Projet BPO

Montage de films de caractères

Groupe 106

KUMAR Aman

ROBALO RODRIGES Flavio

Table des matières

I INTRODUCTION

Une société de production cinématographique souhaite révolutionner la diffusion des films qu'elle produit. Pour se faire ils ont décidé de réduire de façon considérable le volume de données nécessaire à la diffusion d'un film. La société a pris la décision de réaliser ses films avec un nouveau logiciel de montage de film utilisant le format .txt.

La société de production nous demande de pouvoir faire une multitude de modification a ses films. Ainsi, pour ce projet nous avons eu à coder certains éléments afin de développer une bibliothèque. Une bibliothèque qui doit permettre de combiner des films existants et d'y faire des opérations tels que :

- Répéter un film, il s'agit de rejouer les images du film un nombre de fois donné.

- Obtenir un extrait d'un film. Cet extrait doit être désigné par le numéro de la première et de la dernière image. La première image du film est désignée par le numéro 0.

- Encadrer un film. Le film à encadrer doit être entouré d'étoiles "*" positionnées au bord de l'écran. - Coller deux films, un film est joué puis un autre directement à la fin de celui-ci.

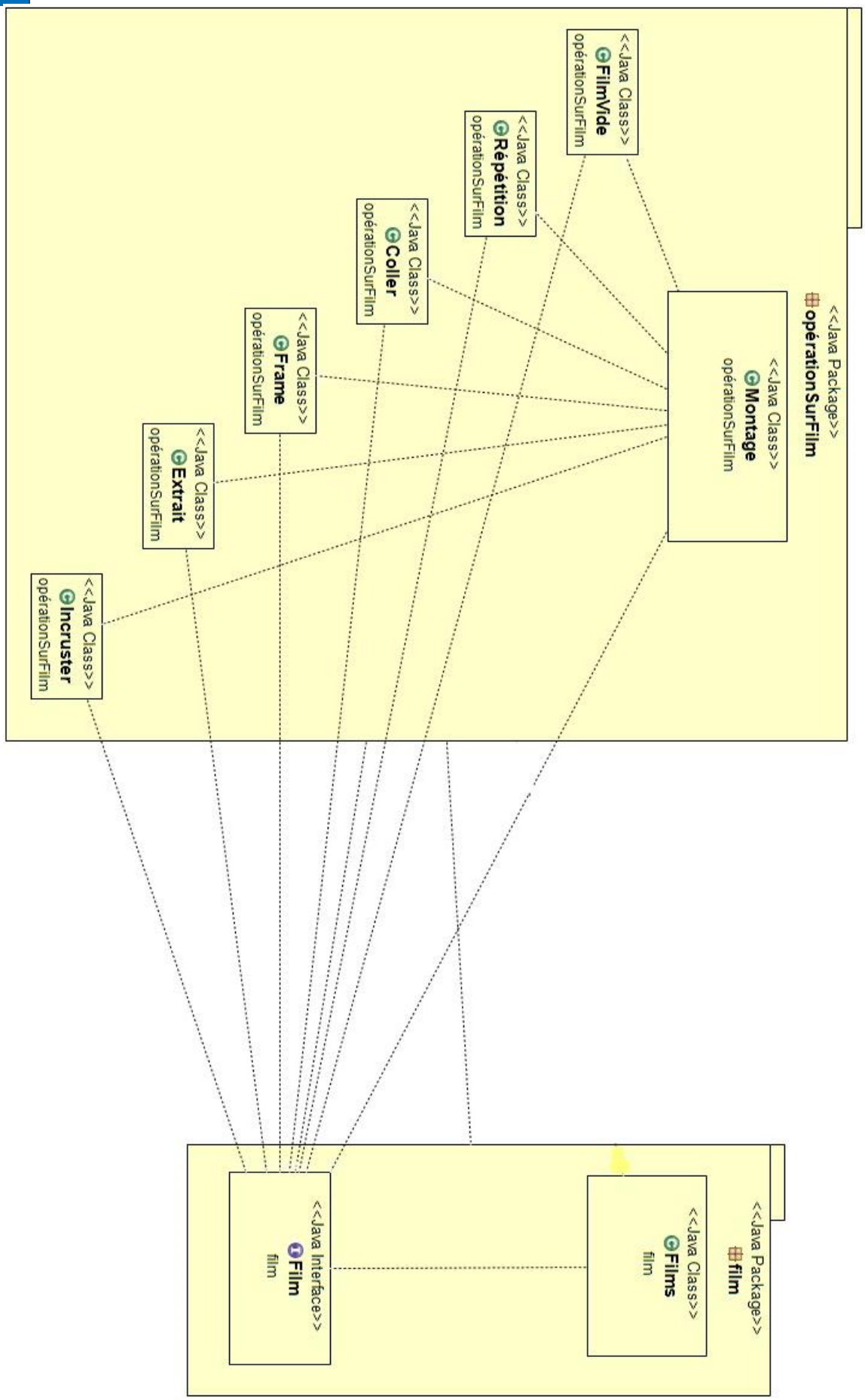
- D'incruster un film dans un film. Ce film est incrusté grâce au numéro de ligne et de colonne qui s'incrustera directement dans le coin en haut à gauche du film devant être incrusté, dans les images du film où il est incrusté.

Toutes ces opérations ont été développés dans une classe OpérationSurFilm qui implémente Film

Utiliser la bibliothèque OpérationSurFilm :

L'utilisation des différentes opérations sur les films se fait à travers la classe Montage.java qui contient des méthodes *static* permettant ainsi de proposer les diverses opérations.

II DIAGRAMME UML



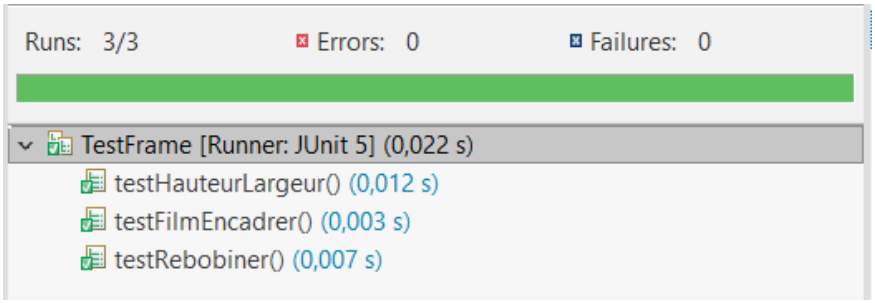
III Les tests unitaires des classes

■ TestFrame.java

```
1.  package tests;
2.
3. import static org.junit.jupiter.api.Assertions.*;
4.
5. import org.junit.jupiter.api.Test;
6.
7. import film.Film;
8. import film.Films;
9. import opérationSurFilm.Montage;
10.  import tests.films.FilmNormal;
11.  import tests.films.FrameAttendu;
12.
13.  class TestFrame {
14.
15.          /*Avoir l'écran de film sous forme d'un string
16.           * @param f film pour le quell on veut les images
17.           * @return sb string contnant les images/les écrans
18.           * */
19.          private static String toString(Film f){
20.          String sb = "";
21.          char[][] écran = new char[f.hauteur()][f.largeur()];
22.                  while(f.suivante(écran)){
23.             sb += Films.toString(écran) + "\n";
24.          }
25.          return sb;
26.      }
27.
28.          /*Test méthode encadrer film.
29.           *On compare les deux string: celle attendu et celle géérer par la
   bibliothèque montage
30.           * */
31.          @Test
32.          void testFilmExtrait() {
33.                  Film filmOriginal = new FilmNormal();
34.                  Film filmAttendu = new FrameAttendu();
35.                  filmOriginal = Montage.encadrerFilm(filmOriginal);
36.
37.                  String s1 = toString(filmAttendu);
38.
39.                  String s2 = toString(filmOriginal);
40.
41.                  assertEquals(s1,s2);
42.          }
43.
44.          /*Test si le nouveau film est de même harteur et de largeur que le film
   attendu !
45.           */
46.          @Test
47.          void testHauteurLargeur() {
48.                  Film filmOriginal = new FilmNormal();
49.                  Film filmAttendu = new FrameAttendu();
50.                  filmOriginal = Montage.encadrerFilm(filmOriginal);
51.
52.                  assertEquals(filmAttendu.hauteur(),filmOriginal.hauteur());
53.                  assertEquals(filmAttendu.largeur(),filmOriginal.largeur());
```

```
54.         }
55.
56.         /*Test rebobiner
57.          * */
58.         @Test
59.         void testRebobiner() {
60.                 Film filmOriginal = new FilmNormal();
61.
62.                 //On projet le film pour aller jusqu'a la dernière image
63.
64.                 filmOriginal = Montage.encadrerFilm(filmOriginal);
65.                 String s1 = toString(filmOriginal);
66.
67.                 //On rebobine pour voir si on peut recomencer le film depuis le
   depuis
68.
69.                 filmOriginal.rembobiner();
70.                 String s2 = toString(filmOriginal);
71.
72.                 //On compare s1 : (Film avant d'être rebobiner) et s2 : (Film
   Aprés avoir rebobiner).
73.                 assertEquals(s1,s2);
74.         }
75.
76.
77.
78.  }
```

Test s’exécute avec sucées :

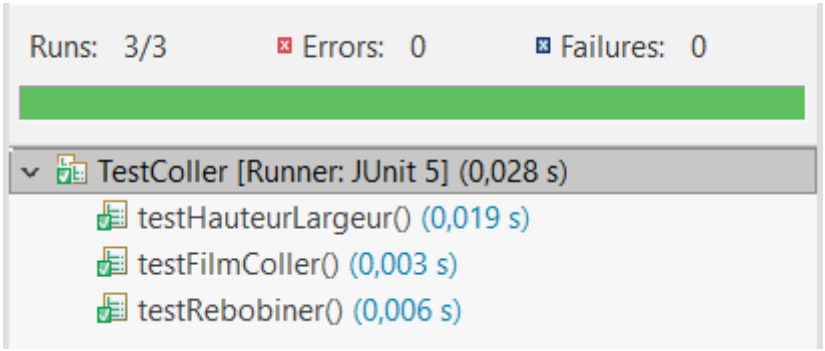


TestColler.java

```
1. package tests;
2.
3. import static org.junit.jupiter.api.Assertions.*;
4.
5. import org.junit.jupiter.api.Test;
6.
7. import film.Film;
8. import film.Films;
9. import opérationSurFilm.Montage;
10.  import tests.films.CollerAttendu;
11.  import tests.films.FilmNormal;
12.  import tests.films.FilmNormal2;
13.
14.  class TestColler {
15.
16.          /*Avoir l'écran de film sous forme d'un string
17.           * @param f film pour le quell on veut les images
18.           * @return sb string contnant les images/les écrans
19.           * */
20.          private static String toString(Film f){
21.          String sb = "";
22.          char[][] écran = new char[f.hauteur()][f.largeur()];
23.                  while(f.suivante(écran)){
24.             sb += Films.toString(écran) + "\n";
25.          }
26.          return sb;
27.      }
28.
29.          /*Test méthode encadrer film.
30.           *On compare les deux string: celle attendu et celle géérer par la
   bibliothèque montage
31.           * */
32.          @Test
33.          void testFilmColler() {
34.                  Film film1 = new FilmNormal();
35.                  Film film2 = new FilmNormal2();
36.
37.                  Film filmAttendu = new CollerAttendu(film1,film2);
38.
39.                  String s1 = toString(filmAttendu);
40.
41.                  film1.rembobiner();
42.                  film2.rembobiner();
43.
44.                  Film FilmOpération = Montage.collerDeuxFilms(film1, film2);
45.
46.                  String s2 = toString(FilmOpération);
47.
48.                  assertEquals(s1,s2);
49.          }
50.
51.          /*Test si le nouveau film est de même harteur et de largeur que le film
   attendu !*/
52.          @Test
53.          void testHauteurLargeur() {
54.                  Film film1 = new FilmNormal();
55.                  Film film2 = new FilmNormal2();
56.
57.                  Film filmAttendu = new CollerAttendu(film1,film2);
58.                  Film FilmOpération = Montage.collerDeuxFilms(film1, film2);
59.
```

```
60.                 assertEquals(filmAttendu.hauteur(),FilmOpération.hauteur());
61.                 assertEquals(filmAttendu.largeur(),FilmOpération.largeur());
62.         }
63.
64.         /*Test rebobiner
65.          * */
66.         @Test
67.         void testRebobiner() {
68.                 Film film1 = new FilmNormal();
69.                 Film film2 = new FilmNormal2();
70.
71.                 //On projet le film pour aller jusqu'a la dernière image
72.
73.
74.                 Film FilmOpération = Montage.collerDeuxFilms(film1, film2);
75.
76.                 String s1 = toString(FilmOpération);
77.
78.                 //On rebobine pour voir si on peut recomencer le film depuis le
   depuis
79.
80.                 FilmOpération.rembobiner();
81.                 String s2 = toString(FilmOpération);
82.
83.                 //On compare s1 : (Film avant d'être rebobiner) et s2 : (Film
   Apré avoir rebobiner).
84.                 assertEquals(s1,s2);
85.         }
86.  }
```

Test s'exécute avec sucées :

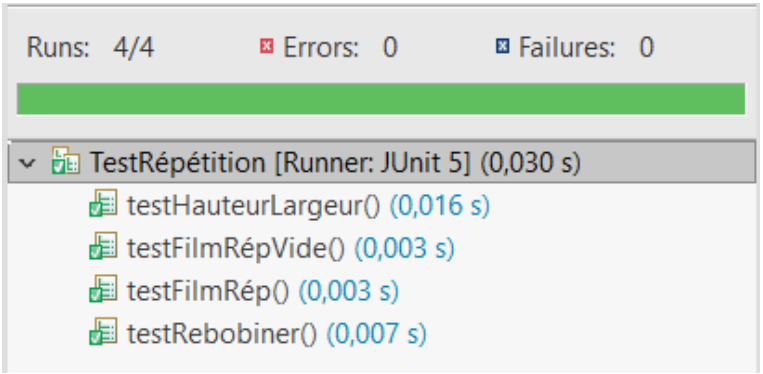


TestRépétition.java

```
1. package tests;
2.
3. import static org.junit.jupiter.api.Assertions.*;
4.
5. import org.junit.jupiter.api.Test;
6.
7. import film.Film;
8. import film.Films;
9. import opérationSurFilm.FilmVide;
10.  import opérationSurFilm.Montage;
11.  import tests.films.ExtraitAttendu;
12.  import tests.films.FilmNormal;
13.  import tests.films.RépétitionAttendu;
14.
15.  public class TestRépétition {
16.
17.          /*Avoir l'écran de film sous forme d'un string
18.           * @param f film pour le quell on veut les images
19.           * @return sb string contnant les images/les écrans
20.           * */
21.          private static String toString(Film f){
22.          String sb = "";
23.          char[][] écran = new char[f.hauteur()][f.largeur()];
24.                  while(f.suivante(écran)){
25.             sb += Films.toString(écran) + "\n";
26.          }
27.          return sb;
28.      }
29.
30.          /*Test méthode répéter de la classe/Biblithéque Montage
31.           *En comparant avec le film attendu.
32.           */
33.          @Test
34.          void testFilmRép() {
35.                  Film filmOriginal = new FilmNormal();
36.                  Film filmAttendu = new RépétitionAttendu();
37.                  filmOriginal = Montage.répéter(2, filmOriginal);
38.
39.                  String s1 = toString(filmAttendu);
40.
41.                  String s2 = toString(filmOriginal);
42.
43.                  assertEquals(s1,s2);
44.          }
45.
46.          /*Test si le nouveau film est de même harteur et de largeur que le film
   attendu !
47.           * */
48.          @Test
49.          void testHauteurLargeur() {
50.                  Film filmOriginal = new FilmNormal();
51.                  Film filmAttendu = new ExtraitAttendu();
52.                  filmOriginal = Montage.répéter(2, filmOriginal);;
53.
54.                  assertEquals(filmAttendu.hauteur(),filmOriginal.hauteur());
55.                  assertEquals(filmAttendu.largeur(),filmOriginal.largeur());
56.          }
57.
58.          /*Test rebobiner
59.           * */
60.          @Test
```

```
61.         void testRebobiner() {
62.                 Film filmOriginal = new FilmNormal();
63.
64.                 //On projet le film pour aller jusqu'a la dernière image
65.
66.                 filmOriginal = Montage.répéter(2, filmOriginal);;
67.                 String s1 = toString(filmOriginal);
68.
69.                 //On rebobine pour voir si on peut recomencer le film depuis le
   depuis
70.
71.                 filmOriginal.rembobiner();
72.                 String s2 = toString(filmOriginal);
73.
74.                 //On compare s1 : (Film avant d'être rebobiner) et s2 : (Film
   Apré avoir rebobiner).
75.                 assertEquals(s1,s2);
76.         }
77.
78.         /*Test en cas de répétition négative ou sup à nombre d'image de film
79.          *On returne donc une film vide!
80.          */
81.         @Test
82.         void testFilmRépVide() {
83.                 Film filmOriginal = new FilmNormal();
84.                 Film filmV = new FilmVide();
85.                 filmOriginal = Montage.répéter(-1, filmOriginal);
86.
87.                 String s1 = toString(filmV);
88.
89.                 String s2 = toString(filmOriginal);
90.
91.                 assertEquals(s1,s2);
92.         }
93.
94.  }
```

Test s'exécute avec sucées :

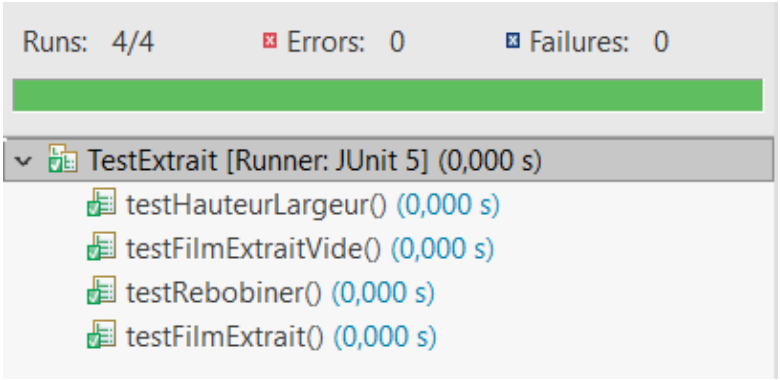


TestExtrait.java

```
1. package tests;
2.
3. import static org.junit.jupiter.api.Assertions.*;
4.
5. import org.junit.jupiter.api.Test;
6.
7. import film.Film;
8. import film.Films;
9. import opérationSurFilm.FilmVide;
10.  import opérationSurFilm.Montage;
11.  import tests.films.ExtraitAttendu;
12.  import tests.films.FilmNormal;
13.
14.  class TestExtrait {
15.
16.          /*Avoir l'écran de film sous forme d'un string
17.           * @param f film pour le quell on veut les images
18.           * @return sb string contnant les images/les écrans
19.           * */
20.          private static String toString(Film f){
21.          String sb = "";
22.          char[][] écran = new char[f.hauteur()][f.largeur()];
23.                  while(f.suivante(écran)){
24.             sb += Films.toString(écran) + "\n";
25.          }
26.          return sb;
27.      }
28.
29.          /*Test méthode film extrait.
30.           *On compare les deux string: celle attendu et celle géérer par la
   bibliothèque montage
31.           * */
32.          @Test
33.          void testFilmExtrait() {
34.                  Film filmOriginal = new FilmNormal();
35.                  Film filmAttendu = new ExtraitAttendu();
36.                  filmOriginal = Montage.extraitFilm(filmOriginal, 2, 4);
37.
38.                  String s1 = toString(filmAttendu);
39.
40.                  String s2 = toString(filmOriginal);
41.
42.                  assertEquals(s1,s2);
43.          }
44.
45.          /*Test si le nouveau film est de même harteur et de largeur que le film
   attendu !
46.           * */
47.          @Test
48.          void testHauteurLargeur() {
49.                  Film filmOriginal = new FilmNormal();
50.                  Film filmAttendu = new ExtraitAttendu();
51.                  filmOriginal = Montage.extraitFilm(filmOriginal, 2, 4);
52.
53.                  assertEquals(filmAttendu.hauteur(),filmOriginal.hauteur());
54.                  assertEquals(filmAttendu.largeur(),filmOriginal.largeur());
55.          }
56.
57.          /*Test rebobiner
58.           * */
59.          @Test
60.          void testRebobiner() {
```

```
61.                 Film filmOriginal = new FilmNormal();
62.
63.                 //On projet le film pour aller jusqu'a la dernière image
64.
65.                 filmOriginal = Montage.extraitFilm(filmOriginal, 2, 4);
66.                 String s1 = toString(filmOriginal);
67.
68.                 //On rebobine pour voir si on peut recomencer le film depuis le
    depuis
69.
70.                 filmOriginal.rembobiner();
71.                 String s2 = toString(filmOriginal);
72.
73.                 //On compare s1 : (Film avant d'être rebobiner) et s2 : (Film
    Apré avoir rebobiner).
74.                 assertEquals(s1,s2);
75.         }
76.
77.         /*Test film Vide.
78.          *Film vide est retouner par la méthose ssi les parmétres sasie dans la
    méthode ne sont pas correcte
79.          *Film vide possible ssi numéro de 1èr image a extraire est négative ou
    supérieure à dernière image a extraire.
80.          * */
81.         @Test
82.         void testFilmExtraitVide() {
83.                 Film filmOriginal = new FilmNormal();
84.                 Film filmV = new FilmVide();
85.                 filmOriginal = Montage.extraitFilm(filmOriginal, -2, 6);
86.
87.                 String s1 = toString(filmV);
88.
89.                 String s2 = toString(filmOriginal);
90.
91.                 assertEquals(s1,s2);
92.         }
93.
94.  }
```

Test s’exécute avec sucées :

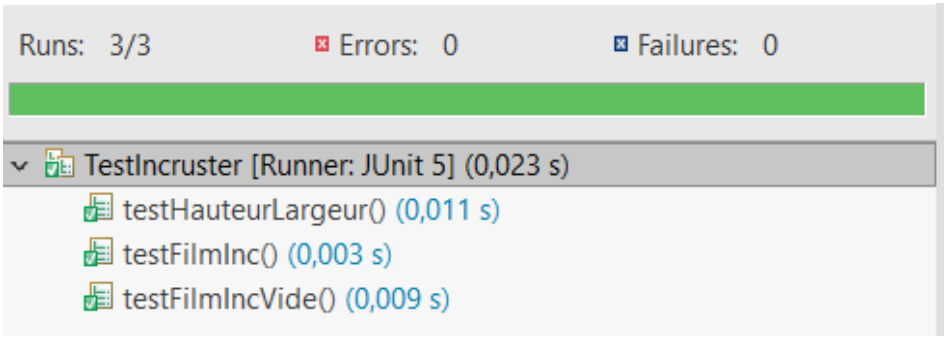


TestIncruster.java

```
1. package tests;
2.
3. import static org.junit.jupiter.api.Assertions.*;
4.
5. import org.junit.jupiter.api.Test;
6.
7. import film.Film;
8. import film.Films;
9. import opérationSurFilm.FilmVide;
10.  import opérationSurFilm.Montage;
11.  import tests.films.ExtraitAttendu;
12.  import tests.films.FilmNormal;
13.  import tests.films.FilmNormal2;
14.  import tests.films.IncrusterAttendu;
15.  import tests.films.RépétitionAttendu;
16.
17.  class TestIncruster {
18.
19.          /*Avoir l'écran de film sous forme d'un string
20.           * @param f film pour le quell on veut les images
21.           * @return sb string contnant les images/les écrans
22.           * */
23.          private static String toString(Film f){
24.          String sb = "";
25.          f.rembobiner();
26.          char[][] écran = new char[f.hauteur()][f.largeur()];
27.                  while(f.suivante(écran)){
28.             sb += Films.toString(écran) + "\n";
29.          }
30.          return sb;
31.      }
32.
33.          /*Test méthode répéter de la classe/Biblithéque Montage
34.           *En comparant avec le film attendu.
35.           */
36.          @Test
37.          void testFilmInc() {
38.                  Film film1 = new FilmNormal();
39.                  Film film2 = new FilmNormal2();
40.                  Film filmAttendu = new IncrusterAttendu(film1,film2,2,1);
41.
42.                  Film fInc = Montage.inscusterDeuxFilms(film1, film2, 2, 1);
43.
44.                  String s1 = toString(filmAttendu);
45.
46.                  String s2 = toString(fInc);
47.
48.                  assertEquals(s1,s2);
49.          }
50.
51.          /*Test si le nouveau film est de même harteur et de largeur que le film
   attendu !
52.           * */
53.          @Test
54.          void testHauteurLargeur() {
55.                  Film film1 = new FilmNormal();
56.                  Film film2 = new FilmNormal2();
57.                  Film filmAttendu = new IncrusterAttendu(film1,film2,2,2);
58.                  film1.rembobiner();
59.                  film2.rembobiner();
60.                  Film fInc = Montage.inscusterDeuxFilms(film1, film2, 2, 2);
61.
```

```
62.                 assertEquals(filmAttendu.hauteur(),fInc.hauteur());
63.                 assertEquals(filmAttendu.largeur(),fInc.largeur());
64.         }
65.
66.
67.         /*Test en cas de intervale non-valide
68.          *On returne donc une film vide!
69.          */
70.         @Test
71.         void testFilmIncVide() {
72.                 Film film1 = new FilmNormal();
73.                 Film film2 = new FilmNormal2();
74.                 Film filmOriginal = new IncrusterAttendu(film1,film2,2,2);
75.
76.                 Film filmV = new FilmVide();
77.
78.                 filmOriginal = Montage.inscusterDeuxFilms(film1, film2, -2, -5);
79.
80.                 String s1 = toString(filmV);
81.
82.                 String s2 = toString(filmOriginal);
83.
84.                 assertEquals(s1,s2);
85.         }
86.
87. }
```

Test s'exécute avec sucées :



IV Le code complet de projet

■ Montage.java

```
1. package opérationSurFilm;
2.
3. import film.Film;
4. import film.Films;
5.
6. public class Montage {
7.
8.   /**
9.    * Méthode permettant de modifier un film en le repetant autant de fois qu'on
   le souhaite.
10.          * @param nbFois : le nombre de repetition à faire
11.          * @param f : le film original à repeter
12.          * @return : Return soit un film vide si nbFois < 0 soit le film répété
13.          */
14.         public static Film répéter(int nbFois,Film f)  {
15.                 assert(f != null ) : "Le film entré est de réfèrences null";
16.                 f.rembobiner();
17.                 if(nbFois > 0) {
18.                         Répétition filmRép = new Répétition(f,nbFois);
19.                         return filmRép;
20.                 }
21.                 return (Film) new FilmVide(f);
22.         }
23.
24.
25.         /**
26.          * Méthode permettant d'optenir un extrait des images d'un film à
   partir de deux positions.
27.          * @param f : le film à extraire
28.          * @param numPImage : numero de la 1ere image à inclure
29.          * @param numDImaga : numero de la derniere image à inclure
30.          * @return : return soit film vide si  condition non vérifié ou
   l'extrait de film
31.          */
32.         public static Film extraitFilm(Film f,int numPImage, int numDImaga) {
33.                 assert(f != null ) : "Le film entré est de réfèrences null";
34.                 f.rembobiner();
35.                 int numPImg = (numPImage <= 0)? 0 : numPImage ;
36.                 int numDImg = numDImaga;
37.                 if((numPImg <= numDImg) && numDImg >= 0) {
38.                         Extrait ef = new Extrait(f, numPImg , numDImg );
39.                  return ef;
40.                 }
41.                 return (Film) new FilmVide(f);
42.     }
43.
44.
45.         /**
46.          * Méthode permettant de avoir un frame/cadre sur le simages de film
47.          * @param f : le film à encardrer
48.          * @return : Le nouveau film avec des images encadré
49.          */
50.         public static Film encadrerFilm(Film f) {
51.                 assert(f != null ) : "Le film entré est de réfèrences null";
52.                 f.rembobiner();
53.          Frame fE = new  Frame(f);
```

```
54.          return fE;
55.      }
56.
57.
58.          /**
59.           * Méthode permettant de coller les deux films l'un aprés l'autre
60.           * @param f1 : première film
61.           * @param f2 : deuxième film à coller à la première
62.           * @return : Returne un nouveau film qui contient f1 et f2
63.           */
64.          public static Film collerDeuxFilms(Film f1,Film f2) {
65.                  assert(f1 != null && f2 != null) : "Les films entrées ont des
   réfèrences null";
66.                  f1.rembobiner();
67.                  f2.rembobiner();
68.                  Coller fC = new  Coller(f1,f2);
69.          return fC;
70.      }
71.
72.
73.          /**
74.           * Méthode permettant de modifier un film en incrustant un autre film
   dans un autre film à une position donné.
75.           * @param f1 : le film recevrant l'incrustation
76.           * @param f2 : le film devant être inscruter
77.           * @param ligne : le numéro de la ligne où le film doit être incrusté
78.           * @param collone : le numéro de la colonne où le film doit être
   incrusté
79.           * @return : Returne soit film vide si les canditions ne sont pas
   valide soit le film incrusté.
80.           */
81.          public static Film inscusterDeuxFilms(Film f1,Film f2,int ligne,int
   collone) {
82.                  assert(f1 != null && f2 != null) : "Les films entrées ont des
   réfèrences null";
83.                  f1.rembobiner();
84.                  f2.rembobiner();
85.                  int numPImage = (ligne < 0)? 0 : ligne;
86.                  int numDImage = (collone < 0)? 0 : collone;
87.                  if(numPImage <= f2.hauteur() && numDImage <= f2.largeur() ) {
88.                          Incruster fI = new  Incruster(f1,f2,numPImage ,numDImage
   );
89.                   return fI;
90.                  }
91.                  return (Film) new FilmVide(f1);
92.      }
93.
94.          /**
95.           * Déterminer le nombre d'image d'un film (Visibilté que dan sle
   package)
96.           * @param f: film
97.           * @return i: nb images
98.           * */
99.          static int nbImages(Film f) {
100.                 int i = 0;
101.                 f.rembobiner();
102.                 char[][] écran = Films.getEcran(f);
103.                 while(f.suivante(écran)) {
104.                         i++;
105.                 }
106.                 f.rembobiner();
107.                 return i;
108.         }
109.
110.         // Cette classe n'a pas vocation à être instanciée car elle ne contient
   que
```

```
111.         // des méthodes de classe (i.e. statiques).
112.         private Montage() {
113.         }
114.
115. }
```

FilmVide.java

```
1. package opérationSurFilm;
2.
3. import film.Film;
4. /**
5.  * La classe FilmVide implémente l'interface Film.
6.  * Cette classe créer un film vide mais gardes l'hauteur et la largeur de
   film original.
7.  * @author KUMAR Aman - ROBALO RODRIGUES Flavio
8.  */
9. public class FilmVide implements Film {
10.         private Film f;
11.
12.         /**
13.          * Construteur de class FilmVide
14.          * @param fo : Film orignale
15.          */
16.         public FilmVide (Film fo) {
17.                 this.f = fo;
18.         }
19.
20.         public int hauteur() {
21.
22.          return f.hauteur();
23.      }
24.
25.      public int largeur() {
26.
27.          return f.largeur();
28.      }
29.
30.
31.      public boolean suivante(char[][] écran) {
32.          return false;
33.      }
34.
35.      public void rembobiner() {
36.      }
37. }
```

Frame.java

```java
package opérationSurFilm;

import film.Film;

/**
 * La classe Frame implémente l'interface Film.
 * Cette classe encadre un film avec des "*" sur les 4 coins de l'écran.
 * @author KUMAR Aman - ROBALO RODRIGUES Flavio
 */

 public class Frame implements Film{

	private Film f;
	private final int increaseScreen = 2;

	/**
	 * Constructeur de la class Frame
	 * @param fInt : Film à encadrer.
	 */
	public Frame(Film fInt) {
		this.f = fInt;
	}

	@Override
	public int hauteur() {
		return f.hauteur() + increaseScreen;
	}

	@Override
	public int largeur() {
		return f.largeur() + increaseScreen;
	}

	@Override
	public boolean suivante(char[][] ecranEncadré) {
		char[][] écran2 = new char[f.hauteur()][f.largeur()];
		if(this.f.suivante(écran2)) {
			// Encadrer l'écran
			for (int i = 0; i < hauteur(); i++) {
				ecranEncadré[i][0] = '*';
				ecranEncadré[i][(largeur()) - 1] = '*';
			}
			for (int j = 0; j < largeur() ; j++) {
				ecranEncadré[0][j] = '*';
				ecranEncadré[(hauteur()) - 1][j]= '*';
			}

			// copier les images de film sur l'écran
			for (int i = 1; i <= f.hauteur(); i++) {
				for (int j = 1; j <= f.largeur() ; j++) {
					ecranEncadré[i][j] = écran2[i-1][j-1];
				}
			}

			return true;
		}
		return false;
	}

	@Override
	public void rembobiner() {
		f.rembobiner();
```

```
63.         }
64.  }
```

■ Coller.java

```
1.   package opérationSurFilm;
2.
3. import film.Film;
4.
5. /**
6.  * La classe Coller implémente l'interface Film.
7.  * Cette classe colle deux films l'un suite à l'autre.
8.  * @author KUMAR Aman - ROBALO RODRIGUES Flavio
9.  */
10.  public class Coller implements Film {
11.
12.         // Les deux films à coller.
13.         private Film f1,f2;
14.         private int cmptEqual;
15.
16.         /**
17.          * Constructeur de la class Coller.
18.          * @param fInt1 : première film.
19.          * @param fInt2 : deuxième film à coller à la première.
20.          */
21.         public Coller(Film fInt1, Film fInt2) {
22.                 this.f1 = fInt1;
23.                 this.f2 = fInt2;
24.                 cmptEqual = 2;
25.         }
26.
27.         @Override
28.         public int hauteur() {
29.                 if((f1.hauteur() - f2.hauteur()) < 0 )
30.                         return f2.hauteur();
31.                 return f1.hauteur();
32.         }
33.
34.         @Override
35.         public int largeur() {
36.                 if((f1.largeur() - f2.largeur()) < 0 )
37.                         return f2.largeur();
38.                 return f1.largeur();
39.         }
40.
41.         @Override
42.         public boolean suivante(char[][] écranFilm) {
43.                 if (f1.equals(f2)) {
44.                         if( this.cmptEqual != 0) {
45.                                 if (!f1.suivante(écranFilm)) {
46.                                         this.rembobiner();
47.                                         this.cmptEqual --;
48.                                         return f1.suivante(écranFilm);
49.                                 }
50.                                 else {
51.                                         return true;
52.                                 }
53.                         }
54.                         else {
55.                                 return false;
56.                         }
57.                 }
58.                 else {
```

```
59.                         if(!f1.suivante(écranFilm)) {
60.                                 //Si il n'y a plus d'image dans f1 on écrit sur
   l'écran avec les images de f2
61.                                 if(!f2.suivante(écranFilm)) {
62.                                         return false;
63.                                 }
64.                         }
65.                 }
66.
67.                 return true;
68.
69.         }
70.
71.         @Override
72.         public void rembobiner() {
73.                 f1.rembobiner();
74.                 f2.rembobiner();
75.         }
76.
77.  }
```

Répétition.java

```
1.	package opérationSurFilm;
2.
3. import film.Film;
4. /**
5.  * La classe Répétition implémente l'interface Film.
6.  * Cette classe modifie un film en le repetant autant de fois qu'on le souhaite.
7.  * @author KUMAR Aman - ROBALO RODRIGUES Flavio
8.  */
9. public class Répétition implements Film  {
10.
11.		/* nbRep : Le nombre de répétition du film */
12.		private int nbRep;
13.
14.		/* cptRep : Compte le nombre de répétition effectué */
15.		private int cmptRép;
16.
17.		/* Film "originl" à repeter */
18.		private Film f;
19.
20.		/**
21.		 * Constructeur
22.		 * @param fInt : le film à repeter
23.		 * @param nbfois : le nombre de repetition à faire
24.		 */
25.		public Répétition(Film fInt,int nbfois) {
26.			this.nbRep = nbfois;
27.			this.f = fInt;
28.			this.cmptRép = 0;
29.		}
30.
31.		@Override
32.		public int hauteur() {
33.			return f.hauteur();
34.		}
35.
36.		@Override
37.		public int largeur() {
38.			return f.largeur();
39.		}
40.
41.		@Override
42.		public boolean suivante(char[][] écran) {
43.
44.			if(!f.suivante(écran)) {
45.				++this.cmptRép;
46.				// Si le nb de répétition est atteint --> return false.
47.				if(this.cmptRép == this.nbRep)
48.					return false;
49.				// Sinon on rebobine le film.
50.				f.rembobiner();
51.				return f.suivante(écran);
52.			}
53.			return true;
54.		}
55.
56.		@Override
57.		public void rembobiner() {
58.			f.rembobiner();
59.			this.cmptRép = 0;
60.		}
61.
62.  }
```

Extrait.java

```java
package opérationSurFilm;

import film.Film;
/**
 * La classe Extrait implémente l'interface Film. <br>
 * Cette classe permet d'extraire une partie d'un film désigné par les numéros de la
 * premieres et dernieres images à inclure du film. <br>
 * La 1ere image d'un film porte le numéro 0. <br>
 *       _ Si le num de la 1ere images à extraire est negatif, l'extrait démarre
 *             alors au début du film original soit à la position 0.
 *       _ Si le num de la derniere images à extraire est superireur au nombre d'image
 *             total du film, l'extrait va alors s'arreter à la fin du film original.
 * @author KUMAR Aman - ROBALO RODRIGUES Flavio
 */

public class Extrait implements Film {

        // -1 car la position peut commencer par 0
        private int numFrame = -1;
        /**
         * Numero du 1er image à inclure
         */
        private int numDebut;
        /**
         * Numero du dernier image à inclure
         */
        private int numFin;

        // Films à extraire
        private Film filmOriginal;

        /**
         * Constructeur de la class ModifExtraire
         * @param film : le film à extraire
         * @param numD : numero de la 1ere image à inclure
         * @param numF : numero de la derniere image à inclure
         */
        public Extrait(Film film, int numD, int numF) {
                filmOriginal = film;
                this.numDebut = numD;
                this.numFin = numF;
        }

        @Override
        public boolean suivante(char[][] écran) {
                do {
                        ++numFrame;
                        if(!filmOriginal.suivante(écran))
                                return false;
                } while(numFrame < numDebut);
                return (numFrame > numFin)? false : true;
        }

        @Override
        public void rembobiner() {
                filmOriginal.rembobiner();
                this.numFrame = -1;
        }

```

```
60.          @Override
61.          public int hauteur() {
62.                  return this.filmOriginal.hauteur();
63.          }
64.
65.          @Override
66.          public int largeur() {
67.                  return this.filmOriginal.largeur();
68.          }
69.
70.  }
```

Incruster.java

```
1. package opérationSurFilm;
2.
3. import film.Film;
4.
5. /**
6.  * La classe Incruster implémente l'interface Film.
7.  * Cette classe modifie un film en incrustant un autre film dans ce film à une
   position donnée.
8.  * @author KUMAR Aman - ROBALO RODRIGUES Flavio
9.  */
10.
11.  public class Incruster implements Film {
12.
13.         private Film f1;
14.         private Film f2;
15.         private int ligne,col;
16.
17.         int hauteurEcranIncrustrer ;
18.         int largeurEcranIncrustrer ;
19.
20.         /**
21.          * Constructeur de la class Incruster.
22.          * @param fInt1 : le film devant être inscruter
23.          * @param fInt2 : le film recevrant l'incrustation
24.          * @param ligne : le numéro de la ligne où le film doit être incrusté
25.          * @param col   : le numéro de la colonne où le film doit être incrusté
26.          */
27.         public Incruster(Film fInt1, Film fInt2, int ligne, int col) {
28.                 this.f1 = fInt1;
29.                 this.f2 = fInt2;
30.                 this.ligne = ligne;
31.                 this.col = col;
32.
33.                 // Déterminer les position en fonction d'hauteur et largeur des
   deux films.
34.
35.                 if(f1.hauteur() - (f2.hauteur() - ligne)  > 0) {
36.                         hauteurEcranIncrustrer =  f1.hauteur() - (f1.hauteur() -
   (f2.hauteur() - ligne));
37.                 }
38.                 else {
39.                         hauteurEcranIncrustrer = f1.hauteur() ;
40.                 }
41.
42.                 if(f1.largeur() - (f2.largeur() - col) > 0) {
43.                         largeurEcranIncrustrer =  f1.largeur() - (f1.largeur() -
   (f2.largeur() - col)) ;
44.                 }
45.                 else {
46.                         largeurEcranIncrustrer = f1.largeur() ;
47.                 }
48.
49.         }
50.
51.         @Override
52.         public int hauteur() {
53.                 return f2.hauteur();
54.         }
55.
56.         @Override
57.         public int largeur() {
58.                 return f2.largeur();
```

```
59.         }
60.
61.         @Override
62.         public boolean suivante(char[][] écranFinal) {
63.
64.                 char[][] écran1 = new char[f1.hauteur()][f1.largeur()];
65.
66.                 if(f2.suivante(écranFinal) ) {
67.                         if(f1.suivante(écran1)) {
68.                                 int i = this.ligne;
69.                                 int j = this.col;
70.
71.                                 for(int k = 0; k < hauteurEcranIncrustrer; k++) {
72.
73.                                         for(int l = 0; l < largeurEcranIncrustrer
   ;l++) {
74.                                                 écranFinal[i][j++] =
   écran1[k][l];
75.                                         }
76.                                         i++;
77.                                         j = this.col;
78.                                 }
79.                         }
80.                         return true;
81.                 }
82.
83.                 return false;
84.         }
85.
86.         @Override
87.         public void rembobiner() {
88.                 f2.rembobiner();
89.                 f1.rembobiner();
90.         }
91.
92.  }
```

V BILAN

Notre deuxième projet en langage JAVA, nous a permis du travailler dans un autre aspect de la programmation à savoir le polymorphisme. De plus, le cahier des charges qui nous a été demandé est respecté. Notre application possède les fonctionnalités minimales demandées. Ce projet, nous a permis d'appliquer et d'approfondir nos connaissances en programmation et surtout en langage JAVA, malgré quelques difficultés rencontrées. Nous avons pu aussi comprendre l'utilité des tests et à quels points ils sont importants dans un projet.

Les difficultés rencontrées :

La principale difficulté que nous avons rencontrée, ce fut au début du projet. En effet, nous avons eu un peu de mal à bien structurer notre projet en différentes classes et donc de bien choisir quelles classes /objet était vraiment utile et efficace. Est-ce que c'est mieux d'utiliser une classe abstraite pour la bibliothèque ou grader l'interface Film.

Nous avons, peut ainsi comprendre qu'à quel point la programmation objet est très efficace puisque à travers les interfaces et les méthodes bien structurées on peut créer des programmes très efficaces et bien optimisés. La deuxième difficulté était d'imaginer et coder les algorithmes afin de respecter différents fonctionnements de la bibliothèque. Certains algorithmes utilisés ont une complexité assez élevée.

✓ Ce qui est réussi :

Le cahier des charges est respecté. Notre bibliothèque fonctionne bien avec des classes bien définies et ordonnées. Ensuite, nous avons des méthodes dans la portée au niveau du paquetage est bien défini. La classe Montage java permet de proposer juste les méthodes utiles aux utilisateurs et qui assez simple à prendre aux mains.

Ce qui peut être amélioré :

Ce qu'on peut améliorer ce sont principalement les classes qui permettent de gérer des nouveaux films en fonction des opérations demandées. En effet, on peut constater que certaines méthodes ne sont pas modifiées dans ces classes, mais sont présentes. Il sera plus efficace de juste

Il sera plus efficace de juste modifier les méthodes qui on veut modifier. Pour cela on aurait pu passer par une classe abstraite qui implémente l'interface Film et donc coder les méthodes qui sont identiques aux tous les films dans cette classe.