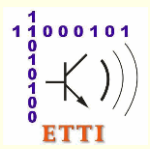


Universitatea POLITEHNICA din Bucuresti
Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Catedra de Telecomunicatii

ITNQ
2011

Laboratorul **SAIM** – sala **B123**

25/03/2012

POO – an II

TPI – an III

Inginerie Software in Comunicatii (ISC)

PSNIM
2012

2011-2012

Laborator 3

Crearea diagramelor UML de masini de stari (FSM). Modificarea modelelor UML, reconstructia codului (*refactoring*). Studiu de caz complex. Predarea temelor de casa specificate la lucrarile 1 si 2

Descrierea laboratorului

In aceasta lucrare de laborator vor fi acoperite urmatoarele probleme:

- **Crearea diagramelor UML de masini de stari (FSM)** in NetBeans 6.1
- **Modificarea modelelor UML si reconstructia codului (*Refactoring*)** in NetBeans 6.1
- **Studiu de caz complex de inginerie a codului Java bazata pe utilizarea UML** in NetBeans 6.1

(include **rezolvarea enunturilor** din ISw_0809_Subiecte_Examen_Subiecte_2A_2B.pdf)

Atentie: **La inceputul laboratoarelor stergeti mai intai toate proiectele existente** (click dreapta pe nodul proiectului in fereastra ***Projects***, selectati ***Delete*** si **confirmati ca doriti sa fie sterse sursele – in cazul proiectelor Java**). **La finalul laboratoarelor stergeti proiectele create**.

3.1. Crearea diagramelor UML de masini de stari (FSM)

In continuare veti invata **cum sa folositi functiile UML ale IDE pentru a crea diagrame UML de masini de stari (FSM)**.

Diagrama UML de masini de stari (cunoscuta si ca *Finite State* Machine, sau FSM) este o reprezentare vizuala a masinii de stari a aplicatiei.

Diagrama UML de masini de stari prezinta **starile prin care trece o entitate** (un obiect sau (sub)system) de la aparitie (creare) pana la disparitie (distrugere). In acest tip de diagrama **comportamentul este specificat ca o secventa de stari** prin care trece o entitate **ca raspuns la evenimentele care apar** pe timpul existentei ei, dar **si raspunsurile entitatii** respective la aceste evenimente.

Diagrama UML de masini de stari poate fi foarte utila in cazul aplicatiilor de **control al proceselor in timp real** sau a subsistemelor care implica **prelucrari concurente** (in paralel, distribuite, etc.), inclusiv a celor de **comunicatie**.

De asemenea, diagrama UML de masini de stari poate fi utilizata pentru a exprima **comportamentul unei entitati pentru un grup de cazuri de utilizare** (spre deosebire de diagramele MSC sau de comunicatie care prezinta in general comportamente din interiorul unui caz de utilizare).

3.1.1. Crearea unui proiect UML independent de platforma

1. Pentru a crea un proiect UML, selectaţi ***File> New Project*** şi apoi faceţi următoarele:
   Sub ***Categories***, selectati ***UML***.
   Sub ***Projects***, selectati ***Platform-Independent Model***.
   Faceti ***Click*** pe ***Next***.
2. In campul ***Project Name*** completati **MyUMLProject**
3. Pentru campul ***Project Location***, click ***Browse***, si navigaţi la orice director de pe computer (in laborator alegeti drive-ul **D:** si directorul **\isw**, si creati sau selectati un **subdirector cu numarul grupei**, apoi creati un **subdirector cu nume diferit de cele existente**, de exemplu: **D:\isw\441E\Proiect1**).
4. Faceti ***Click*** pe ***Finish***. IDE-ul creeaza proiectului UML si apare caseta de dialog ***Create New Diagram.***
5. Din lista ***Diagram type*** alegeti ***State Diagram***.
6. In campul ***Diagram Name*** completati **SodaMachineStateDiagram**.
7. Lasati valoarea implicita in campul ***Namespace*** si click pe ***Finish***. IDE-ul realizeaza următoarele:
   Creeaza un proiect UML independent de platforma numit **MyUMLProject**
   Adaugă nodul proiectului in fereastra ***Projects***.
   Creeaza un nod **SodaMachineStateDiagram** sub nodul ***Model***
   Afişează noua diagramă în editorul de diagrame (diagrama este goala în acest moment)
   Deschide (*Modeling*) ***Palette***

3.1.2. Adaugarea elementelor de tip stare

Aceasta sectiune contine urmatoarele proceduri:

3.1.2.1. Adaugarea unui element stare initiala

1. Din sectiunea ***Basic*** a ***Pallete*** selectati pictograma ***Initial State***  .
2. Faceti click in coltul din stanga sus a diagramei de masini de stari pentru a plasa elementul.
3. Deselectati pictograma ***Initial State*** prin right-click.

3.1.2.2. Adaugarea unor elemente stare simpla

1. Din sectiunea ***Basic*** a ***Pallete*** selectati pictograma ***Simple State*** .
2. Faceti click in dreapta elementului stare initiala pentru a plasa noul element.
3. Deselectati pictograma ***Simple State***.
4. Faceti dublu-click pe noul element si denumiti-l **Type Displaying**, apoi apasati ***Enter***.
5. Adaugati inca 3 elemente de tip ***Simple State*** si denumiti-le

   Refunding
   Calculating
   Processing

3.1.2.3. Adaugarea unui element stare finala

1. Din sectiunea ***Basic*** a ***Pallete*** selectati pictograma ***Final State*** .
2. Faceti click in partea de jos a diagramei de masini de stari pentru a plasa noul element.
3. Deselectati pictograma ***Final State***.

4. Faceti right-click pe starea finala (nu pe numele ei) si selectati din meniul pop-up ***Labels*** > ***Final State Name*** pentru a elimina numele starii finale (**Unnamed**).

3.1.3. Adaugarea legaturilor (tranzitiilor) intre stari

Un eveniment poate produce **tranzitia** dintr-o stare in alta. Intr-o diagrama de masini de stari tranzitia intre doua stari este reprezentata printr-o **sageata care conecteaza cele doua stari**, element care se numeste ***State Transition***.

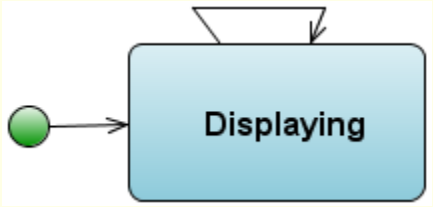
Aceasta sectiune contine urmatoarele proceduri:

3.1.3.1. Adaugarea unui element tranzitie intre doua stari distincte

1. Din sectiunea ***Basic*** a ***Pallete*** selectati pictograma ***State Transition*** .
2. Faceti click mai intai pe elementul ***Initial State*** si apoi pe elementul stare **Displaying**.
3. Deselectati pictograma ***State Transition***.

3.1.3.2. Adaugarea unui element tranzitie reflexiva

1. Din sectiunea ***Basic*** a ***Pallete*** selectati pictograma ***State Transition***.
2. Faceti click - mai intai pe elementul stare **Displaying**,
   - apoi putin deasupra acestui element,
   - apoi putin in dreapta si
   - in final din nou pe elementul stare **Displaying**.



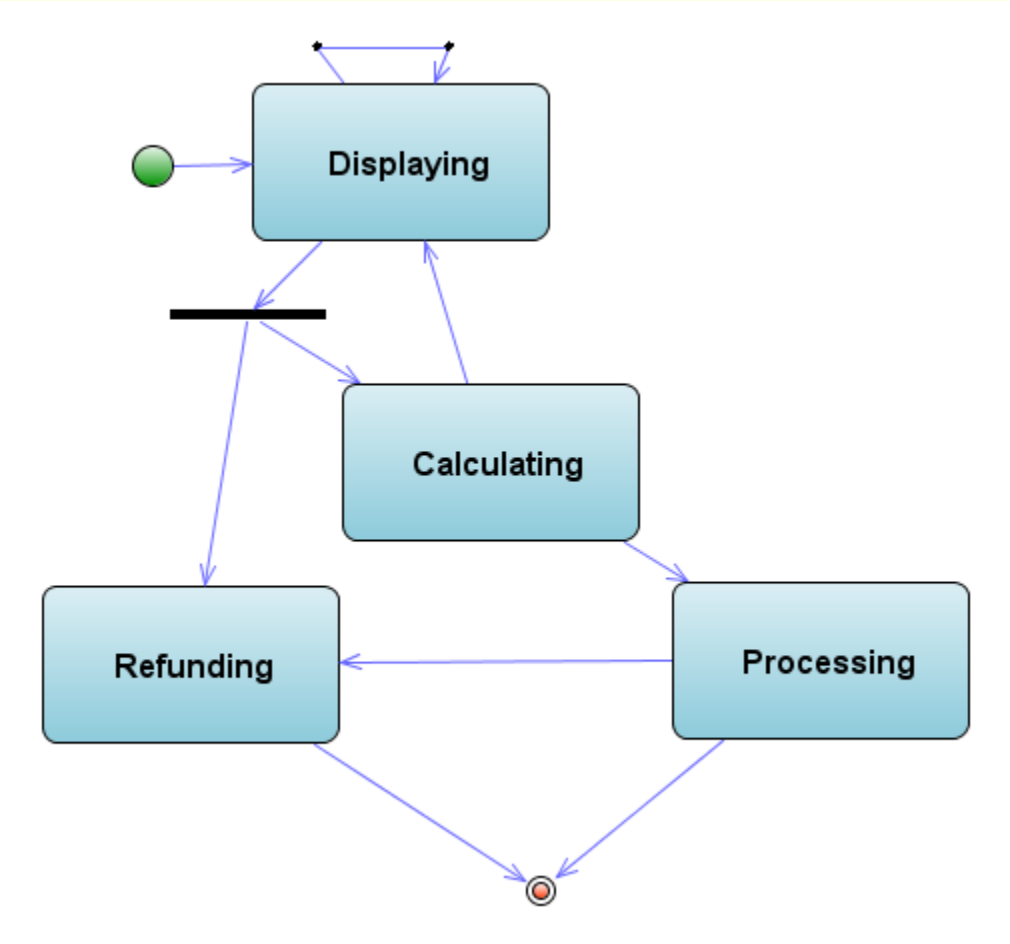
3. Deselectati pictograma ***State Transition***.

3.1.3.3. Adaugarea unui element join/merge orizontal

1. Din sectiunea ***Basic*** a ***Pallete*** selectati pictograma ***Horizontal Join/Merge*** .
2. Faceti click sub elementul stare **Displaying**.
3. Deselectati pictograma ***Horizontal Join/Merge***.

3.1.3.4. Adaugarea mai multor elemente tranzitie

1. Din sectiunea ***Basic*** a ***Pallete*** selectati pictograma ***State Transition***.
2. Adaugati 8 elemente tranzitie:
   - De la starea **Displaying** la elementul ***Horizontal Join/Merge***
   - De la elementul ***Horizontal Join/Merge*** la starea **Refunding**
   - De la elementul ***Horizontal Join/Merge*** la starea **Calculating**
   - De la starea **Calculating** la starea **Processing**
   - De la starea **Calculating** la starea **Displaying**
   - De la starea **Processing** la starea **Refunding**
   - De la starea **Refunding** la ***Final State***
   - De la starea **Processing** la ***Final State***
3. Deselectati pictograma ***State Transition***.



3.1.3.5. Etichetarea tranzitiilor cu nume de evenimente

Tranzitiile sunt in general **declansate de evenimente** (cu exceptia tranzitiilor automate).

1. Faceti right-click pe tranzitia (legatura) dintre ***Initial State*** si starea **Displaying**.
2. Selectati ***Label*** > ***Name*** si scrieti **Coins Entered.**
3. In acelasi fel etichetati cu nume de evenimente urmatoarele tranzitii:

Tranzitia	Numele evenimentului
De la starea **Displaying** la starea **Displaying**	**Coins Entered**
De la starea **Displaying** la elementul ***Horizontal Join/Merge***	**Button Pressed**
De la starea **Calculating** la starea **Displaying**	**Calculating Finished**
De la starea **Calculating** la starea **Processing**	**Calculating Finished**
De la starea **Processing** la ***Final State***	**Soda Ejected**
De la starea **Processing** la starea **Refunding**	**Soda Ejected**
De la starea **Refunding** la ***Final State***	**Coins Refunded**

3.1.3.6. Etichetarea tranzitiilor cu pre-/post-conditii

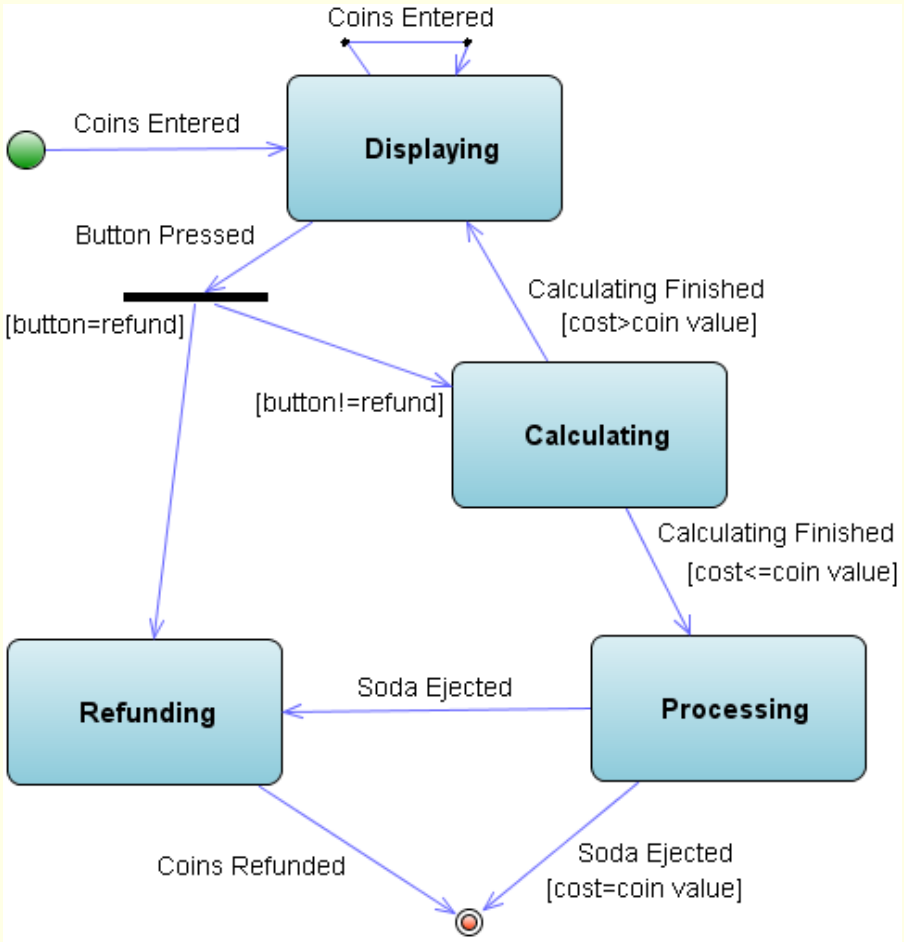
Anumite tranzitii sunt **realizate conditionat**. Pentru etichetarea lor pe legaturi (tranzitii) se folosesc **paranteze drepte intre care sunt scrise conditiile logice**.

1. Faceti right-click pe tranzitia (legatura) dintre ***Horizontal Join/Merge*** si starea **Refunding**.

2. Selectati ***Label > Pre Condition*** si scrieti **button=refund.**

3. In acelasi fel etichetati cu pre-conditii urmatoarele tranzitii:

Tranzitia	**Pre-conditia**
De la ***Horizontal Join/Merger*** la starea **Calculating**	**button!=refund**
De la starea **Calculating** la starea **Processing**	**cost<=coin value**
De la starea **Calculating** la starea **Displaying**	**cost>coin value**
De la starea **Processing** la ***Final State***	**cost=coin value**



3.2. Modificarea modelelor UML si reconstructia codului (Refactoring)

3.2.1. Pregatirea proiectelor demonstrative Java si UML

1. Din meniul principal al IDE-ului selectati ***File > New Project***. Apare asistentul ***New Project***.

2. În pagina ***Choose Project***, în panoul ***Categories*** expandaţi nodul ***Samples*** si selectati nodul ***UML***. Panoul ***Projects*** este actualizat cu proiectele UML disponibile.

3. În panoul ***Projects***, selectaţi **UML Bank App Sample** şi faceţi click pe ***Next***.

4. În pagina ***Name and Location***, lasati valoarea implicită **UMLBankAppSample** pentru **nume proiectului Java**.

5. Pentru campul ***Project Location***, click pe ***Browse*** pentru a naviga la folderul in care doriti sa salvati fisierele pentru proiectele demonstrative.

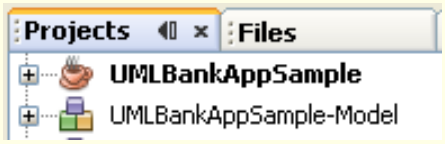
Selectati drive-ul **D:** directorul **\isw**, **subdirectorul cu numarul grupei** (de exemplu: **D:\isw\441E**).

De mentionat faptul ca atunci cand schimbati valoarea campului ***Projects Location***, IDE-ul completeaza automat valorile din campurile ***Java Project Folder*** si ***UML Project Folder***.

6. Lăsaţi valoarea implicită **UMLBankAppSample-Model** pentru **numele proiectului UML**.

7. Faceti click pe ***Finish***. Apare caseta de dialog ***Opening Projects***.

In fereastra ***Projects*** apar proiectele: **UMLBankAppSample** si **UMLBankAppSample-Model**.



8. In fereastra ***Projects***, sub nodul **UMLBankAppSample-Model** expandati nodul ***Model*** si expandati nodul **bankpack**.

9. Selectati directorul **bankpack** si toate elementele de sub directorul **bankpack** prin apasarea tastei ***Shift*** sau a tastei ***Ctrl*** atunci când faceti selecţia.

10. Faceţi click dreapta pe elementele selectate şi alegeţi ***Create Diagram From Selected Elements*** din meniul de pop-up. Apare asistentul ***Create New Diagram***.

11. In lista ***Diagram Type*** selectati ***Class Diagram***.

12. Scrieti **BankClassDiagram** in campul ***Diagram Name***, lasati **UMLBankAppSample-Model** in campul ***Namespace*** si faceti click pe ***Finish***. IDE-ul realizeaza urmatoarele:

Sub nodul ***Model*** creează un nod **BankClassDiagram**

In editorul de diagrama se afiseaza noua diagrama

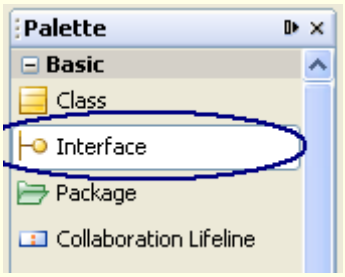
Deschide ***Palette***

3.2.2. Modificarea proiectului UML

In continuare veti invata **cum sa modificati un model UML**, direct sau utilizand diagrame grafice. Veti observa ca toate modificarile sunt automat sincronizate in proiectele asociate Java si UML.

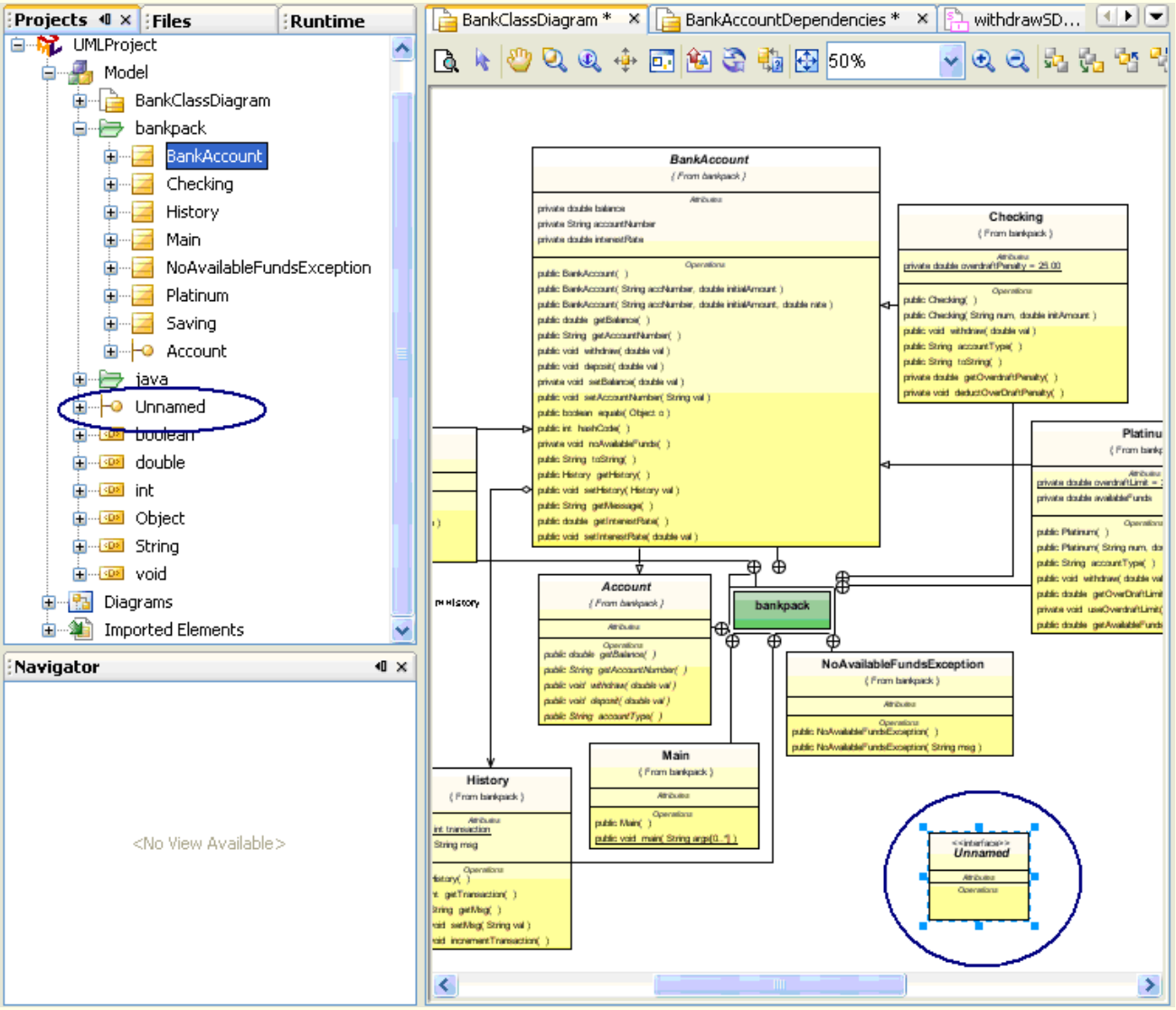
1. Activati diagrama **BankClassDiagram** facand click pe fila diagramei sau prin dublu click in arborele de proiect.

2. Selectati ***Interface*** in zona ***Palette***.



3. Faceti click pe butonul stanga al mouse-ului in spatiu liber din zona diagramei.

Veti obtine un nou element ***Interface*** in modelul UML



Observatie: Pana cand elementul nu are un nume nicio sursa nu este asociata cu el. Numele implicit este **Unnamed**, si poate fi schimbat.

Observatie: Aveti posibilitatea sa glisati un element catre un alt pachet, in arborele de proiect sau sa schitati link-ul de la elementul dvs. catre pachete de pe diagrama.

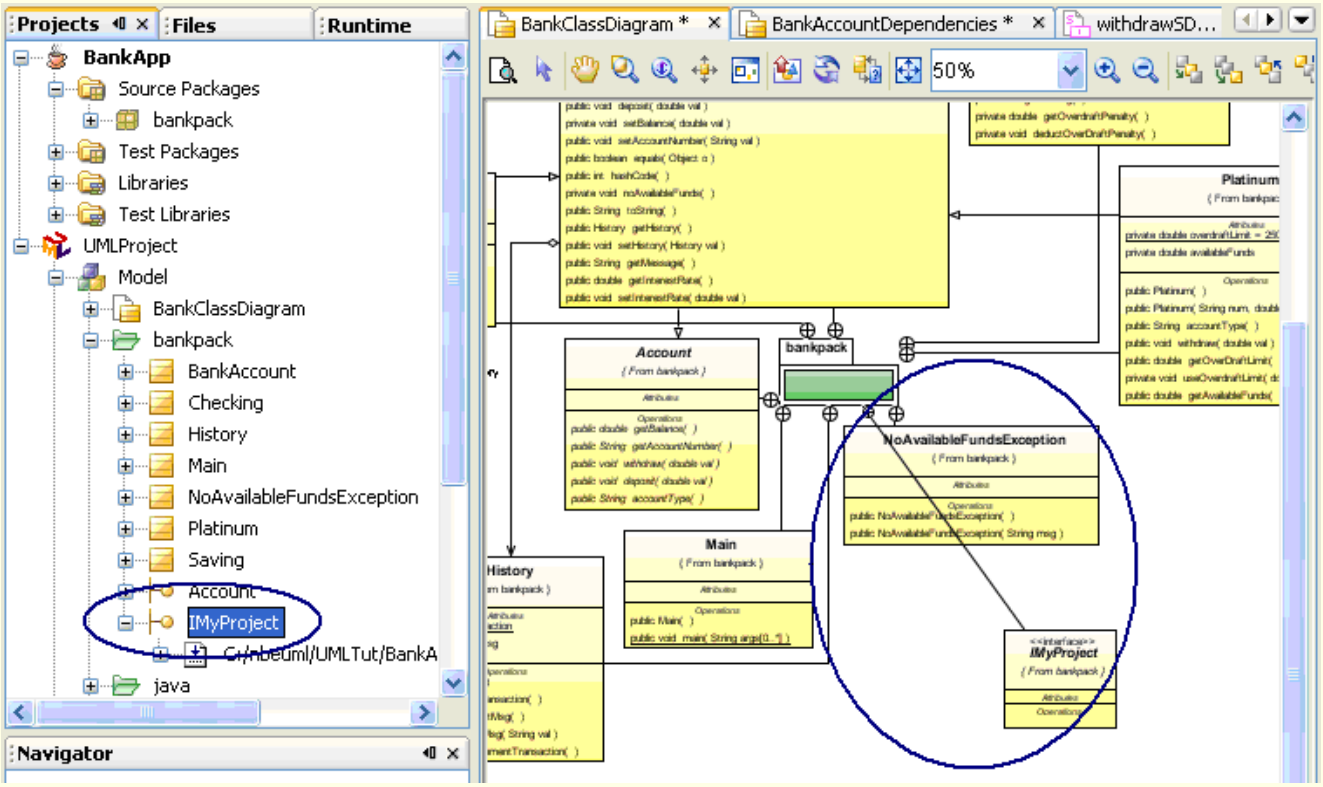
4. Selectati elementul ***Interface*** (daca nu este selectat). Scrieti numele **IMyProject** si apasati ***Enter***.

Acum aveti sursa implicita pentru interfata **IMyProject** in aplicatia Java asociata.

Observatie: In cazul in care incepeti sa scrieti imediat dupa selectie, numele vechi va fi inlocuit cu unul nou. In caz care aveti nevoie de o corectie pentru nume faceti dublu click pe nume spre a intra in modul de editare.

5. Selectati ***Nested Link*** in paleta UML. Faceti click pe interfata **IMyProject**. Faceti click pe pachetul **bankpack**.

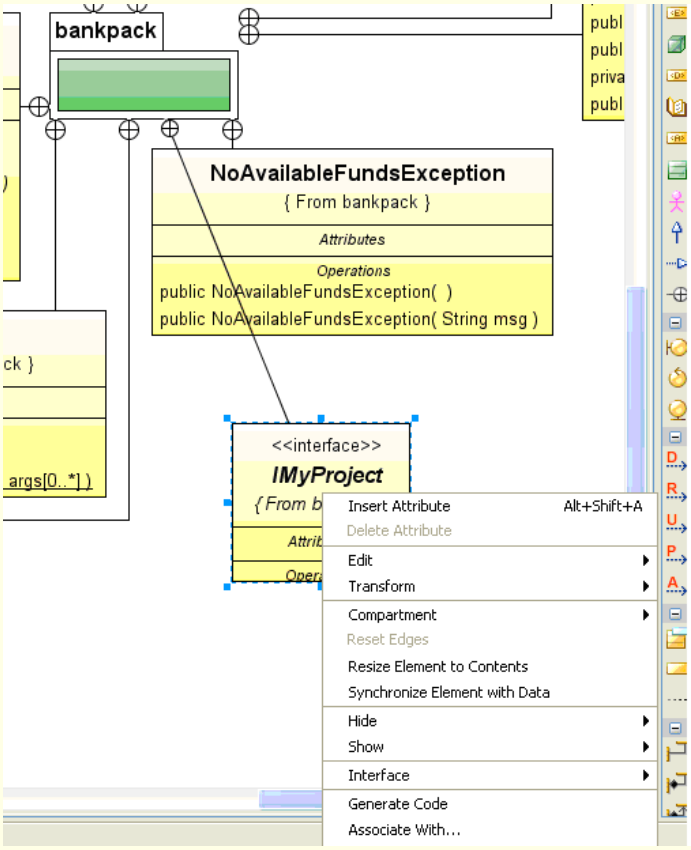
Acum, noua interfata este situata in pachetul de baza (in model si in proiectul Java).



6. Deselectati instrumentul ***Nesting Link*** facand click pe butonul dreapta al mouse-ului (sau prin ***ESC***).

7. Selectati **100%** in caseta de marire si derulati in zona de diagrama pana la interfata **IMyProject**.

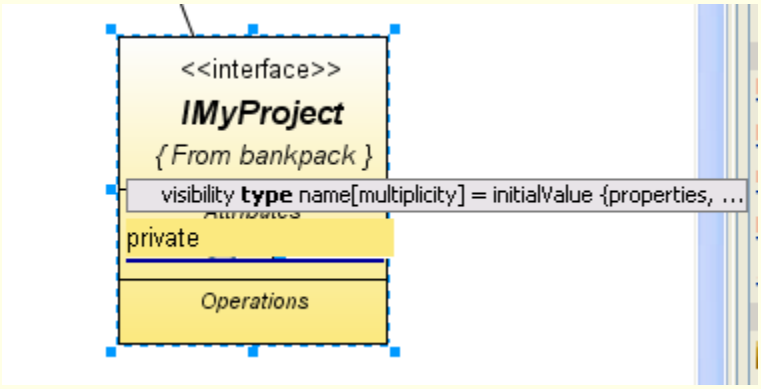
8. Invocati meniul contextual (*pop-up*) cu click dreapta pe titlul ***Attributes*** in cadrul interfetei.



9. Faceti click pe meniul ***Insert Attribute***.

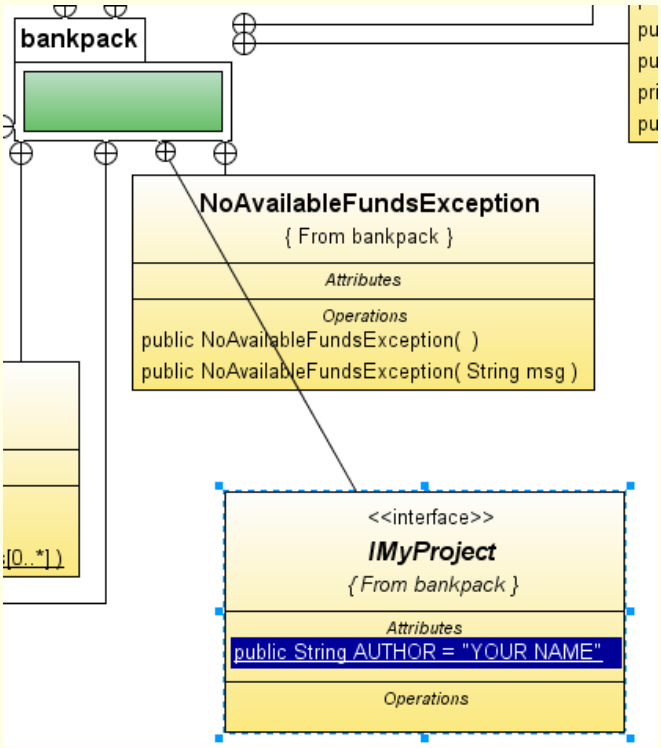
Un atribut cu nume implicit a fost adaugat in model, nicio sursa nu este asociata cu atributul pana cand unul are nume implicit.

10. Stergeti numele implicit (cu ***backspace***) si stergeti tipul implicit al atributului (**int**)



11. Scrieti **String** ca tip de atribut si **AUTHOR** ca nume implicit, apasati = si scrieti **"YOUR NAME"**, apoi apasati ***Enter***.

Acum aveti atributul in model si in sursa, si orice modificare a atributului din model va afecta codul sursa.



12. Invocati ***Navigate to Source*** din meniul contextual (*pop-up*) al elementului ***Interface*** (este necesar mai intai sa folositi ***Generate Code...***). Veti vedea sursa pentru noua interfata.

```
package bankpack;
public interface IMyProject {
   public static final String AUTHOR = "YOUR NAME";
}
```

Observatie: Puteti utiliza ***Navigate to Source*** din elementul nodului in proiectul UML, de asemenea in arborele de proiect sau accesand direct sursa din proiectul Java.

13. Activati fila diagramei **BankClassDiagram**. Adaugati o clasa diagramei (utilizati paleta), dandu-i numele **Utils**.

14. Mutati **Utils** in pachetul **bankpack**.

Ar trebui sa vedeti **Utils.class** in pachetul **bankpack** din proiectul Java cu sursa implicita.

```
package bankpack;
```

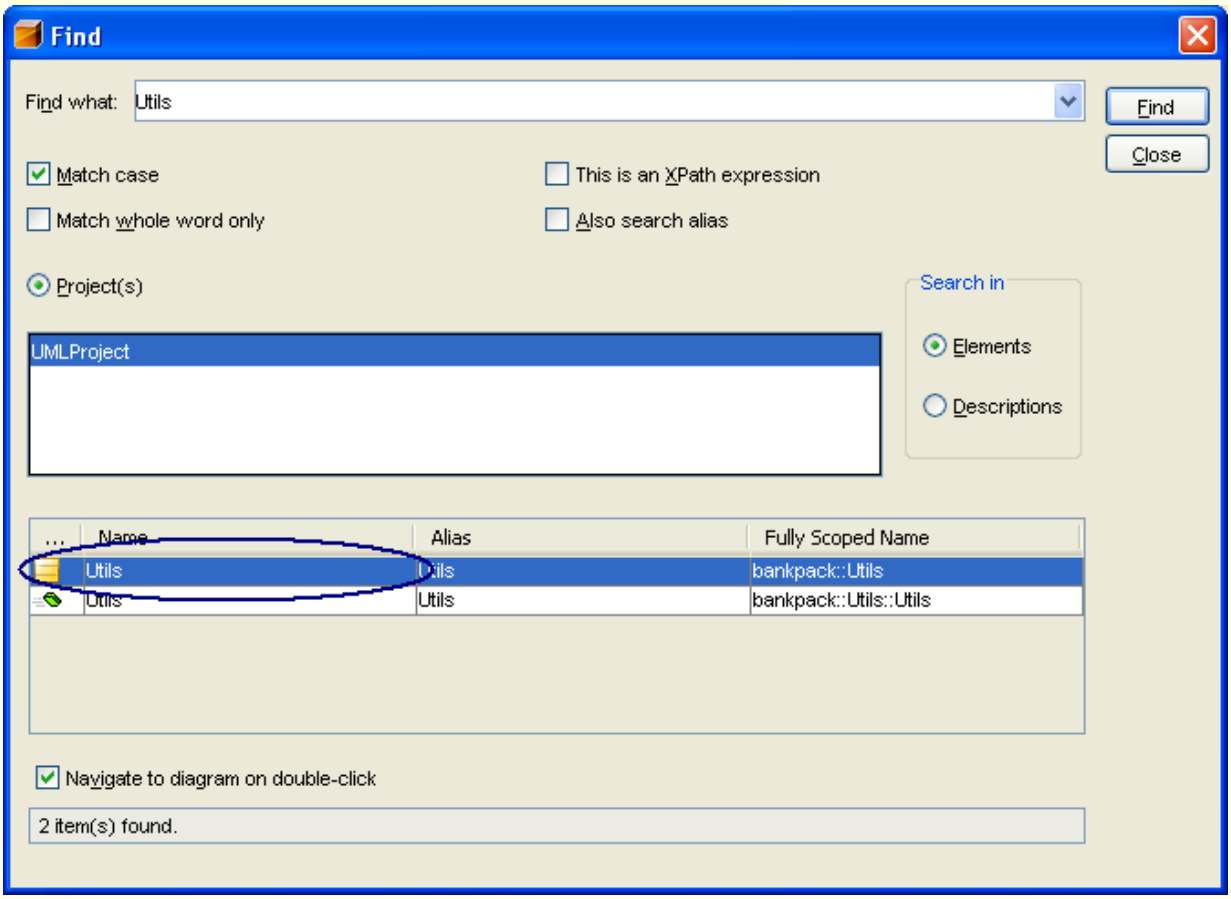
```
public class Utils {
    public Utils() {
    }
}
```

Observatie: Puteti adauga noi clase sau interfete la model folosind arborele proiectului, invocand meniul contextual (*pop-up*) pentru pachetul dorit si selectand ***Add*** > ***Element item***.

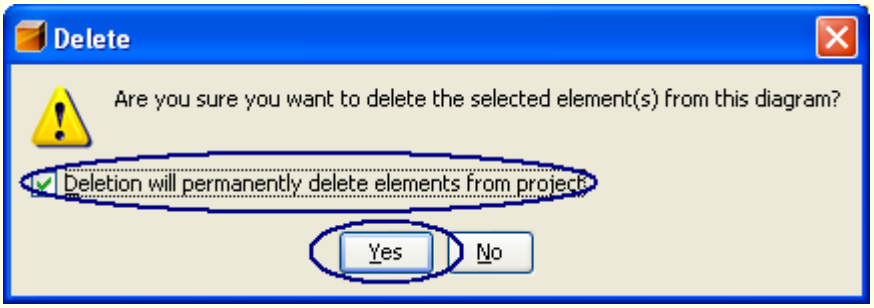
15. Activati fila diagramei **BankClassDiagram**. Selectati butonul ***Orthogonal Layout*** din bara de instrumente a diagramei si faceti click pe ***Yes*** in caseta de dialog de avertisment.

16. Selectati clasa **Utils** in diagrama.

Observatie: Puteti gasi clasa **Utils** invocand ***Edit*** > ***Find in Model...*** din meniul principal. Doar scrieti **Utils** pentru a descoperi si apasati ***Find***. Puteti da dublu click pe rezultatul cautat si diagrama corespunzatoare se va deschide, iar clasa **Utils** va fi selectata pe diagrama.



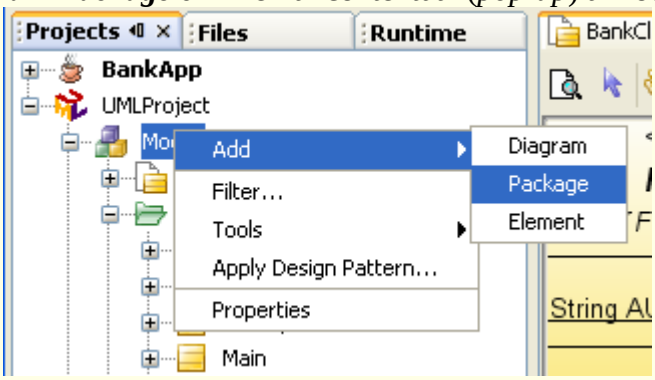
17. Apasati tasta ***Delete***. Confirmati stergerea obiectului (nu debifati caseta).



Clasa este eliminata atat din modelul UML (din arborele proiectului si toate diagramele) cat si din proiectul Java.

Observatie: Aveti posibilitatea de a elimina atribute si operatii din model si din sursa selectand si apasand pe ***Delete*** sau invocand ***Delete*** din meniul contextual (*pop-up*). **Atentie**, daca aveti de gand sa eliminati pachetul din diagrama cu optiunea stergere din model bifata, tot continutul pachetului va fi eliminat din model si din proiectul Java (daca diagrama este situata in pachet, atunci si diagrama va fi stearsa) .

18. Invocati ***Add*** > ***Package*** din meniul contextual (*pop-up*) al nodului ***Model***.

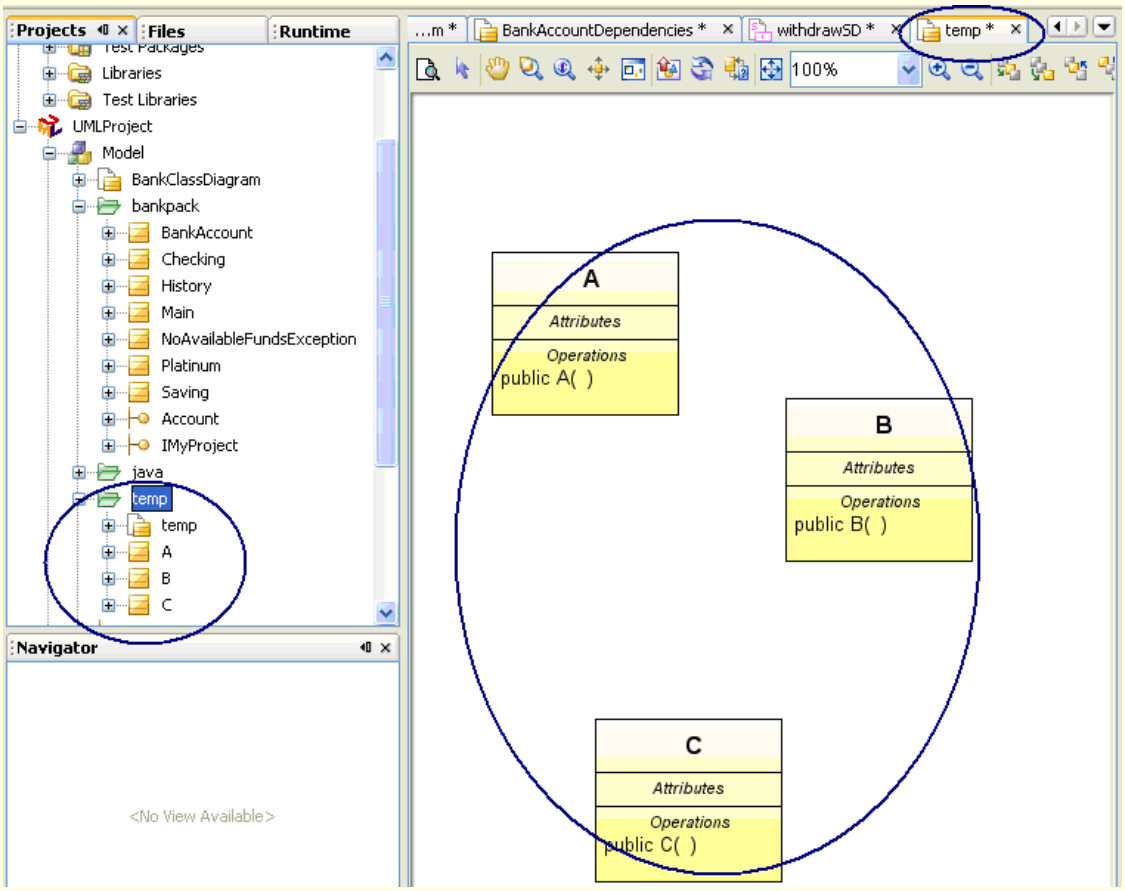


19. Scrieti numele pachetului **temp** in asistent.

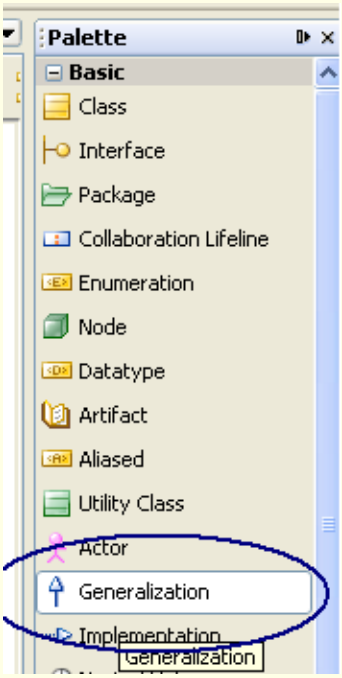
20. Selectati caseta ***Create Scoped Diagram***. Apasati pe ***OK***. Pachetul cu diagrama *scoped* apare in model. Diagrama este deschisa dupa creare. Pachetul gol nu este propagat catre proiectul Java.

21. Selectati ***Class*** in paleta. Faceti click pe zona din diagrama **temp**. Noua clasa **Unnamed** apare.

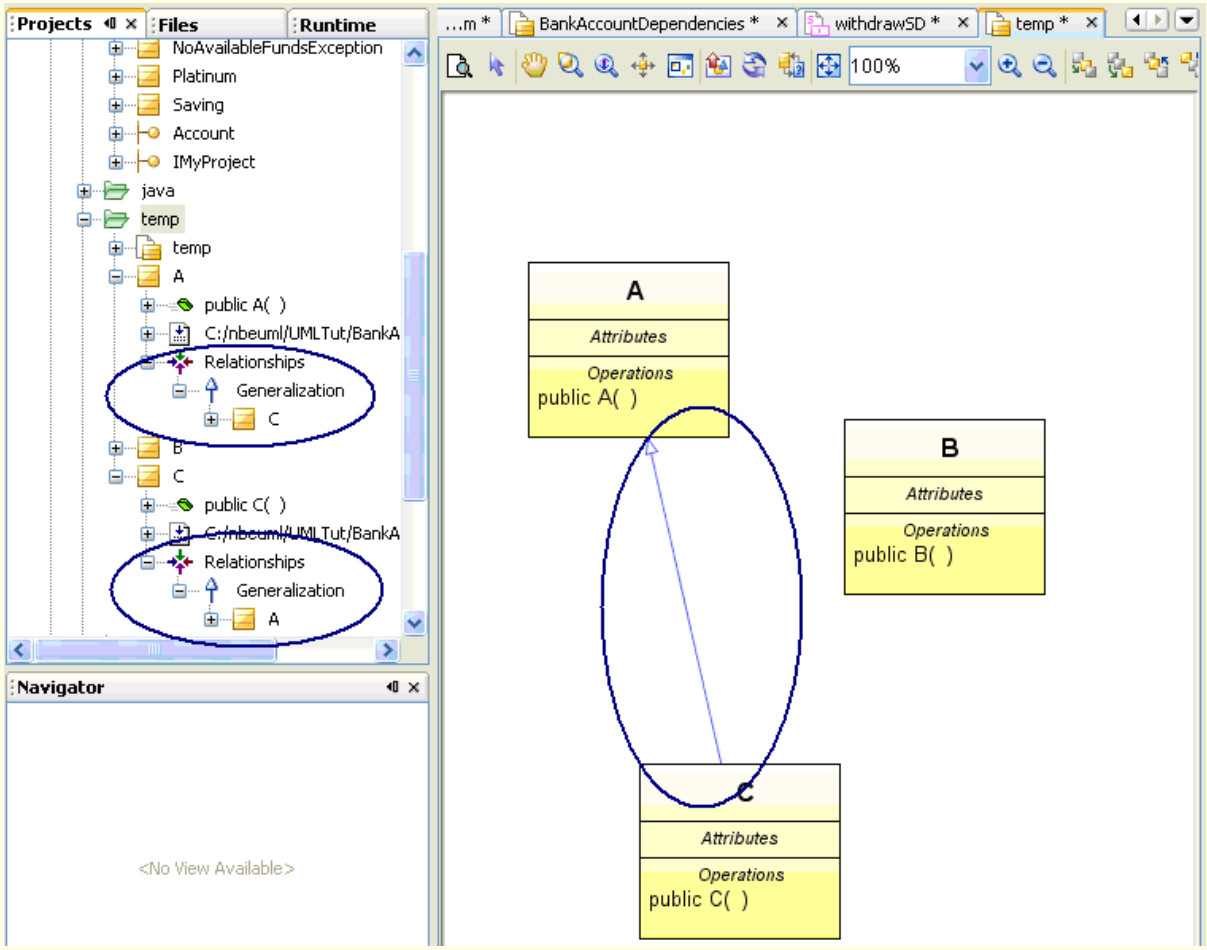
22. Scrieti numele **A** si faceti click in zona libera a diagramei. Prima clasa a fost numita **A,** si apare o a doua clasa, nedenumita. Denumiti-o **B** si creati o a treia clasa **C**. Toate clasele apar pe diagrama, in proiectul Java si in arborele proiectului.



23. Selectati legatura ***Generalization*** in paleta.

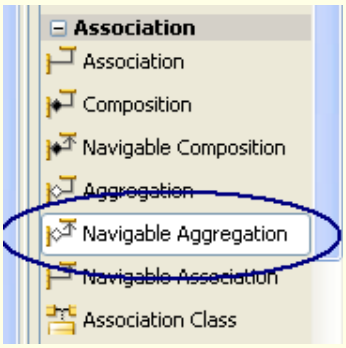


24. Faceti click pe clasa **C** din diagrama. Faceti click pe clasa **A** din diagrama. Relatia de generalizare apare pe diagrama, in arborele proiectului UML si se propaga catre proiectul Java.



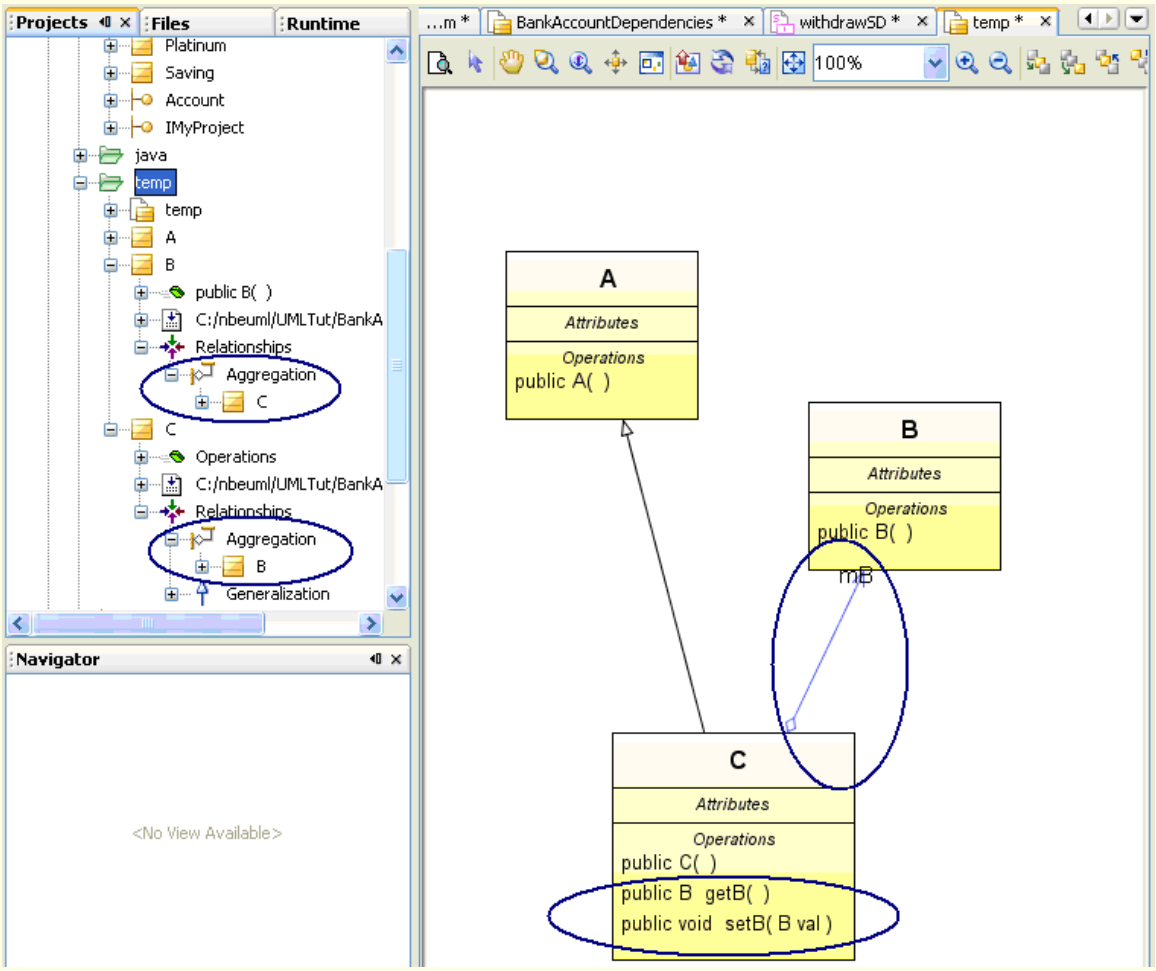

```
package temp;
public class C extends A {
    public C() {
    }
}
```

25. Selectati legatura ***Navigable Aggregation*** in paleta.



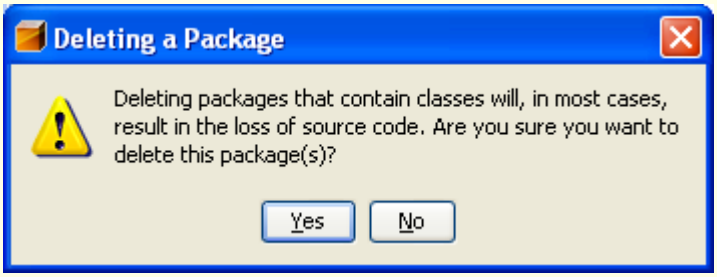
26. Faceti click pe clasa **C** si apoi pe clasa **B** de pe diagrama. O legatura ***Navigable Aggregation*** apare pe diagrama, in arborele modelului si se propaga catre proiectul Java.

Metode accessor sunt create in model si in sursa.




```
package temp;
public class C extends A {
    private B mB;
    public C() {
    }
    public B getB() {
        return mB;
    }
    public void setB(B val) {
        this.mB = val;
    }
}
```

27. Selectati pachetul **temp** din arborele proiectului. Invocati ***Delete*** din meniul contextual (*pop-up*). Selectati ***Yes*** in fereastra de dialog pentru a sterge obiectul. Selectati ***Yes*** pentru a confirma stergerea pachetului.



28. Selectati ***Yes*** pentru a confirma stergerea celorlalte elemente.

Pachetul **temp** cu tot continutul, incluzand toate clasele si diagrama de clase, este eliminat din proiectul UML. Pachetul de test cu clasele sale este eliminat din proiectul de Java. Toate filele pentru diagrama si clasele sterse sunt inchise.

3.2.3. Modificarea surselor Java

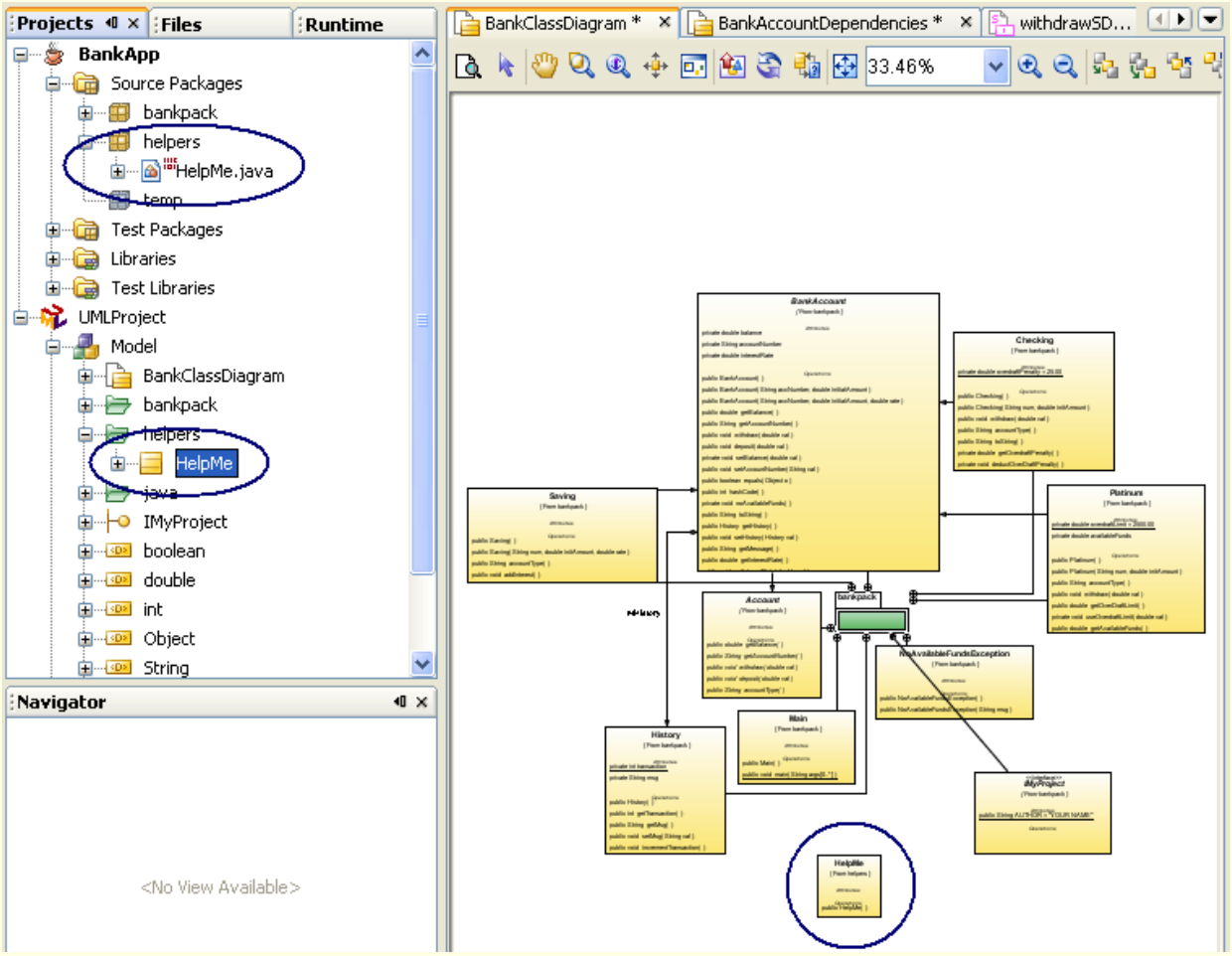
In continuare veti invata **cum sa modificati sursele Java**.

1. In proiectul Java adaugati un pachet numit **helpers**. Pachetul gol apare in proiectul Java, dar nu se propaga catre proiectul UML.

2. Adaugati in pachet clasa **HelpMe**. Folositi ***Reverse Engineer...***.

Pachetul **helpers** si clasa **HelpMe** apar in model. Clasa si pachetul pot fi glisate catre diagrama. Sau pot fi modificate din cadrul proiectului UML.

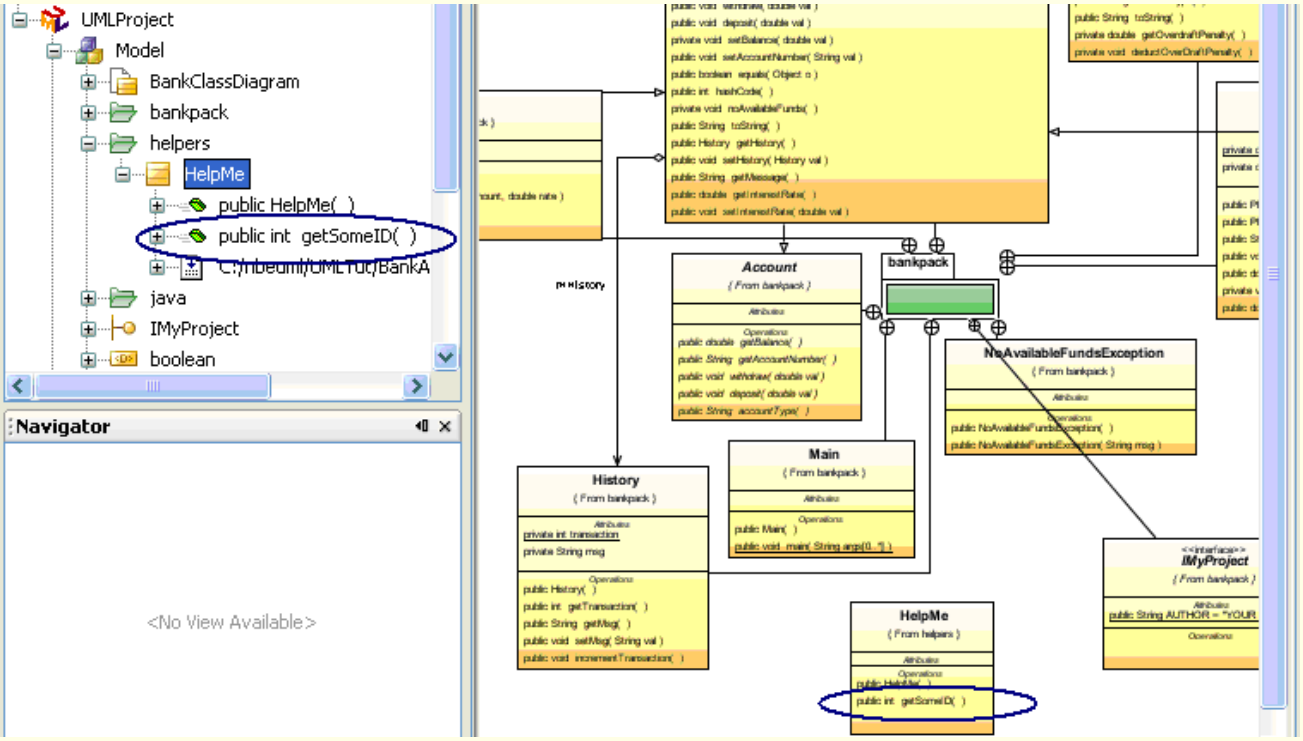
3. Activati diagrama **BankClassDiagram**. Glisati clasa **HelpMe** catre diagrama.



4. Activati fila sursa. Adaugati o noua metoda in codul sursa al clasei **HelpMe**

```
public int getSomeID () {
    return 0;
}
```

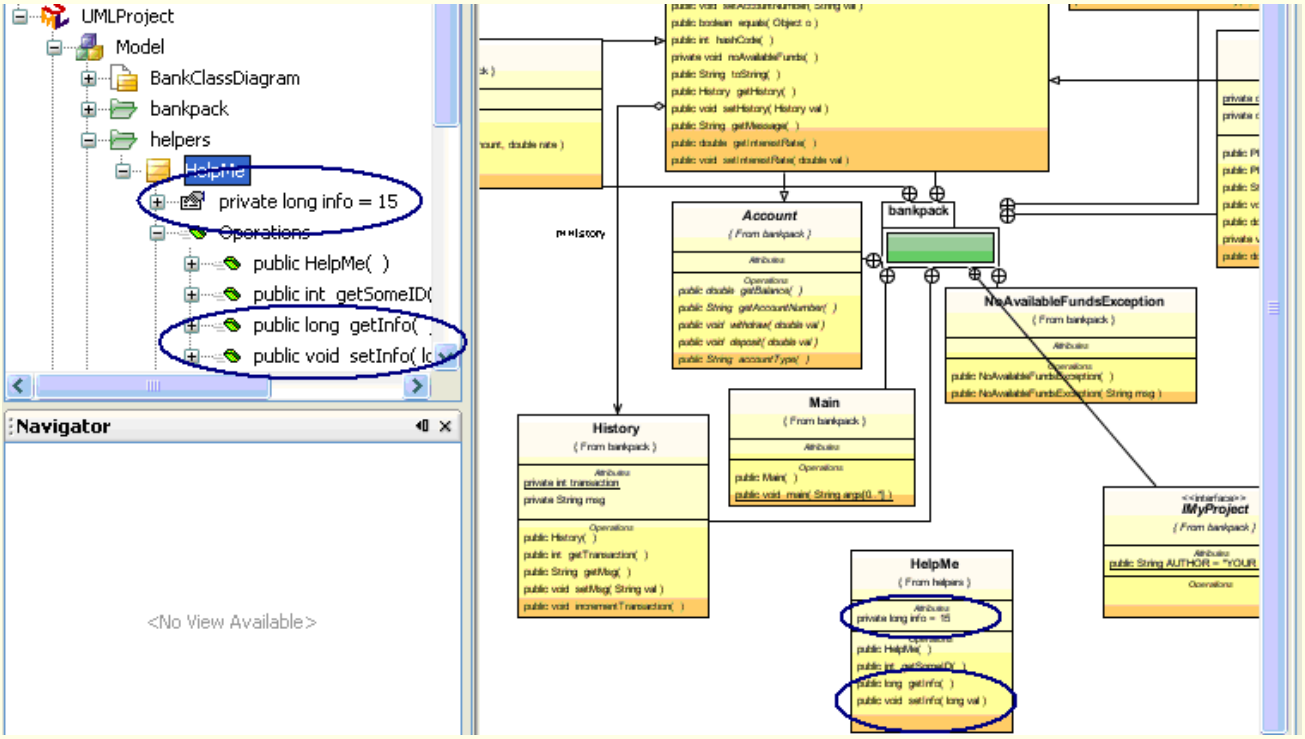
Metoda apare in model in nodul **HelpMe** si pe diagrame (daca folositi ***Reverse Engineer...***).



5. Adaugati un atribut in codul sursa al clasei **HelpMe**

```
private long info = 15;
```

Metode de tip accesor (**getInfo()** si **setInfo()**) sunt generate in sursa. Atributul si metodele accesor sunt propagate catre model (daca folositi ***Reverse Engineer...***).



6. Adaugati in codul sursa al clasei **HelpMe**

```
import bankpack.*;
```

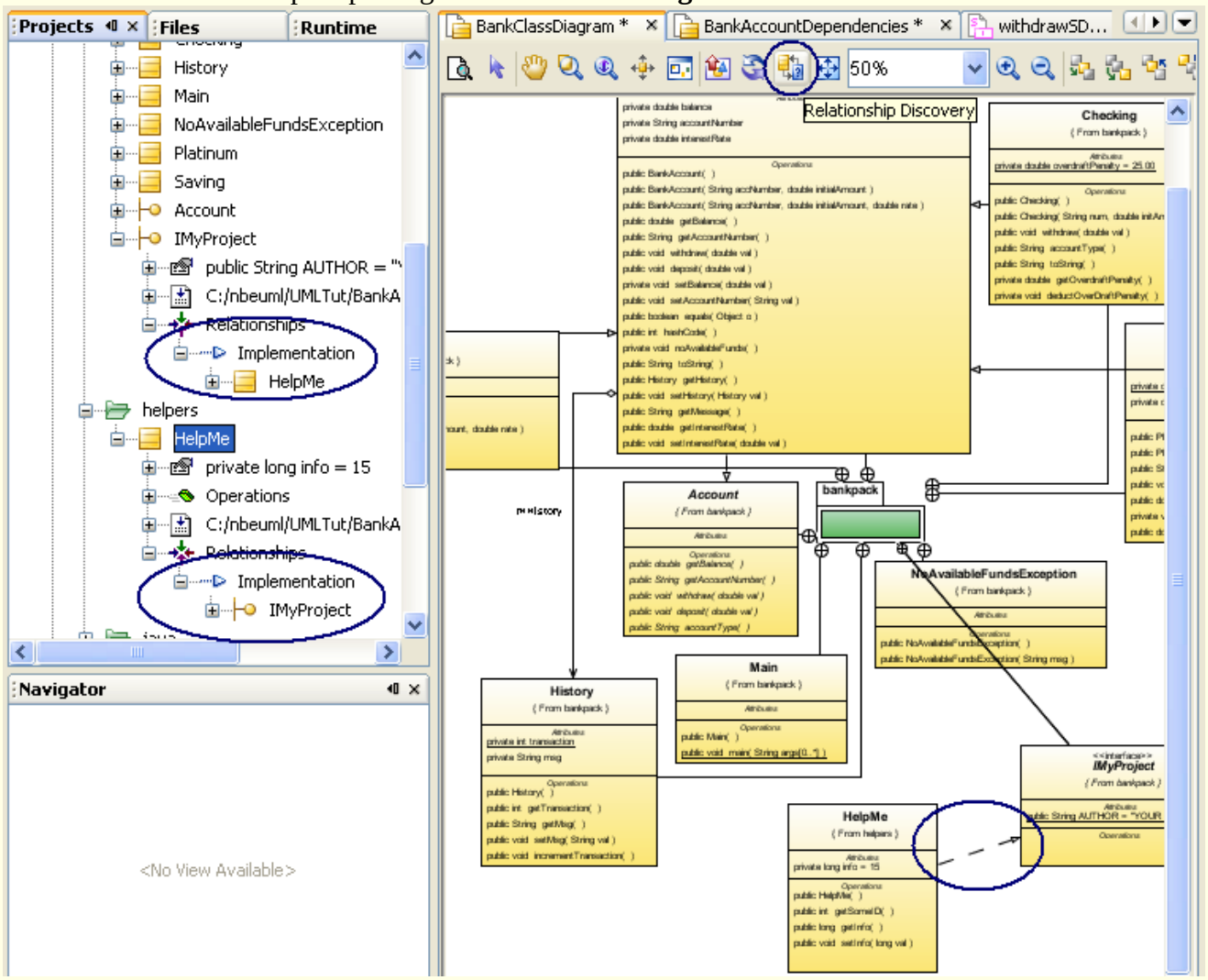
7. Adaugati in declaratia clasei **HelpMe**

```
implements IMyProject
```

Relatia este propagata catre modelul UML (daca folositi ***Reverse Engineer...***).

8. Activati diagrama **BankClassDiagram**. Apasati butonul ***Relationship Discovery*** din bara de instrumente.

Relatia ar trebui sa apara pe diagrama **BankClassDiagram**.



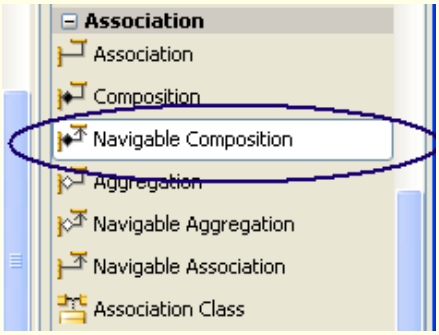
3.2.4. Reconstructia codului (*Refactoring*)

In continuare veti invata **cum sa reconstruiti codurile sursa Java**.

1. Adaugati un nou element ***Class*** pe diagrama BankClassDiagram. Numiti-o **HelpUser**.

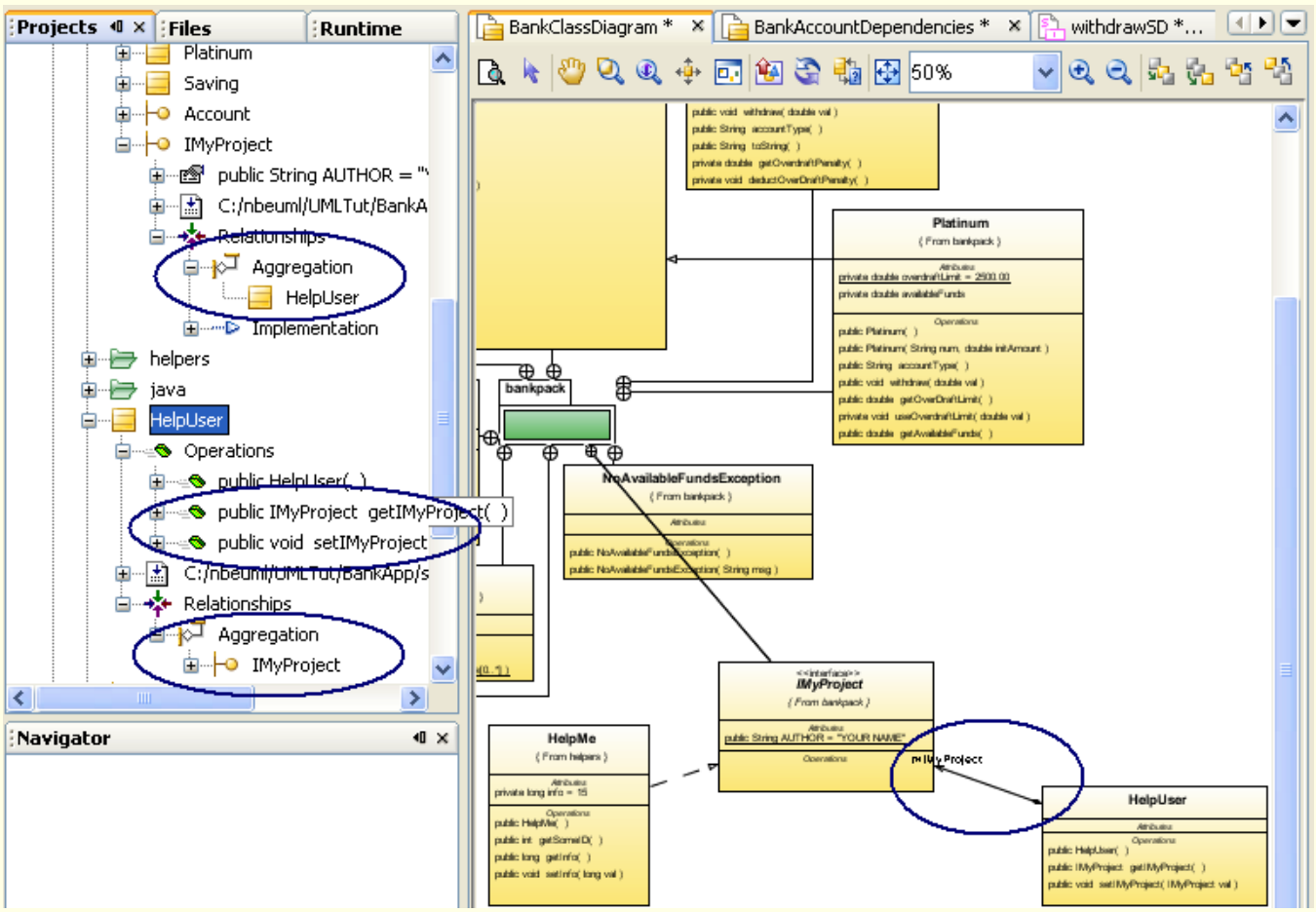
Elementul clasa apare pe diagrama si in model.

2. Selectati legatura ***Navigable Composition*** in paleta.

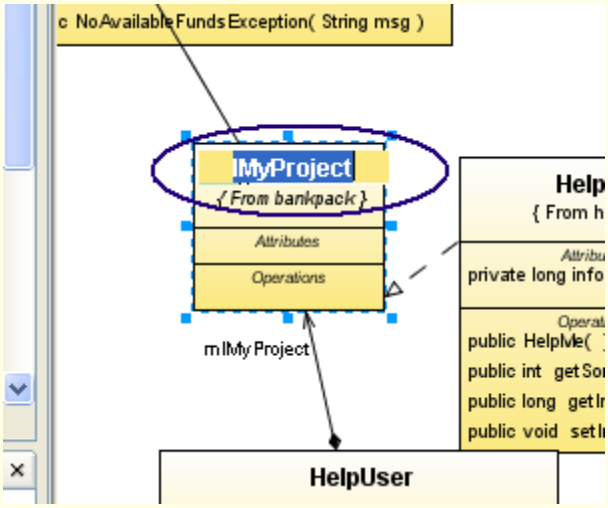


3. Faceti click pe clasa **HelpUser** si apoi pe clasa **IMyProject** din diagrama.

Navigable Composition apare pe diagrama, in arborele modelului si se propaga in proiectul Java. Metode accesor sunt create in model si in sursa.



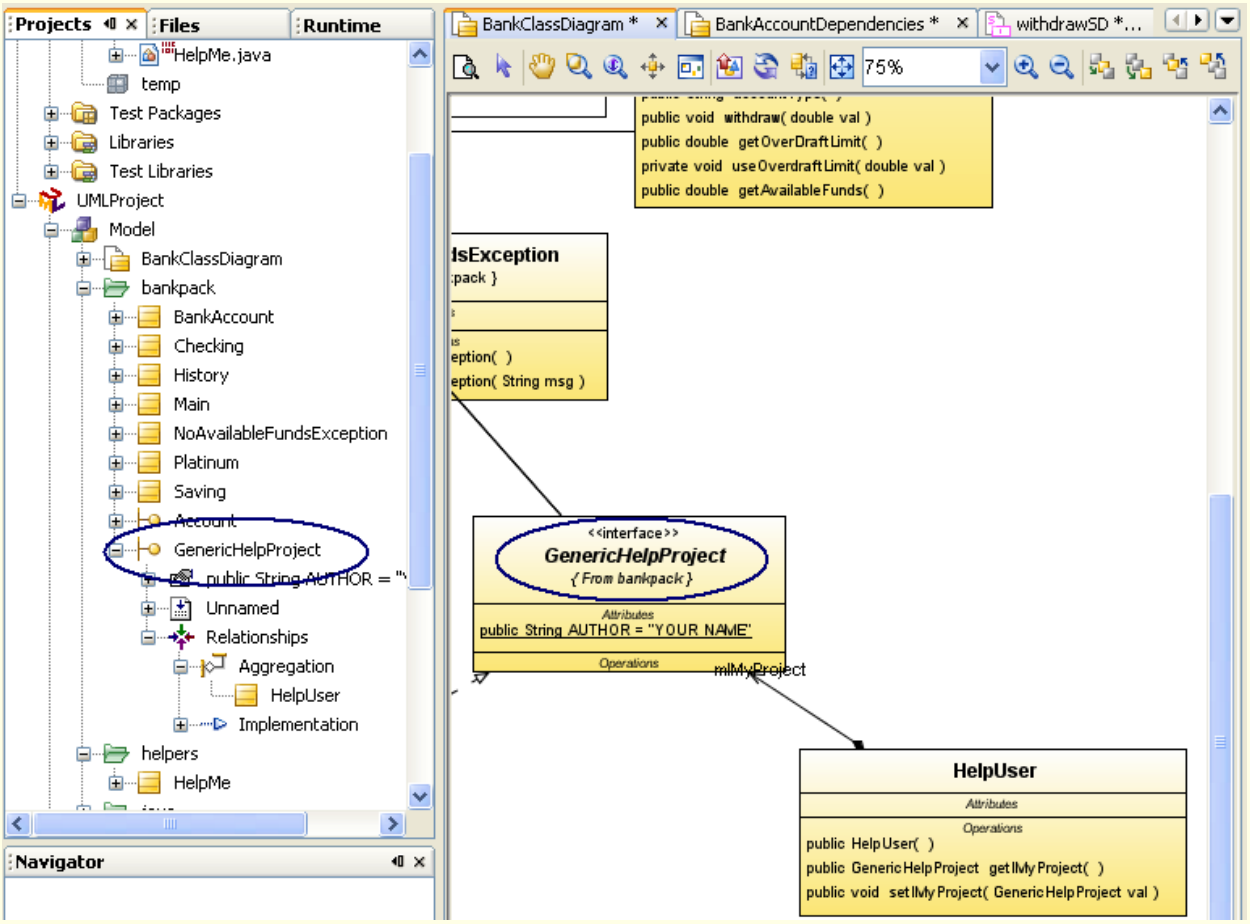
4. Faceti dublu click pe numele **IMyProject** din elementul diagrama **IMyProject** pentru a deschide editorul de nume. Acum puteti sa scrieti noul nume.



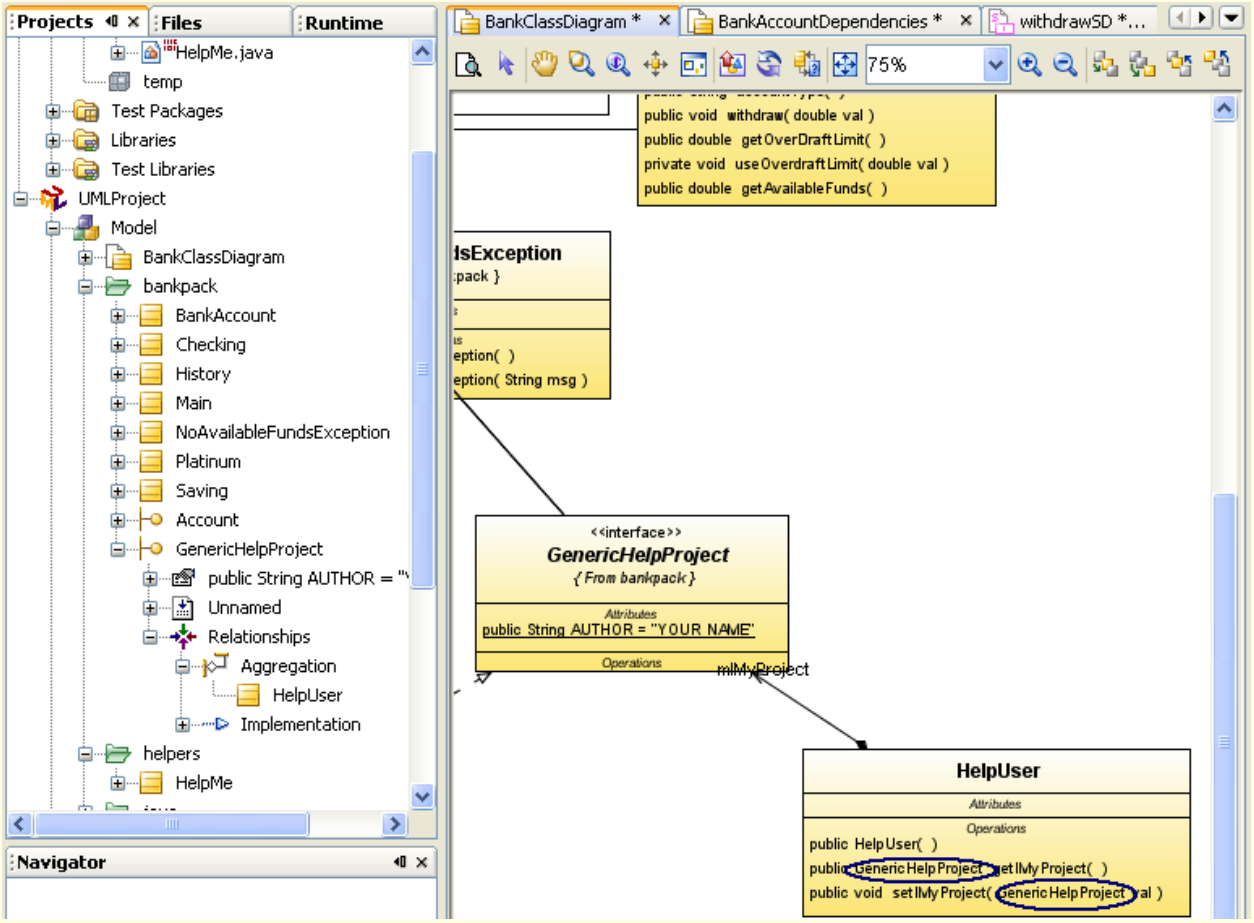
5. Redenumiti interfata **IMyProject** in **GenericHelpProject**.

Este redenumit elementul ***Interface*** in diagrama, in model si in sursa.

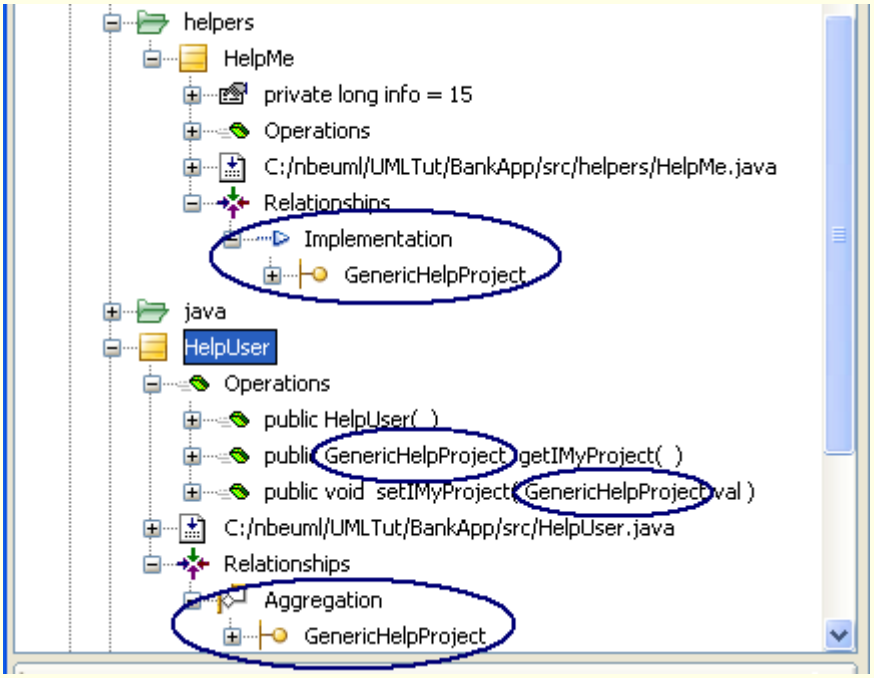
```
package bankpack;
public interface GenericHelpProject {
   public static final String AUTHOR = "YOUR NAME";
}
```



6. Dupa redenumirea elementului, este realizata automat reconstructia (***Refactoring***) care **redenumeste toate utilizarile vizibile pe diagrama**: adica **tipul parametrului** si **tipul returnat** in **cazul metodelor accesor** generate pentru legatura ***Composition***.

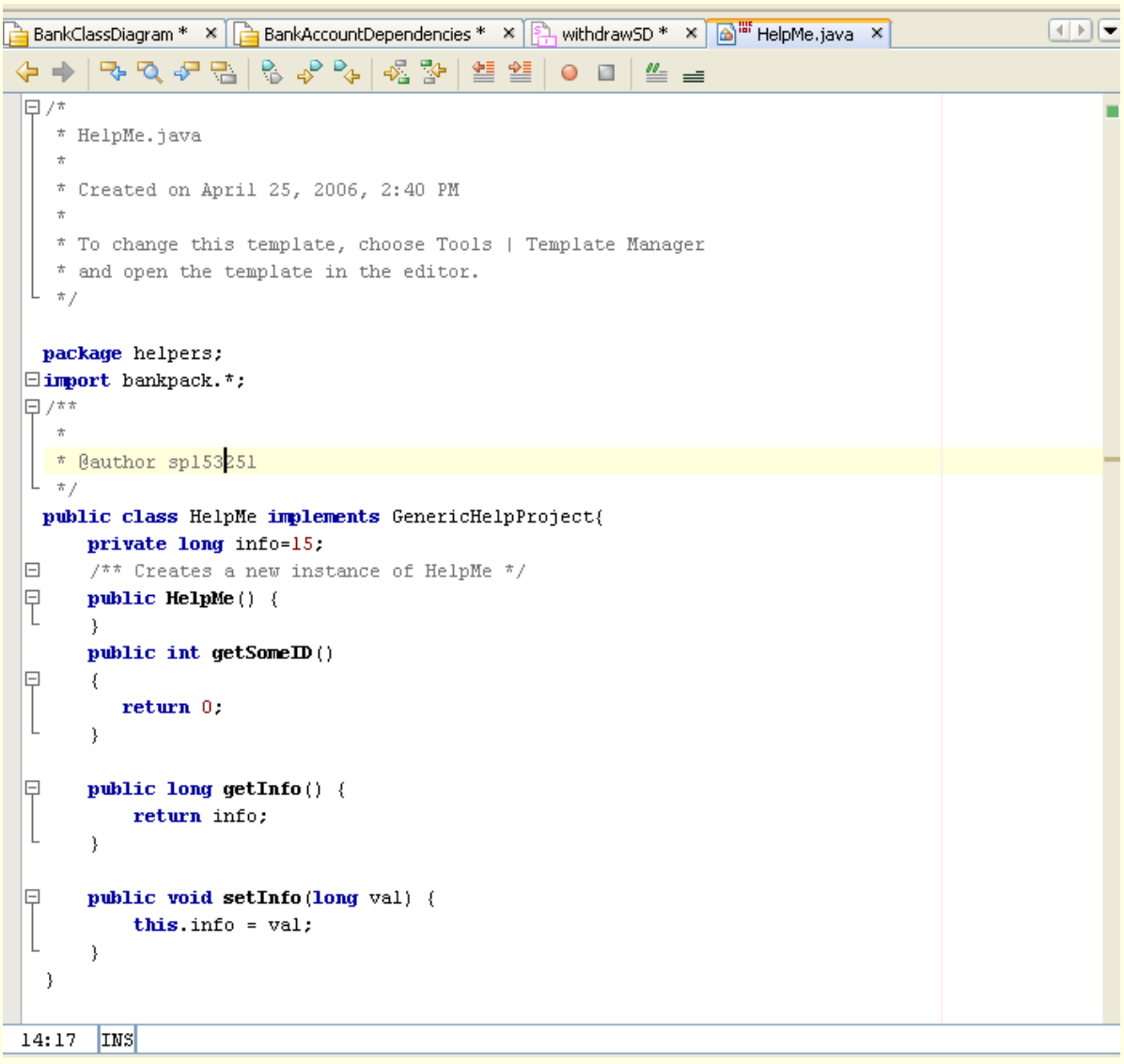


Sunt **redenumite si utilizarile elementelor modificate**: in relatia ***Generalization*** si in metodele accesor generate de relatia ***Composition***.



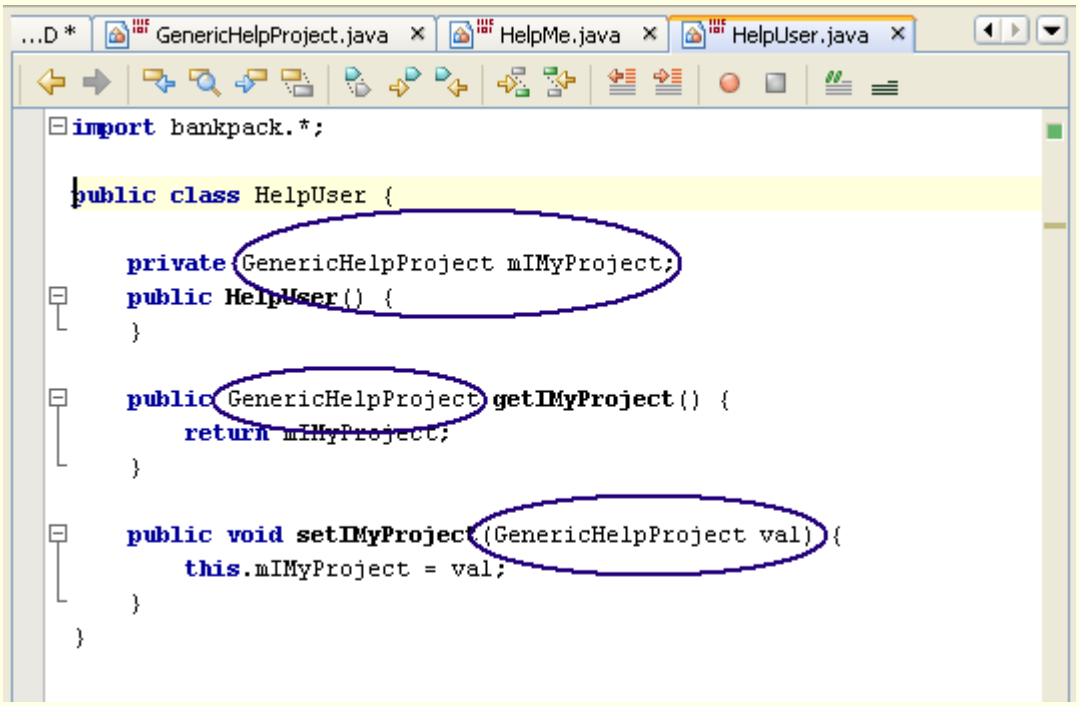
7. Deschideti fisierul sursa pentru clasa **HelpMe**.

Observati ca **este redenumita si utilizarea** elementului ***Interface*** **in clasa extinsa**



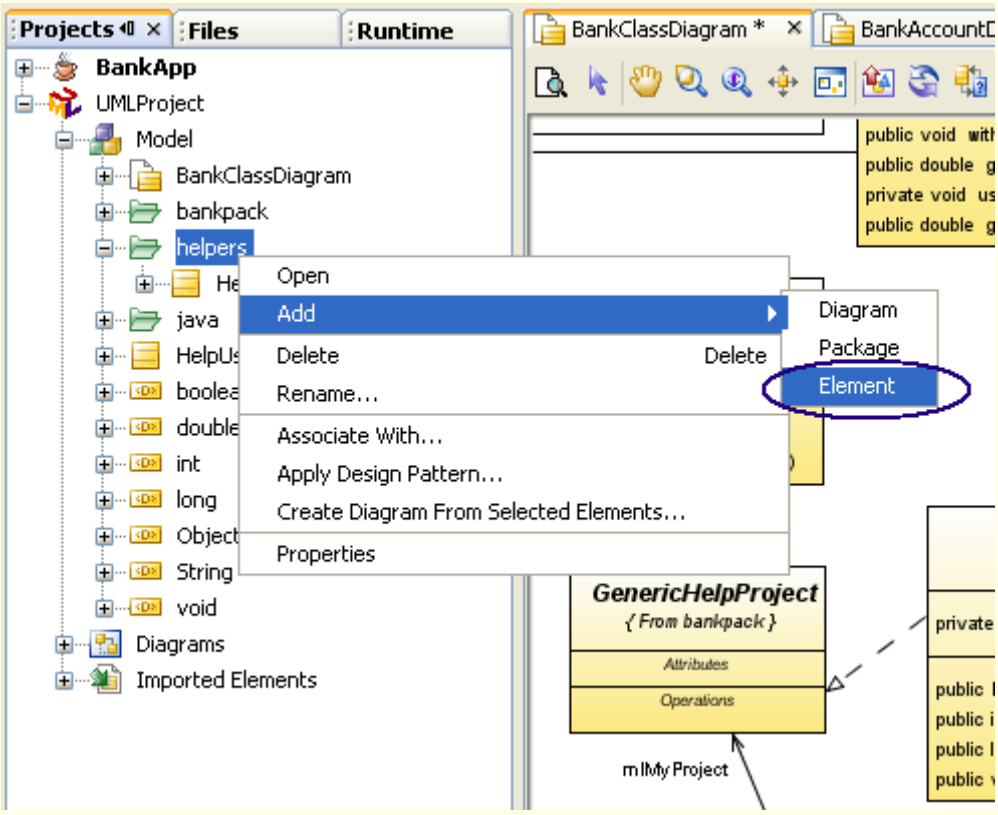
8. Deschideti fisierul sursa pentru clasa **HelpUser**

Observati ca **sunt redenumite si tipul atributului si tipurile din metodele accesor.**

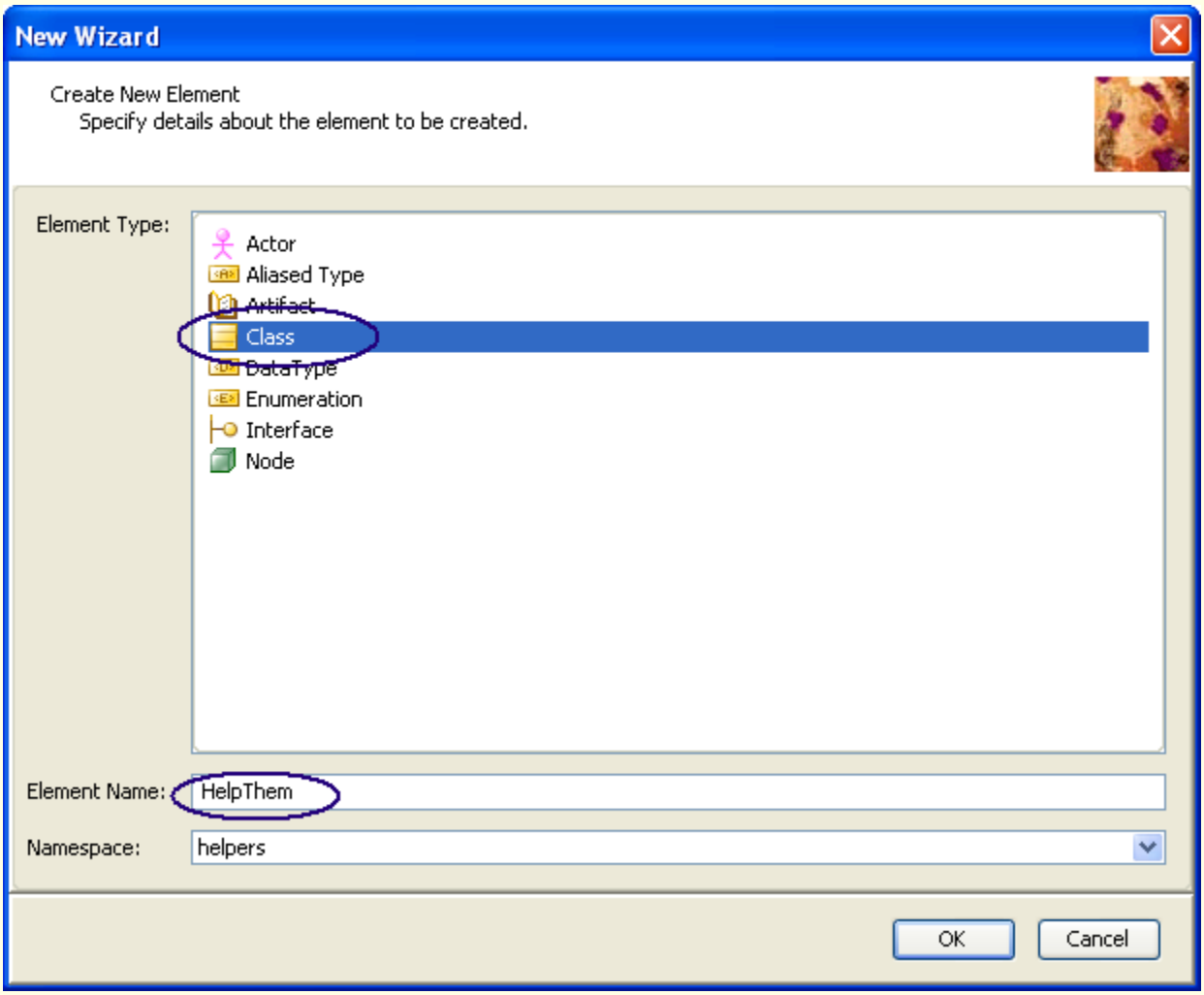


3.2.5. Modificarea modelului UML din arborele proiectului

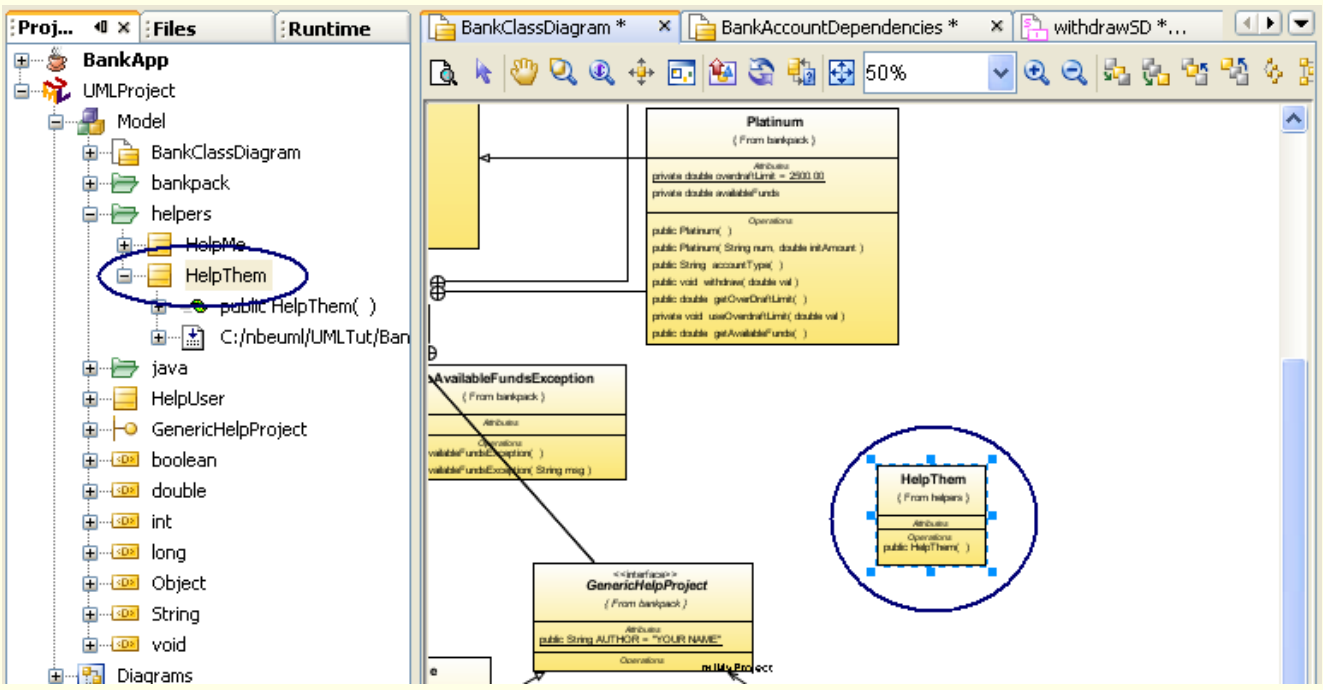
1. Faceti click cu butonul dreapta al mouse-ului pe nodul helpers din arborele proiectului UML. In meniul contextual deschis, selectati ***Add*** > ***Element***



2. In asistent selectati ***Class*** la ***Element Type***. Scrieti **HelpThem** in campul ***Element Name***. Apasati **OK**.

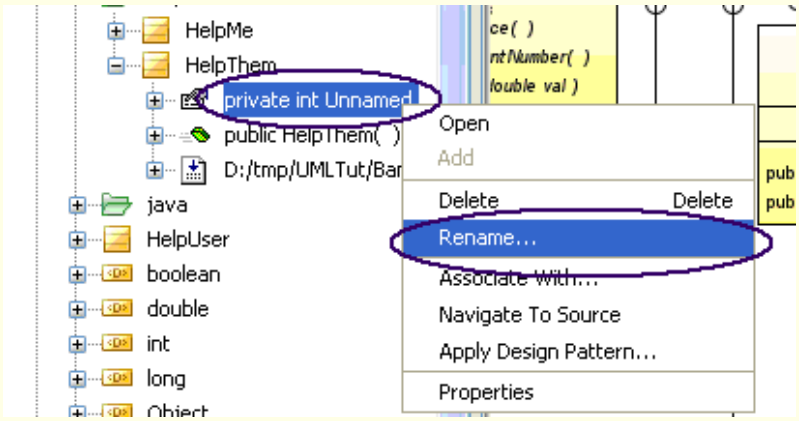


3. Un nou nod clasa **HelpThem** este creat. Expandati nodul clasei create pentru a vedea constructorul creat. Glisati clasa **HelpThem** din arbore in diagrama.

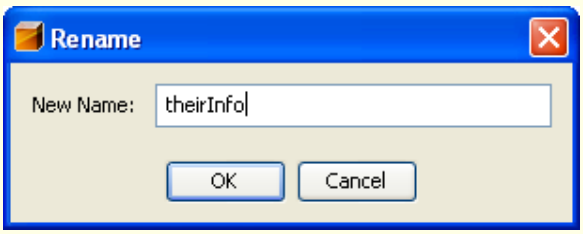


4. Deschideti meniul contextual pentru nodul clasei si invocati comanda ***Add > Atributte***. Este creat un nou atribut **private int Unnamed**.

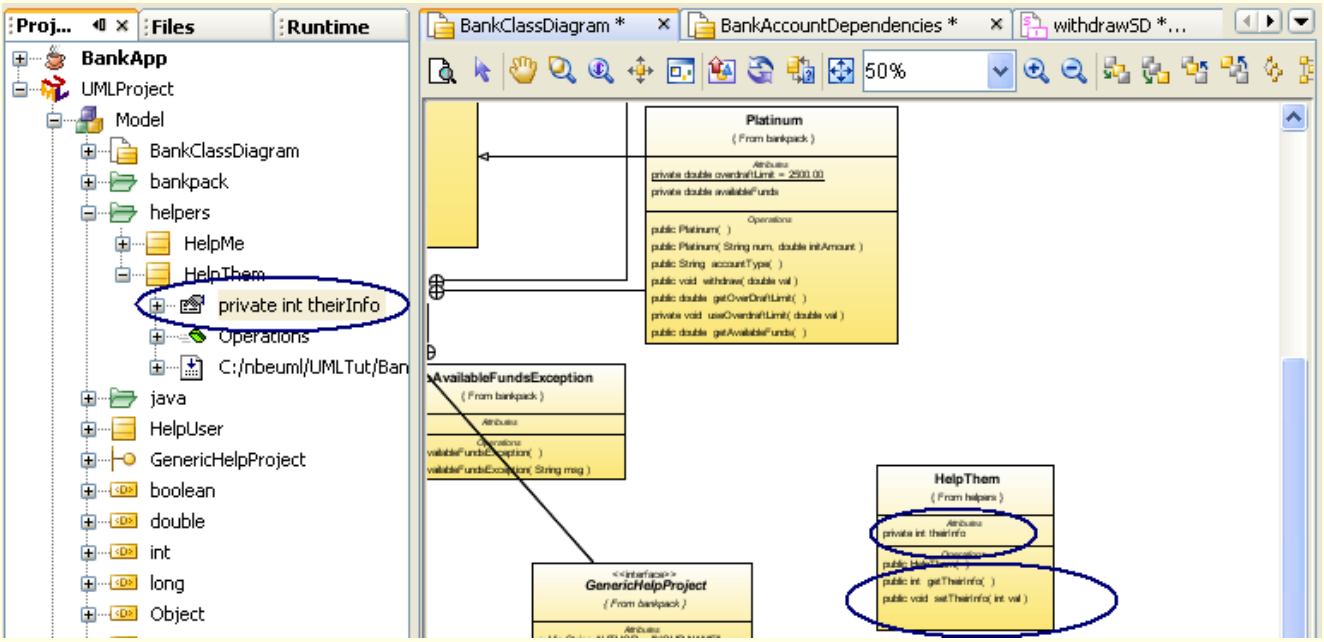
5. Deschideti meniul contextual pentru nodul atribut creat. Invocati comanda ***Rename***.



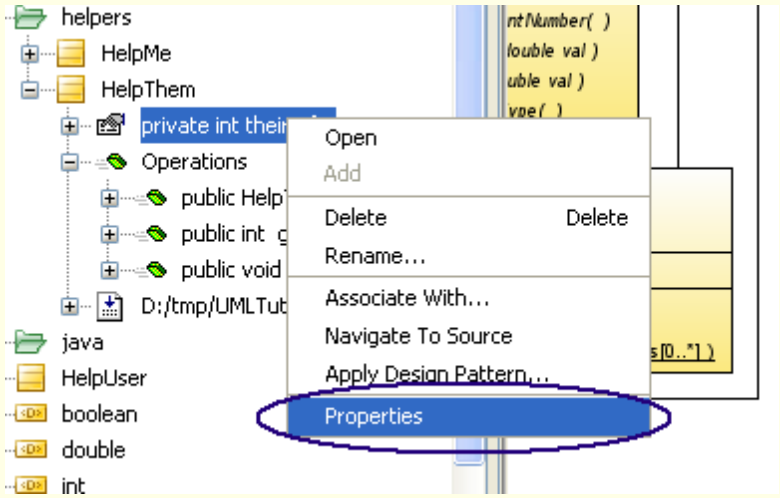
6. Scrieti noul nume, **theirInfo,** si apasati butonul ***OK***



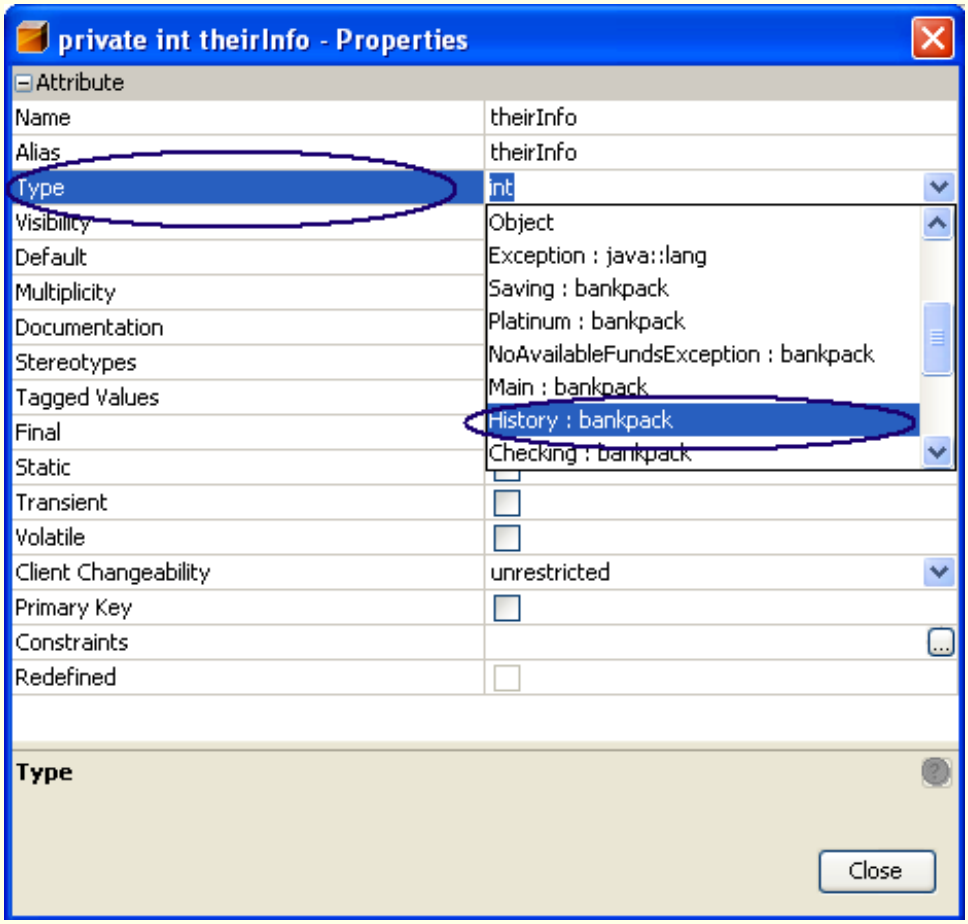
7. Atributul este redenumit. Sunt create metode accesor.



8. Deschideti ***Properties*** pentru nodul atribut **theirInfo**.

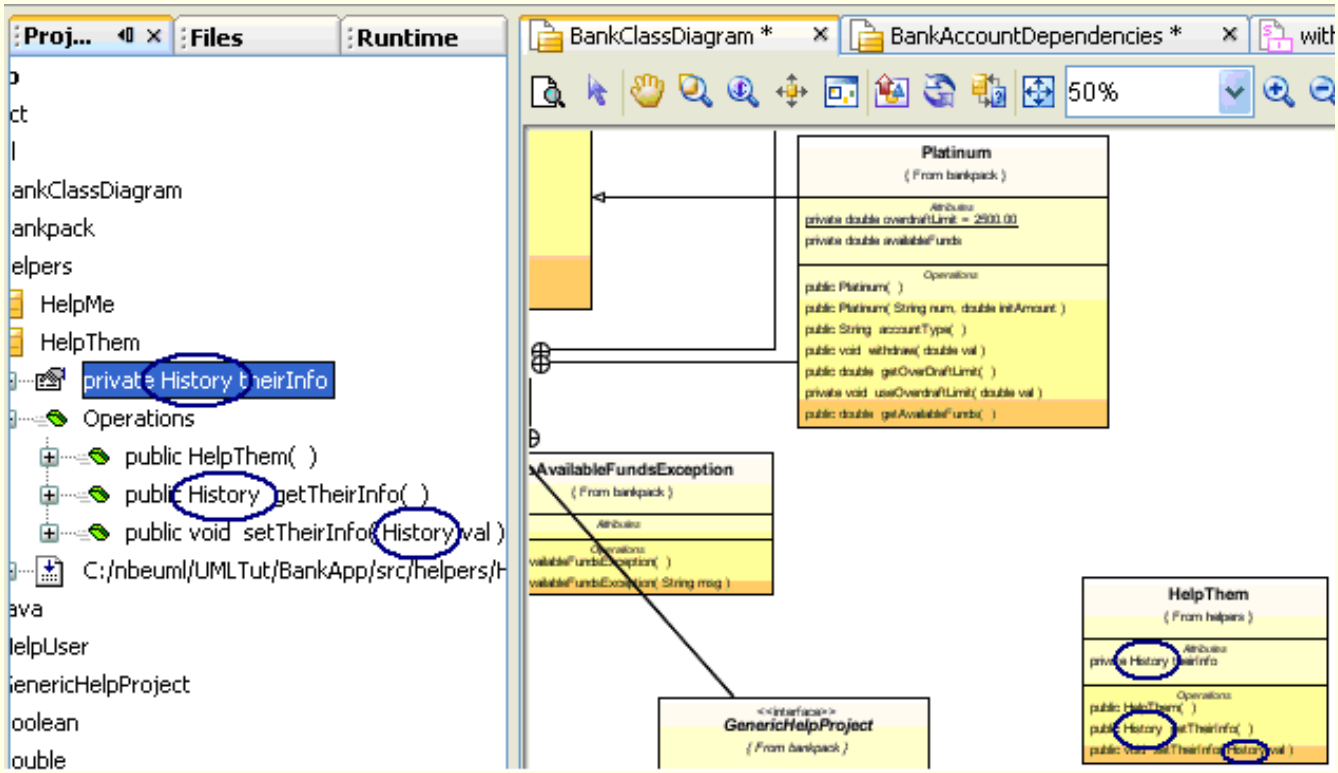


9. In campul ***Type*** schimbati din **int** in **History: bankpack**



Inchideti fereastra ***Properties***.

10. Tipul atributului si tipurile folosite in metodele accesor au fost schimbate.

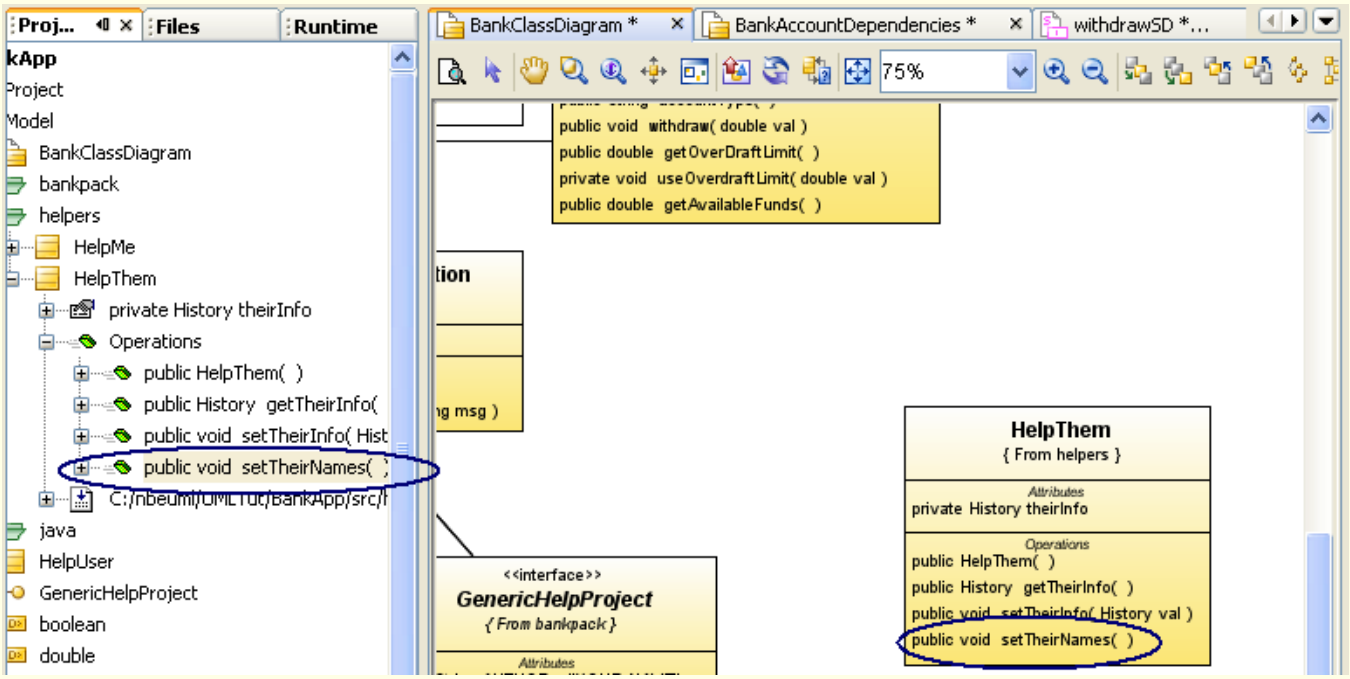


11. Deschideti meniul contextual pentru nodul clasei si invocati comanda ***Add > Operation***.

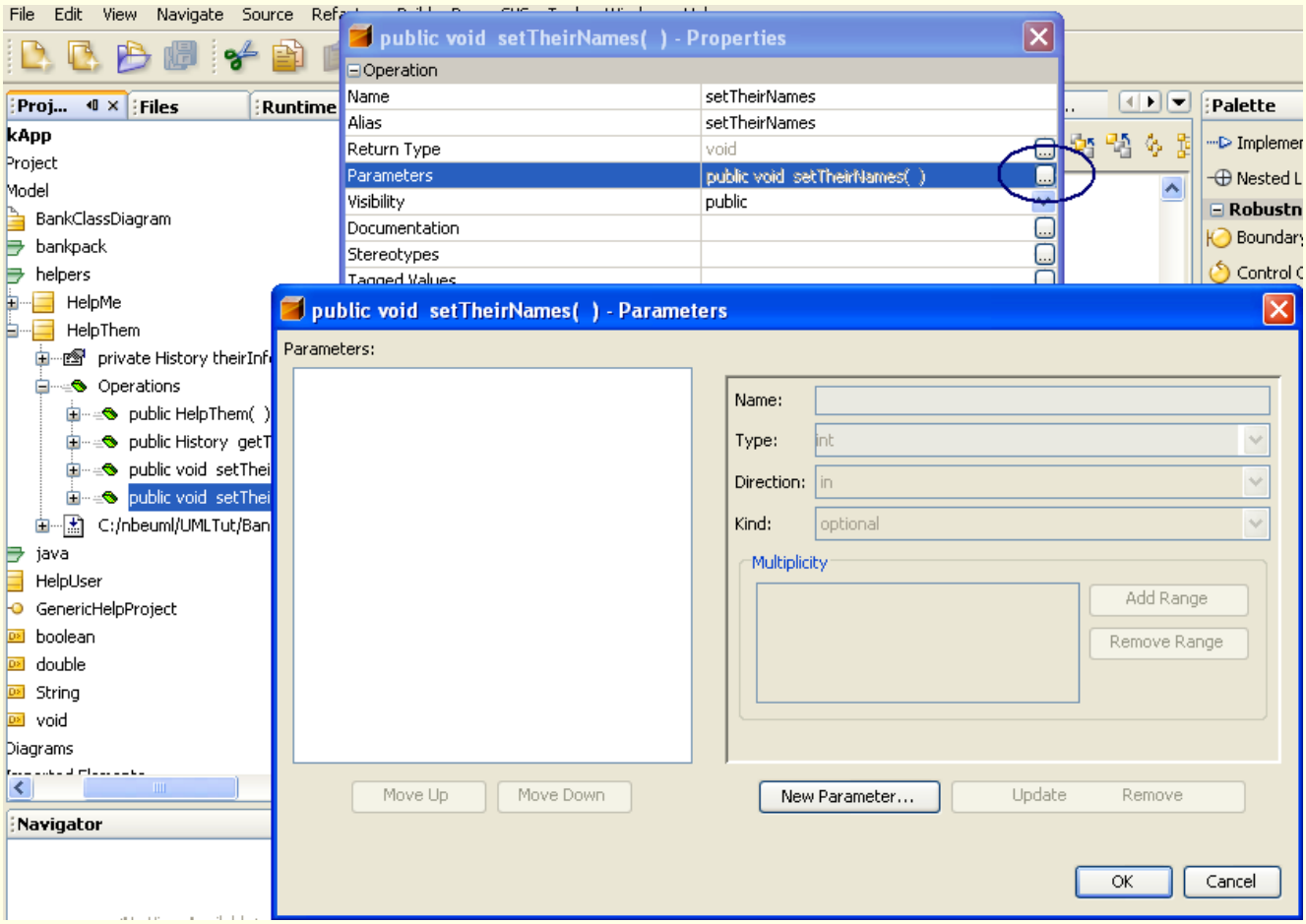
O noua operatie **public void Unnamed()** este creata.

12. Deschideti meniul contextual pentru nodul operatiei create. Invocati comanda ***Rename***. Introduceti noul nume: **setTheirNames** si apasati butonul ***OK.***

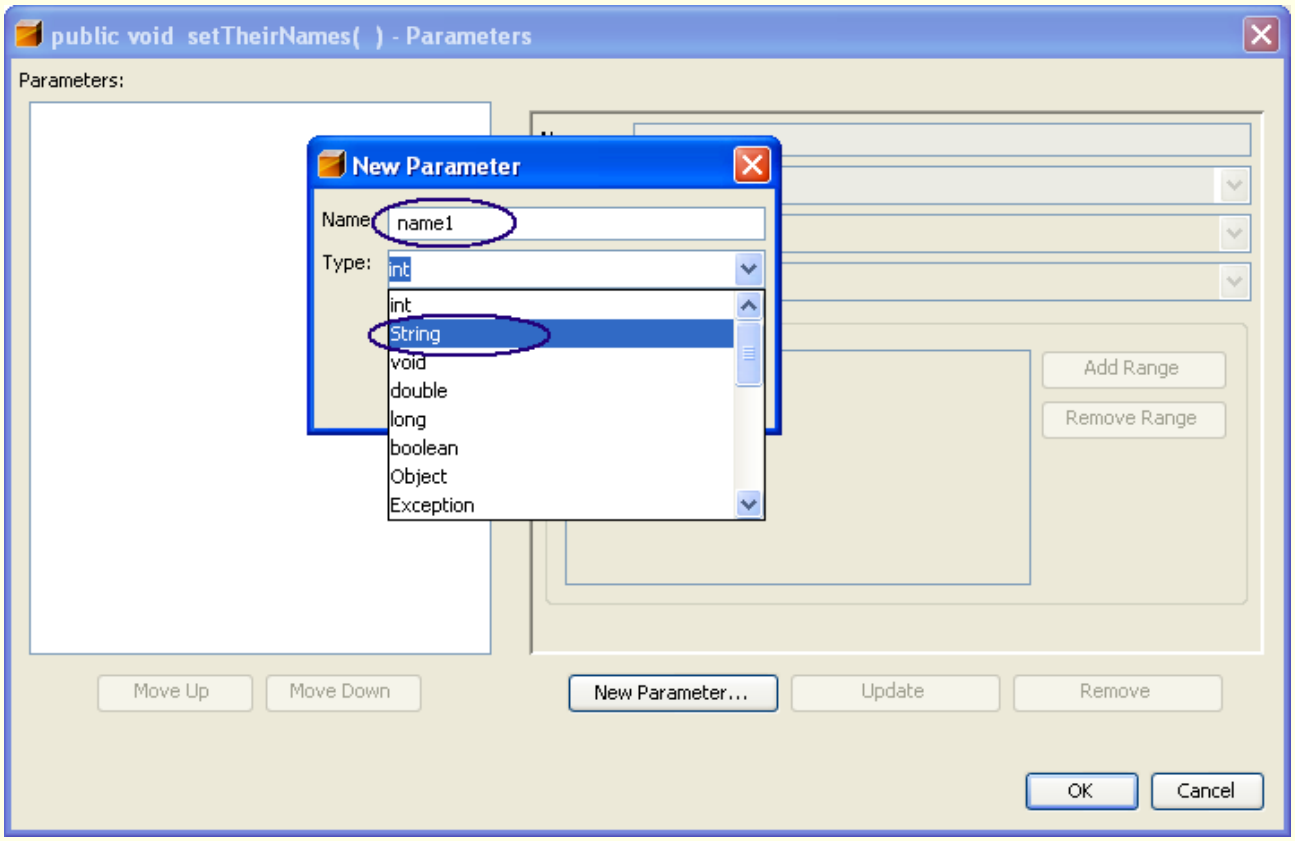
Operatia este redenumita.



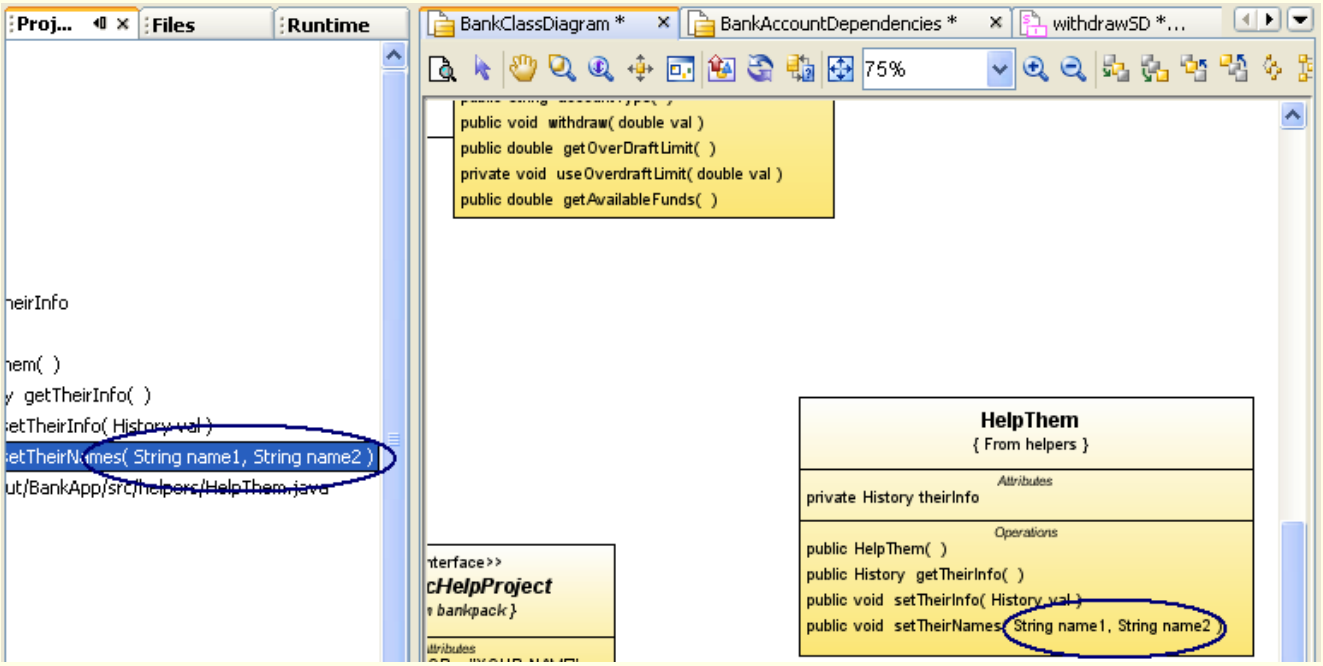
13. Deschideti ***Properties*** pentru nodul functiei **setTheirNames**. Faceti click pe butonul '...' pentru a particulariza valoarea ***Parameters***.



14. In fereastra de dialog care se deschide, **public void setTheirNames( ) - Parameters** apasati butonul ***New Parameter...*** Scrieti numele **name1**. Schimbati tipul in **String**.



16. Adaugati un alt parametru de acelasi tip cu numele **name2.** Apasati butonul ***OK*** si inchideti fereastra ***Properties***. Modificarile apar in clasa.



3.2.6. Rezumat

In acest tutorial ati invatat sa efectuati urmatoarele:

Sa **adaugati noi elemente in model** si sa **intelegeti relatia model UML – sursa Java**.

Sa **intelegereti utilizarea si impactul operatiei de reconstructie** (*refactoring*) in timpul modificarii elementelor diagramelor.

Atentie: **La finalul laboratoarelor stergeti proiectele create** (click dreapta pe nodul proiectului in fereastra ***Projects***, selectati ***Delete*** si **confirmati ca doriti sa fie sterse sursele – in cazul Java**)

3.3. Studiu de caz complex de inginerie a codului Java bazata pe utilizarea UML

In continuare vor fi **rezolvate o parte dintre problemele din subiectele de examen** din documentul ISw_0809_Subiecte_Examen_Subiecte_2A_2B.pdf (studiu de caz si la lucrarile a 2-a si a 3-a).

3.3.1. Incarcarea proiectelor (realizate in lucrarea a 2-a)

1. Verificati daca exista directorul numit **\MonitoringCaseStudy2** pe drive-ul **D:** in directorul **\isw**, in **subdirectorul cu numarul grupei**, (exemplu: **D:\isw\441E\MonitoringCaseStudy2**). Daca exista, stergeti-l.

2. Descarcati arhiva **MonitoringCaseStudy2.zip**, **in subdirectorul cu numarul grupei**, (de exemplu: **D:\isw\441E**).

3. Dezarhivati fisierul descarcat, folosind WinZip sau WinRar, cu click dreapta pe numele fisierului **MonitoringCaseStudy2.zip** selectand apoi optiunea ***Extract Here***.

Automat va fi creat subdirectorul **\MonitoringCaseStudy2** iar in el subdirectoarele **\UMLMonitoringApp** (in care se afla proiectul UML) si **\JavaMonitoringApp** (in care se afla proiectul Java).

4. Din meniul principal selectaţi ***File> Open Project.*** Apare asistentul ***Open Project***

Sub ***Look in***, selectati calea proiectului UML anterior dezarhivat (de forma: **D:\isw\441E\MonitoringCaseStudy2\UMLMonitoringApp**).

Lasati selectat ***Open as Main Project***

Faceti ***Click*** pe ***Open Project***.

In ***Projects*** apare nodul **UMLMonitoringApp**.

5. Din meniul principal selectaţi ***File> Open Project.*** Apare asistentul ***Open Project***

Sub ***Look in***, selectati calea proiectului Java anterior dezarhivat (de forma: **D:\isw\441E\ MonitoringCaseStudy2\JavaMonitoringApp**).

Deselectati ***Open as Main Project***

Faceti ***Click*** pe ***Open Project***.

In ***Projects*** apare nodul **JavaMonitoringApp**.

3.3.2. Explorarea proiectelor

1. In zona ***Projects***, sub nodul **UMLMonitoringApp**, sub nodul ***Model*** gasiti diagrama de clase **MainClassDiagram** si o deschideti in editorul de diagrame.

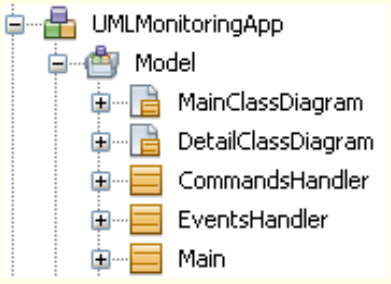


Diagrama **MainClassDiagram** ar trebui sa arate astfel.

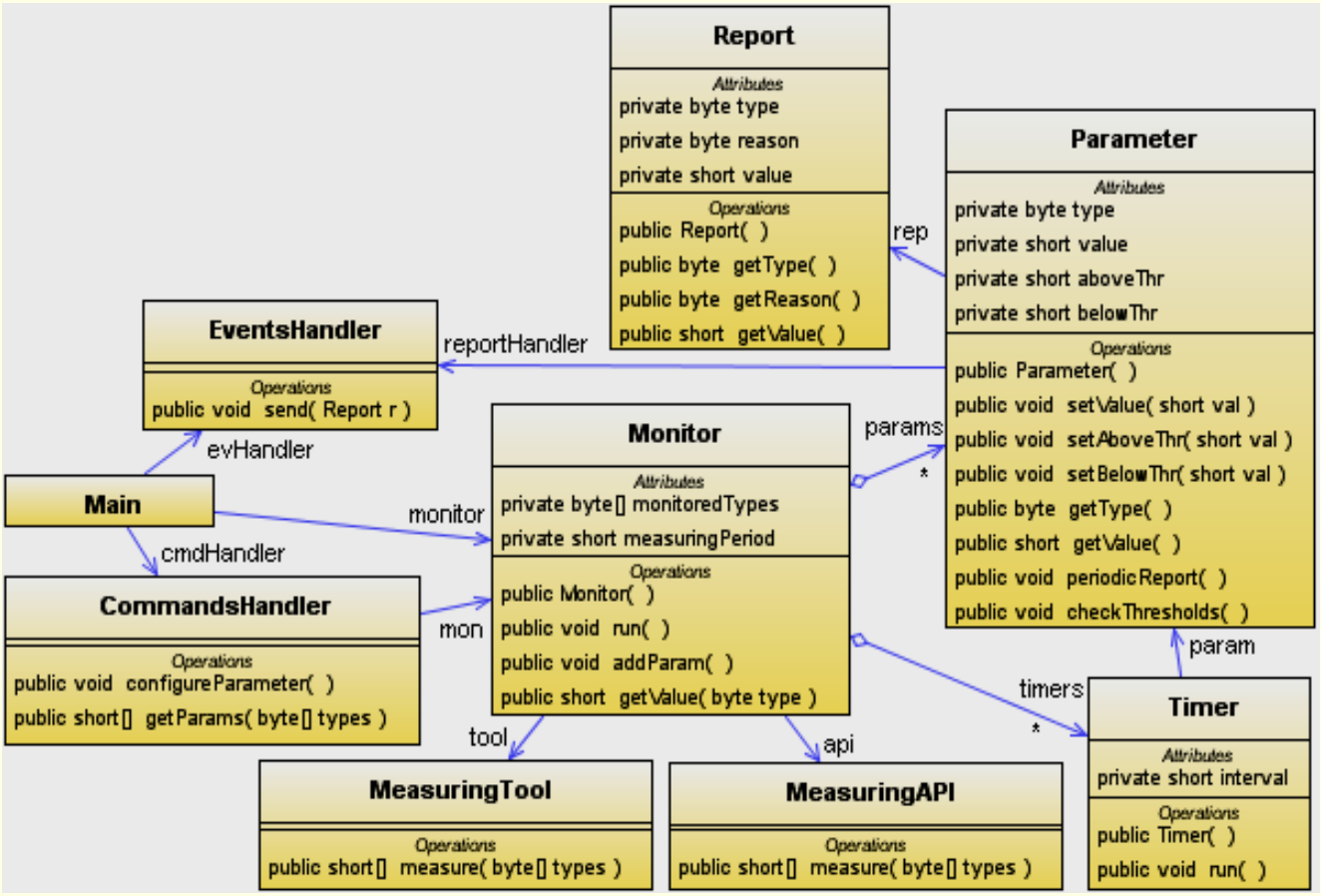
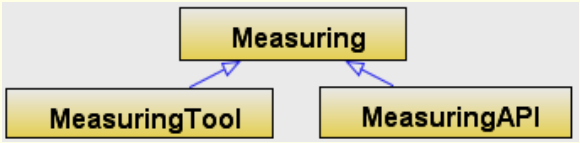
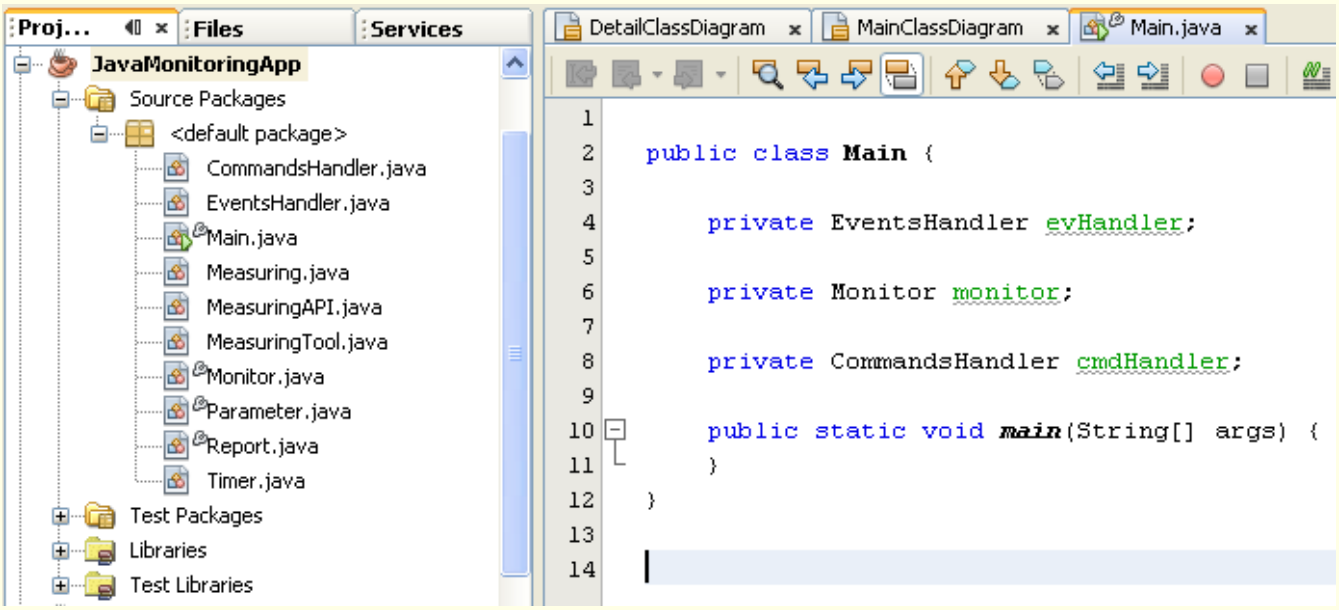


Diagrama **DetailClassDiagram** (in care se introduce clasa **Measuring** ca o generalizare a claselor **MeasuringTool** si **MeasuringAPI**, clase **reconstruite astfel incat sa extinda** clasa **Measuring**) ar trebui sa arate astfel.



2. In zona ***Projects***, sub nodul **JavaMonitoringApp**, sub nodul ***Source Packages***, sub nodul ***<default package>***, gasiti codurile claselor (generate din diagrama de clase anterioara) si le puteti deschide in editorul de diagrame.



In continuare vor fi **prezentate codurile sursa Java generate din diagrama de clase de mai sus.**

```
1  public class Main {
2      private EventsHandler evHandler;
3      private Monitor monitor;
4      private CommandsHandler cmdHandler;
5  }
```

```
1  public class CommandsHandler {
2      private Monitor mon;
3      public void configureParameter () {
4      }
5      public short[] getParams (byte[] types) {
6          return null;
7      }
8  }
```

```
1  public class EventsHandler {
2      public void send (Report r) {
3      }
4  }
```

```
1  public class Monitor {
2      private byte[] monitoredTypes;
3      private short measuringPeriod;
4      private Timer timers;
5      private Parameter params;
6      private MeasuringTool tool;
7      private MeasuringAPI api;
8      private boolean toolOrAPI;
9      public Monitor () {
10     }
11     public void run () {
12     }
13     public void addParam () {
14     }
15     public short getValue (byte type) {
16         return 0;
17     }
18 }
```

```
1  public class Parameter {
2      private short value;
3      private short aboveThr;
4      private short belowThr;
5      private byte type;
6      private Report rep;
7      private EventsHandler reportHandler;
8      public Parameter () {
9      }
10     public void setAboveThr (short val) {
11     }
12     public void setBelowThr (short val) {
13     }
14     public void setValue (short val) {
15     }
16     public void periodicReport () {
17     }
18     public void checkThresholds () {
19     }
20     public byte getType () {
21         return 0;
22     }
23     public short getValue () {
24         return 0;
25     }
26 }
```

```
1  public class MeasuringAPI extends Measuring {
2      public short[] measure (byte[] types) {
3          return null;
4      }
```

```
}
```

```
public class MeasuringTool extends Measuring {
    public short[] measure (byte[] types) {
        return null;
    }
}
```

```
public class Measuring {
    public Measuring () {
    }
}
```

```
public class Report {
    private byte type;
    private byte reason;
    private short value;
    public Report () {
    }
    public byte getReason () {
        return 0;
    }
    public byte getType () {
        return 0;
    }
    public short getValue () {
        return 0;
    }
}
```

```
public class Timer {
    private short interval;
    private Parameter param;
    public Timer () {
    }
    public void run () {
    }
}
```

3.3.3. Rezolvarea enunturilor care privesc scrierea codurilor Java

Enunt: Clasa **Main** creeaza obiectele din clasa **Monitor** (care extinde **Thread**), **EventsHandler** si **CommandsHandler** si face legaturile dintre ele.

Posibila solutie (acest cod nu facea obiectul subiectelor de examen):

```
public class Main {
    private static short           aboveThr;
    private static short           belowThr;
    private static CommandsHandler cmdHandler;
    private static EventsHandler   evHandler;
    private static short           interval;
    private static Monitor         monitor;
    private static byte            type;

    public static void main(String[] args) {
        short   measuringPeriod = 1000;
        boolean toolOrAPI       = true;

        evHandler  = new EventsHandler();
        monitor    = new Monitor(measuringPeriod, toolOrAPI, evHandler);
        cmdHandler = new CommandsHandler(monitor);

        // Alte coduri, pentru testare sistem
    }
}
```

Enunt: Constructorul **Monitor()** primeste
- **perioada efectuarii masuratorilor**,

- un **flag care indica metoda de masurare** (**Tool** sau **API**) si
- o **referinta catre un obiect** de tip **EventsHandler** (care va fi transmisa obiectelor de tip **Parameter** si **Report** in momentul crearii acestora), si trebuie (*pe langa initializarile evident necesare*):
- **sa creeze spatiul necesar tratarii a 50 de parametri**, si
- **sa lanseze firul de executie** al obiectului **Monitor**.

1. (A) Sa se scrie codul constructorului **Monitor()**. **(10 pct.)**

Posibila solutie:

```
 1  public class Monitor extends Thread {
 2      private MeasuringAPI api;
 3      private EventsHandler evHandler;
 4      private short measuringPeriod;
 5      private byte[] monitoredTypes;
 6      private byte nrParam;
 7      private Parameter[] params;
 8      private Timer timers;
 9      private MeasuringTool tool;
10      private boolean toolOrAPI;
11
12      // 1.A.
13      public Monitor(short period, boolean toolOrAPI, EventsHandler ev) {
14          this.measuringPeriod = period;       // initializare evident necesara
15          this.toolOrAPI = toolOrAPI;          // initializare evident necesara
16
17          if (toolOrAPI) {                     // initializare evident necesara
18              this.tool = new MeasuringTool();
19          } else {
20              this.api = new MeasuringAPI();
21          }
22          this.evHandler = ev;                 // initializare evident necesara
23          monitoredTypes = new byte[50];  // spatiul necesar stocarii tipurilor
24          params = new Parameter[50];     // spatiul necesar stocarii parametrilor
25          nrParam = 0;                    // variabila contor parametri current stocati
26          this.start();
27      }
28  }
```

Enunt: Apelul **configureParameter()** (*metoda a clasei* **CommandsHandler**) primeste
- **tipul** parametrului a carui monitorizare este dorita,
- **valoarea intervalului de raportare periodica** in milisec. (valoarea 0 indica lipsa raportarii periodice), si
- **valori ale pragurilor de alertare** (superior si / sau inferior, valoarea 0 indica lipsa pragului).

Posibila solutie (acest cod nu facea obiectul subiectelor de examen):

```
 1  public class CommandsHandler {
 2      private Monitor mon;
 3      public CommandsHandler(Monitor mon) {
 4          this.mon = mon;
 5      }
 6
 7      public void configureParameter(byte type, short interval, short aboveThr,
 8                          short belowThr) {
 9          mon.addParam(type, interval, aboveThr, belowThr);
10      }
11  }
```

Enunt: Efectul este apelul **addParam()** prin care obiectul **Monitor**
- creeaza si stocheaza un nou parametru (transmitandu-i si referinta de tip **EventsHandler**), si
- ii stocheaza separat tipul.

Daca primeste valori nenule pentru **interval**, **addParam()**
- creeaza un obiect de tip **Timer** (care extinde **Thread**)
- pe care il initializeaza cu intervalul primit si cu referinta obiectului de tip **Parameter**, si
- lanseaza firul de executie al obiectului de tip **Timer**.

Constructorul **Parameter()** initializeaza noul obiect cu valorile primite.

1. (B) Sa se scrie codul metodei **addParam()**. **(10 pct.)**

Posibila solutie:

```
1     // 1.B.
2     public void addParam(byte type, short interval,short aboveThr,short belowThr){
3         // creare si stocare parametru
4         params[nrParam] = new Parameter(type, aboveThr, belowThr, evHandler);
5
6         // stocare tip parametru (separat)
7         monitoredTypes[nrParam] = type;
8
9         // creare si lansare Timer (conditionata)
10        if (interval > 0) {
11            Timer t = new Timer(interval, params[nrParam]);
12            t.start();
13        }
14        nrParam++;
15    }
```

Enunt: Metoda **run()** a obiectelor de tip **Monitor**

- foloseste apelul **Thread.sleep(period)** pentru a-si suspenda periodic executia.

La reluarea executiei

- se apeleaza metoda **measure()** (a obiectului de tip **MeasuringTool** sau **MeasuringAPI**)
  - furnizandu-i un tablou cu tipurile curent monitorizate, iar
- cu valorile obtinute se actualizeaza valorile curente ale fiecarui parametru, si apoi
- se genereaza catre obiectele **Parameter** cereri **checkThresholds()**.

2. (A) Sa se scrie codul metodei **run()** a obiectelor **Monitor**. **(10 pct.)**

Posibila solutie:

```
1     // 2.A.
2     public void run() {
3
4         while (true) {
5             if (nrParam > 0) {
6                 // referinta catre tabloul valorilor masurate
7                 short[] values;
8
9                 // crearea unui tablou cu tipurile monitorizate
10                byte[] types = new byte[nrParam];
11                for (int i = 0; i < nrParam; i++) {
12                    types[i] = monitoredTypes[i];
13                }
14
15                // apelul metodei measure()
16                if (toolOrAPI) {
17                    values = tool.measure(types);
18                } else {
19                    values = api.measure(types);
20                }
21
22                // pentru toate obiectele Parameter curent existente
23                for (int i = 0; i < nrParam; i++) {
24                    // actualizarea valorilor curente
25                    params[i].setValue(values[i]);
26                    // cereri checkThresholds() catre obiectele Parameter
27                    params[i].checkThresholds();
28                }
29
30            }
31            // suspendarea periodica a executiei
32            try {
33                Thread.sleep(measuringPeriod);
34            } catch (InterruptedException ex) {
35            }
36        }
37    }
```

Enunt: Metoda **checkThresholds()** (*metoda a clasei* **Parameter**)
- compara valoarea curenta cu pragurile (daca acestea au valori nenule).

Daca unul dintre praguri este depasit,
- este creat un obiect **Report** initializat cu
  - **motivul generarii lui** (depasirea pragului superior/ inferior codificata prin 1/ 2),
  - **tipul parametrului** si **valoarea curenta**,
- si este trimis catre **EventHandler** prin apelul **send()**.

3. (A) Sa se scrie codul metodei **checkThresholds()**. **(10 pct.)**

Posibila solutie:

```
1     // 3.A.
2     public void checkThresholds() {
3         byte reason;
4
5         // comparatia valorii curente cu pragurile (daca acestea sunt nenule)
6         if ((aboveThr != 0) && (value > aboveThr)) {
7             reason = 1;
8             rep = new Report(reason, type, value);
9             reportHandler.send(rep);
10        }
11
12        if ((belowThr != 0) && (value < belowThr)) {
13            reason = 2;
14            rep = new Report(reason, type, value);
15            reportHandler.send(rep);
16        }
17    }
```

Enunt: Apelul **getParams()** (*metoda a clasei* **CommandsHandler**)
- primeste **un tablou cu tipurile parametrilor a caror valori sunt dorite** si
- returneaza **un tablou cu valorile parametrilor respectivi**,
- facand apel la metoda **getValue()** a obiectului **Monitor**.

Enunt: Apelul **getValue()** (*metoda a clasei* **Monitor**)
- primeste **tipul unui parametru a carui valoare este dorita**,
- cauta si returneaza **valoarea parametrului respectiv**.

2. (B) Sa se scrie codurile metodelor **getParams()** si **getValue()**. **(10 pct.)**

Posibila solutie:

```
1     // 2.B.a.
2     public short[] getParams(byte[] types) {
3         // crearea tabloului valorilor
4         short[] values = new short[types.length];
5
6         // obtinerea valorilor si popularea tabloului
7         for (int i = 0; i < types.length; i++) {
8             values[i] = mon.getValue(types[i]);
9         }
10        return values;
11    }
```

```
1     // 2.B.b.
2     public short getValue(byte type) {
3         short value = 0;
4
5         // pentru toti parametrii existenti
6         for (int i = 0; i < nrParam; i++) {
7             // identificarea parametrului care are tipul primit ca argument
8             if (type == monitoredTypes[i]) {
9                 // obtinerea valorii parametrului
10                value = params[i].getValue();
11            }
12        }
13        return value;
14    }
```

Enunt: Constructorul **Timer()**
- primeste pe langa
  - intervalul de raportare si
  - o referinta catre obiectul de tip **Parameter**
- pe care o stocheaza.

Metoda **run()** a obiectelor de tip **Timer**
- foloseste apelul **Thread.sleep(interval)** pentru a-si suspenda periodic executia si
- a genera catre obiectul clasei **Parameter** o cerere **periodicReport()**.

Enunt: Metoda **periodicReport()** (*metoda a clasei* **Parameter**)
- creeaza un obiect **Report** pe care
- il initializeaza cu
  - **motivul generarii lui** (codificat prin valoarea 0),
  - **tipul parametrului** si
  - **valoarea curenta**, si
- il trimite catre **EventHandler** prin apelul **send()**.

3. (B) Sa se scrie codul metodei **run()** a clasei **Timer** si codul metodei **periodicReport()**. **(10 pct.)**

Posibila solutie:

```
 1  public class Timer {
 2      private short interval;
 3      private Parameter param;
 4
 5      public Timer(short interval, Parameter param) {
 6          this.interval = interval;
 7          this.param    = param;
 8      }
 9
10      // 3.B.a.
11      public void run() {
12          try {
13              Thread.sleep(interval);
14          } catch (InterruptedException ex) {}
15
16          param.periodicReport();
17      }
18  }
```

```
 1  public class Parameter {
 2      private short value;
 3      private short aboveThr;
 4      private short belowThr;
 5      private byte type;
 6      private Report rep;
 7      private EventsHandler reportHandler;
 8
 9      public Parameter(byte type, short aboveThr, short belowThr, EventsHandler ev) {
10          this.type = type;
11          this.aboveThr = aboveThr;
12          this.belowThr = belowThr;
13          reportHandler = ev;
14      }
15
16      // 3.B.b.
17      public void periodicReport() {
18          byte reason = 0;
19          rep = new Report(reason, type, value);
20          reportHandler.send(rep);
21      }
```

3.3.4. Un proiect Java care contine rezolvarile si cod pentru testare

1. Verificati daca exista directorul numit **\MonitoringCaseStudyComplete** pe drive-ul **D:** in directorul **\isw**, in **subdirectorul cu numarul grupei**, (exemplu: **D:\isw\441E\MonitoringCaseStudyComplete**). Daca exista, stergeti-l.

2. Descarcati arhiva **MonitoringCaseStudyComplete.zip**, **in subdirectorul cu numarul grupei**, (de exemplu: **D:\isw\441E**\).

3. Dezarhivati fisierul descarcat, folosind WinZip sau WinRar, cu click dreapta pe numele fisierului **MonitoringCaseStudyComplete.zip** selectand apoi optiunea ***Extract Here***.

Automat va fi creat subdirectorul **\MonitoringCaseStudyComplete** iar in el subdirectorul **\JavaMonitoringApplication** (in care se afla proiectul Java).

4. Din meniul principal selectaţi ***File> Open Project.*** Apare asistentul ***Open Project***

Sub ***Look in***, selectati calea proiectului Java anterior dezarhivat (de forma: **D:\isw\441E\MonitoringCaseStudyComplete\JavaMonitoringApplication**).

Lasati selectat ***Open as Main Project.*** Faceti ***Click*** pe ***Open Project***.

In ***Projects*** apare nodul **JavaMonitoringApplication**.

5. In fereastra ***Projects***, cu right-click pe nodul **JavaMonitoringApplication** selectati din meniul pop-up comanda ***Build***.

6. Cu right-click pe nodul **JavaMonitoringApplication** selectati ***Run*** din meniul pop-up.

Clasa principala (**Main**) a proiectului Java contine **un cod suplimentar** adaugat **pentru testare.** Au fost astfel **creati si configurati parametri de 7 categorii (tipuri):**

- tipul 1 – caruia ii este monitorizat **doar pragul superior** (stabilit la valoarea **200**)

- tipul 2 – caruia ii este monitorizat **doar pragul inferior** (stabilit la valoarea **100**)

- tipul 3 – care este monitorizat **doar prin raportare periodica** (cu perioada **5 sec**)

- tipul 4 – caruia ii este monitorizat **atat pragul superior** (stabilit la valoarea **200**)
cat si pragul inferior (stabilit la valoarea **100**)

- tipul 5 – care este monitorizat **atat d.p.d.v. al pragului superior** (stabilit la valoarea **200**)
cat si prin raportare periodica (cu perioada **6 sec**)

- tipul 6 – care este monitorizat **atat d.p.d.v. al pragului inferior** (stabilit la valoarea **100**)
cat si prin raportare periodica (cu perioada **7 sec**)

- tipul 7 – care este monitorizat **atat d.p.d.v. al pragului superior** (stabilit la valoarea **200**)
cat si d.p.d.v. al pragului inferior (stabilit la valoarea **100**)
si prin raportare periodica (cu perioada **11 sec**)

```
 1  public class Main {
 2      private static short           aboveThr;
 3      private static short           belowThr;
 4      private static CommandsHandler cmdHandler;
 5      private static EventsHandler   evHandler;
 6      private static short           interval;
 7      private static Monitor         monitor;
 8      private static byte            type;
 9
10      public static void main(String[] args) {
11          short   measuringPeriod = 1000;
12          boolean toolOrAPI       = true;
13          evHandler  = new EventsHandler();
14          monitor    = new Monitor(measuringPeriod, toolOrAPI, evHandler);
15          cmdHandler = new CommandsHandler(monitor);
16
```

```
17          // Parametru tip 1 - mon. prag sup.
18          type     = 1;
19          interval = 0;
20          aboveThr = 200;
21          belowThr = 0;
22          cmdHandler.configureParameter(type, interval, aboveThr, belowThr);
23
24          // Parametru tip 2 - mon. prag inf.
25          type     = 2;
26          interval = 0;
27          aboveThr = 0;
28          belowThr = 100;
29          cmdHandler.configureParameter(type, interval, aboveThr, belowThr);
30
31          // Parametru tip 3 - mon. periodica
32          type     = 3;
33          interval = 5000;
34          aboveThr = 0;
35          belowThr = 0;
36          cmdHandler.configureParameter(type, interval, aboveThr, belowThr);
37
38          // Parametru tip 4 - mon. prag sup.+inf.
39          type     = 4;
40          interval = 0;
41          aboveThr = 200;
42          belowThr = 100;
43          cmdHandler.configureParameter(type, interval, aboveThr, belowThr);
44
45          // Parametru tip 5 - mon. prag sup.+per.
46          type     = 5;
47          interval = 6000;
48          aboveThr = 200;
49          belowThr = 0;
50          cmdHandler.configureParameter(type, interval, aboveThr, belowThr);
51
52          // Parametru tip 6 - mon. prag inf.+per.
53          type     = 6;
54          interval = 7000;
55          aboveThr = 0;
56          belowThr = 100;
57          cmdHandler.configureParameter(type, interval, aboveThr, belowThr);
58
59          // Parametru tip 7 - mon. prag inf.+sup.+per.
60          type     = 7;
61          interval = 11000;
62          aboveThr = 200;
64          belowThr = 100;
65          cmdHandler.configureParameter(type, interval, aboveThr, belowThr);
66      }
67  }
```

Metoda send() a clasei EventsHandler contine un cod adaugat pentru testare.

```
1  public class EventsHandler {
2      public void send(Report r) {
3          System.out.println("EventsHandler.send() cu reason " + r.getReason() +
4                  " cu type " + r.getType() + " cu value " + r.getValue());
5      }
6  }
```

In metoda run() a clasei Monitor a fost adaugat un cod pentru testare.

```
1  import java.util.Calendar;
2
3     public void run() {
4        Calendar now;
5        while (true) {
6           // alte coduri ...
7           now = Calendar.getInstance();
8           System.out.println("Momentul curent:" + now.get(Calendar.HOUR_OF_DAY) +":"
9             + now.get(Calendar.MINUTE) + ":" + now.get(Calendar.SECOND) + ":");
10          }
11    }
```

Efectul unei executii de 18 sec. a programului poate fi urmarit mai jos.

```
compile:
run:
Parameter cu type 1 cu aboveThr 200 cu belowThr 0
Monitor.addParam()

Parameter cu type 2 cu aboveThr 0 cu belowThr 100
Monitor.addParam()

Parameter cu type 3 cu aboveThr 0 cu belowThr 0
Monitor.addParam()
Timer.run() cu interval 7000

Parameter cu type 4 cu aboveThr 200 cu belowThr 100
Monitor.addParam()

Parameter cu type 5 cu aboveThr 200 cu belowThr 0
Monitor.addParam()
Timer.run() cu interval 5000

Parameter cu type 6 cu aboveThr 0 cu belowThr 100
Monitor.addParam()
Timer.run() cu interval 6000

Parameter cu type 7 cu aboveThr 200 cu belowThr 100
Monitor.addParam()
Timer.run() cu interval 11000
```

Initializari parametri
- tip 1
- tip 2
- tip 3
- tip 4
- tip 5
- tip 6
- tip 7

```
Momentul curent: 16:26:11:
EventsHandler.send() cu reason 1 cu type 4 cu value 208
EventsHandler.send() cu reason 1 cu type 5 cu value 201

Momentul curent: 16:26:12:
EventsHandler.send() cu reason 2 cu type 2 cu value 92

Momentul curent: 16:26:13:
EventsHandler.send() cu reason 2 cu type 6 cu value 87

Momentul curent: 16:26:14:
EventsHandler.send() cu reason 0 cu type 3 cu value 118

Momentul curent: 16:26:15:
```

Raportare parametri
- **depasire prag sup.**
- depasire prag sup.
- **depasire prag inf.**
- depasire prag inf.
- **raportare periodica**

```
EventsHandler.send() cu reason 2 cu type 4 cu value 94
EventsHandler.send() cu reason 1 cu type 5 cu value 210
EventsHandler.send() cu reason 0 cu type 5 cu value 210

Momentul curent: 16:26:16:
EventsHandler.send() cu reason 1 cu type 7 cu value 203
EventsHandler.send() cu reason 0 cu type 6 cu value 185

Momentul curent: 16:26:17:
```

- depasire prag inf.
- **depasire prag sup.**
- **raportare periodica (aceeasi valoare!!!)**
- **nimic de raportat**

```
Momentul curent: 16:26:18:
Momentul curent: 16:26:19:
EventsHandler.send() cu reason 1 cu type 4 cu value 201
EventsHandler.send() cu reason 2 cu type 6 cu value 88

Momentul curent: 16:26:20:
EventsHandler.send() cu reason 0 cu type 7 cu value 146

Momentul curent: 16:26:21:
EventsHandler.send() cu reason 1 cu type 7 cu value 203
Momentul curent: 16:26:22:
Momentul curent: 16:26:23:
EventsHandler.send() cu reason 1 cu type 5 cu value 219
Momentul curent: 16:26:24:
EventsHandler.send() cu reason 1 cu type 5 cu value 205
Momentul curent: 16:26:25:
Momentul curent: 16:26:26:
EventsHandler.send() cu reason 2 cu type 6 cu value 84
Momentul curent: 16:26:27:
BUILD STOPPED (total time: 18 seconds)
```

- nimic de raportat
...

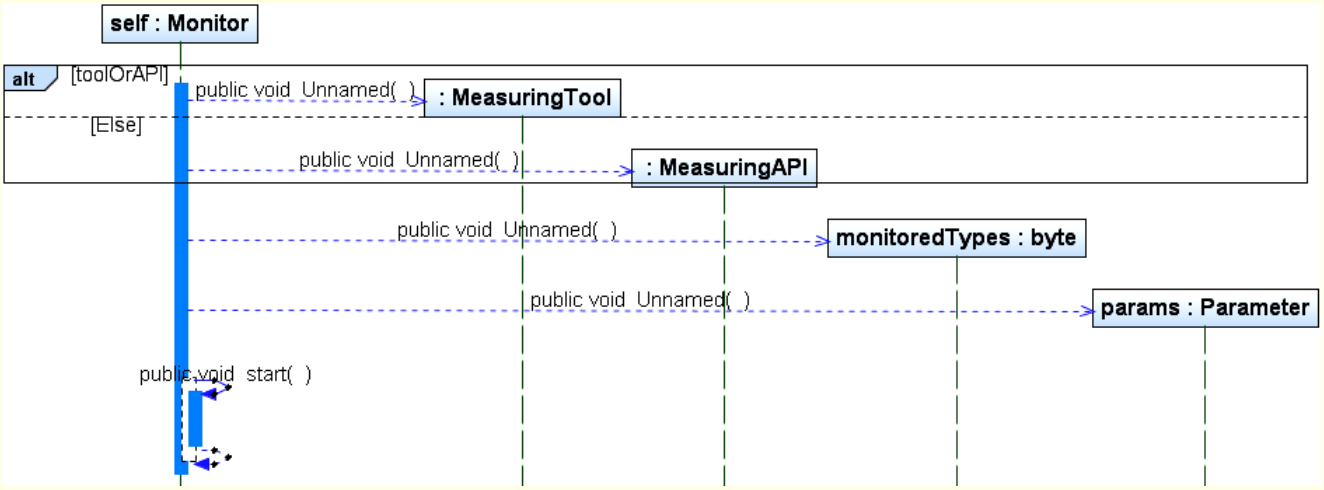
Si in metodele measure() a fost adaugat **un cod pentru testare** (care genereaza aleator valorile „masurate” in gama **80..220**, astfel incat acestea sa poata iesi intr-o mica proportie din gama **100..200).**

```
1    public short[] measure(byte[] types) {
2
3        short[] values = new short[types.length];
4
5        for (int i = 0; i < types.length; i++) {
6            values[i] = (short) (80 + 140 * Math.random());
7        }
8        return values;
9    }
```

3.3.5. Rezolvarea enunturilor care privesc crearea diagramelor UML

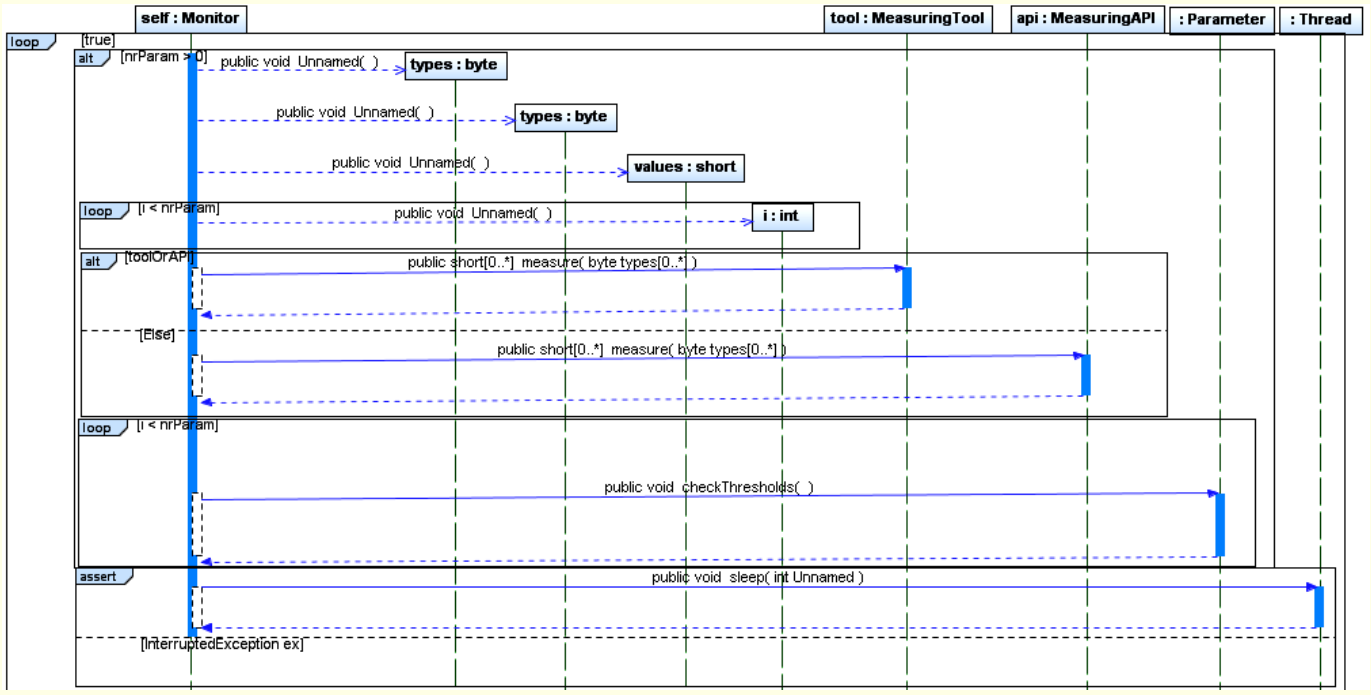
Enunt: **4.(A)** Sa se descrie **sub forma unei diagrame MSC** (continand crearea obiectelor, mesajele schimbate, parametrii acestora) **scenariul** in care este **creat obiectul de tip Monitor**. **(5 pct.)**

Posibila solutie: (cu observatia ca in loc de **Unnamed()** ar trebui sa fie apelurile constructorilor)



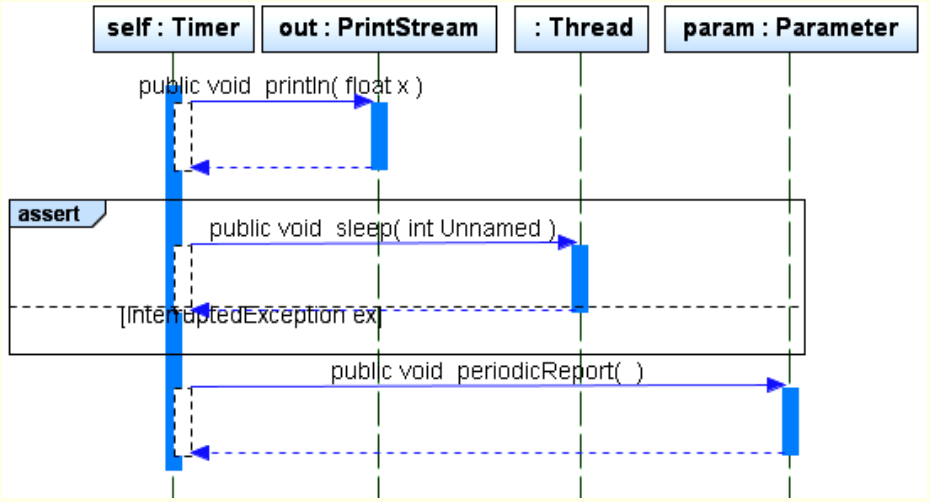
Enunt: **5.(A)** Sa se descrie **sub forma unei diagrame MSC scenariul** in care sunt **masurati parametrii, verificate pragurile si generat un raport avand ca motivatie depasirea valorii unui prag**. **(5 pct.)**

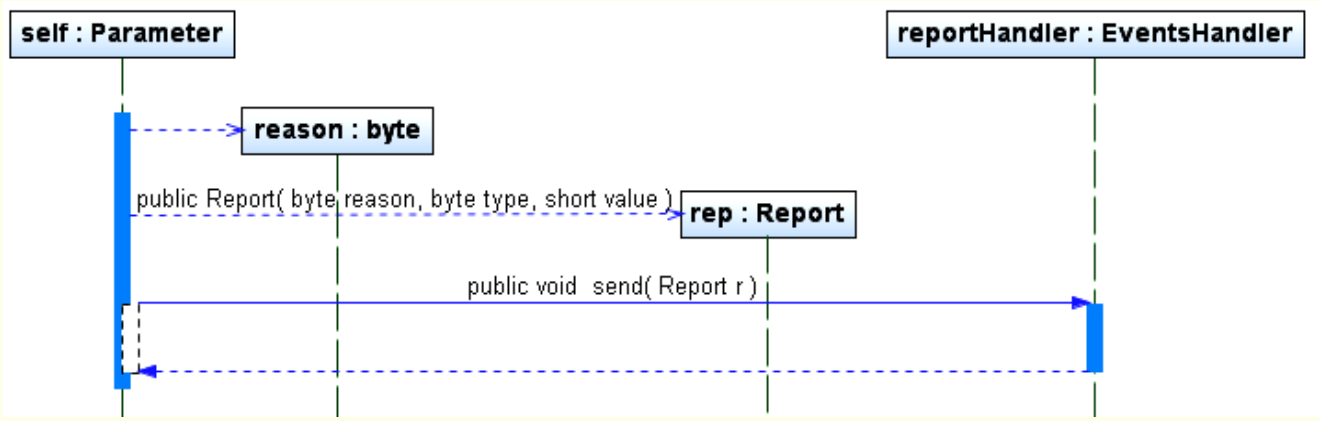
Posibila solutie:



Enunt: **4.(B)** Sa se descrie **sub forma unei diagrame MSC** (continand crearea obiectelor, mesajele schimbate, parametrii acestora) **scenariul** in care **un parametru este monitorizat prin raportare periodica**. **(5 pct.)**

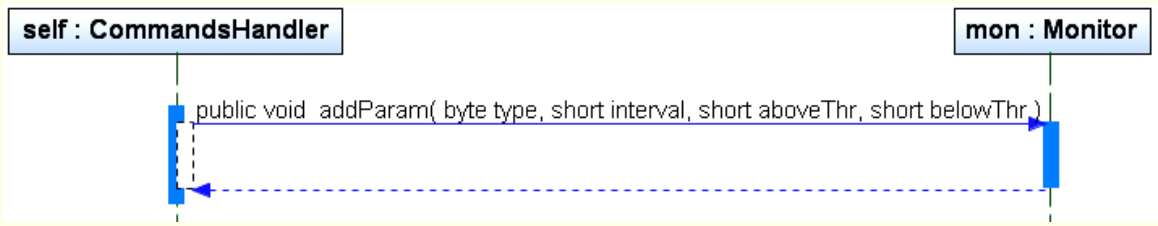
Posibila solutie:

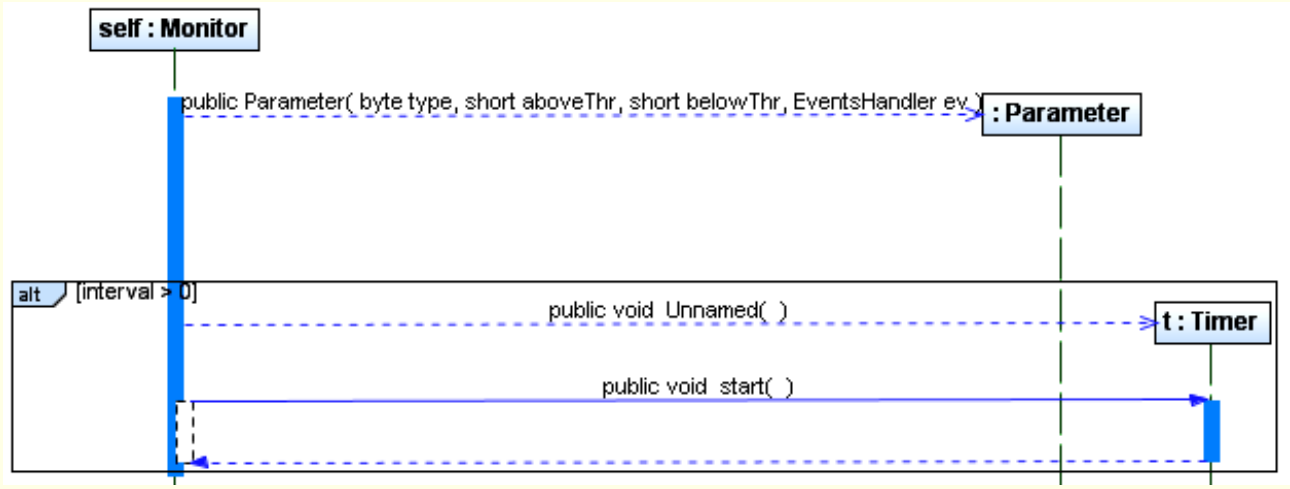




Enunt: **5.(B)** Sa se descrie **sub forma unei diagrame MSC scenariul** in care **un nou parametru este creat si configurat**. **(5 pct.)**

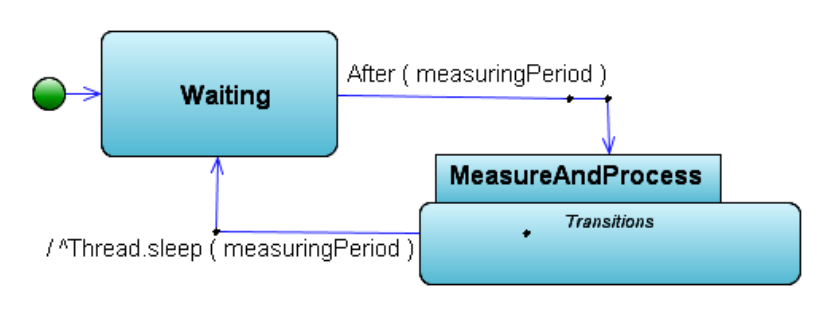
Posibila solutie:





Enunt: **6.(A)** Sa se descrie **sub forma unei diagrame FSM** (masina de stari, **continand stari, tranzitii, evenimente, actiuni, activitati**) **evolutia starilor obiectului** de tip **Monitor** (care trece periodic din starea **asteptare** trecere perioada in starea **masurare si verificare**, ultima formata din substarile **masurare** parametri, **actualizare** valori si **verificare** praguri). **(5 pct.)**

Posibila solutie:
- la nivel inalt - superstarile **asteptare (Waiting)** si **masurare si verificare (MeasureAndProcess)**



- automatul superstarii **MeasureAndProcess** - substarile **masurare** parametri (**Measure**), **actualizare** valori (**Update**) si **verificare** praguri (**Check**)