By FiberFarra
For FiberFarra

PUNTS CLAU OCL

Aclariment: Col·lecció és el conjunt de files que tenim al anar-nos movent per les diferents branques. Apartat és cadascuna de les 3 funcionalitats que ens demana l'enunciat.

Elements dels contractes de les operacions

- **Operació(Context):** nom i paràmatres que tindrà l'operació o funcionalitat. Aquesta inclou també, si es requereix, el valor retornat.
- **Precondicions(Pre):** Condicions que cal satisfer per poder executar l'operació.
- **Postcondicions(Post):** Canvis d'estat que es produeixen com a conseqüència de l'execució:
    - altes/baixes d'instàncies de classes d'objectes i d'associacions
    - modificació d'atributs
    - generalització o especialització d'un objecte
    - canvi de subclasse d'un objecte

  Ho hem d'interpretar com que ens explica el que obtenim després de fer l'operació.
- **Sortida(Body):** En cas de que no es realitzin canvis al sistema (Consultores), llavors tindrem el body, que substitueix a la postcondició. En aquest apartat tindrem la descripció de la sortida i l'obtenció d'aquesta.

Exemple modificadora amb retorn:

```
context: Sistema::nomOperació(attr1:tipus, attr2: tipus, ...): NomClasse2
pre: NomClasse.allInstances()->exists(v | n.val1 = attr1)
post: NomClasse2.allInstances()->exists(v2 | v2.oclIsNew() and v2 | v2.val1 =
attr1 AND [Condicions] AND result= v2)
```

Exemple modificadora sense retorn:

```
context: Sistema::nomOperació(attr1:tipus, attr2: tipus, ...)
pre: NomClasse.allInstances()->exists(v | n.val1 = attr1)
post: NomClasse2.allInstances()->exists(v2 | v2.oclIsNew() and v2 | v2.val1 =
attr1 AND [Condicions])
```

Exemple Consultora:

```
context: Sistema::nomOperació(attr1:tipus, attr2: tipus, ...): Set ( tupleType
(ret1:tipus, ret2:tipus, ret3: Set(Tipus)...))
pre: NomClasse.allInstances()->exists(v | n.val1 = attr1)
body: result = NomClasse2.allInstances()->select (v2 | v2.val1 = attr1 AND ...)
-> collect (v2 | tuple {ret1= v2.val1, ret2=v2.val2,...})
```

Tipus de llistes d'elements

- **Conjunt (Set):** Llista d'elements que no es poden repetir i que tampoc tenen ordre. Seria equivalent a la estructura Set de c++.
- **Bossa (Bag):** Llista d'elements que es poden repetir pero que no segueixen cap ordre. Seria l'equivalent a una List de c++.
- **Sequència (Sequence):** Llista d'elements que es poden repetir i que tenen un ordre concret. Seria l'equivalent a un vector de c++.

Operacions que podem realitzar

- **allInstances()**: Ens permet agafar tots els elements que conté un objecte sense poder-los classificar. Sempre ens retorna un Set d'elements.

```
Empleats.allInstances() //Ens retorna tots els elements d'Empleats
```

- **exists(*condicions*)**: Ens permet evaluar si hi ha algun element de la selecció que tinguem abans de fer l'exists que compleix les condicions que establim dins d'aquest. Ens retorna un booleà que ens indica si existeix o no.

```
Empleats.allInstances()-> exists(e | e.nom = 'Alex') //Ens retorna cert
si existeix un empleat amb el nom Alex.
```

- **oclIsNew()**: Es fa servir, en una postcondició, per indicar que un objecte ha estat creat com a efecte de l'operació. S'ha de posar sempre que estiguem creant una nova instància.

```
post: Empleats.allInstances()-> exists(e | e.oclIsNew and e.nom = 'Alex')
//Estem creant una nova instància amb el nom Alex
```

- **@pre**: Es fa servir, en una postcondició, per comprovar si un objecte existia abans d'executar l'operació.

```
post: if(not Empleats.allInstances()@pre->exists(e | e.nom='Alex')) then
Empleats.allInstances()->exists(e | e.oclIsNew and e.nom='Alex') //Estem
creant una nova instància amb el nom Alex en cas que no existís
```

- **Select(*condicions*):** Es fa servir, en el body, quan volem seleccionar una part de tots els elements d'un objecte o d'un conjunt d'objectes.

```
body: result = Empleats.allInstances()->select(e | e.edat > 65) //Estem
seleccionant tots els/les treballadors/es majors de 65 anys.
```

- **Collect(llista)**: Es fa servir, en el body, per a especificar quins elements volem que es retornin i de quina manera volem que ho facin.

```
body: result = Empleats.allInstances()->select(e | e.edat > 65) ->
collect(e | tuple {nom = e.nom, edat = e.edat}) //Estem posant en un Set
els diferents noms i edats dels treballadors/es majors de 65 anys
```

- **size()**: Ens dona un natural que representa el nombre d'instàncies que conté la classe o la selecció que haguem fet de la classe. Ens servirà sobretot si ens demanen que hi hagi un mínim o un màxim d'instàncies.

```
pre: Departament.allInstances()->exists(d | d.empleats->size() > 5 )
//Els departaments que tenen més de 5 empleats assignats  un mateix
departament
```

- **count(ocurrencia)**: Ens dona un nombre natural que representa el nombre de tuples que tenen aquesta ocurrència que posem al parèntesis. Només funcionarà en cas de que siguin exactes, no val que sigui una part del paràmatre. Podem posar un string o el valor d'un ennum.

```
pre: Departament.allInstances()->exists(d |
d.empleats.rol->count(TRols::Directiu) ) //Ens retorna el nombre de
persones per departament que tenen el rol de Directiu.
```

- **includes(ocurrencia)**: Ens retorna tots els elements/un booleà dels diferents elements que incloguin l'objecte. Ens serveix sobretot quan volem comprovar en una relació si s'inclou una clau.

```
pre: Empleat.allInstances()->exists(e |
e.departament.nom->includes("Disseny") ) //Ens retorna cert en cas de que
l'empleat sigui del departament disseny. Podriem continuar la condició i
seguir reduint la mostra, ja que ens guardaria els que compleixen la cond
```

- **includesAll(objecte/enum)**: Ens retorna cert si tots els elements d'un ennum/classe estan relacionats per alguna instància de la classe.

```
pre: Empleat.allInstances()->exists(e |
e.departament.nom->includesAll(TDepartament) ) //Ens retornarà cert si
entre tots els empleats com a mínim hi ha un empleat assignat a cada
departament.
```

- **excludes(ocurrencia)**: Ens retorna tots els elements/un booleà dels diferents elements que no incloguin l'objecte. Ens serveix sobretot quan volem comprovar que en una relació no s'inclou la clau.

```
pre: Empleat.allInstances()->exists(e |
e.departament.nom->excludes("Becari") ) //Ens retorna cert en cas de que
l'empleat no estigui en el departament de becaris.
```

- **excludesAll(objecte/enum):** Ens retorna cert si cap dels elements d'un ennum/classe estan relacionats per alguna instància de la classe.

```
pre: Empleat.allInstances()->exists(e |
e.departament.nom->excludesAll(TDepartament) ) //Ens retornarà cert si
els empleats e no pertanyen a cap departament.
```

- **isEmpty():** Ens retorna cert en cas de que a la col·lecció no tinguem cap element.

```
pre: Empleat.allInstances()->exists(e | e.nom = "Alex" and
e.departament->isEmpty()) //Ens retornarà cert si l'empleat Alex no està
assignat a cap departament.
```

- **notEmpty():** Ens retorna cert en cas de que a la col·lecció tinguem algun element.

```
pre: Empleat.allInstances()->exists(e | e.nom = "Alex" and
e.departament->notEmpty()) //Ens retornarà cert si l'empleat Alex està
assignat a algun departament.
```

- **forAll(condicions)**: Totes les instàncies de la col·lecció han de complir les condicions que s'estableixen. En cas contrari, retornarà fals.

```
pre: Empleat.allInstances()->forAll(e1, e2 | e1 <> e2 and e1.nom<>
e2.nom) //Ens retornarà cert si el nom de tots els empleats és diferent.
```

- **isUnique(condicions)**: Ens permet comprovar que totes les instàncies sota el mateix element són diferents i no es repeteixen.

```
pre: Empleat.allInstances()->isUnique(nom) //Ens retornarà cert si el nom
de tots els empleats és diferent.
```

- **oclIsTypeOf(objecte/enum)**: Ens permet comprovar/dir que una instància del objecte forma part d'una subclasse del objecte. Només ens serveix per a relacionar una col·lecció de la superclasse amb una subclasse.

```
pre: Empleat.allInstances()->exists(e | e.oclIsTypeOf("becaris")) //Ens
retornarà els empleats que formen part de la subclasse becaris.
```

- **oclAsType(objecte/enum):** Ens permet accedir a elements de la subclasse des de la classe. Només es necessita del pare al fill, del fill al pare es pot fer directament.

```
pre: Empleat.allInstances()->exists(e | e.oclIsTypeOf("becaris") and
e.oclAsType("becaris").sou <500) //Ens retornarà els empleats que formen
part de la subclasse becaris i que el seu sou és inferior a 500.
```

- **asSet():** Ens permet eliminar els elements repetits en una col·lecció o bossa.

```
pre: Empleat.allInstances()->exists(e | e.projecte.nom->asSet()) //Ens
retornarà un set amb els noms dels diferents projectes on hi ha algun
empleat assignat sense repetir-los.
```

Operadors lògics

Operació	Notació	Resultat
and	a and b	booleà
or	a or b	booleà
or exclusiu	a xor b	booleà
negació	not a	booleà
igualtat	a = b	booleà
desigualtat	a <> b	booleà
implicació (Si a es compleix, llavors b es compleix)	a implies b	booleà
if-then-else-endif	if a then b else c endif	b/c
concatenació	string.concat(string)	string
substring	string.substring(int,int)	string

Punts Clau

1. Si hi ha diferents apartats a realitzar, s'ha de fer un diagrama de sequüencia per apartat, mai s'han de posar tots junts en un mateix diagrama, significaria que fas totes les funcions seguides.
2. Si una frase comença per: "Aquesta funcionalitat només es pot portar a terme si..." ens estan dient la precondició que s'ha de complir.
3. És important saber si la col·lecció que tenim seleccionada té una sola instància o està formada per un conjunt d'instàncies, la qual cosa ens condicionarà les operacions que podem fer.
4. A una subclasse tenim accés a tots els atributs i relacions de la seva superclasse, però a la inversa no ho podem fer. Una superclasse no pot

accedir directament als atributs del seu full, ha d'utilitzar oclAsType("nomSubclasse").

5. Sempre posar el nom de l'atribut a comparar, no quedar-nos al nom de la classe (e | ~~e.empresa = nomEmpresa~~ -> e | e.empresa**.nom** = nomEmpresa)
6. Si en una relació entre dues classes hi ha una associativa, primer hem de passar per l'associativa per a anar d'una a l'altra (~~Festa - Persona~~ -> Festa **Assistent** - Persona)
7. Si diuen: "introdueix tota la informació necessària", ens tocarà fer nosaltres la cerca d'elements necessaris per a crear una nova instància de la classe demanada, ja que l'enunciat no ho especificarà. Cal mirar les que tenen una multiplicitat mínima d'1, a part dels atributs de la classe i les subclasses.
8. Els apartats són independents, és a dir, si nosaltres en el primer apartat comprovem que existeixi un element en una classe, en el següent apartat, si es requereix, cal tornar-ho a comprovar.
9. En cas de que l'esquema UML inclogui noms de rol, utilitzarem els noms de rol abans d'utilitzar els noms de la pròpia classe, ja que ens ajudaran a evitar confusions.
10. Si el post d'un loop és =, llavors només quedarà registrat l'ultima iteració del bucle en cas de que no hi hagi un oclIsNew. Per tant, és molt important utilitzar un exists o un include.
11. Si hi ha un pre que ja està cobert per una RT, llavors no s'ha de posar.
12. No s'ha de comprovar que no existeixi la instància que anem a crear/insertar. La RT se'n ocupa.