další prvky již ne
- forEach i forOne jsou popsány v specifikaci v sekci 8.2.2.6