Code Java : Test et Intégration

Classe Livre (abstraite)

```java
package book;

public abstract class Livre {


	protected String titleBook ;
	/*
	 * Title of the book
	 */
	protected String authorBook;
	/*
	 * Author of the book
	 */
	protected String summaryBook;
	/*
	 * Summary of the book
	 */
	protected String pathToImage;
	/*
	 * It is the path to get the picture of a book
	 */


	public Livre(String title, String author, String summary)
	{
		this.titleBook = title;
		this.authorBook = author;
		this.summaryBook = summary;
	}

	public Livre(String title)
	{
		this.titleBook = title;
		getDatasBook(title);
	}

	public abstract void getDatasBook(String title);
	/*
	 * Initialise the author, summary, image and/or the book itself
	 */

	// Getters and Setters

	public String getTitleBook() {
		return titleBook;
	}
	public void setTitleBook(String titleBook) {
		this.titleBook = titleBook;
	}
	public String getAuthorBook() {
		return authorBook;
	}
	public void setAuthorBook(String authorBook) {
		this.authorBook = authorBook;
	}
	public String getSummaryBook() {
```

```
            return summaryBook;
      }
      public void setSummaryBook(String summaryBook) {
            this.summaryBook = summaryBook;
      }

}
```

Classe LivreLibrary

```
package book;

public class LivreLibrary extends Livre{

      public LivreLibrary(String title, String author, String summary)
      /*
       * Here is the class that represents a book in the servor
       */

      {
            super(title, author, summary);

      }

      @Override
      public void getDatasBook(String title)
      /*
       * Save the author, summary and image of the book by its title
       */
      {
            // TO BE COMPLETED

      }

      public LivreMyCollection getBook(String link)
      /*
       * The link to download the book
       */
      {
            // TO BE COMPLETED


            LivreMyCollection newBook = new LivreMyCollection("", "", "");
            return newBook;

      }

}
```

Classe LivreMyCollection

```
package book;

import java.util.ArrayList;
import ambiance.Ambiance;
import ambiance.AmbianceOlfactive;
import ambiance.AmbianceSonore;

public class LivreMyCollection extends Livre
/*
 * It represents the book that will be saved by the user
 */
{

	private String contentBook;
	/*
	 * The path to get the .epub or .txt of the book
	 */

	private ArrayList<String> currentText ;
	/*
	 * It represents the current page of the user
	 */

	private int numberOfPage ;
	/*
	 * It represents the number of page of the book
	 * It depends of how is splited the book by the
	 * application
	 */

	private int currentNumberPage;
	/*
	 * It represents the counter of the page of the user
	 */

	private ArrayList<AmbianceOlfactive> ambianceOlfactiveBook;
	/*
	 * This is the big list where will be all the smell atmosphere of the full book
	 */
	private ArrayList<AmbianceSonore> ambianceSonoreBook;
	/*
	 * This is the big list where will be all the song atmosphere of the full book
	 */

	private ArrayList<AmbianceOlfactive> currentAmbianceOlfactive;
	/*
	 * The current list of olfactive atmosphere
	 */
	private ArrayList<AmbianceSonore> currentAmbianceSonore;
	/*
	 * The current list of song atmosphere
	 */


	public LivreMyCollection(String title, String author, String summary) {
		super(title, author, summary);
		currentNumberPage = 0;
	}

	@Override
	public void getDatasBook(String title)
	/*
```

```
	 * Save the author, summary, image and .epub/.txt of the book by its title
	 */
	{
		// TO BE COMPLETED

	}

	public void updateAtmosphere()
	/*
	 * Update the current atmosphere with the good context of the book
	 */
	{
		// TO BE COMPLETED
	}


	// Getters and Setters

	public ArrayList<AmbianceOlfactive> getCurrentAmbianceOlfactive() {
		return currentAmbianceOlfactive;
	}



	public void setCurrentAmbianceOlfactive(ArrayList<AmbianceOlfactive>
currentAmbianceOlfactive) {
		this.currentAmbianceOlfactive = currentAmbianceOlfactive;
	}



	public ArrayList<AmbianceSonore> getCurrentAmbianceSonore() {
		return currentAmbianceSonore;
	}



	public void setCurrentAmbianceSonore(ArrayList<AmbianceSonore>
currentAmbianceSonore) {
		this.currentAmbianceSonore = currentAmbianceSonore;
	}



	public ArrayList<String> getCurrentText() {
		return currentText;
	}

	public void setCurrentText(ArrayList<String> currentText) {
		this.currentText = currentText;
	}
	public int getNumberOfPage() {
		return numberOfPage;
	}

	public void setNumberOfPage(int numberOfPage) {
		this.numberOfPage = numberOfPage;
	}
}
```

Classe Ambiance

```
package ambiance;

public abstract class Ambiance {

	protected String path;

	public Ambiance(String path)
	{
		this.path = path;
	}

	public abstract void play();
	/*
	 * It will display the atmosphere
	 */
}
```

Classe Ambiance Sonore

```
package ambiance;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import javazoom.jl.decoder.JavaLayerException;
import javazoom.jl.player.Player;

public class AmbianceSonore extends Ambiance{


	public AmbianceSonore(String path) {
		super(path);

	}

	public void play()
	/*
	 * It displays the song in the computer for the moment
	 */
	{
		try
		{
			File file = new File(this.path);
```

```
            FileInputStream fis = new FileInputStream(file);
            BufferedInputStream bis = new BufferedInputStream(fis);

            try
            {
                Player player = new Player(bis);
                player.play();

            }catch (JavaLayerException e )
            {
                System.out.println("can not open the file");
            }

        } catch (IOException e)
        {
            e.printStackTrace();
        }

    }

    public String getPath()
    {
        return path;
    }


    }
```

Classe Ambiance Olfactive

```
package ambiance;

public class AmbianceOlfactive extends Ambiance {

      public AmbianceOlfactive(String path) {
            super(path);

      }

      @Override
      public void play() {
            /*
             * To be completed
             * It requires a connection to the Raspberry Pi
             */
      }

}
```

Classe User

```
package user;

import java.awt.print.Book;
import java.util.ArrayList;

import ambiance.AmbianceSonore;
import ambiance.AmbianceOlfactive;
import book.LivreLibrary;
import book.LivreMyCollection;
import book.Livre;
public class User {

	private LivreMyCollection currentBook;
	/*
	 * This is the book that is currently open by the user
	 */
	private ArrayList<LivreMyCollection> bookSavedByUser;
	/*
	 * This is the list of books that was downloaded by the user
	 */

	public User()
	{

	}


	public boolean isConnectedRaspberryPi()
	{
		// TO BE COMPLETED
		return false;
	}

	public boolean displaySongWithoutRP(ArrayList<AmbianceSonore> currentSong )
	/*
	 * Display the song on the phone
	 * Return true if it has been done correctly
	 */
	{
		// TO BE COMPLETED
		return false;
	}

	public boolean displaySongRP(ArrayList<AmbianceSonore> currentSong )
	/*
	 * Display the song on the raspberry Pi
	 * Return true if it has been done correctly
	 */
	{

		if (!isConnectedRaspberryPi())
		{
			System.out.println("The raspberry Pi is not connected");
		}

		// TO BE COMPLETED
		return false;
	}
```

```
public boolean displaySmellRP(ArrayList<AmbianceOlfactive> currentSong )
/*
 * Send the datas of the current smell atmosphere
 * Return true if it has been done correctly
 */
{
	// TO BE COMPLETED
	return false;
}



public boolean downloadBook(LivreLibrary bookToDownload)
/*
 * The user wants to save and download this book
 * Return true if it has been done correctly
 */
{
	// TO BE COMPLETED
	bookSavedByUser.add(bookToDownload.getBook(""));
	return false;
}

public int getSizeBook(Livre book)
/*
 * Return the size of the book that it takes in the application
 */
{
	// TO BE COMPLETED

	return -1;
}



// Getters and Setters

public LivreMyCollection getCurrentBook() {
	return currentBook;
}

public void setCurrentBook(LivreMyCollection currentBook) {
	this.currentBook = currentBook;
}

public ArrayList<LivreMyCollection> getBookSavedByUser() {
	return bookSavedByUser;
}

public void setBookSavedByUser(ArrayList<LivreMyCollection> bookSavedByUser) {
		this.bookSavedByUser = bookSavedByUser;
	}

}
```

Classe UISound (interface graphique de lecture de son)

```
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

import ambiance.AmbianceSonore;
import javazoom.jl.decoder.JavaLayerException;
import javazoom.jl.player.Player;

public class UISound extends JFrame implements Runnable{


	private AmbianceSonore mySound;

	private JButton search,play,stop;

	private JPanel panel1,panel2,panel;

	private boolean onPlay = false;

	private Thread t;

	private PlaySound currentPlay;

	public UISound()
	{
		super ("Listen Songs");


	}

	public void initialise()
	{
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setResizable(true);
		this.setSize(500, 100);
		panel = new JPanel();
		panel1 = new JPanel();
		panel2 = new JPanel();

		this.setLayout(new GridLayout(2, 3));
		search = new JButton("Search");
		search.setSize(new Dimension(10,10));
		play = new JButton("Play");
		play.setSize(new Dimension(10,10));
		stop = new JButton("Stop");
		stop.setSize(new Dimension(10,10));

		panel.add(search);
		panel1.add(play);
		panel2.add(stop);

		this.add(panel);
		this.add(panel1);
		this.add(panel2);

		search.addActionListener(new Search());
		play.addActionListener(new Play());
		this.setVisible(true);
	}

		private void createError1()
		{
			String string2 = "Erreur de Lecture";
			JOptionPane.showMessageDialog(this, "Aucun fichier n'a été ajouté",
					string2, JOptionPane.WARNING_MESSAGE);
		}


		private void addPath()
		{
			this.add(new JLabel(mySound.getPath()));
			this.validate();
		}



	public class Play implements ActionListener
	{

		@Override
		public void actionPerformed(ActionEvent e) {
			if (mySound==null)
			{
				createError1();
				}
				else
				{
					currentPlay = new PlaySound();
					t = new Thread(currentPlay);
					play.setEnabled(false);
					t.start();

				}

			}


		}

		public class Search implements ActionListener
		{

			@Override
			public void actionPerformed(ActionEvent e) {
				JFileChooser fc = new JFileChooser();
				int result = fc.showOpenDialog(null);
				if (result== JFileChooser.APPROVE_OPTION)
				{
					mySound = new
	AmbianceSonore(fc.getSelectedFile().getAbsolutePath());
					addPath();
					onPlay = true;
				}

			}

		}

		public class Stop implements ActionListener
		{

			@Override
			public void actionPerformed(ActionEvent e) {
				if (t!=null)
				{
					currentPlay.stopThread();
					play.setEnabled(true);
				}

			}

		}



		public class PlaySound implements Runnable
```

```
{
        private boolean isPlayed = false;
        private void play()
        {

        while (isPlayed)
        {

        try
        {

                File file = new File(mySound.getPath());
                FileInputStream fis = new FileInputStream(file);
                BufferedInputStream bis = new BufferedInputStream(fis);

                try
                {
                        Player player = new Player(bis);
                        player.play();

                }catch (JavaLayerException e )
                {
                        System.out.println("can not open the file");
                }

        } catch (IOException e)
        {
                e.printStackTrace();
        }


        }
        }
        @Override
        public void run() {
                isPlayed = true;
                this.play();

        }
        public synchronized void stopThread()
        {
                this.isPlayed = false;
        }

}



@Override
public void run() {
        initialise();
```

```
	}
}
```

Classe Tests

```
import java.util.ArrayList;
import ambiance.*;
import book.*;
import javazoom.jl.decoder.JavaLayerException;
import javazoom.jl.player.Player;
import user.*;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.BufferedInputStream;
import java.io.File;
import javax.swing.JFileChooser;
import java.lang.Thread;


public class Tests {

	public static void main(String[] args) {


		User userExample = new User();
		String path = "../../ambianceMp3/Car.mp3";

		class PlaySound implements Runnable
			{
				private boolean isPlayed = false;
				private void play()
				{

				while (isPlayed)
				{

				try
				{

					File file = new File(path);
					FileInputStream fis = new FileInputStream(file);
					BufferedInputStream bis = new BufferedInputStream(fis);

					try
					{
						Player player = new Player(bis);
						player.play();

					}catch (JavaLayerException e )
					{
						System.out.println("can not open the file");
					}

				} catch (IOException e)
```

```
                        {
                                e.printStackTrace();
                        }


                        }
                        }
                        @Override
                        public void run() {
                                isPlayed = true;
                                this.play();

                        }
                        public void stopThread()
                        {
                                this.isPlayed = false;
                        }

                }

        Thread t = new Thread(new UISound());
        t.start();

    }


    // TESTS


    public long downloadTimeTest(User user, LivreLibrary bookToDownload)
    /*
     * Gives the time to download a book
     */
    {
        long startTime = System.currentTimeMillis();
        user.downloadBook(bookToDownload);
        long stopTime = System.currentTimeMillis();
        return (stopTime - startTime);
    }

    public boolean isDisplayedWithoutBox(User user)
    /*
     * Returns a boolean : true if the action has been well made
     */
    {

        boolean bool =
user.displaySongWithoutRP(user.getCurrentBook().getCurrentAmbianceSonore());
        return bool;
    }
```

```
	public boolean isSavedCurrentBook(User user)
	/*
	 * Returns if a new book is well saved
	 */
	{
		LivreMyCollection bookForTest = new LivreMyCollection("", "", "");
		user.setCurrentBook(bookForTest);
		return (user.getCurrentBook()==bookForTest);
	}

	public boolean isSavedManualMode(User user)
	/*
	 * Returns if the modification of the current Atmospheres is well saved
	 */
	{
		ArrayList<AmbianceSonore> ambianceToTest = new
ArrayList<AmbianceSonore>();
		LivreMyCollection currentBook = user.getCurrentBook();
		currentBook.setCurrentAmbianceSonore(ambianceToTest);
		return (currentBook.getCurrentAmbianceSonore()==ambianceToTest);
	}




}
```

Classe Main

```
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import javax.swing.JFileChooser;

import ambiance.AmbianceSonore;
import javazoom.jl.decoder.JavaLayerException;
import javazoom.jl.player.Player;

public class Main {

	public static void main(String[] args) {

		JFileChooser fc = new JFileChooser();
		/*
		 * To open and search the .mp3
		 */

		class PlaySound implements Runnable
			{
				private boolean isPlayed = false;
				/*
		 * Boolean to specify if the .mp3 is played or not
		 */
		private void play()
		{

		while (isPlayed)
		{

		try
		{
			int result = fc.showOpenDialog(null);
			if (result== JFileChooser.APPROVE_OPTION)

			{
			File file = new File(fc.getSelectedFile().getAbsolutePath());
			FileInputStream fis = new FileInputStream(file);
			BufferedInputStream bis = new BufferedInputStream(fis);

			try
			{
				Player player = new Player(bis);
				player.play();

			}catch (JavaLayerException e )
			{
				System.out.println("can not open the file");
			}
			}
		} catch (IOException e)
		{
			e.printStackTrace();
		}
		}


		}
		@Override
		public void run() {
			/*
			 * Play the .mp3
			 */
			isPlayed = true;
			this.play();

		}
		public void stopThread()
		{
			this.isPlayed = false;
		}

	}
```

```
                PlaySound mySong = new PlaySound();
                Thread t = new Thread(mySong);
                t.start();
                mySong.stopThread();

        }

}
```