RAPPORT

PROJET 2

BPO

CHRISTIAN GAMO
(102)
MONICA HUYNH

2019
2020

TABLE DES MATIERES

PRESENTATION DU PROJET

Ce second projet dans le cadre du module des Bases de la programmation orientée objet consiste à réaliser une bibliothèque permettant d'effectuer des différents montages avec des films, qui sont ici des successions d'images de même taille (une largeur et une hauteur) contenant des caractères ASCII. Les principales méthodes de montage à coder ont été de répéter un film un nombre donné de fois, d'obtenir un extrait d'un film existant, d'encadrer un film, de créer un film en y assemblant deux films à la suite de l'autre et d'incruster un film dans un film.

DIAGRAMME UML

Nous avons trois packages : `bibliotheque`, `film` et `exemple`. `exemple` utilise le package `film`, `bibliotheque` utilise les deux autres packages et `film` n'utilise aucun package. Certaines classes de bibliotheque utilise le package exemple car nous effectuons des tests « boîte blanche » : on utilise la classe LaDiagonaleDuFou pour effectuer nos tests utilitaires.

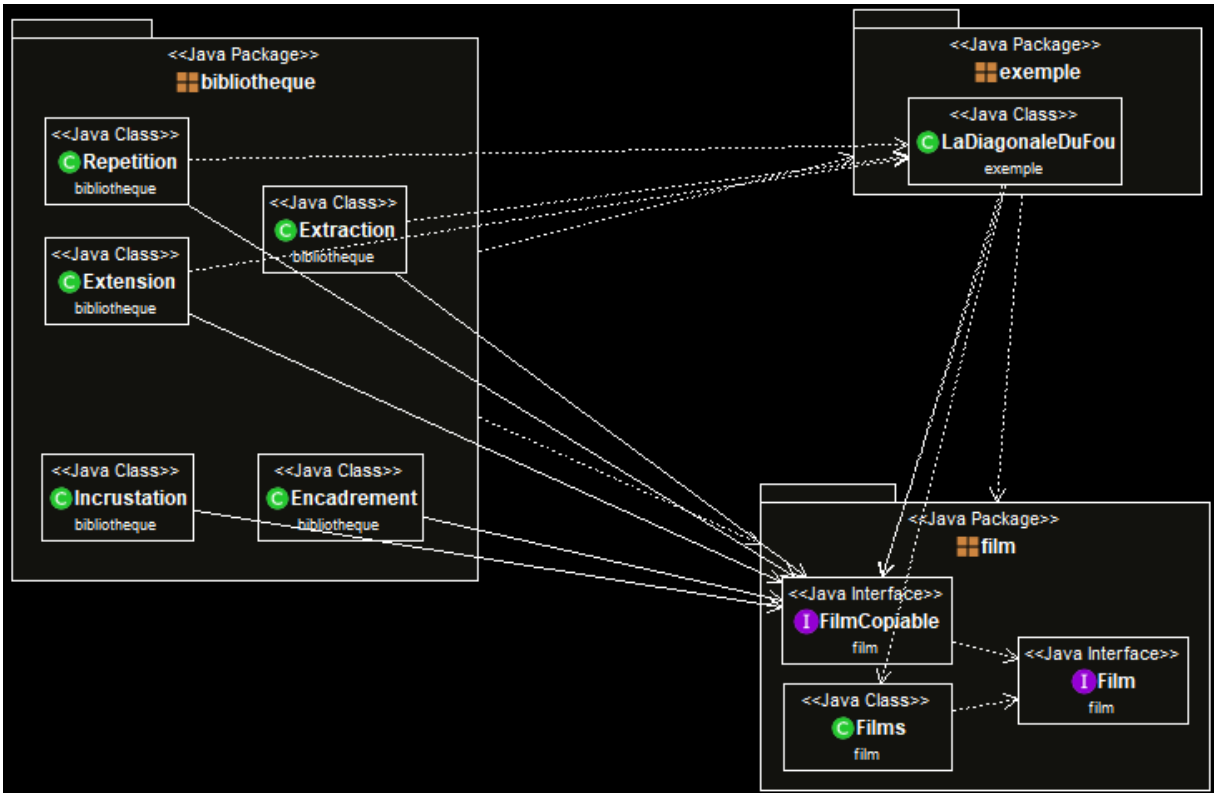

FIGURE 1. DIAGRAMME DE PAQUETAGE

TESTS UNITAIRES

Voici le code Java de nos tests unitaires. Tous se sont exécutés avec succès. Nous avons testé chaque méthode de chaque classe du package `bibliotheque` sauf les constructeurs.

Test Encadrement

```
/**
 * @file EncadrementTest.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 17/05/2020
 * @brief Test jUnit de la classe Encadrement
 */

package bibliotheque;

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

import org.junit.jupiter.api.Test;

import exemple.LaDiagonaleDuFou;
import film.FilmCopiable;
import film.Films;

class EncadrementTest {

	@Test
	void testSuivante() {
		FilmCopiable film = new LaDiagonaleDuFou();
		FilmCopiable cadre = new Encadrement(film.copie());
		char[][] écran1 = Films.getEcran(film);
		char[][] écran2 = Films.getEcran(cadre);
		boolean egal = true;

		while (film.suivante(écran1) && cadre.suivante(écran2)) {
			// Vérification que le film a bien été encadré
			for (int col = 0; col < cadre.largeur(); ++col) {
				// Première ligne d'étoiles
				assertEquals(écran2[0][col], '*');
				//Dernière ligne
				assertEquals(écran2[cadre.hauteur() - 1][col], '*');
			}

			// Les autres lignes
			for (int lig = 1; lig < cadre.hauteur(); ++lig) {
				assertEquals(écran2[lig][0], '*');
				assertEquals(écran2[lig][cadre.largeur() - 1], '*');
			}

			// Vérification que l'intérieur du cadre correspond bien au film
			for (int i = 0; i < film.hauteur(); ++i) {
				for (int j = 0; j < film.largeur(); ++j) {
					if (écran1[i][j] != écran2[i + 1][j + 1]) {
						egal = false;
					}
				}
			}
			assertTrue(egal);
```

```
                Films.effacer(écran1);
                Films.effacer(écran2);
            }
        }

        @Test
        void testRembobiner() {
            FilmCopiable film = new LaDiagonaleDuFou();
            FilmCopiable cadre = new Encadrement(film);
            char[][] écran = Films.getEcran(cadre);
            char[][] frame1 = Films.getEcran(cadre);

            cadre.suivante(frame1);
            cadre.rembobiner();
            cadre.suivante(écran);

            assertEquals(Films.toString(écran), Films.toString(frame1));

        }

        @Test
        void testClone() {
            FilmCopiable film = new LaDiagonaleDuFou();
            FilmCopiable cadre = new Encadrement(film);
            FilmCopiable copie = cadre.copie();
            assertTrue(copie.getClass() == cadre.getClass() && copie != cadre
                    && copie.equals(cadre));
        }

        @Test
        void testHauteur() {
            FilmCopiable film = new LaDiagonaleDuFou();
            FilmCopiable cadre = new Encadrement(film);

            assertEquals(film.hauteur() + 2, cadre.hauteur());
        }

        @Test
        void testLargeur() {
            FilmCopiable film = new LaDiagonaleDuFou();
            FilmCopiable cadre = new Encadrement(film);

            assertEquals(film.largeur() + 2, cadre.largeur());
        }
}
```

Test Extension

```
/**
 * @file ExtensionTest.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 17/05/2020
 * @brief Test jUnit de la classe Extension
 */
package bibliotheque;

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

```java
import org.junit.jupiter.api.Test;

import exemple.LaDiagonaleDuFou;
import film.FilmCopiable;
import film.Films;

class ExtensionTest {

	@Test
	void testSuivante() {
		FilmCopiable diag1 = new LaDiagonaleDuFou();
		FilmCopiable diag2 = new LaDiagonaleDuFou();
		FilmCopiable extension = new Extension(diag1.copie(),diag2.copie());

		int cptPartie1 = 0;
		int cptPartie2 = 0;
		int cptExtension = 0;

		char[][] écran1 = Films.getEcran(diag1);
		char[][] écran2 = Films.getEcran(diag2);
		char[][] écran3 = Films.getEcran(extension);

		while (diag1.suivante(écran1))
			++cptPartie1;


		while (diag2.suivante(écran2))
			++cptPartie2;

		while (extension.suivante(écran3))
			++cptExtension;

		assertEquals(cptPartie1+cptPartie2,cptExtension);

		extension.rembobiner();
		diag1.rembobiner();
		diag2.rembobiner();
		Films.effacer(écran1);
		Films.effacer(écran2);
		Films.effacer(écran3);

		for (int i = 0; i<cptPartie1 ; ++i) {
			diag1.suivante(écran1);
			extension.suivante(écran3);
			assertEquals(Films.toString(écran1),Films.toString(écran3));
			Films.effacer(écran1);
			Films.effacer(écran3);
		}

		for (int i = 0; i<cptPartie2 ; ++i) {
			diag2.suivante(écran2);
			extension.suivante(écran3);
			assertEquals(Films.toString(écran2),Films.toString(écran3));
			Films.effacer(écran2);
			Films.effacer(écran3);
		}


	}

	@Test
	void testRembobiner() {
```

```
		FilmCopiable diag1 = new LaDiagonaleDuFou();
		FilmCopiable diag2 = new LaDiagonaleDuFou();
		FilmCopiable extension = new Extension(diag1.copie(),diag2.copie());
		char[][] écran = Films.getEcran(extension);
		char[][] frame1 = Films.getEcran(extension);

		extension.suivante(frame1);
		extension.rembobiner();
		extension.suivante(écran);

		assertEquals(Films.toString(écran),Films.toString(frame1));

		Extension.testRembobinerPartie1Fini();

	}

	@Test
	void testClone() {
		FilmCopiable diag1 = new LaDiagonaleDuFou();
		FilmCopiable diag2 = new LaDiagonaleDuFou();
		FilmCopiable extension = new Extension(diag1.copie(),diag2.copie());
		FilmCopiable copie = extension.copie();
		assertTrue(copie.getClass() == extension.getClass()
				&& copie != extension && copie.equals(extension));
	}

	@Test
	void testHauteur() {
		FilmCopiable diag1 = new LaDiagonaleDuFou();
		FilmCopiable diag2 = new LaDiagonaleDuFou();
		FilmCopiable extension = new Extension(diag1.copie(),diag2.copie());

		assertEquals(diag1.hauteur(), extension.hauteur());
	}

	@Test
	void testLargeur() {
		FilmCopiable diag1 = new LaDiagonaleDuFou();
		FilmCopiable diag2 = new LaDiagonaleDuFou();
		FilmCopiable extension = new Extension(diag1.copie(),diag2.copie());

		assertEquals(diag1.largeur(), extension.largeur());
	}

}
```

Test Extraction

```
/**
 * @file ExtractionTest.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 17/05/2020
 * @brief Test jUnit de la classe Extraction
 */
package bibliotheque;

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

import org.junit.jupiter.api.Test;

import exemple.LaDiagonaleDuFou;
import film.FilmCopiable;
import film.Films;
```

```
class ExtractionTest {

	@Test
	void testSuivante() {
		int nbDeb = 2;
		int nbFin = 5;
		FilmCopiable diag = new LaDiagonaleDuFou();
		FilmCopiable extrait = new Extraction(diag.copie(),nbDeb,nbFin);
		char[][] écran = Films.getEcran(diag);
		char[][] écran2 = Films.getEcran(extrait);
		char[][] écranPoubelle = Films.getEcran(extrait);

		for(int i = 0; i < nbDeb; ++i) {
			diag.suivante(écranPoubelle);
		}

		for(int i = 0; i <= nbFin - nbDeb;++i) {
			diag.suivante(écran);
			extrait.suivante(écran2);

			assertEquals(Films.toString(écran),Films.toString(écran2));
			Films.effacer(écran);
			Films.effacer(écran2);
		}

	}

	@Test
	void testRembobiner() {
		FilmCopiable diag = new LaDiagonaleDuFou();
		int noDeb = 2;
		int noFin = 3;
		FilmCopiable extraction = new Extraction(diag,noDeb,noFin);
		char[][] écran = Films.getEcran(extraction);
		char[][] frame1 = Films.getEcran(extraction);

		extraction.suivante(frame1);
		extraction.rembobiner();
		extraction.suivante(écran);

		assertEquals(Films.toString(écran),Films.toString(frame1));
		Extraction.testRembobinerCpt();

	}

	@Test
	void testClone() {
		FilmCopiable diag = new LaDiagonaleDuFou();
		int noDeb = 2;
		int noFin = 3;
		FilmCopiable extraction = new Extraction(diag,noDeb,noFin);
		FilmCopiable copie = extraction.copie();
		assertTrue(copie.getClass() == extraction.getClass()
				&& copie != extraction && copie.equals(extraction));
	}

	@Test
	void testHauteur() {
		FilmCopiable diag = new LaDiagonaleDuFou();
		int noDeb = 2;
		int noFin = 3;
```

```
		FilmCopiable extraction = new Extraction(diag,noDeb,noFin);

		assertEquals(diag.hauteur(), extraction.hauteur());
	}

	@Test
	void testLargeur() {
		FilmCopiable diag = new LaDiagonaleDuFou();
		int noDeb = 2;
		int noFin = 3;
		FilmCopiable extraction = new Extraction(diag,noDeb,noFin);

		assertEquals(diag.largeur(), extraction.largeur());
	}

}
```

Test Incrustation

```
/**
 * @file IncrustationTest.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 17/05/2020
 * @brief Test jUnit de la classe Incrustation
 */
package bibliotheque;

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

import org.junit.jupiter.api.Test;

import exemple.LaDiagonaleDuFou;
import film.FilmCopiable;
import film.Films;

class IncrustationTest {

	@Test
	void testSuivante() {
		FilmCopiable normal = new LaDiagonaleDuFou();
		FilmCopiable normal2 = new LaDiagonaleDuFou();
		int ligne = 2;
		int colonne = 3;
		FilmCopiable incrustation = new Incrustation(normal.copie(),
				normal2.copie(), ligne, colonne);

		// Film avec incrustation
		char[][] écranIncruste = Films.getEcran(incrustation);
		// Film de base
		char[][] écranBase = Films.getEcran(normal);
		// Film à incruster
		char[][] écranAIncruste = Films.getEcran(normal2);
		boolean egal = true;
		int colIncrus = colonne;
		int ligIncrus = ligne;

		while (normal2.suivante(écranAIncruste) && normal.suivante(écranBase)) {
		     // Le film de base au point d'incrustation, ne possède pas le film
		à incruster
		    for (int i = 0; ligIncrus < normal.hauteur() - ligne;
		                    ++i, ++ligIncrus, colIncrus = colonne) {
		      for (int j = 0; j < normal.largeur() - colonne; ++j, ++colIncrus) {
```

```
                        if(écranAIncruste[i][j] != écranBase[ligIncrus][colIncrus])
                                egal = false;

                        }
                }
                Films.effacer(écranIncruste);
                Films.effacer(écranBase);
        }
        assertEquals(egal, false);

        normal2.rembobiner();
        colIncrus = colonne;
        ligIncrus = ligne;

        Films.effacer(écranIncruste);
        Films.effacer(écranAIncruste);
        incrustation.suivante(écranIncruste);
        normal2.suivante(écranAIncruste);
        while (incrustation.suivante(écranIncruste)
                        && normal2.suivante(écranAIncruste)) {
                // Le film de base avec normalement l'incrustation, possède le
                film à incruster
                for (int i = 0; ligIncrus < normal.hauteur() - ligne;
                         ++i, ++ligIncrus, colIncrus = colonne) {
                        for (int j = 0; j < normal.largeur()
                                        - colonne; ++j, ++colIncrus) {
                                if ((écranAIncruste[i][j]
                                        == écranIncruste[ligIncrus][colIncrus]))
                                        egal = true;

                                else
                                        egal = false;
                        }
                }
                Films.effacer(écranIncruste);
                Films.effacer(écranAIncruste);
        }
        assertEquals(egal, true);
}

@Test
void testRembobiner() {
        FilmCopiable diag1 = new LaDiagonaleDuFou();
        FilmCopiable diag2 = new LaDiagonaleDuFou();
        int ligne = 2;
        int colonne = 3;
        FilmCopiable incrustation = new Incrustation(diag1.copie(),
                        diag2.copie(), ligne, colonne);

        char[][] écran = Films.getEcran(incrustation);
        char[][] frame1 = Films.getEcran(incrustation);

        incrustation.suivante(frame1);
        incrustation.rembobiner();
        incrustation.suivante(écran);

        assertEquals(Films.toString(écran), Films.toString(frame1));

}
```

```
	@Test
	void testClone() {
		FilmCopiable diag1 = new LaDiagonaleDuFou();
		FilmCopiable diag2 = new LaDiagonaleDuFou();
		int ligne = 2;
		int colonne = 3;
		FilmCopiable incrustation = new Incrustation(diag1.copie(),
				diag2.copie(), ligne, colonne);

		FilmCopiable copie = incrustation.copie();
		assertTrue(copie.getClass() == incrustation.getClass()
				&& copie != incrustation && copie.equals(incrustation));
	}

	@Test
	void testHauteur() {
		FilmCopiable diag1 = new LaDiagonaleDuFou();
		FilmCopiable diag2 = new LaDiagonaleDuFou();
		int ligne = 2;
		int colonne = 3;
		FilmCopiable incrustation = new Incrustation(diag1.copie(),
				diag2.copie(), ligne, colonne);

		assertEquals(diag1.hauteur(), incrustation.hauteur());
	}

	@Test
	void testLargeur() {
		FilmCopiable diag1 = new LaDiagonaleDuFou();
		FilmCopiable diag2 = new LaDiagonaleDuFou();
		int ligne = 2;
		int colonne = 3;
		FilmCopiable incrustation = new Incrustation(diag1.copie(),
				diag2.copie(), ligne, colonne);

		assertEquals(diag1.largeur(), incrustation.largeur());
	}

}
```

Test Repetition

```
/**
 * @file RepetitionTest.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 2 - 23/05/2020
 * @brief Test jUnit de la classe Repetition
 */
package bibliotheque;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import exemple.LaDiagonaleDuFou;
import film.FilmCopiable;
import film.Films;

class RepetitionTest {

	@Test
```

```
	void testRembobiner() {
		FilmCopiable diag = new LaDiagonaleDuFou();
		Repetition repet = new Repetition(diag.copie(), 1);
		char[][] écran = Films.getEcran(repet);
		char[][] écran2 = Films.getEcran(repet);
		Repetition.testBlancCptRepet0();
		Repetition.testBlancCptRepet();
		repet.rembobiner();
		Repetition.testBlancCptRepet0();

		Films.effacer(écran);
		repet.suivante(écran);
		diag.suivante(écran2);

		assertEquals(Films.toString(écran2),Films.toString(écran));

	}

	@Test
	void testSuivante() {
		int nbRepet = 4;
		FilmCopiable diag = new LaDiagonaleDuFou();
		Repetition repet = new Repetition(diag.copie(), nbRepet);
		char[][] écran = Films.getEcran(repet);

		int cptOriginal = 0;
		int cptRepet = 0;
		while (diag.suivante(écran))
			++cptOriginal;
		while(repet.suivante(écran))
			++cptRepet;
		assertTrue(cptRepet == nbRepet*cptOriginal);
	}

	@Test
	void testClone() {
		int nbRepet = 4;
		FilmCopiable diag = new LaDiagonaleDuFou();
		Repetition repet = new Repetition(diag, nbRepet);
		FilmCopiable copie = repet.copie();
		assertTrue(copie.getClass() == repet.getClass()
				&& copie != repet && copie.equals(repet));
	}


	@Test
	void testHauteur() {
		FilmCopiable diag = new LaDiagonaleDuFou();
		Repetition repet = new Repetition(diag, 2);

		assertEquals(diag.hauteur(), repet.hauteur());
	}

	@Test
	void testLargeur() {
		FilmCopiable diag = new LaDiagonaleDuFou();
		Repetition repet = new Repetition(diag, 2);

		assertEquals(diag.largeur(), repet.largeur());
	}

}
```

CODE JAVA DU PROJET

Film.java (interface donnée et non modifiée)

```
package film;

/**
 * Interface devant être respectée par tout film.
 *
 * Attention, pour qu'un film puisse être projeté ou sauvergardé (voir la classe
 * film.Outils), il doit être composé d'un nombre fini d'images. En conséquence,
 * la méthode suivante() doit nécessairement retourner false au bout d'un nombre
 * fini d'appels.
 */
public interface Film {
	/**
	 * 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.
	 */
	int 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.
	 */
	int 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
	 */
	boolean suivante(char[][] écran);

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

}
```

Films.java (classe donnée et non modifiée)

```
package film;

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Arrays;

/**
 * Ensemble de méthodes de classe utilitaires pour la projection (sur System.in)
```

```
 * et la sauvegarde (dans un fichier) d'un film. Le format de fichier employé
 * pour la sauvegarde est compatible avec le player graphique.
 */
public class Films {
	/**
	 * Projette un film dans sa totalité sur System.in. Attention, la méthode ne
	 * se termine pas si le film est infini.
	 *
	 * @param f
	 *            Le film devant être projeté.
	 */
	public static void projeter(Film f) {
		char[][] écran = getEcran(f);
		while (f.suivante(écran)) {
			System.out.println(toString(écran));
			pause(1. / 12);
			effacer(écran);
		}
	}

	/**
	 * Sauvegarder un film dans un fichier.
	 *
	 * @param f
	 *            Le film à sauvegarder.
	 * @param nom
	 *            le nom du fichier où sauvegarder le film.
	 * @throws FileNotFoundException
	 *             Si le nom du fichier ne permet pas de le créer.
	 */
	public static void sauvegarder(Film f, String nom) throws FileNotFoundException
{
		try (PrintWriter out = new PrintWriter(nom)) {
			char[][] écran = getEcran(f);
			out.println(f.largeur() + " " + f.hauteur());
			while (f.suivante(écran)) {
				out.println(toString(écran));
				out.println("\\newframe");
				effacer(écran);
			}
		}
	}

	/**
	 * Construit un écran adapté à la projection d'un film.
	 *
	 * @param f
	 *            Le film pour lequel un écran doit être constuit.
	 * @return L'écran adapté au film.
	 */
	public static char[][] getEcran(Film f) {
		char[][] écran = new char[f.hauteur()][f.largeur()];
		effacer(écran);
		return écran;
	}

	/**
	 * Efface un écran.
	 *
	 * @param écran
	 *            L'écran à effacer
	 */
```

```
	public static void effacer(char[][] écran) {
		for (char[] ligne : écran)
			Arrays.fill(ligne, ' ');
	}

	/**
	 * Convertit en chaine de caractère un écran.
	 *
	 * @param écran
	 *            L'écran à convertir
	 * @return La chaine correspondante à l'écran.
	 */
	public static String toString(char[][] écran) {
		StringBuilder s = new StringBuilder();
		for (char[] ligne : écran)
			s.append(new String(ligne) + System.lineSeparator());
		return s.toString();
	}

	/**
	 * Endort le programme pendant un temps donné.
	 *
	 * @param d
	 *            Le temps d'endormissement exprimé en seconde.
	 */
	public static void pause(double d) {
		try {
			Thread.sleep((long) (d * 1000));
		} catch (InterruptedException e) {
			Thread.currentThread().interrupt();
		}
	}

	// Cette classe n'a pas vocation à être instanciée car elle ne contient que
	// des méthodes de classe (i.e. statiques).
	// Introduire un constructeur privé interdit toute instanciation (en dehors
	// de la classe elle même).
	private Films() {
	}
}
```

FilmCopiable.java (Interface)

```
/**
 * @file FilmCopiable.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 01/05/2020
 * @brief Sous-interface de Film permettant le clonage d'un film
 */

package film;

/** Sous-interface de Film permettant le clonage d'un film
 *
 * Attention, lorsque des films ne sont pas "indépendants"
 * (lorsqu'ils référencent une même instance), leur combinaison
 * au sein d'un même montage peut donner des résultats incohérents.
 * Pour éviter ce problème, on peut appeler la méthode clone()
 * dans l'instanciation d'une classe de la bibliothèque
 * dépendant de 1 ou plusieurs FilmCopiable.
 *
```

```
 * Aussi, si vous souhaitez projeter ou sauvegarder deux films à la suite
 * qui ne sont pas indépendants, il faut utiliser la méthode rembobiner()
 * entre les deux projetages/sauvegardes.
 */


public interface FilmCopiable extends Film {

	/**
	 * Méthode abstraite pour cloner un FilmCopiable
	 */
	FilmCopiable copie();


}
```

LaDiagonaleDuFou.java (classe donnée et modifiée légèrement)

```
package exemple;

import java.io.FileNotFoundException;

import film.FilmCopiable;
import film.Films;

/**
 * Un exemple basique d'implémentation de l'interface Film.
 */
public class LaDiagonaleDuFou implements FilmCopiable {
	private int num = 0;
	private static final int NB_IMAGES = 20;



	@Override
	public int hauteur() {
		return 10;
	}

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


	@Override
	public boolean suivante(char[][] écran) {

		if (num == NB_IMAGES)
			return false;
		écran[num % hauteur()][num % hauteur()] = 'a'; // un 'a' se balade sur

		// la diagonale
		++num;
		return true;
	}

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

	@Override
```

```
	public LaDiagonaleDuFou copie() {
		return new LaDiagonaleDuFou();
	}

	/**
	 * La projection (puis la sauvegarde) d'un tel film.
	 */
	public static void main(String[] args) {
		FilmCopiable film = new LaDiagonaleDuFou();
		Films.projeter(film);
		film.rembobiner();
		try {
			Films.sauvegarder(film, "fou.txt");
		} catch (FileNotFoundException e) {
			System.err.println("Le fichier 'fou.txt' n'a pas pu être créé.");
		}
	}
}
```

Encadrement.java

```
/**
 * @file Encadrement.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 01/05/2020
 * @brief Type de donnée représentant un film encadré
 */

package bibliotheque;

import java.util.Arrays;

import film.FilmCopiable;

/** Type de donnée représentant un film encadré */

public class Encadrement implements FilmCopiable {
	/** Le film que l'on va encadrer */
	private FilmCopiable original;
	/** Nombre de caractères nécessaire pour le placement du cadre */
	private static final int BORDS = 2;
	/** Caractère formant le cadre */
	private static final char CADRE = '*';

	/**
	 * Constructeur
	 *
	 * @param f Le film que l'on va encadrer (obligatoire)
	 */
	public Encadrement(FilmCopiable f) {
		assert (f != null);
		this.original = f;
	}

	/**
	 * Spécialise la méthode copie() de FilmCopiable
	 *
	 * @return Retourne une copie d'une instance de Encadrement
	 */
	@Override
	public FilmCopiable copie() {
		return new Encadrement(original);
```

```
}

/**
 * * 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 this.original.hauteur() + BORDS;
}

/**
 ** 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 this.original.largeur() + BORDS;
}

/**
 * Spécialise la méthode suivante() de FilmCopiable.
 * 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) {
	assert(écran != null);
	char[][] écran2 = new char[original.hauteur()][original.largeur()];
	for (char[] ligne : écran2)
		Arrays.fill(ligne, ' ');

	if (!original.suivante(écran2))
		return false;

	for (int lig = 0; lig < original.hauteur(); ++lig) {
		for (int col = 0; col < original.largeur(); ++col) {
			écran[lig + 1][col + 1] = écran2[lig][col];
		}
	}

	for (int i = 0; i < largeur(); ++i) {
		écran[0][i] = CADRE; // Bord du haut
		écran[hauteur() - 1][i] = CADRE; // Bord du bas
	}

	for (int j = 0; j < hauteur(); ++j) {
		écran[j][0] = CADRE; // Bord de gauche
		écran[j][largeur() - 1] = CADRE; // Bord de droite
	}

	return true;
}

/**
```

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


	/**
	 * Spécialise la méthode equals() de la classe Object
	 * Vérifie que tous les attributs possèdent la même valeur
	 */
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Encadrement other = (Encadrement) obj;
		if (original == null) {
			if (other.original != null)
				return false;
		}
		else if (!original.equals(other.original))
			return false;
		return true;
	}
}
```

Repetition.java

```
/**
 * @file Repetition.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 2 - 23/05/2020
 * @brief Type de donnée représentant un film répété
 */

package bibliotheque;
import static org.junit.Assert.*;

import exemple.LaDiagonaleDuFou;
import film.FilmCopiable;
/** Type de donnée représentant un film répété */


public class Repetition implements FilmCopiable {
	/** Film que l'on va répéter */
	private FilmCopiable original;
	/** Nombre de répétitions voulu */
	private int nbRepet;
	/** Compteur du nombre de répétitions effectué */
	private int cptRepet;

	/**
	 * Constructeur
	 * @param f Le film que l'on va répété (obligatoire)
	 * @param nb Le nombre de répétitions voulu (obligatoire)
```

```
     */
    public Repetition(FilmCopiable f, int nb) {
           assert(f != null);
           this.original = f;
           this.nbRepet = nb;
           this.cptRepet = 0;

    }

    /**
      * Méthode statique crée pour le test unitaire.
      * Permet de vérifier la valeur de l'attribut cptRepet
      * après instanciation
      */
    public static void testBlancCptRepet0() {
           FilmCopiable diag = new LaDiagonaleDuFou();
        Repetition repet = new Repetition(diag, 1);
        assertEquals(0, repet.cptRepet);
    }

    /**
      * Méthode statique crée pour le test unitaire.
      * Permet de vérifier la valeur de l'attribut cpt
      * avant et après rembobinage.
      */
    public static void testBlancCptRepet() {
        FilmCopiable diag = new LaDiagonaleDuFou();
        Repetition repet = new Repetition(diag, 1);
        char[][] écran = new char[repet.hauteur()][repet.largeur()];
           while(repet.suivante(écran));
        assertFalse(repet.cptRepet == 0);
}


    /**
     * Spécialise la méthode copie() de FilmCopiable
     * @return Retourne une copie d'une instance de Repetition
     */
    @Override
    public FilmCopiable copie() {
           return new Repetition(original, nbRepet);
    }

    /**
     * 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 this.original.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 this.original.largeur();
}

/**
 * Spécialise la méthode suivante() de FilmCopiable
 * 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, donc a atteint
 *               le nombre de répétitions souhaité
 */
@Override
public boolean suivante(char[][] écran) {
        assert(écran != null);

        if (!original.suivante(écran)) {
                original.rembobiner();
                if(!original.suivante(écran))
                        return false;
                ++cptRepet;
        }
        return cptRepet < nbRepet;
}

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

/**
 * Spécialise la méthode equals() de la classe Object
 * Vérifie que tous les attributs possèdent la même valeur
 */
@Override
public boolean equals(Object obj) {
        if (this == obj)
                return true;
        if (obj == null)
                return false;
        if (getClass() != obj.getClass())
                return false;
        Repetition other = (Repetition) obj;
        if (cptRepet != other.cptRepet)
                return false;
        if (nbRepet != other.nbRepet)
                return false;
        if (original == null) {
                if (other.original != null)
                        return false;
        }
        else
                if (!original.equals(other.original))
                        return false;
        return true;
}
```

```
}
```

Extraction.java

```
/**
 * @file Extraction.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 01/05/2020
 * @brief Type de donnée représentant un extrait de film
 */

package bibliotheque;

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

import java.util.Arrays;

import exemple.LaDiagonaleDuFou;
import film.FilmCopiable;

/** Type de donnée représentant un extrait de film */

public class Extraction implements FilmCopiable {

	/** Film dont on va extraire une partie */
	private FilmCopiable original;
	/** Numéro de la première frame de l'extrait */
	private int noDebut;
	/** Numéro de la dernière frame de l'extrait */
	private int noFin;
	/** Compteur du numéro de frame */
	private int cpt;

	/**
	 * Constructeur
	 * @param f Le film dont on va extraire une partie (obligatoire)
	 * @param noDeb Le numéro de frame du début de l'extrait (obligatoire)
	 * @param noFin Le numéro de frame de la fin de l'extrait (obligatoire)
	 */
	public Extraction(FilmCopiable f, int noDeb, int noFin) {
		assert(f != null);
		this.original = f;
		this.noDebut = noDeb;
		this.noFin = noFin;
		this.cpt = 0;
	}

	 /**
	  * Méthode statique créée pour le test unitaire.
	  * Permet de vérifier la valeur de l'attribut cpt
	  * avant et après rembobinage.
	  */
	 public static void testRembobinerCpt() {
		FilmCopiable diag = new LaDiagonaleDuFou();
		int noDeb = 2;
		int noFin = 18;
		int compteur = 0;
		Extraction extraction = new Extraction(diag,noDeb,noFin);
		char[][] écran = new char[extraction.hauteur()][extraction.largeur()];

		while(extraction.suivante(écran));
		compteur = noFin + 1;
```

```
     //= aux nombres de frames avant et pendant l'extraction du film de base

        assertEquals(extraction.cpt,compteur);
        extraction.rembobiner();
        assertEquals(extraction.cpt,0);
   }

/**
 * Spécialise la méthode copie() de FilmCopiable
 * @return Retourne une copie d'une instance de Extraction
 */
@Override
public FilmCopiable copie(){
       return new Extraction(original, noDebut, noFin);
}

/**
 * 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 this.original.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 this.original.largeur();
}

/**
 * Spécialise la méthode suivante() de FilmCopiable
 *
 * 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) {
       assert(écran != null);
       // écran2 est une variable qui va contenir les images
       // dont on a pas besoin pour l'extrait
       char[][] écran2 = new char[hauteur()][largeur()];
       for (char[] lignes : écran2)
     Arrays.fill(lignes, ' ');

       while (cpt < noDebut) {
              ++cpt;
              original.suivante(écran2);
       }
```

```
			if (cpt <= noFin) {
				++cpt;
				return original.suivante(écran);
			}
			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() {
			this.cpt = 0;
			this.original.rembobiner();
		}

		/**
		 * Spécialise la méthode equals() de la classe Object
		 * Vérifie que tous les attributs possèdent la même valeur
		 */
		@Override
		public boolean equals(Object obj) {
			if (this == obj)
				return true;
			if (obj == null)
				return false;
			if (getClass() != obj.getClass())
				return false;
			Extraction other = (Extraction) obj;
			if (cpt != other.cpt)
				return false;
			if (noDebut != other.noDebut)
				return false;
			if (noFin != other.noFin)
				return false;
			if (original == null) {
				if (other.original != null)
					return false;
			}
			else
				if (!original.equals(other.original))
					return false;
			return true;
		}
}
```

Entension.java

```
/**
 * @file Extension.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 1 - 01/05/2020
 * @brief Type de donnée représentant un film composé d'un collage de deux films
 */

package bibliotheque;

import static org.junit.Assert.*;

import exemple.LaDiagonaleDuFou;
```

```java
import film.FilmCopiable;

/** Type de donnée représentant un film composé d'un collage de deux films */

public class Extension implements FilmCopiable {
	/** Le film qui compose la première partie du film collé */
	private FilmCopiable partie1;
	/** Le film qui compose la deuxième partie du film collé */
	private FilmCopiable partie2;
	/** Booléan indiquant si la première partie a été vue */
	private boolean partie1Fini;

	/**
	 * Constructeur
	 *
	 * @param premier  Le film qui compose la première partie du film collé
	 *                 (obligatoire)
	 * @param deuxieme Le film qui compose la deuxième partie du film collé
	 *                 (obligatoire)
	 */
	public Extension(FilmCopiable premier, FilmCopiable deuxieme) {
		assert (premier != null && deuxieme != null);
		this.partie1 = premier;
		this.partie2 = deuxieme;
		this.partie1Fini = false;
	}

	/**
	 * Méthode statique créée pour le test unitaire. Permet de vérifier la valeur
	 * de l'attribut partie1Fini avant et après rembobinage.
	 */
	public static void testRembobinerPartie1Fini() {
		FilmCopiable diag1 = new LaDiagonaleDuFou();
		FilmCopiable diag2 = new LaDiagonaleDuFou();
		Extension extend = new Extension(diag1, diag2);
		char[][] écran = new char[extend.hauteur()][extend.largeur()];

		while (extend.suivante(écran))
			;

		assertTrue(extend.partie1Fini);
		extend.rembobiner();
		assertFalse(extend.partie1Fini);
	}

	/**
	 * * Spécialise la méthode copie() de FilmCopiable
	 *
	 * @return Retourne une copie d'une instance de Extension
	 */
	@Override
	public FilmCopiable copie() {
		return new Extension(partie1, partie2);
	}

	/**
	 * * 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() {
	if (partie1.hauteur() > partie2.hauteur())
		return partie1.hauteur();
	else
		return partie2.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() {
	if (partie1.largeur() > partie2.largeur())
		return partie1.largeur();
	else
		return partie2.largeur();

}

/**
 * Spécialise la méthode suivante() de FilmCopiable.
 * 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 (les deux parties sont terminés)
 */
@Override
public boolean suivante(char[][] écran) {
	assert(écran != null);

	if (!partie1Fini) {
		if (partie1.suivante(écran))
			return true;
		else {
			partie1Fini = true;
			partie2.rembobiner();
			return partie2.suivante(écran);
		}
	}
	else
		return partie2.suivante(écran);
}

/**
 * Rembobine le film (en rembobinant les 2 parties) en permettant de rejouer
 * le film dans sa totalité (via des appels successifs à la méthode
 * suivante()).
 */
@Override
public void rembobiner() {
	this.partie1.rembobiner();
	this.partie2.rembobiner();
	partie1Fini = false;
}

/**
 * Spécialise la méthode equals() de la classe Object.
 * Vérifie que tous les attributs possèdent la même valeur
```

```
	 */
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Extension other = (Extension) obj;
		if (partie1 == null) {
			if (other.partie1 != null)
				return false;
		}
		else if (!partie1.equals(other.partie1))
			return false;
		if (partie1Fini != other.partie1Fini)
			return false;
		if (partie2 == null) {
			if (other.partie2 != null)
				return false;
		}
		else if (!partie2.equals(other.partie2))
			return false;
		return true;
	}

}
```

Incrustation.java

```
/**
 * @file Incrustation.java
 * Projet BPO - IUT de Paris
 * @author Christian Gamo & Monica Huynh
 * @version 2 - 23/05/2020
 * @brief Type de donnée représentant l'incrustation d'un film sur un autre
 */

package bibliotheque;

import java.util.Arrays;

import film.FilmCopiable;

/** Type de donnée représentant l'inscrustation d'un film sur un autre */

public class Incrustation implements FilmCopiable{
	/** Le film servant de fond */
	private FilmCopiable base;
	/** Le film inscrusté */
	private FilmCopiable aIncruste;
	/** Numéro de la ligne du point d'incrustation du film aIncruste */
	private int ligne;
	/** Numéro de la colonne du point d'incrustation du film aIncruste */
	private int colonne;

	/**
	 * Constructeur
	 * @param f1 Le film servant de base
	 * @param f2 Le film à incruster dans la base
	 * @param x Le numéro de la ligne
```

```
 * du point d'incrustation du film à incruster
 * @param y Le numéro de la colonne
 * du point d'incrustation du film à incruster
 */
public Incrustation(FilmCopiable f1, FilmCopiable f2, int x, int y) {
	assert(f1 != null && f2 != null);
	if(x<0)
		x = 0;
	if(y<0)
		y = 0;
	base = f1;
	aIncruste = f2;
	ligne = x;
	colonne = y;
}

/**
 * Spécialise la méthode copie() de FilmCopiable
 * @return Retourne une copie d'une instance de Incrustation
 */
@Override
public FilmCopiable copie() {
		return new Incrustation(base, aIncruste, ligne, colonne);
}

/**
 * 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 base.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 base.largeur();
}

/**
 * Spécialise la méthode suivante() de FilmCopiable.
 * 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) {
	assert(écran != null);
	if (!base.suivante(écran))
		return false;
```

```
            char[][] écran2 = new char[aIncruste.hauteur()][aIncruste.largeur()];
            for (char[] lignes : écran2)
           Arrays.fill(lignes, ' ');

            int ligIncrus = 0;
            int colIncrus = 0;

            if (aIncruste.suivante(écran2) && ligne>=0 && colonne>=0) {
                  for (int i = ligne; i < base.hauteur();
                              ++i, ++ligIncrus, colIncrus = 0) {
                        for (int j = colonne; j < base.largeur(); ++j, ++colIncrus)
{
                              if(ligIncrus < aIncruste.hauteur() &&
                                 colIncrus < aIncruste.largeur()) {
                                     écran[i][j] = écran2[ligIncrus][colIncrus];
                              }
                        }
                  }
            }
            return true;
      }

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

      /**
       * Spécialise la méthode equals() de la classe Object
       * Vérifie que tous les attributs possèdent la même valeur
       */
      @Override
      public boolean equals(Object obj) {
            if (this == obj)
                  return true;
            if (obj == null)
                  return false;
            if (getClass() != obj.getClass())
                  return false;
            Incrustation other = (Incrustation) obj;
            if (aIncruste == null) {
                  if (other.aIncruste != null)
                        return false;
            }
            else
                  if (!aIncruste.equals(other.aIncruste))
                        return false;
                  if (base == null) {
                        if (other.base != null)
                              return false;
            }
            else
                  if (!base.equals(other.base))
                        return false;
                  if (colonne != other.colonne)
                        return false;
                  if (ligne != other.ligne)
```

```
                    return false;
                return true;
        }
}
```

BILAN DE PROJET

RETOUR D'EXPERIENCE

Les difficultés rencontrées

La plus grosse difficulté rencontrée lors de ce projet a été les problèmes de référence. Le fait d'utiliser un film déjà existant et de le mettre en attribut de notre nouveau film nous a posé des soucis. En effet, nous agissions directement sur le film et non pas une nouvelle instance du film. Nous ne voulions pas interdire (en précisant dans la documentation) que pour effectuer un montage, on ne pouvait pas passer en paramètre un film déjà employé. Pour essayer de remédier à ce problème, nous avions décidé d'essayer d'effectuer des copies et ainsi utiliser la copie du film et non pas le film original. Cette solution a été un peu difficile à mettre en place car nous n'avions pas réellement de vraies connaissances quant à l'utilisation de la méthode .clone() contenue dans la bibliothèque standard. Avec beaucoup d'aide, de recherches et d'essais, nous avons réussi à créer cette copie et à l'utiliser seulement dans les cas où nous avons besoin de nous en servir. Maintenant les problèmes plus mineurs, ont été les tests unitaires : au départ, ça a été un peu flou sur comment tester une méthode, nous ne voulions pas créer des méthodes ou des getters/setters en plus seulement pour tester une méthode. Après quelques conseils, nous avons réalisé des méthodes statiques dans certaines classes de notre bibliothèque pour effectuer des tests unitaires. Nous avons également, grâce aux tests unitaires, trouvé un certain bug qui nous a été difficile à résoudre (nous avons passé pas mal de temps à ne pas comprendre). En comparant le contenu des écrans (où l'on diffuse le film), nous trouvions que dans un tableau où était contenu du vide (un espace) n'était pas égal à un autre tableau où celui-ci contenait également du vide. Nous n'avions pas initialisé correctement nos écrans, ainsi l'un contenait réellement un espace et l'autre contenait « null » ... Nous savons maintenant que lorsqu'on affiche le caractère *null*, cela s'affiche comme si c'était un espace (et qu'il est égal à \u0000).

Ce qui est réussi

Premièrement, nous avons réussi à créer d'autres fichiers textes avec le résultat que nous espérions avoir. Nous pensons ainsi avoir une bibliothèque qui permet de réaliser plusieurs petits montages. Nous n'avons pas de problèmes de référence et nous pouvons réutiliser un film déjà employé dans un autre film sans soucis (en respectant les précisions de la documentation). Nous avons fait de notre mieux pour avoir une bibliothèque assez stable, qui repose sur une interface ce qui permet d'obtenir un code polymorphe.

- VERS DES AMELIORATIONS

En revanche, même si nous sommes contents d'avoir réussi à implémenter des copies pour éviter les problèmes de référence, nous sommes au courant que cette solution possède un coût en mémoire, que ce n'est pas la meilleure solution mais nous n'avons pas su trouver un autre moyen. C'est un peu dommage, peut-être, d'avoir dû concevoir une nouvelle interface seulement pour pouvoir copier un film, et que tous les films se reposent maintenant sur cette interface et non plus sur Film. Concernant les tests unitaires, il est sûrement possible de les améliorer et peut-être éviter de créer des méthodes statiques dans nos classes.