MiASI Lista 3

*ver. 1.1.0*

Z1.

**a) Należy zdefiniować w języku OCL ograniczenia jakie muszą spełniać asocjacje „rodzice-dzieci” oraz „mąż-żona”.**

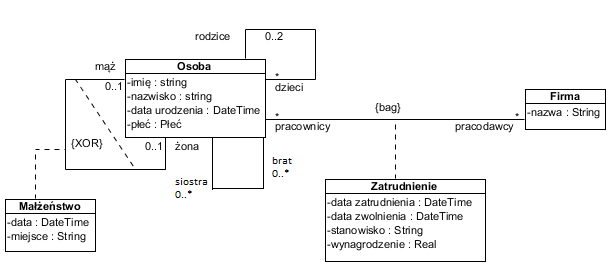
*context Osoba inv*

*self.dzieci →size() ≥0*

*self.rodzice→size() ≥ 0 AND self.rodzice→size() ≤ 2*

*(self.maz→size() = 0 AND self.maz→size() = 1) XOR (self.maz→size() = 1 AND self.maz→size() = 0)*

**b) Na podstawie asocjacji „rodzice-dzieci” zdefiniować nową asocjację „brat-siostra”.**



**c) Należy zdefiniować w języku OCL następujące ograniczenia związane z zatrudnieniem:**

**- jeśli osoba zatrudnia się w danej firmie kolejny raz, to wcześniej powinna być z tej firmy zwolniona**

*context Osoba::zatrudnij(f irma: Firma) inv:*

*self.zatrudnienie →select(z | z.pracodawca = firma) → forAll(z2 | z2.data\_zwolnienia < DateTime.now)*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *self.zatrudnienie* | *Bag(Zatrudnienie)* |
| *self.zatrudnienie →select(z | z.pracodawca = firma)* | *Bag(Zatrudnienie)* |
| *DateTime.now* | *DateTime* |
| *z2.data\_zwolnienia < DateTime.now* | *Boolean* |
| *forAll(z2 | z2.data\_zwolnienia < DateTime.now)* | *Boolean* |

**oraz funkcje:**

**- suma przychodów uzyskanych w danym miesiącu (miesiąc, rok) przez daną osobę,**

*context Osoba::przychodWMiesiacu(miesiąc : DateTime) : Real*

*body: result = self.zatrudnienie → select(z | z.data\_zatrudnienia ≤ miesiąc AND z.data\_zwolnienia ≥ miesiąc).wynagrodzenie → sum()*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *self.zatrudnienie* | *Bag(Zatrudnienie)* |
| *z.data\_zatrudnienia* | *DateTime* |
| *z.data\_zatrudnienia ≤ miesiąc* | *Boolean* |
| *z.data\_zwolnienia* | *DateTime* |
| *z.data\_zwolnienia ≥ miesiąc* | *Boolean* |
| *self.zatrudnienie → select(z | z.data\_zatrudnienia ≤ miesiąc AND z.data\_zwolnienia ≥ miesiąc* | *Bag(Zatrudnienie)* |
| *self.zatrudnienie → select(z | z.data\_zatrudnienia ≤ miesiąc AND z.data\_zwolnienia ≥ miesiąc).wynagrodzenie → sum()* | *Real* |

**- suma przychodów uzyskanych w danym miesiącu przez dane małżeństwo,**

*context Małżeństwo::wspolnyPrzychodWMiesiacu(miesiąc : DateTime) : Real*

*pre: self.data ≤ miesiąc*

*post: result = self.maz →przychodWMiesiacu(miesiąc) + self.zona → przychodWMiesiacu(miesiąc)*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *self.maz* | *Osoba* |
| *self.zona* | *Osoba* |
| *przychodWMiesiacu(miesiąc)* | *Real* |

**- suma przychodów uzyskanych przez daną osobę w zadanym okresie jej zatrudnienia**

*context Osoba::przychodWOkresie(początek : DateTime, koniec : DateTime) : Real*

*pre: początek < koniec*

*post: result = self.przychodWMiesiacu(początek) + self.przychodWOkresie(początek + DateTime.Month, koniec)*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *self.przychodWMiesiacu(początek)* | *Real* |
| *DateTime.Month* | *DateTime* |
| *początek + DateTime.Month* | *DateTime* |
| *self.przychodWOkresie(początek + DateTime.Month, koniec)* | *Real* |

**- suma przychodów uzyskanych przez małżeństwo w zadanym okresie od zawarcia małżeństwa.**

*context Małżeństwo::wspolnyPrzychodWOkresie(początek : DateTime, koniec : DateTime) : Real*

*pre: self.data ≥ początek*

*post: result = self.maz → przychodWOkresie(początek, koniec) + self.zona → przychodWOkresie(początek, koniec)*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *self.maz* | *Osoba* |
| *przychodWOkresie(początek, koniec)* | *Real* |
| *self.zona* | *Osoba* |

Z2)

**Należy wyrazić w języku OCL następujące ograniczenia:**

**a) każda osoby, która nie prowadzi zajęć jest uczniem,**

*context Osoba inv:*

*self.oclIsTypeOf(Prowadzacy) implies Dziennik.allInstance() →forAll(d | not d.uczniowie → exists(self))*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *self.oclIsTypeOf(Prowadzacy)* | *Boolean* |
| *Dziennik.allInstance()* | *Set(Dziennik)* |
| *d.uczniowie* | *Set(Osoba)* |
| *exists(self)* | *Boolean* |
| *not d.uczniowie → exists(self)* | *Boolean* |
| *forAll(d | not d.uczniowie → exists(self))* | *Boolean* |

*albo lepiej:*

*context Osoba inv:*

*self.oclIsTypeOf(Prowadzący) implies self.dziennik →notEmpty()*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *self.oclIsTypeOf(Prowadzacy)* | *Boolean* |
| *self.dziennik* | *Dziennik* |
| *self.dziennik →notEmpty()* | *Boolean* |

**b) ilość osób w dzienniku musi zawierać się w przedziale [20,40],**

*context Dziennik inv:*

*self.uczniowie →size() ≥20 AND self.uczniowie →size() ≤ 40*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *self.oclIsTypeOf(Prowadzacy)* | *Boolean* |
| *self.dziennik* | *Dziennik* |
| *self.dziennik →notEmpty()* | *Boolean* |

**c) dziennik zawiera określone przedmioty a ich ilość musi wynosić przynajmniej 10,**

*context Przedmiot def:*

*okleslonePrzedmioty : Set(Przedmiot) = Set{P1,…,Pn}*

*context Dziennik inv:*

*self.przedmiot →includesAll(Przedmiot.okreslonePrzedmioty)*

*self.przedmiot →size() ≤ 10*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *self.przedmiot* | *Set(Przedmiot)* |
| *okreslonePrzedmioty* | *Set(Przedmiot)* |
| *includesAll(okreslonePrzedmioty)* | *Boolean* |
| *size() ≤ 10* | *Boolean* |

**d) osoba musi mieć niepuste dane tj. imię i nazwisko, a także jeżeli osoba nie jest w dzienniku, to nie może posiadać w nim ocen,**

*context Osoba inv:*

*self.nazwisko.size() > 0 AND self.imie.size() > 0*

*self.oceny →notEmpty() IMPLIES (self.dziennik → notEmpty() AND self.dziennik.ocena → includesAll(self.oceny))*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *self.nazwisko.size()* | *Boolean* |
| *self.imie.size()* | *Boolean* |
| *self.oceny* | *Set(Ocena)* |
| *self.oceny →notEmpty()* | *Boolean* |
| *self.dziennik* | *Dziennik* |
| *self.dziennik.ocena* | *Set(Ocena)* |
| *self.dziennik.ocena → includesAll(self.oceny)* | *Boolean* |

**e) ocena musi być wystawiona dla przedmiotu i osoby z tego samego dziennika, w którym znajduje się ta ocena**

context Ocena inv:

self.przedmiot.dziennik → notEmpty()

self.przedmiot.dziennik → includes(self.dziennik)

self.dziennik = self.osoba.dziennik

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| self.przedmiot | *Przedmiot* |
| self.przedmiot.dziennik | *Set(Dziennik)* |
| self.dziennik | *Dziennik* |
| self.osoba | *Osoba* |
| self.osoba.dziennik | *Dziennik* |
| includes(self.dziennik) | *Boolean* |

**f) ocena z danego przedmiotu może być wystawiona tylko przez osobę, która ten przedmiot prowadzi oraz ocena jest z zakresu [2..5],**

*context Ocena inv:*

*self.wartosc ≥ 2 AND self.wartosc ≤ 5*

*self.przedmiot.prowadzacy → includes(self.prowadzacy)*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *self.wartosc ≥ 2* | *Boolean* |
| *self.przedmiot* | *Przedmiot* |
| *self.przedmiot.prowadzący* | *Set(Prowadzący)* |
| *self.prowadzacy* | *Prowadzący* |
| *includes(self.prowadzacy)* | *Boolean* |

**g) wszystkie oceny, które zostały wystawione przez prowadzącego są z przedmiotów, które ten prowadzący prowadzi,**

*context Prowadzący inv:*

*self.wystawił → forAll(o : Ocena | self.prowadzi →includes(o.przedmiot))*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *self.wystawił* | *Set(Ocena)* |
| *self.prowadzi* | *Set(Przedmiot)* |
| *o.przedmiot* | *Przedmiot* |
| *includes(o.przedmiot)* | *Boolean* |
| *forAll(o : Ocena | self.prowadzi →includes(o.przedmiot))* | *Boolean* |

**h) każda z ocen wystawionych dla przedmiotu jest w dzienniku w którym jest ten przedmiot oraz osoba prowadząca dany przedmiot jest osobą, która wystawiła ocenę.**

*context Przedmiot inv:*

*self.ocena→forAll(o: Ocena | self.dziennik→includes(o.dziennik))*

*self.ocena→forAll(o: Ocena | self.prowadzacy→includes(o.wystawiający))*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| self.ocena | *Set(Ocena)* |
| self.prowadzacy | *Set(Prowadzący)* |
| *o.wystawiający* | *Prowadzący* |
| *includes(o.wystawiający)* | *Boolean* |
| *forAll(o: Ocena | self.prowadzacy→includes(o.wystawiający))* | *Boolean* |
| *self.dziennik* | *Set(Dziennik)* |
| *o.dziennik* | *Dziennik* |
| *includes(o.dziennik)* | *Boolean* |
| *forAll(o: Ocena | self.dziennik→includes(o.dziennik))* | *Boolean* |

**Zdefiniować warunki wstępne i końcowe dla operacji:**

**a) sredniaOcenaUcznia(o : Osoba) : Real – operacja oblicza średnią arytmetyczną z wszystkich ocen danej osoby – operacja klasy Dziennik,**

*context Dziennik::sredniaOcenaUcznia(o: Osoba) : Real*

*pre: self.uczeniowie →includes(o) AND o.oceny→size()>0*

*post: result = o.oceny.wartosc→sum() / o.oceny→size()*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *self.uczen* | *Set(Osoba)* |
| *includes(o)* | *Boolean* |
| *o.oceny* | *Set(Ocena)* |
| *size()>0* | *Boolean* |
| *o.oceny.wartosc→sum()* | *Integer* |
| *o.oceny→size()* | *Integer* |
| *o.oceny.wartosc→sum() / o.oceny→size()* | *Real* |

***b) wystawOcenę(komu : Osoba, z\_czego : Przedmiot, jaka : Integer) : Ocena – operacja wpisywania oceny z przedmiotu (z\_czego) osobie (komu) o wartości (jaka) – jest to operacja klasy Prowadzący. W wyniku tej operacji zwracany jest obiekt klasy Ocena – jest to nowo stworzona ocena,***

*context Prowadzacy::wystawOcene(komu: Osoba, z\_czego: Przedmiot, jaka: Integer) : Ocena*

*pre: komu.dziennik→notEmpty()*

*pre: z\_czego.dziennik→exists(d : Dziennik | d = komu.dziennik)*

*pre: jaka ≥ 2 AND jaka ≤ 5*

*post: result = self.wystawil.wartosc = jaka AND self.wystawil.przedmiot = z\_czego AND self.wystawil.osoba = komu AND result.OclIsNew()*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *komu.dziennik* | *Dziennik* |
| *komu.dziennik→notEmpty()* | *Boolean* |
| *z\_czego.dziennik* | *Set(Dziennik)* |
| *exists(d : Dziennik | d = komu.dziennik)* | *Boolean* |
| *jaka ≥ 2 AND jaka ≤ 5* | *Boolean* |
| *self.wystawil.wartosc* | *Integer* |
| *self.wystawil.przedmiot* | *Przedmiot* |
| *self.wystawil.osoba* | *Osoba* |

***Alternatywne rozwiązanie***

*context Prowadzacy::wystawOcene(komu: Osoba, z\_czego: Przedmiot, jaka: Integer) : Ocena*

*pre:*

*not komu → oclIsUndefined() AND*

*not z\_czego → oclIsUndefined() AND*

*not jaka → oclIsUndefined() AND*

*jaka ≥ 2 AND jaka ≤ 5 AND*

*komu.dziennik → notEmpty() AND*

*komu.dziennik.oceny → select(o : Ocena | o.przedmiot = z\_czego AND o.prowadzący = self) →*

*size() = 1*

*post:*

*not result →oclIsUndefined AND result → oclIsNew() AND*

*result.wartosc = jaka @PRE AND*

*result.przedmiot = z\_czego @PRE AND*

*result.oceniany = komu @PRE AND*

*result.wystawiajacy = self AND*

*result.dziennik = komu @PRE.DZIENNIK*

*kumu.oceny @PRE →includes(result)*

**c) poprawOcene(o : Ocena, nowa\_wartosc : Integer) : oclAny – popraw ocenę (o) na nową wartość – jest to operacja klasy Prowadzący. Operacja nie zwraca wyniku, stąd dowolny typ oclAny.**

*context Prowadzacy::poprawOcene(o: Ocena, nowa\_wartosc:Integer) : oclAny*

*pre: self.wystawil→exists(o)*

*pre: nowa\_wartosc ≥2 AND nowa\_wartosc ≤5*

*post: result = oclAny AND self.wystawil→select( oc : Ocena | oc = o).wartosc = nowa\_wartosc*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *self.wystawil* | *Set(Ocena)* |
| *exists(o)* | *Boolean* |
| *nowa\_wartosc ≥2 AND nowa\_wartosc ≤5* | *Boolean* |
| *self.wystawil* | *Set(Ocena)* |
| *self.wystawil→select( oc : Ocena | oc = o)* | *Set(Ocena)* |
| *self.wystawil→select( oc : Ocena | oc = o).wartosc* | *Integer* |

***Z3)***

***Dla diagramu klas stanowiącego model systemu kartotek wyrazić w OCL następujące ograniczenia:***

***a) Istnieje dokładnie jedna kartoteka, który nie jest podkartoteką innej kartoteki.***

*context Kartoteka inv:*

*Kartoteka.allInstance() → select(k | k.kartoteka → size() = 0) → size() = 1*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *Kartoteka* | *OCLType* |
| *Kartoteka.allInstance()* | *Set(Kartoteka)* |
| *k.kartoteka* | *Kartoteka* |
| *k.kartoteka → size() = 0* | *Boolean* |
| *Kartoteka.allInstance() → select(k | k.kartoteka → size() = 0)* | *Set(Kartoteka)* |

**b) Największy poziom zagnieżdżenia kartotek nie przekracza zadanej liczby n.**

*context Kartoteka inv:*

*let parentsCount(k : Kartoteka) : int*

*if k.kartoteka →oclIsUndefind() then*

*0*

*else*

*1 + parentsCount(k.kartoteka)*

*endif*

*in*

*parentsCount(self) ≤ n*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *k.kartoteka* | *Kartoteka* |
| *k.kartoteka →oclIsUndefind()* | *Boolean* |
| *parentsCount(k.kartoteka)* | *Integer* |
| parentsCount(self) ≤ n | *Boolean* |

**c) Łączna liczba plików w danym systemie nie może przekroczyć zadanej liczby n.**

*context Plik inv:*

*Plik.allInstance() → size() ≤ n*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *Plik* | *OCLType* |
| *Plik.allInstance()* | *Set(Plik)* |

**d) Łączna liczba kartotek (podkartotek) w danym systemie nie może przekroczyć zadanej liczby n.**

*context Kartoteka inv:*

*Kartoteka.allInstance() →size() ≤ n*

|  |  |
| --- | --- |
| *Wyrażenie* | *Typ wyrażenia* |
| *Kartoteka* | *OCLType* |
| *Kartoteka.allInstance()* | *Set(Kartoteka)* |

**Dodatek**

|  |  |  |
| --- | --- | --- |
| **Kryterium** | **Ordered** | **Not Ordered** |
| **Unique** | *OrderedSet* | *Set* |
| **Not Unique** | *Sequence* | *Bag* |