Exercice 2 (copie 2, 45 minutes environ, répondre sur le sujet)

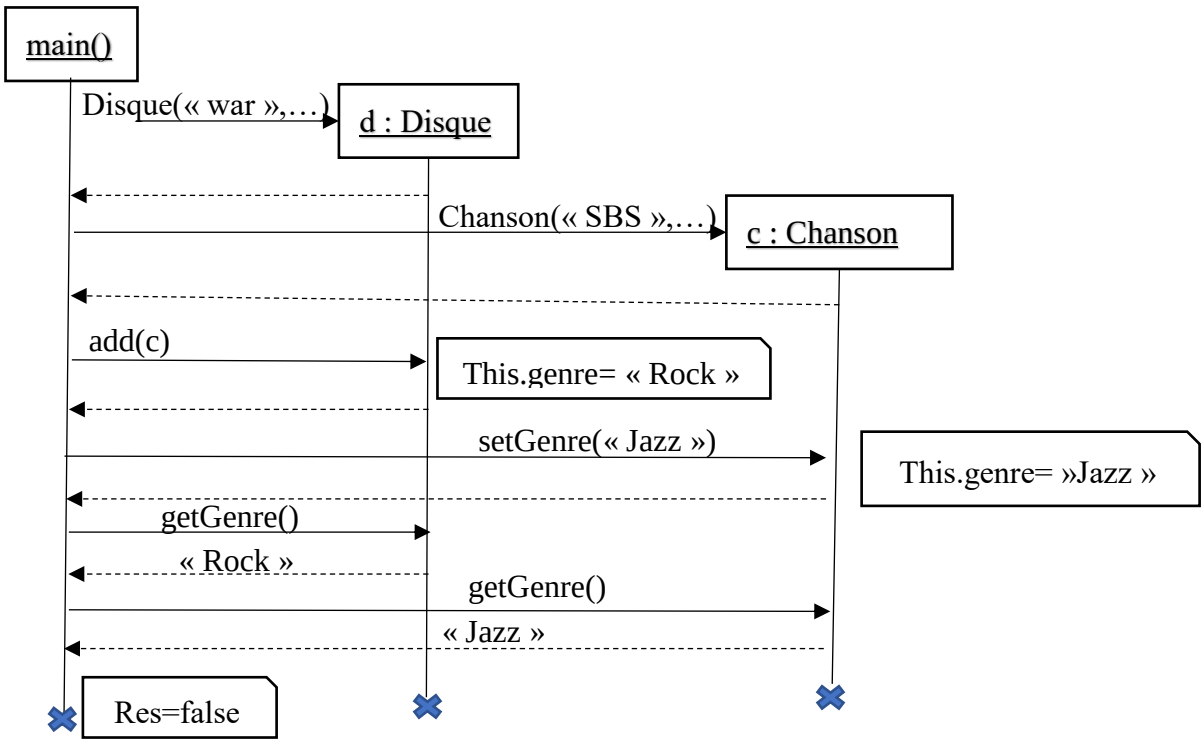
Question 1

a) La classe **`Chanson`** est-elle immuable ? Justifiez votre réponse.

Elle n'est pas immuable car elle hérite avec usufruit de la méthode setGenre() permettant de changer la valeur de l'attribut genre. Il est donc possible de changer l'état d'un objet Chanson après sa création/initialisation.

b) On considère le code ci-dessous. Donnez le *diagramme de séquence* correspondant à son exécution. On ne dessinera pas les messages vers les **`ArrayList`** ni aucune des actions réalisées dans la méthode **`add()`** (juste ce message et non ce qu'il génère comme instructions)

```
Public static void main(String[]args){
{
      Disque d= new Disque("War", "Rock", "28 février 1983");
      Chanson c = new Chanson("SBS", "Rock","28 février 1983");
      d.add(c);
      c.setGenre("Jazz");
      boolean res=d.getGenre().equals(c.getGenre());
}
```



c) Quel problème de sécurité met en évidence le code de la question b) si l'on souhaite dorénavant qu'un disque soit le seul maître des chansons qui le composent ?

On constate qu'un disque n'est pas maître de son état car les chansons qui lui sont associées ne sont pas copiées de manière défensive lors d'un add(). Comme la classe Chanson n'est pas immuable l'appelant

(disposant de l'adresse) a donc tout loisir de modifier après coup l'état d'une chanson stockée sur un disque sans que celui-ci n'en soit informé et en particulier le genre d'une chanson qui pourrait ne pas rester cohérent avec celui du disque.

d) On souhaite régler le problème identifié à la question c). Quelles méthodes de la classe **`Disque`** sont à modifier et pour faire quoi (on ne demande pas d'écrire ce code) ?

Il faut modifier la version actuelle de la méthode add() pour assurer une copie défensive de la chanson reçue en paramètre.

e) On propose d'ajouter le constructeur ci-dessous à la classe **`Chanson`** : un constructeur par copie (**copy-constructor**). Ce code génère des erreurs de compilation ! Expliquez ce qui ne va pas. Proposez une version opérationnelle, élégante et sûre. Indiquez, si cela est nécessaire, les éventuelles modifications à faire sur le reste du code pour que votre constructeur fonctionne.

Le constructeur proposé (qui ne compile pas)	*Expliquez ici pourquoi il y a ces erreurs*

`public Chanson(Chanson c){`
`  super(c.titre, c.genre, c.date);`
`}`
	Même si ce sont deux instances de chansons qui dialoguent l'une avec l'autre les 3 attributs titre, genre, date ont été hérités sans usufruit (privés dans la super-classe Œuvre) dans la chanson appelée. Les 3 accès inter-objets sont donc illégaux.

Donnez ici votre proposition de constructeur
`public Chanson(Chanson c){`
`  super(« », « », « ») ;`
`  if (c !=null){ // version avec test`
`  this.setDate(c.getDate()) ;`
`  this.setGenre(c.getGenre());`
`  this.setTitre(c.getTitre());`
`  }`
`}`
 	
`public Chanson(Chanson c){// sans test`
`super(c.getTitre(),c.getGenre(),c.getDate()) ;`
`}`
	Ici les modifications éventuelles à faire ailleurs
	
	Il n'y a aucun moyen de récupérer la valeur de l'attribut date en l'état. Il faut rajouter un getter String getDate(). Si de plus on veut pouvoir tester la non nullité de c, il faut rajouter 2 setter (pour date et titre) car le super placé en première ligne ne permet pas le test et qu'il est impossible pour une chanson d'accéder aux attributs hérités dans son code (visibilité privée donc sans usufruit).

Question 2

On veut écrire le code d'une sous classe **`ChansonSans`** de la classe **`Chanson`**. Située dans le package **`production`**, elle a un attribut privé **`aCapella`** de type **`boolean`**. Elle comporte deux méthodes publiques : un constructeur et une méthode **`void joue()`** qui affiche « je joue la chanson a capella » si **`aCapella`** vaut **`true`** sinon la même chose que la classe **`Chanson`**.

a) Donnez le code complet de **`ChansonSans`**.

```
package  production ;

public class ChansonSans extends Chanson{

     private boolean aCapella ;

     public ChansonSans(String titre, String genre, String date,
Boolean aCap){

          super(titre, genre, date);

          this.aCapella=aCap;

     }

     public void joue(){

          if (this.aCapella){ System.out.println("je joue a capella");}

          else{super.joue();}

   }

}
```

b) On souhaite avoir des chansons normales et des chansons a capella sur le même disque. Doit-on modifier le code de la Classe **`Disque`** ? Justifiez votre réponse.

Non, le code de Disque n'est pas à modifier si l'on ne fait pas de copies défensives des chansons dans add(). En effet en Java héritage rime avec sous-typage. Donc partout dans le code deDisque où l'on attend une chanson, on peut mettre une ChansonSans. Il est donc possible de stocker dans l'ArrayList<Chanson> les adresses aussi bien d'instance de Chanson que de ChansonSans. On travaille ensuite dans le code de Disque uniquement avec des types Chanson.

Notez que si l'on voulait réaliser une copie défensive dans add(), il faudrait se soucier de copier le bon type de chanson (une Chanson ou une ChansonSans). Une modification du code add()) seraient alors nécessaire. Deux techniques seraient utilisables : introspection (instanceOf() ou clonage qui impose de rendre clonable les classes Chanson et ChansonSans).