BPO

MONTAGE DE FILMS CARACTERES

MILLO CHELSEY GRP 109

YE FANGUYEN LISA GRP 109

Table des matières

INTRODUCTION

Présentation du projet

Notre projet consiste à redonner vie à une bibliothèque afin que celle-ci puisse donner naissance à divers montages vidéo.

Cette bibliothèque permettra à celui qui la manipulera de mettre en mouvement / de monter un film à partir de films existants respectant le format TXT.

Spécification du projet

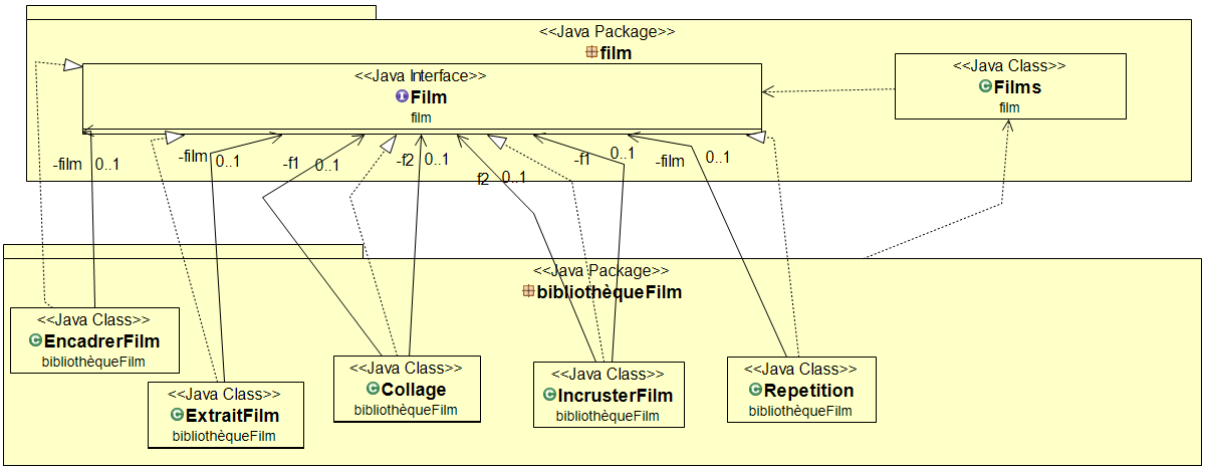
Cette dernière fera mettre en marche plusieurs options de montages :

- répétition d'un film n fois
- récupérer un extrait particulier d'un film
- encadrer un film d'étoiles
- coller deux films l'un à la suite de l'autre
- incruster un film dans un autre

Sans oublier ces méthodes qui pourront satisfaire les désirs de l'utilisateur :

- hauteur(), largeur() pour la définition des images
- suivante() pour obtenir les images du film
- rembobiner () pour récupérer toutes les images du film
- projeter() pour visualiser un film (fourni par prof)
- sauvegarder() pour comme son nom l'indique enregistrer un film (fourni par prof)

DIAGRAMME UML



CODE JUNITS

Nous avons fait de simples tests unitaires pour chaque classe que nous avons créés afin de vérifier le fonctionnement des codes, mais ils ne suffisent pas pour vérifier tout le fonctionnement total de ce que nous avons coder. Finalement c'est avec les tests que le professeur nous fournit nous avons à réussi de retrouve les problèmes de nos codes mais nous n'avons pas réussi à résoudre tous les problèmes qui nous conduirait à 0 échec. Nous avons réussi à faire exécuter notre code avec les tests basic et extra mais pas celui de composition.

Test fait par nous-mêmes

```
package tests;

/**
 * @author : MILLO Chelsey, YE Fangyuan Lisa
 * @date : 24/05/2020
 *
 * Enumeration des etats de l'affichage des images : les états pour faire tests
*/
public enum EtatSuivant {
	OUI,
	NON
}
```

Class EncadrerFilmTest

```
package bibliothèqueFilmTests;

import static org.junit.jupiter.api.Assertions.*;
/**
 * @author : MILLO Chelsey, YE Fangyuan Lisa
 * @date : 24/05/2020
 *
 * Tester le classe EncadrerFilm
*/

import org.junit.jupiter.api.Test;

import bibliothèqueFilm.EncadrerFilm;
import exemple.LaDiagonaleDuFou;
import film.Film;
import film.Films;



class EncadrerFilmTest {
	private char[][] écran; // l'écran où afficher l'image

	//creer un film f1 qui a 20 images dont l'hauteur et la largeur sont 10
	Film f1 = new LaDiagonaleDuFou();
	// créer un film f3 qui a encadrer le film f1 par symbole *
	Film f3 = new EncadrerFilm(f1);

	/**
	 * Obtenir EtatSuivant d'un film
	 *
	 * @param film : le film à verifie
```

```
 * @return EtatSuivant.OUI si l'image suivante a été affichée sur l'écran
 *                   et EtatSuivant.NON si le film est terminé
 */
private EtatSuivant getEtatSuivant(Film f) {
        écran = Films.getEcran(f);
        if(f.suivante(écran))
                return EtatSuivant.OUI;
        else
                return EtatSuivant.NON;
}

/**
 * Tester que les films f1 et f3 sont bien crées
 */
@Test
void testEncadrerFilm() {
        assertNotNull(f1);
        assertNotNull(f3);
}

/**
 * Tester que le film f3 a deux case en plus de l'hauteur pour mettre les *
 */
@Test
void testHauteur() {
        assertEquals (f1.hauteur() + 2,f3.hauteur());
}

/**
 * Tester que le film f3 a deux case en plus de la largeur pour mettre les *
 */
```

```
	@Test
	void testLargeur() {
		assertEquals (f1.largeur() + 2,f3.largeur());
	}

	/**
	 * Tester que le film encadre f3 a le même nombre des images que le film f1
	 */
	@Test
	void testSuivante() {
		// les images de numéro 0 à 19 retourne vraie
		for(int i = 0; i < 20; ++i) {
		assertEquals(EtatSuivant.OUI,getEtatSuivant(f3));
		}
		// la dernier image retourne faute
		assertEquals(EtatSuivant.NON,getEtatSuivant(f3));
	}

	//@Test
	void testRembobiner() {
	}

}
```

Class CollageTest

```
package bibliothèqueFilmTests;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

import bibliothèqueFilm.Collage;
import exemple.LaDiagonaleDuFou;
import film.Film;
import film.Films;

/**
 * @author : MILLO Chelsey, YE Fangyuan Lisa
 * @date : 24/05/2020
 *
 * Tester le classe Collage
*/

class CollageTest {
	private char[][] écran; // l'écran où afficher l'image

	//creer un film f1 qui a 20 images dont l'hauteur et la largeur sont 10
	Film f1 = new LaDiagonaleDuFou();
	// creer un autre film f2 qui a 30 images dont l'hauteur et la largeur sont
20
	Film f2 = new UnAutreFilm();
	// créer un film f3 qui a collé les films f1 et f2
	Film f3 = new Collage(f1,f2);

	/**
	 * Obtenir EtatSuivant d'un film
	 *
	 * @param film : le film à verifie
	 * @return EtatSuivant.OUI si l'image suivante a été affichée sur l'écran
	 *           et EtatSuivant.NON si le film est terminé
	 */
	private EtatSuivant getEtatSuivant(Film f) {
		écran = Films.getEcran(f);
		if(f.suivante(écran))
			return EtatSuivant.OUI;
		else
			return EtatSuivant.NON;
	}

	/**
	 * Tester que les films f1, f2 et f3 sont bien crées
	 */
	@Test
	void testCollage() {
		assertNotNull(f1);
		assertNotNull(f2);
		assertNotNull(f3);
	}

	/**
	 * Tester que le film collé a une hauteur suffisamment grande
	 */
	@Test
```

```
	void testHauteur() {
		// verifier que l'hauteur de film collée f3 est 20
		assertEquals (f2.hauteur(),f3.hauteur());
	}

	/**
	 * Tester que le film collé a une largeur suffisamment grande
	 */
	@Test
	void testLargeur() {
		// verifier que la largeur de film collée f3 est 20
		assertEquals (f2.largeur(),f3.largeur());
	}

	/**
	 * Tester que le film collé a bien 50 images
	 */
	@Test
	void testSuivante() {
		// les images de numéro 0 à 49 retourne vraie
		for(int i = 0; i < 50; ++i) {
		assertEquals(EtatSuivant.OUI,getEtatSuivant(f3));
		}
		// la dernier image retourne faute
		assertEquals(EtatSuivant.NON,getEtatSuivant(f3));
	}

	//@Test
	void testRembobiner() {
		fail("Not yet implemented");
	}

}
```

Class ExtraitFilmTest

```
package bibliothèqueFilmTests;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

import bibliothèqueFilm.ExtraitFilm;
import exemple.LaDiagonaleDuFou;
import film.Film;
import film.Films;
/**
 * @author : MILLO Chelsey, YE Fangyuan Lisa
 * @date : 24/05/2020
 *
 * Tester le classe ExtraitFilm
*/

class ExtraitFilmTest {
	private char[][] écran; // l'écran où afficher l'image

	//creer un film f1 qui a 20 images dont l'hauteur et la largeur sont 10
	Film f1 = new LaDiagonaleDuFou();
	// créer un film f3 qui est un extrait de film f1 (de 8eme à 11eme images)
	Film f3 = new ExtraitFilm(f1,8,11);

	/**
	 * Obtenir EtatSuivant d'un film
	 *
	 * @param film : le film à verifie
	 * @return EtatSuivant.OUI si l'image suivante a été affichée sur l'écran
	 *           et EtatSuivant.NON si le film est terminé
	 */
	private EtatSuivant getEtatSuivant(Film f) {
		écran = Films.getEcran(f);
		if(f.suivante(écran))
			return EtatSuivant.OUI;
		else
			return EtatSuivant.NON;
	}

	/**
	 * Tester que les films f1, et f3 sont bien crées
	 */
	@Test
	void testExtraitFilm() {
		assertNotNull(f1);
		assertNotNull(f3);
	}

	/**
	 * Tester que l'hauteur de film f3 egale celle de film f1
	 */
	@Test
	void testHauteur() {
		assertEquals (f3.hauteur(),10);
	}
```

```
	/**
	 * Tester que la largeur de film f3 egale celle de film f1
	 */
	@Test
	void testLargeur() {
		assertEquals (f3.largeur(),10);
	}

	/**
	 * Tester que le film f3 a bien 4 images
	 */
	@Test
	void testSuivante() {
		// les images de numéro 7 à 10 retourne vraie
		for(int i = 0; i < 4; ++i) {
		assertEquals(EtatSuivant.OUI,getEtatSuivant(f3));
		}
		// la dernier image retourne faute
		assertEquals(EtatSuivant.NON,getEtatSuivant(f3));
	}

	//@Test
	void testRembobiner() {
		fail("Not yet implemented");
	}

}
```

Class IncrusterFilmTest

```
package bibliothèqueFilmTests;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

import bibliothèqueFilm.IncrusterFilm;
import exemple.LaDiagonaleDuFou;
import film.Film;
import film.Films;


/**
 * @author : MILLO Chelsey, YE Fangyuan Lisa
 * @date : 24/05/2020
 *
 * Tester le classe  IncrusterFilm
*/

class IncrusterFilmTest {
	private char[][] écran; // l'écran où afficher l'image

	//creer un film f1 qui a 20 images dont l'hauteur et la largeur sont 10
	Film f1 = new LaDiagonaleDuFou();
	// creer un autre film f2 qui a 30 images dont l'hauteur et la largeur sont
20
	Film f2 = new UnAutreFilm();
	// créer un film f3 qui incruster le film f1 dans le film f2 dans le
coordonnée (3,2)
	Film f3 = new IncrusterFilm(f1,f2,3,2);


	/**
	 * Obtenir EtatSuivant d'un film
	 *
	 * @param film : le film à verifie
	 * @return EtatSuivant.OUI si l'image suivante a été affichée sur l'écran
	 *           et EtatSuivant.NON si le film est terminé
	 */
	private EtatSuivant getEtatSuivant(Film f) {
		écran = Films.getEcran(f);
		if(f.suivante(écran))
			return EtatSuivant.OUI;
		else
			return EtatSuivant.NON;
	}

	/**
	 * Tester que les films f1, f2, f3 et f4 sont bien crées
	 */
	@Test
	void testIncrusterFilm() {
		assertNotNull(f1);
		assertNotNull(f2);
		assertNotNull(f3);
		//assertNotNull(f4);
	}
```

```
	/**
	 * Tester que le film f3 a une hauteur egale celle de la film f2
	 */
	@Test
	void testHauteur() {
		assertEquals(f2.hauteur(),f3.hauteur());
		//assertEquals(f1.hauteur(),f4.hauteur());
	}

	/**
	 * Tester que le film f3 a une largeur egale celle de la film f2
	 */
	@Test
	void testLargeur() {
		assertEquals(f2.largeur(),f3.largeur());
		//assertEquals(f1.largeur(),f4.largeur());
	}

	/**
	 * Tester que le film f3 a même nombre de images que celle de la film f2
	 */
	@Test
	void testSuivante() {
		// les images de numéro 0 à 29 retourne vraie
		for(int i = 0; i < 30; ++i) {
		assertEquals(EtatSuivant.OUI,getEtatSuivant(f3));
		}
		// la dernier image retourne faute
		assertEquals(EtatSuivant.NON,getEtatSuivant(f3));


	}

	/**
	 * Tester que le film f4 a même nombre de images que celle de la film f1
	 */
	@Test
	void testSuivante2() {
		Film f4 = new IncrusterFilm(f2,f1,3,2);
		// les images de numéro 0 à 29 retourne vraie
		for(int j = 0; j < 20; ++j) {
		assertEquals(EtatSuivant.OUI,getEtatSuivant(f4));
		}
		// la dernier image retourne faute
		assertEquals(EtatSuivant.NON,getEtatSuivant(f4));
	}

	//@Test
	void testRembobiner() {
		fail("Not yet implemented");
	}

}
```

Class RepetitionTest

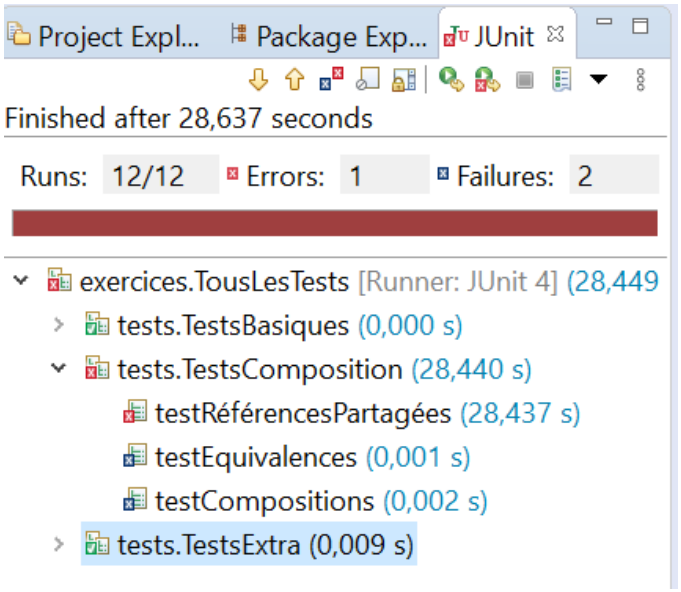
```
package bibliothèqueFilmTests;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

import bibliothèqueFilm.Repetition;
import exemple.LaDiagonaleDuFou;
import film.Film;
import film.Films;



/**
 * @author : MILLO Chelsey, YE Fangyuan Lisa
 * @date : 24/05/2020
 *
 * Tester le classe Repetition
*/

class RepetitionTest {
	private char[][] écran; // l'écran où afficher l'image

	//creer un film f1 qui a 20 images dont l'hauteur et la largeur sont 10
	Film f1 = new LaDiagonaleDuFou();
	// créer un film f3 qui est un film repeter 3 fois le film f1
	Film f3 = new Repetition(f1,3);

	/**
	 * Obtenir EtatSuivant d'un film
	 *
	 * @param film : le film à verifie
	 * @return EtatSuivant.OUI si l'image suivante a été affichée sur l'écran
	 *           et EtatSuivant.NON si le film est terminé
	 */
	private EtatSuivant getEtatSuivant(Film f) {
		écran = Films.getEcran(f);
		if(f.suivante(écran))
			return EtatSuivant.OUI;
		else
			return EtatSuivant.NON;
	}

	/**
	 * Tester que les films f1 et f3 sont bien crées
	 */
	@Test
	void testRepetition() {
		assertNotNull(f1);
		assertNotNull(f3);
	}

	/**
	 * Tester que le film f3 a la même hauteur que le film f1
	 */
	@Test
	void testHauteur() {
		assertEquals (f1.hauteur(),f3.hauteur());
```

```
	}

	/**
	 * Tester que le film f3 a la même largeur que le film f1
	 */
	@Test
	void testLargeur() {
		assertEquals (f1.largeur(),f3.largeur());
	}

	/**
	 * Tester que le film f3 a bien 60 images
	 */
	@Test
	void testSuivante() {
		// les images de numéro 0 à 59 retourne vraie
		for(int i = 0; i < (3*20); ++i) {
		assertEquals(EtatSuivant.OUI,getEtatSuivant(f3));
		}
		// la dernier image retourne faute
		assertEquals(EtatSuivant.NON,getEtatSuivant(f3));
	}

	/**
	 * Tester que repeter le film une nombre de fois nulle retourne false
	 */
	@Test
	void testSuivante2() {
		Film f4 = new Repetition(f1,0);
		assertEquals(EtatSuivant.NON,getEtatSuivant(f4));
	}

	/**
	 * Tester que repeter le film une nombre de fois negative retourne false
	 */
	@Test
	void testSuivante3() {
		Film f5 = new Repetition(f1,-2);
		assertEquals(EtatSuivant.NON,getEtatSuivant(f5));
	}

	//@Test
	void testRembobiner() {
		fail("Not yet implemented");
	}

}
```

Class UnAutreFilm

```
package bibliothèqueFilmTests;

import film.Film;

/**
* Classe d’exemple d’un Film
*/

public class UnAutreFilm implements Film{
	private static int num = 0;
	private static final int NB_IMAGES = 30;

	@Override
	public int hauteur() {
		return 20;
	}

	@Override
	public int largeur() {
		return hauteur(); // ce sera un carré
	}

	@Override
	public boolean suivante(char[][] écran) {
		if (num == NB_IMAGES)
			return false;
		for (int i = 0; i < hauteur(); ++i) {
			for (int j = 0; j < largeur(); ++j ) {
				écran[i][j] = 'b';
			}
		}
		++num;
		return true;
	}

	@Override
	public void rembobiner() {
		num = 0;
	}
}
```

Test fait par prof



CODE JAVA DU PROJET

Class Collage

```
package bibliothèqueFilm;


import film.Film;
/**
 * @author : MILLO Chelsey, YE Fangyuan Lisa
 * @date : 23/05/2020
*/


/**
 * Une classe implemente l'Interface Film, il redefinir l'ensemble de methode de
 * l'Interface Film pour coller deux films, l'un à la suite de l'autre afin de créer
 * un nouveau film, les ecran de ce film doit suiffisamment grand pour afficher les
 * images.
 */
public class Collage implements Film {
	private Film f1,f2;

	public Collage(Film f1, Film f2) {
		this.f1 = f1;
		this.f2 = f2;
	}

	/**
	 * Obtenir une hauteur plus long entre les hauteurs de deux films
	 *
	 * @return Hauteur minimale de l'écran pour pouvoir afficher les images de
	 *	ce film.
	 */
```

```
@Override
public int hauteur() {
	if (f1.hauteur() >= f2.hauteur())
		return f1.hauteur();
	else
		return f2.hauteur();
}

/**
 * Obtenir une largeur plus long entre les largeurs de deux films
 *
 * @return largeur minimale de l'écran pour pouvoir afficher les images de
 *	ce film.
 */
@Override
public int largeur() {
	if(f1.largeur() >= f2.largeur())
		return f1.largeur();
	else
		return f2.largeur();
}

/**
 * Obtenir l'image suivante (s'il y en a une).
 *
 * @param écran : L'écran où afficher l'image
 * @return vrai Si l'image suivante a été affichée sur l'écran et faux si le
 *	film est terminé
 */
@Override
public boolean suivante(char[][] écran) {
```

```
            if(f1.suivante(écran)) { // Obtenir d'abord les images de premier film
                return true;
            }
            else {
                if(f2.suivante(écran)) { // Obtenir ensuite les images de deuxieme film
                    return true;
                }
                else {
                    rembobiner(); // il fait un boucle en testes Compositions !!!!
                    return false;
                }
            }
        }

        /**
         * Rembobine le film en permettant de rejouer le film dans sa totalité (via
         * des appels successifs à la méthode suivante()).
         */
        @Override
        public void rembobiner() {
            f1.rembobiner();
            f2.rembobiner();
        }

}
```

Class EncadrerFilm

```
package bibliothèqueFilm;


import film.Film;
/**
 * @author : MILLO Chelsey, YE Fangyuan Lisa
 * @date : 24/05/2020
*/


/**
 * Une classe implemente l'Interface Film, il redefinir l'ensemble de methode de
 * l'Interface Film pour encadrer un film (par quatre ligne d'étoiles – '*' – au
 * bord de l'écran).
 */
public class EncadrerFilm implements Film {
	private Film film;
	private static final int NB_HLAJOUTER = 2; // nombre de lignes et collones supplementaire
                                                                                    // pour
ajouter les * en bord d'écran
	private static int num = 0;
	private static char sympole = '*'; //sympole pour encadrer le film.


	public EncadrerFilm(Film film) {
		this.film = film;
	}


	/**
	 * Obtenir une hauteur qui a ajouter les espaces pour le sympole d'encradrement.
	 *
	 * @return Hauteur minimale de l'écran pour pouvoir afficher les images de
	 *	ce film.
```

```
	 */
	@Override
	public int hauteur() {
		return film.hauteur() + NB_HLAJOUTER;
	}

	/**
	 * Obtenir une largeur qui a ajouter les espaces pour le sympole d'encradrement.
	 *
	 * @return largeur minimale de l'écran pour pouvoir afficher les images de
	 *         ce film.
	 */
	@Override
	public int largeur() {
		return film.largeur() + NB_HLAJOUTER;
	}

	/**
	 * Obtenir l'image suivante (s'il y en a une).
	 *
	 * @param écran : L'écran où afficher l'image
	 * @return vrai Si l'image suivante a été affichée sur l'écran et faux si le
	 *         film est terminé
	 */
	@Override
	public boolean suivante(char[][] écran) {
		char[][] écran2 = new char[film.hauteur()][film.largeur()]; // obtenir l'écran vide de
1er film
		if(film.suivante(écran2)) {
			for(int i = 0; i < largeur(); ++i) { // obtenir les symbole d'encadrement au bord
de l'écran
				écran[num][i] = sympole;
```

```
                    écran[hauteur() - 1][i] = sympole;
                }
                for (int i = 1; i < hauteur(); ++i) {
                    écran[i][num]  = sympole;
                    écran[i][largeur() - 1] = sympole;
                }
                for(int i = 1; i < hauteur()-1; ++i) {
                    for(int j =1; j < largeur()-1; ++j) { // obtenir les écrans de 1er film
tourne autour de symbole '*'
                        écran[i][j] = écran2[i-1][j-1];
                    }
                }
                return true;
            }
            film.rembobiner(); // rembobiner le film
            return false;
        }

        /**
         * Rembobine le film en permettant de rejouer le film dans sa totalité (via
         * des appels successifs à la méthode suivante()).
         */
        @Override
        public void rembobiner() {
            num = 0;
        }
}
```

Class ExtraitFilm

```
package bibliothèqueFilm;


import java.util.Arrays;


import film.Film;
/**
 * @author : MILLO Chelsey, YE Fangyuan Lisa
 * @date : 24/05/2020
*/


/**
 * Une classe implemente l'Interface Film, il redefinir l'ensemble de methode de
 * l'Interface Film pour obtenir un extrait d'un film. L'extrait est désigné
 * par les numéros de la première et de la dernière images à inclure et la
 * première image du film porte le numéro 0.
 */
public class ExtraitFilm implements Film{
	private Film film;
	private int première,dernière;
	private static int num = 0;



	public ExtraitFilm(Film film, int première, int dernière) {
		this.film = film;
		this.première = première;
		this.dernière = dernière;
	}


	/**
	 * Indique la hauteur des images de ce film (en nombre de caractères).
```

```
 *
 * @return Hauteur minimale de l'écran pour pouvoir afficher les images de
 *         ce film.
 */
@Override
public int hauteur() {
            return film.hauteur();
}

/**
 * Indique la largeur des images de ce film (en nombre de caractères).
 *
 * @return largeur minimale de l'écran pour pouvoir afficher les images de
 *         ce film.
 */
@Override
public int largeur() {
            return film.largeur();
}

/**
 * Obtenir l'image suivante (s'il y en a une).
 *
 * @param écran :  L'écran où afficher l'image
 * @return vrai Si l'image suivante a été affichée sur l'écran et faux si le
 *         film est terminé
 */
@Override
public boolean suivante(char[][] écran) {
            if(première < 0)première = 0;
            for(;num < première;++num) { // obtenir l'écran avant de la premier images désire
```

```
                    film.suivante(écran);
                    for (char[] ligne : écran) //effacer l'écran.
                        Arrays.fill(ligne, ' ');
                }
                if(num > dernière) { // si le numéro des images superieur à le dernier images souhaire
                    this.rembobiner();
                    film.rembobiner(); //rembobiner le film
                    return false;
                }
                else {
                    ++num;
                    if(film.suivante(écran)) return true; // sinon obtenir les images (entrer prmiere et derniere images souhaites)
                    else {
                        this.rembobiner(); // rembobiner le film
                        film.rembobiner();
                        return false;
                    }
                }

        }

        /**
         * Rembobine le film en permettant de rejouer le film dans sa totalité (via
         * des appels successifs à la méthode suivante()).
         */
        @Override
        public void rembobiner() {
            num = 0;
        }
```

```
}
```

Class IncrusterFilm

```
package bibliothèqueFilm;

import film.Film;
/**
 * @author : MILLO Chelsey, YE Fangyuan Lisa
 * @date : 24/05/2020
*/

/**
 * Une classe implemente l'Interface Film, il redefinir l'ensemble de methode de
 * l'Interface Film pour incruster un film dans un autre film. Le point d'incrustation
 * sera désigné par les numéros de ligne et de colonne que doit prendre le coin en haut à
 * gauche du film devant être incrusté dans les images du film où il est incrusté
 *
 */
public class IncrusterFilm implements Film{
	private Film f1, f2;
	private int ligne, colonne;

	public IncrusterFilm(Film f1, Film f2, int ligne, int colonne) {
		this.f1 = f1;
		this.f2 = f2;
		this.ligne = ligne;
		this.colonne = colonne;
	}

	/**
	 * Obtenir une hauteur de la film à incrusté
	 *
	 * @return Hauteur minimale de l'écran pour pouvoir afficher les images de
```

```
     *        ce film.
     */
    @Override
    public int hauteur() {
            return f2.hauteur();
    }

    /**
     * Obtenir une largeur de la film à incrusté
     *
     * @return Largeur minimale de l'écran pour pouvoir afficher les images de
     *         ce film.
     */
    @Override
    public int largeur() {
            return f2.largeur();
    }

    /**
     * Obtenir l'image suivante (s'il y en a une).
     *
     * @param écran : L'écran où afficher l'image
     * @return vrai Si l'image suivante a été affichée sur l'écran et faux si le
     *         film est terminé
     */
    @Override
    public boolean suivante(char[][] écran2) {
            char[][] écran1 = new char [f1.hauteur()][f1.largeur()]; // obtenir l'écran vide de film devant être incrusté
            if(f2.suivante(écran2)) { // obtenir les images de film où il est incrusté
                    if(!f1.suivante(écran1)) // si les nombre d'images de film devant être incrusté sont inferieur à celui à incruté
```

```
                    return true;                // obtenir directe les images de la film à
incrusté.
                    if(ligne < 0)ligne = 0;
                    if(colonne < 0)colonne = 0;
                    for(int i = ligne; i < hauteur() && i < f1.hauteur() + ligne; ++i) { // sinon
obtenir les images incrustes
                        for(int j = colonne; j < largeur() && j < f1.largeur() + colonne; ++j) {
                            écran2[i][j] = écran1[i - (ligne)][j - (colonne)];
                        }
                    }
                    return true;
                }
                f1.rembobiner(); // rembobiner les films
                f2.rembobiner();
                return false;
            }

            /**
             * Rembobine le film en permettant de rejouer le film dans sa totalité (via
             * des appels successifs à la méthode suivante()).
             */
            @Override
            public void rembobiner() {
            }

}
```

Class Repetition

```
package bibliothèqueFilm;

import film.Film;
/**
 * @author : MILLO Chelsey, YE Fangyuan Lisa
 * @date : 23/05/2020
*/

/**
 * Une classe implemente l'Interface Film, il redefinir l'ensemble de methode de
 * l'Interface Film pour repeter un film une nombre de fois donné.
 */
public class Repetition implements Film {
	private Film film;
	private int nbRep; // nombre de fois repeter de la film
	private static int num = 0;

	public Repetition(Film film, int nbRep) {
		this.film = film;
		this.nbRep = nbRep;
	}

	/**
	 * Indique la hauteur des images de ce film (en nombre de caractères).
	 *
	 * @return Hauteur minimale de l'écran pour pouvoir afficher les images de
	 *         ce film.
	 */
	@Override
	public int hauteur() {
```

```
        return film.hauteur() ;
}

/**
 * Indique la largeur des images de ce film (en nombre de caractères).
 *
 * @return largeur minimale de l'écran pour pouvoir afficher les images de
 *       ce film.
 */
@Override
public int largeur() {
        return film.largeur();
}

/**
 * Obtenir l'image suivante (s'il y en a une).
 *
 * @param écran :L'écran où afficher l'image
 * @return vrai Si l'image suivante a été affichée sur l'écran et faux si le
 *       film est terminé
 */
@Override
public boolean suivante(char[][] écran) {
        /*if (nbRep <= 0) // repeter un film un nombre de fois negative ou vide
                return false; // conduire à un film vide.*/
        if (nbRep > 0 && num != nbRep) {
                if(!film.suivante(écran)) { // si le film a déjà projecte une fois
                        film.rembobiner();      // rebombiner le film
                        num++;
                        if(num < nbRep) { // tant il n'a pas finir d'afficher nbRep foix le film
```

```
                                        return film.suivante(écran);  // il continuer d'obtenir les
images de film
                                }
                                else {
                                        rembobiner();
                                        return false;
                                }
                        }
                        else return true;
                }
                else return false;
        }

        /**
         * Rembobine le film en permettant de rejouer le film dans sa totalité (via
         * des appels successifs à la méthode suivante()).
         */
        @Override
        public void rembobiner() {
                num = 0;
        }

}
```

BILAN DU PROJET

Difficultés rencontrées

Durant la réalisation de notre projet, nous avons rencontrés de nombreux obstacles qui nous ont empêché d'avancer à la vitesse que nous espérons.

La compréhension totale du sujet était pour nous le problème majeur. Nous savions à peu près quel résultat arriver pour certains cas cependant nous n'avions eu aucune idée par où commencer.

Plus tard, des confusions ont commencé à venir ce qui nous a beaucoup perturber et ralentis. En effet, les solutions que nous pensions être correcte n'étaient en réalités pas celles du professeur, mais en discutant avec nos camarades et l'aide des professeurs, nous avions pu avancer.

C'est avec le jeu de test du professeur que nous avions réussi de localiser les problèmes de notre bibliothèque. A cette occasion nous avions retrouvé 9 échecs sur 12 après la première exécution de jeu d'essai. En effet, une bonne partie des échecs venaient de la méthode rembobiner que nous avions faite.

Nous avons de plus, pas su résoudre quelques problèmes, tels que les tests de composition qui ne fonctionne pas en raison notre algorithme. (par exemple : on retrouve une erreur lorsque l'on souhaite mettre un même film l'un à la suite de ou alors combiner les mêmes opérations)

Ce que nous avons appris grâce à ce projet

Ce projet nous a permis dans un premier temps de mieux nous familiariser avec le langage Java ainsi que son fonctionnement. De plus, ce projet nous a également permis de faire plus attention à l'architecture et la structure de notre code et qui grâce à ça de mieux repérer les éventuels problèmes dans notre code. Sans oublier comment gérer la bibliothèque

Les éventuelles améliorations

Bien que nous sachions mieux organiser l'architecture et la structure de notre code, il n'empêche pas qu'il ait bien évidement une autre manière d'ordonner notre programme.

Nous pensons qu'il serait bien sûr possible d'améliorer notre programme. En effet, on pourrait chercher à optimiser le code le plus possible de sorte qu'il n'occupe le moins d'espace et que l'application s'exécute plus rapidement.

Bilan final

Nous nous sommes vraiment investis dans ce travail en binôme, en essayant de finir le projet. Pour nous, ce projet n'est pas tout à fait terminé, en vue de certains tests qui nous conduisent à des échecs même si notre bibliothèque et méthodes sont complets