Projet sur les Graphes : Banking

Frat.java

```
import java.util.Vector;

public class Frat {

     private String _name;
     private int _budget;
     private Vector<Debt> _debtList;

     ////////// GETTERS + SETTERS //////////
     public String getName() { return _name; }
     public void setName(String name) { _name = name; }
     public boolean isFrat(String fratName) { return (fratName.equals(getName())); }
     public boolean isFrat(Frat otherFrat) { return (this == otherFrat); }

     public int getBudget() { return _budget; }
     public void setBudget(int budget) { _budget = budget; }

     public Vector<Debt> getDebtList() { return _debtList; }
     private Debt getDebtAtIndex(int i){
          return getDebtList().get(i);
     }

     public int getDebt(Frat creditor){
          try {
               int i = 0;
               while (getDebtAtIndex(i).getCreditor() != creditor){
                    ++i;
               }
               return (getDebtAtIndex(i).getAmount());
          }
          catch (ArrayIndexOutOfBoundsException e){
               // No debt has been found
               return 0;
          }
     }
```

```
public void setDebt(Frat creditor, int newAmount){
     try {
          int i = 0;
          while (getDebtAtIndex(i).getCreditor() != creditor){
               ++i;
          }
          if (newAmount != 0){
               getDebtAtIndex(i).setAmount(newAmount);
          }
          else{
               deleteDebt(creditor);
          }
     }
     catch (ArrayIndexOutOfBoundsException e){
          System.out.println("Error while changing debt : creditor could not be found.");
     }
}

public boolean hasDebt(Frat otherFrat){
     /** Returns true if this frat has debts for other frat, false otherwise.*/
     return (getDebt(otherFrat) != 0);
}

////////// CONSTRUCTOR //////////
public Frat(String name, int budget){
     setName(name);
     setBudget(budget);
     _debtList = new Vector<Debt>();
}

public String toString() {
     return getName();
}

////////// WORK METHODS //////////
public void addDebt(Frat creditor, int amount){
     _debtList.add(new Debt(creditor, amount));
}

public void deleteDebt(Frat creditor){
     /** deleteDebt deletes the debt of a creditor. */
     try {

          int i = 0;
          while (getDebtAtIndex(i).getCreditor() != creditor){
               ++i;
```

```
                }
                getDebtList().remove(i);

            } catch (ArrayIndexOutOfBoundsException e){
                System.out.println("Error while deleting debt : creditor could not be found.");
            }
        }

        public void deleteDebt(int index){
            _debtList.remove(index);
        }

        public void changeDebt(Frat creditor, int amount){
            /** changeDebt allows to add an amount to creditor's debt. Positive and
                negative amounts are accepted. */
            try{
                setDebt(creditor, getDebt(creditor) + amount);
            }
            catch ( ArrayIndexOutOfBoundsException e ) {
                System.out.println("Error while changing debt : creditor could not be found.");
            }
        }

}

class Debt {

        private int _amount;
        private Frat _creditor;

        ////////// GETTERS + SETTERS //////////
        public int getAmount() { return _amount; }
        public void setAmount(int amount) { _amount = amount; }

        public Frat getCreditor() { return _creditor; }
        public void setCreditor(Frat creditor) { _creditor = creditor; }

        ////////// CONSTRUCTOR //////////
        public Debt(Frat creditor, int amount){
            setCreditor(creditor);
            setAmount(amount);
        }
}
import java.util.Vector;

public class Frat {
```

```
private String _name;
private int _budget;
private Vector<Debt> _debtList;

////////// GETTERS + SETTERS //////////
public String getName() { return _name; }
public void setName(String name) { _name = name; }
public boolean isFrat(String fratName) { return (fratName.equals(getName())); }
public boolean isFrat(Frat otherFrat) { return (this == otherFrat); }

public int getBudget() { return _budget; }
public void setBudget(int budget) { _budget = budget; }

public Vector<Debt> getDebtList() { return _debtList; }
private Debt getDebtAtIndex(int i){
     return getDebtList().get(i);
}

public int getDebt(Frat creditor){
     try {
          int i = 0;
          while (getDebtAtIndex(i).getCreditor() != creditor){
               ++i;
          }
          return (getDebtAtIndex(i).getAmount());
     }
     catch (ArrayIndexOutOfBoundsException e){
          // No debt has been found
          return 0;
     }
}

public void setDebt(Frat creditor, int newAmount){
     try {
          int i = 0;
          while (getDebtAtIndex(i).getCreditor() != creditor){
               ++i;
          }
          if (newAmount != 0){
               getDebtAtIndex(i).setAmount(newAmount);
          }
          else{
               deleteDebt(creditor);
          }
     }
```

```
        catch (ArrayIndexOutOfBoundsException e){
            System.out.println("Error while changing debt : creditor could not be found.");
        }
    }

    public boolean hasDebt(Frat otherFrat){
        /** Returns true if this frat has debts for other frat, false otherwise.*/
        return (getDebt(otherFrat) != 0);
    }

    ////////// CONSTRUCTOR //////////
    public Frat(String name, int budget){
        setName(name);
        setBudget(budget);
        _debtList = new Vector<Debt>();
    }

    public String toString() {
        return getName();
    }

    ////////// WORK METHODS //////////
    public void addDebt(Frat creditor, int amount){
        _debtList.add(new Debt(creditor, amount));
    }

    public void deleteDebt(Frat creditor){
        /** deleteDebt deletes the debt of a creditor. */
        try {

            int i = 0;
            while (getDebtAtIndex(i).getCreditor() != creditor){
                ++i;
            }
            getDebtList().remove(i);

        } catch (ArrayIndexOutOfBoundsException e){
            System.out.println("Error while deleting debt : creditor could not be found.");
        }
    }

    public void deleteDebt(int index){
        _debtList.remove(index);
    }

    public void changeDebt(Frat creditor, int amount){
```

```
          /** changeDebt allows to add an amount to creditor's debt. Positive and
               negative amounts are accepted. */
          try{
               setDebt(creditor, getDebt(creditor) + amount);
          }
          catch ( ArrayIndexOutOfBoundsException e ) {
               System.out.println("Error while changing debt : creditor could not be found.");
          }
     }

}

class Debt {

     private int _amount;
     private Frat _creditor;

     ////////// GETTERS + SETTERS //////////
     public int getAmount() { return _amount; }
     public void setAmount(int amount) { _amount = amount; }

     public Frat getCreditor() { return _creditor; }
     public void setCreditor(Frat creditor) { _creditor = creditor; }

     ////////// CONSTRUCTOR //////////
     public Debt(Frat creditor, int amount){
          setCreditor(creditor);
          setAmount(amount);
     }
}
```

Graph.java

```
import java.io.*;
import java.util.Vector;

public class Graph {

    private Vector<Frat> _fratList;
    private Vector<Vector<Frat>> _cycles;

    ////////// GETTERS + SETTERS //////////
```

```
    public Vector<Frat> getFratList(){ return _fratList; }
    public int getLength(){ return _fratList.size(); }
    public Frat getFrat(int index){ return _fratList.get(index); }

    public void setFratList(Vector<Frat> fratList){
        _fratList.clear();
        _fratList = new Vector<Frat>(fratList);
    }

    ////////// CONSTRUCTOR //////////

    public Graph(String fileName){

        File file = new File(fileName);
        BufferedReader reader = null;
        try {
            reader = new BufferedReader(new FileReader(file));
            String text = null;
            text = reader.readLine();
            _fratList = new Vector<Frat>(Integer.decode(text));
            Frat tmp;
            Frat tmpCreditor;

            while ((text = reader.readLine()) != null) {
                String[] splitLine = text.split("\\s+");

                if (splitLine.length==2){
                    //System.out.println("1 : *"+splitLine[0]+"*"+splitLine[1]);
                    tmp = new Frat(splitLine[0],Integer.decode(splitLine[1]));
                    _fratList.add(tmp);

                } else if (splitLine.length==3){
                    tmp=null;
                    tmpCreditor = null;

                    for (int i=0;i<_fratList.size();i++){
                        if (_fratList.get(i).isFrat(splitLine[0])){
                            tmp = _fratList.get(i);//get Debitor
                        } else if (_fratList.get(i).isFrat(splitLine[1])){
                            tmpCreditor = _fratList.get(i);//get Creditor
                        }
                    }
                    if (tmp!=null && tmpCreditor!=null){//if both were found, add Debt
                        //System.out.println("adding debt");
                        tmp.addDebt(tmpCreditor,Integer.decode(splitLine[2]));
                    } else{System.out.println("One of the Fraternitys was not found in a debt");}
```

```
                }
            }
        } catch (FileNotFoundException e) {
            System.out.println("File not found");
        } catch (IOException e) {
            System.out.println("Error opening file");
        }try {
            if (reader != null) {
                reader.close();
            }
        } catch (IOException e) {
            System.out.println("Error closing file");
        }
        _cycles = new Vector<Vector<Frat>>();
    }

    ////////// WORK METHODS //////////

    public static void main(String[] argv){
        try{
            Graph a = new Graph(argv[0]);
            a.detectCycles();
            a.reduceCycles();
            a.graphToImage("debtNoCycles");
            System.out.println("Le fichier debtNoCycles.dot contient la situation sans cycles.\nUtilisez la commande : <dot Tpng
debtNoCycles.dot -o debtNoCylces.png>\npour creer l'image.\n");
            a.payBack();
            a.graphToImage("debtRefunded");
            System.out.println("Le fichier debtNoCycles.dot contient la situation actuelle.\nUtilisez la commande : <dot Tpng
debtRefunded.dot -o debtRefunded.png>\npour creer l'image.\n");
        } catch (Exception e) {System.out.println("An error has occured");}
    }

    public void detectCycles(){
        Vector<Frat> visited = new Vector<Frat>();
        for (Frat frat:getFratList()){
            if (! visited.contains(frat)){
                Vector<Frat> path = new Vector<Frat>();
                findCycle(frat, path, visited);
            }
        }
    }

    public void findCycle(Frat currentFrat, Vector<Frat> path, Vector<Frat> visited){
        for (Debt debt:currentFrat.getDebtList()){ // looking for all creditors
                if (path.contains(debt.getCreditor())){
```

```
                    // found cycle
                    path.add(currentFrat);
                    Vector<Frat> cycle = new Vector<Frat>();
                    int i = 0;
                    while (path.get(i) != debt.getCreditor()){
                        i++; // the first frats in the path are not necessarily
                    } // in the cycle. we're skipping them here.
                    do{
                        cycle.add(path.get(i));
                        i++;
                    } while (i<path.size());
                    _cycles.add(cycle);
                    path.remove(currentFrat);
                }
                else {
                    path.add(currentFrat);
                    findCycle(debt.getCreditor(), path, visited);
                    path.remove(currentFrat);
                }
        }
        visited.add(currentFrat);
    }

    public void reduceCycles(){
        int cyclesNumber = _cycles.size();
        for(int cycleID = 0; cycleID<cyclesNumber; cycleID++){
            Vector<Frat> cycle = _cycles.get(cycleID);
            int amountToReduce = minimumDebtOf(cycle);

            // reducing all debts by amountToReduce
            System.out.println(cycleID+1+") Reduction de "+amountToReduce+" :");
            for (int i=0; i<cycle.size(); ++i){
                Frat frat = cycle.get(i);
                Frat nextFrat = cycle.get( (i+1)%cycle.size() );
                System.out.print(frat+" -"+frat.getDebt(nextFrat)+"-> ");
                frat.changeDebt(nextFrat, -amountToReduce);

                if (frat.getDebt(nextFrat) == 0){
                    // this link might be in other cycles, we have to delete the
                    // other cycles containing it
                    cyclesNumber -= deleteCyclesContaining(frat, nextFrat, cycleID+1);
                }
            }
            System.out.println(cycle.get(0)+" ...");
            System.out.println("Nouvelle situation : ");
            printReducedCycle(cycle);
```

```
            System.out.println();
        }
    }

    public int minimumDebtOf(Vector<Frat> cycle){
        int res = cycle.get(0).getDebt(cycle.get(1));
        int newDebt = 0;
        for (int i=1; i<cycle.size(); ++i){
            newDebt = cycle.get(i).getDebt(cycle.get( (i+1)%cycle.size() ));
            if (newDebt < res){
                res = newDebt;
            }
        }
        return res;
    }

    public int deleteCyclesContaining(Frat debitor, Frat creditor, int fromID){
        int cyclesNumber = _cycles.size();
        int cyclesDeleted = 0;
        while (fromID<cyclesNumber){
            boolean toDelete = false;
            Vector<Frat> nextCycle = _cycles.get(fromID);
            int j=-1;
            while (!toDelete && j+1<nextCycle.size()){
                j++;
                if (nextCycle.get(j) == debitor){
                    if (nextCycle.get((j+1)%nextCycle.size()) == creditor){
                        toDelete = true;
                    }
                }
            }
            if (toDelete){
                _cycles.remove(fromID);
                cyclesNumber--;
                cyclesDeleted++;
            }
            else{
                fromID++;
            }
        }
        return cyclesDeleted;
    }

    public void printReducedCycle(Vector<Frat> cycle){
        for (int i=0; i<cycle.size(); ++i){
            Frat frat = cycle.get(i);
```

```
            Frat nextFrat = cycle.get( (i+1)%cycle.size() );
            if (frat.getDebt(nextFrat) != 0){
                System.out.print(frat+" -"+frat.getDebt(nextFrat)+"-> "+nextFrat+" | ");
            }
        }
        System.out.println();
    }

    public void payBack(){
        System.out.println("Ordre des remboursements :\n");
        boolean done = false;//tu check if finished
        while(done==false){
            @SuppressWarnings("unchecked")
            Vector<Frat> start = (Vector<Frat>) _fratList.clone();
            //get in start Frats who wil not be refunded at the moment
            Vector<Debt> debtList = new Vector<Debt>();
            for (Frat tmp:getFratList()){
                debtList = tmp.getDebtList();
                if (tmp.getBudget()>0 && tmp.getDebtList().size()>0){
                    //but Frats must also have some budget and have at least one debt
                    for (Debt tmpDebt:debtList){
                        start.remove(tmpDebt);
                    }
                }else{start.remove(tmp);}
            }
            if(start.size()>0){
                //if at least one Frat fullfills conditions, we are not done yet
                for (Frat tmp:start){//for every Frat in start
                    debtList = tmp.getDebtList();
                    for(int i = 0;i<debtList.size();i++){// and for every debt that Frat has
                        if(tmp.getBudget()>=debtList.get(i).getAmount()){
                            //if can fully refund
                            tmp.setBudget(tmp.getBudget()-debtList.get(i).getAmount());
                            debtList.get(i).getCreditor().setBudget(debtList.get(i).getCreditor().getBudget()
+debtList.get(i).getAmount());
                            System.out.println(tmp.getName()+" ("+debtList.get(i).getAmount()+") ->
"+debtList.get(i).getCreditor().getName());
                            tmp.deleteDebt(debtList.get(i).getCreditor());
                        }else{
                            //else refund max possible
                            debtList.get(i).getCreditor().setBudget(debtList.get(i).getCreditor().getBudget()+tmp.getBudget());
                            tmp.changeDebt(debtList.get(i).getCreditor(),-tmp.getBudget());
                            System.out.println(tmp.getName()+" ("+tmp.getBudget()+") -> "+debtList.get(i).getCreditor().getName()+"
reste "+debtList.get(i).getAmount());
                            tmp.setBudget(0);
                        }
```

```
                    }
                }
            }else{done=true;}
        }
        System.out.println();
    }

    public void graphToImage(String filename){
        Writer writer = null;

        try {
            writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(filename+".dot"), "utf-8"));
            writer.write("digraph G {\n");
            Vector<Debt> debtList;
            for (Frat tmp:getFratList()){//for every node in graph

                String myName = ("\""+tmp.getName()+"\n"+tmp.getBudget()+"\"");
                //myName == string with name and budget of Frat
                writer.write(myName+" [style=filled, fillcolor = orange]"+"\n");
                //create node with orange color and myName as info
                debtList = new Vector<Debt>(tmp.getDebtList());

                for (Debt tmpDebt:debtList){//for every debt a Frat has

                    Frat tmpCreditor = tmpDebt.getCreditor();
                    String debtName = ("\""+tmpCreditor.getName()+"\n"+tmpCreditor.getBudget()+"\"");
                    //same as myName but for creditor
                    String line = " "+myName+" -> "+debtName+"[label=\" "+tmpDebt.getAmount()+"\"];\n";
                    writer.write(line);
                }
            }
        } catch (IOException ex) {
          System.out.println("Could not create .dot file");
        } finally {
            try {
                writer.write("}");
                writer.close();
            } catch (Exception ex) {System.out.println("Could not close .dot file");}
        }
    }

}
```