TD3

Objectifs :

- Connaitre les principes du reverse engineering (du code aux diagrammes)
- Connaitre les principes du forward engineering (des diagrammes au code)

Exercice 1. Reverse engineering

Modélisez par un diagramme de classes de conception le code Java suivant. Donnez un maximum de détails (visibilité des attributs/méthodes, type des attributs/paramètres des méthodes, navigabilités / rôles / multiplicités, relations d'héritage / implémentation / association / agrégation / composition / dépendance, éléments statiques / abstraits, stéréotypes, etc.).

```java
public class A extends B {
	private C c;
	public C getC(){
		return c;
	}
	public void setC(C c ){
		this.c = c;
	}
	public void methode2() {
		getC().affiche(this);
	}
}
```

```java
public abstract class B implements I1 {

	public abstract void methode2();

	public void methode() {
	   System.out.println("zzzzz");
	}
}
```

```java
public interface I1 extends
test.I0, test.I2{

	public void methode2();

}
```

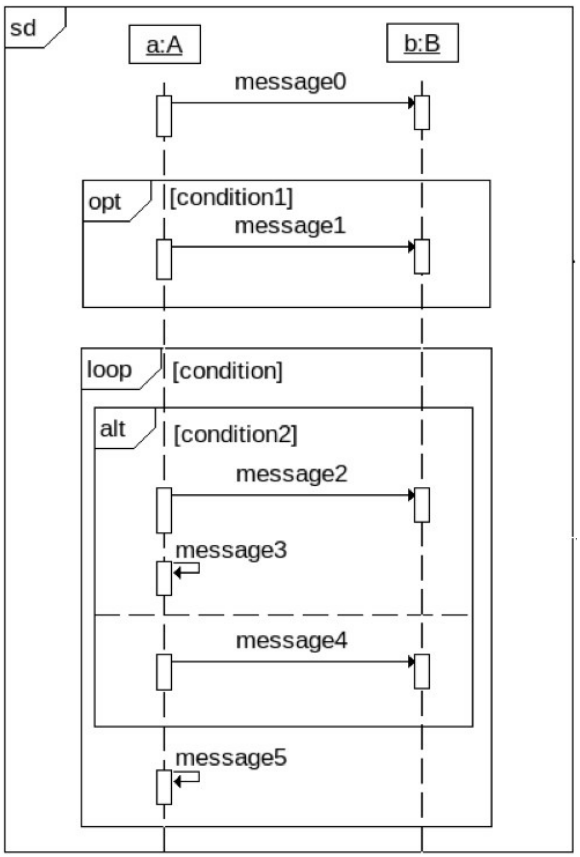
```java
public class C {
   public C() {
       super();
   }
   public void affiche(I2 moche){
       System.out.println(moche.EXAMEN);
   }
}
```

```java
public interface I0 {
  public void methode();
}
```

```java
public interface I2 {
  public static final String   EXAMEN="Super
Examen";
}
```

Exercice 2. Forward engineering en séquence

1. Donnez le pseudocode équivalent au diagramme de séquence suivant.



2. Donnez le diagramme de classe que l'on peut également déduire du diagramme de séquence.

Exercice 3. Reverse engineering en séquence

Modélisez par un diagramme de séquence de conception le code Java suivant. Utilisez à bon escient les paramètres de message, les valeurs de retour des messages synchrones, la création d'objets, les frames, etc. Le commencement correspond à la méthode main().

```
public class Etudiant {

	public void soudoyer(Enseignant ens){
		Cheque c=new Cheque(500);
		boolean accepte=ens.recevoirCheque(c);
		if (accepte)
			ens.demanderAide();
	}

	public static void main(){
		Etudiant chris=new Etudiant();
		Enseignant pedro=new Enseignant();
		chris.soudoyer(pedro);
	}
}
```

```
public class Enseignant {

	public boolean recevoirCheque(Cheque cheque){
		int montant = cheque.getMontant();
		return (montant>100);
	}

	public void demanderAide() {

	}
}
```

```
public class Cheque {
	private int montant;
	public Cheque(int valeur) {
		this.montant=valeur;
	}
	public int getMontant() {
		return montant;
	}
}
```

Exercice 4. Synchronisation

1. Modélisez les 2 classes définies dans le code ci-dessous par un diagramme de classe de conception le plus détaillé possible.

```
public class Synchronisateur {

   private List<Calculateur> calculateurs;

   public Synchronisateur(){
         calculateurs=new ArrayList<Calculateur>();
   }
   public void ajouterCalculateur(Calculateur newCalc){
         calculateurs.add(newCalc);
   }
   public void calculer(String calcul){
     Iterator<Calculateur> it=calculateurs.iterator();
     while (it.hasNext()){
         Calculateur calculateur=it.next();
         System.out.println("Envoi d'un calcul vers
"+calculateur);
         calculateur.calculer(calcul);
     }
   }
   public synchronized void recupereResultat(Calculateur
calc,String res) {
      System.out.println("Resultat recu du calculateur
"+calc + " : "+res);
   }
}
```

```
public class Calculateur {

   private static int nbCalc=0;
   private Synchronisateur mySync;
   private String myName;

   public Calculateur(Synchronisateur sync){
         mySync=sync;
         sync.ajouterCalculateur(this);
         myName="Calculateur"+nbCalc++;
   }
   public String toString(){
         return myName;
   }
   public void calculer(String calcul) {
         System.out.println(myName + ": je lance
le calcul "+calcul);
         //calcul
         mySync.recupereResultat(this, "fin du
calcul");
   }
}
```

2. Identifiez/justifiez les dépendances actuelles entre les 2 classes.

3. Afin de mieux comprendre la mise en place des calculateurs avec un synchronisateur ainsi que les échanges entre eux lors d'un calcul, modélisez par un diagramme de séquence le scénario de test suivant (en détail : utilisez à bon escient les paramètres de message, les valeurs de retour des messages synchrones, la création d'objets, les *frames*, etc.).

```
public class Test {
  public static void main(String[] args) {
         Synchronisateur leSynchro = new Synchronisateur();
         new Calculateur(leSynchro);
         new Calculateur(leSynchro);
         leSynchro.calculer("test");
   }
}
```

4. Nous souhaitons que les 2 classes soient dans des paquetages différents. Proposez une solution (diagramme de classe/paquetage) évitant les cycles de dépendance.