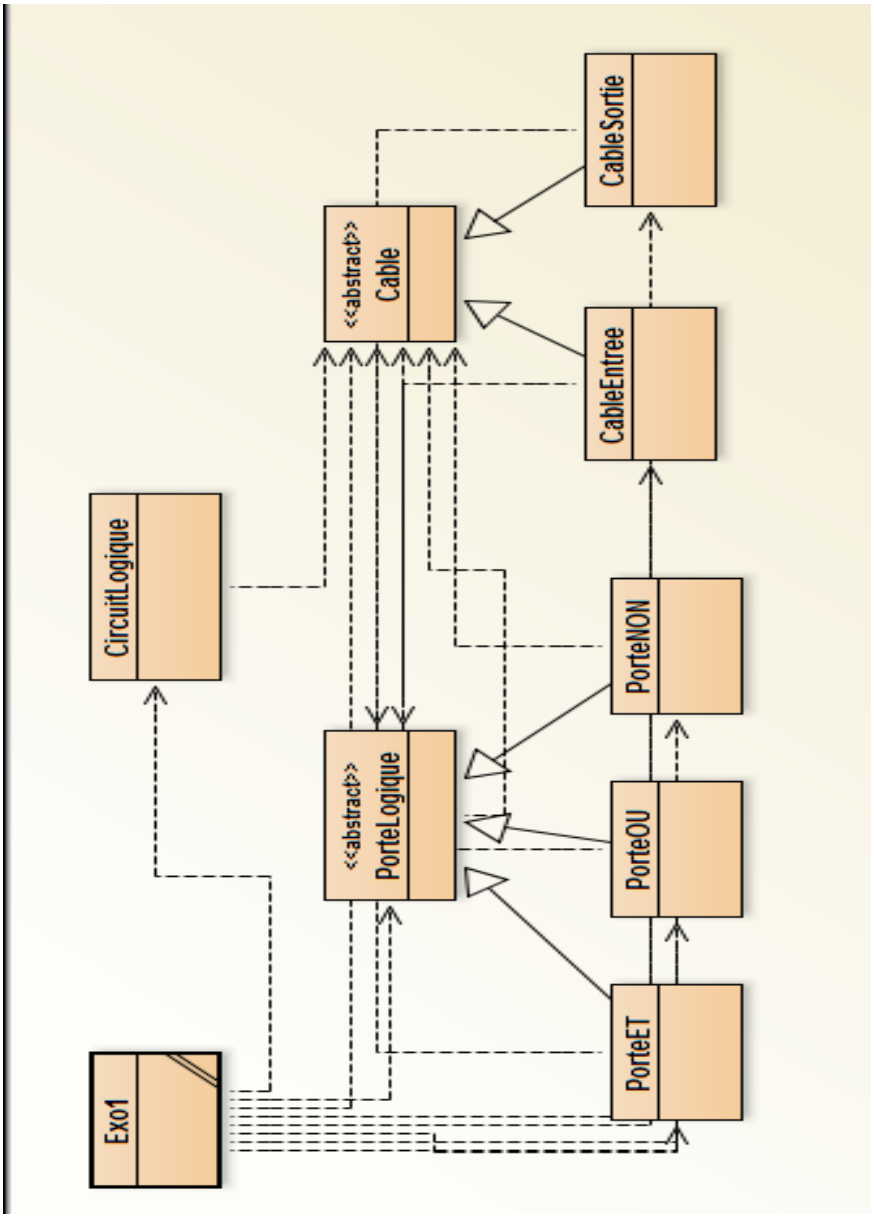
Corrigé de l'examen final

(2 heures)

Exercice 1 (9 points)

1. Diagramme des classes (2 points)



2. Donner le programme java de chacune des classes.

Le programme ci-dessous n'est un corrigé type. Toute autre solution sera étudiée.

```
package Exo1;                          1 point
import java.util.*;

public class CircuitLogique {

ArrayList <Cable> entrees;
CableSortie sortie;

public CircuitLogique(ArrayList<Cable>
entrees, CableSortie sortie){
this.entrees = entrees;
this.sortie = sortie;
}
public boolean
calculerSortie(ArrayList<Boolean>
valeursEntrees)
{
 for (int i = 0; i<entrees.size(); i++)

(entrees.get(i)).setEtat((valeursEntrees.get(i)
).booleanValue());

 System.out.println("Circuit: CalculerSortie");
 return sortie.getEtat();
      }

}
```

```
package Exo1;                0,5 point

public abstract class  Cable {

      protected boolean etat;

      public Cable (String nom){}

      public Cable(boolean valeur){
            etat = valeur;
      }

      public boolean getEtat(){

            return etat;
      }

public void setEtat(boolean valeur){
            etat = valeur;
      }


}
```

```
package Exo1;                1 point

import java.util.*;

public class CableEntree extends Cable{
String nom;

public CableEntree(String nom){
super(nom);}

public boolean getEtat(){
System.out.println(" état du cable " + nom+
":"+ etat);
      return etat;
}
}
```

```
package Exo1;                          1 point
import java.util.*;

public class CableSortie extends Cable {
      PorteLogique source;

public CableSortie(PorteLogique
source,String nom){
            super(nom);
            this.source = source;
      }

      public CableSortie(boolean etat){
            super(etat);
      }


      public boolean getEtat(){
      etat = source.calculerSortie();
            return etat;
      }
}
```

1 point

```
package Exo1;

public abstract class PorteLogique {
  protected String nom;
  protected Cable entree1;
  protected CableSortie sortie;

  public PorteLogique (Cable e1, String
nom){
        this.nom = nom;
         entree1 = e1;
 sortie = new CableSortie(this,
"s"+"nom");
  }


  abstract boolean calculerSortie();

  public CableSortie getSortie(){
               return sortie;
  }



`
```

0,5 point

```
package Exo1;

public class PorteNON extends
PorteLogique{

public PorteNON(Cable e1, String nom){
              super(e1,nom);
       }

boolean calculerSortie(){
System.out.println(nom + ": Sortie = "
+! entree1.getEtat() );

return ! entree1.getEtat();
       }


}
```

1 point

```
package Exo1;

public class PorteET extends
PorteLogique {

private Cable entree2;

public PorteET(Cable e1, Cable e2,
String nom){
              super(e1,nom);
              entree2 = e2;
       }

boolean calculerSortie(){
System.out.println(nom +  ": Sortie =
"+ (entree1.getEtat() &&
entree2.getEtat()));
return (entree1.getEtat() &&
entree2.getEtat()) ;
       }


}
```

1 point

```
package Exo1;

public class PorteOU extends
PorteLogique {
private Cable entree2;

public PorteOU(Cable e1, Cable
e2,String nom){
       super(e1,nom);
       entree2 = e2;
}

       boolean calculerSortie(){
              System.out.println(nom+" :
Sortie = "+ ( entree1.getEtat() &&
entree2.getEtat()));
return (entree1.getEtat() ||
entree2.getEtat()) ;
       }

}
```

Exercice 2 (9 points)

1. la classe étudiant (2 points)

```
package Exo2;

import java.io.*;

public class Etudiant implements Comparable<Etudiant>,
Serializable{

private String matricule, nom, prenom;
float moyennes[];
float moyGenerale;

public Etudiant(String mat, String n, String p, float[]moy){
	matricule = mat;
	nom = n;
	prenom = p;
	moyennes = new float[moy.length];
	float moyGen = 0 ;

	for (int i = 0; i<moy.length; i++){
		moyGen = moyGen + moy[i];
		};

	moyGen = moyGen/moy.length;

	moyGenerale = moyGen;
	}

public int compareTo(Etudiant autre){
	float x = moyGenerale-autre.moyGenerale;

	if( x<0) return 1;
	else if (x>0) return -1;
	else  return 0;
}

public String toString(){

	return matricule+ " "+ nom+ " "+ prenom + " " + moyGenerale;

}

}
```

2. Ecrire le code java de la classe Promotion **(2 points)**

```
package Exo2;
import java.util.*;
import java.io.*;

public class Promotion implements Serializable {

	private TreeSet<Etudiant> promotionClassee;
	private int nbModules;

	public Promotion(int nbModules){
	this.nbModules = nbModules;
	e = new TreeSet<Etudiant>();
	}

	public void ajouterEtudiant(Etudiant e){
	promotionClassee.add(e);
	}


	public TreeSet<Etudiant> getClassement(){
		return promotionClassee;
	}

	public int getNbModules(){
		return nbModules;
	}
}
```

3. Le code donné en page 4 est le squelette de la méthode main. Ecrire le code java de la partie 3 permettant de sauvegarder le contenu de la collection dans le fichier "classement.txt". Pour cela utilisez un flux muni d'un buffer.

```
// 2. Ecriture dans le fichiee
                                                    2 points
try{
out = new BufferedWriter(new FileWriter("classement.txt"));

Iterator<Etudiant> it=promotion.getClassement().iterator();
while (it.hasNext()){
out.write(it.next().toString());
out.newLine();      // Ecrire un séparateur de lignes
out.flush(); // vider le tampon
}
}
catch (Exception ex){
ex.printStackTrace();
  }
```

4. Pour une meilleure flexibilité, on désire donner la possibilité à l'utilisateur du programme de saisir les données des étudiants par parties au lieu de le faire pour la promotion entière en une seule exécution.

a)-Expliquez précisément, et sans donner le code, ce qu'il faut modifier dans le programme

Réponse: Pour assurer cela deux possibilités existent: **1 point**

1. Nous pouvons utiliser le mécanisme de sérialisation qui sauvegarde les objets dans leur état et les restaure au moment voulu. Dans notre cas, il s'agit de l'objet promotion contenant la collection promotionClassee. Les modifications à faire dans le code:

- les classes Promotion et étudiant doivent implémenter l'interface Serializable.
- Déserialiser en début du main et sérialiser en fin de main

2. Nous pouvons au début de chaque exécution construire la collection à partir du contenu du fichier, y ajouter de nouveau objets et écrire à la fin

b) -Donner le code java des instructions à ajouter et/ou modifier. **2 points**

Ici je donne le code correspondant à la solution 1 . Dans la solution 2, on doit faire en début de main, la lecture du fichier, le parsage (en utilisant StringTokenizer par exemple) et la création de la collection avant de pouvoir faire la saisie.

```
// serialisation

ObjectOutputStream save = new ObjectOutputStream(new BufferedOutputStream(
                          new FileOutputStream(new File("classement.dat"))));
// écriture les objets dans un fichier
save.writeObject(promotion);
// Fermer le flux
save.close();
```

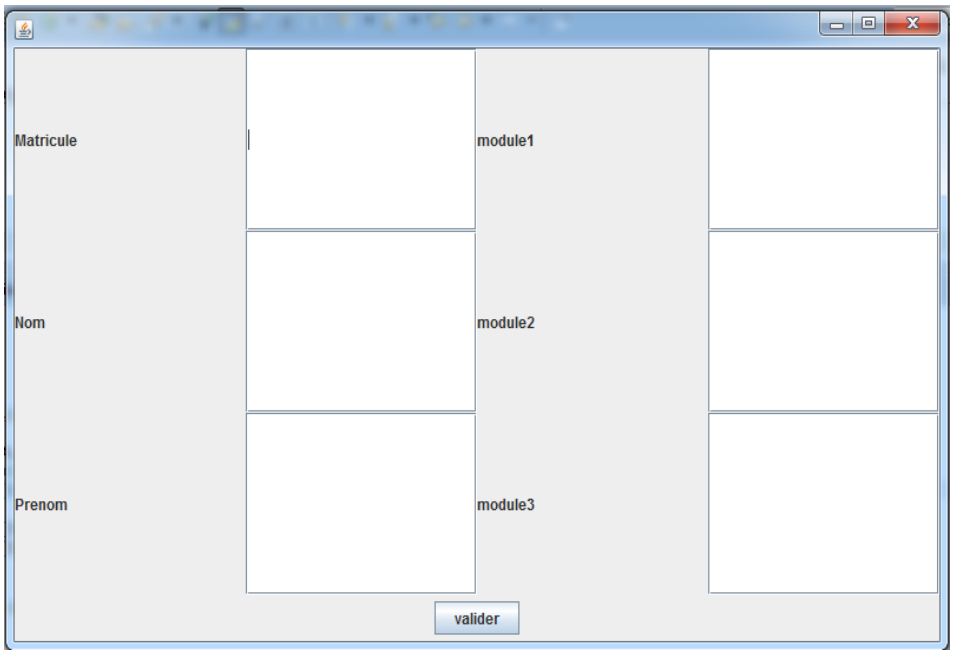
```
//deserialisation
try {
ObjectInputStream in = new ObjectInputStream(new BufferedInputStream(
                           new FileInputStream(new File("classement.dat"))));
promotion = (Promotion) in.readObject();
 in.close();
 } catch (Exception ex) {
       ex.printStackTrace(); }
}
```

Exo3 (2 points)

Il y a deux versions de cet exercice. c'est la disposition des composants qui diffère selon les arguments du GridLayout

Version 1.

```
p.setLayout(new GridLayout(0,2));
p1.setLayout(new GridLayout(3,2));
p2.setLayout(new GridLayout(3,2));
```



Version 2.

```
p.setLayout(new GridLayout(0,2));
p1.setLayout(new GridLayout(6,0));
p2.setLayout(new GridLayout(6,0));
```