UNIVERSITÉ SORBONNE PARIS NORD
GALILEE INSTITUTE
Master 3IR – M2

Java Multimedia

Horloge Analogique et Numerique

Instructors: **Professeur Jalel BEN OTHMAN**

Students: **Tai NGUYEN** **12108339**

Villetaneuse, January 2022

Contents

List of Figures

List of Tables

1 Objective

It is requested to implement a clock which following the click on the mouse allows a analog or digital display (display must include hours, minutes, seconds). In addition, when launching the application, ask the user to enter the time (ie hour: minutes: seconds).

2 Brief history of clocks

The English word "clock" replaced the Old English word "daegmael" meaning day measure. The word "clock" comes from the French word "cloche" meaning bell, which enters the language around the 14th century, around the time when clocks started hitting the mainstream.

2.1 Sundials

As early as 3500 Before Common Era (BCE), the Egyptians used sun clocks to divide the day into parts. The shadow cast by an obelisk, which had been carefully constructed and geographically positioned, enabled people to divide the day into morning and afternoon. The addition of markers to the base of the obelisk could indicate further subdivisions of the day.

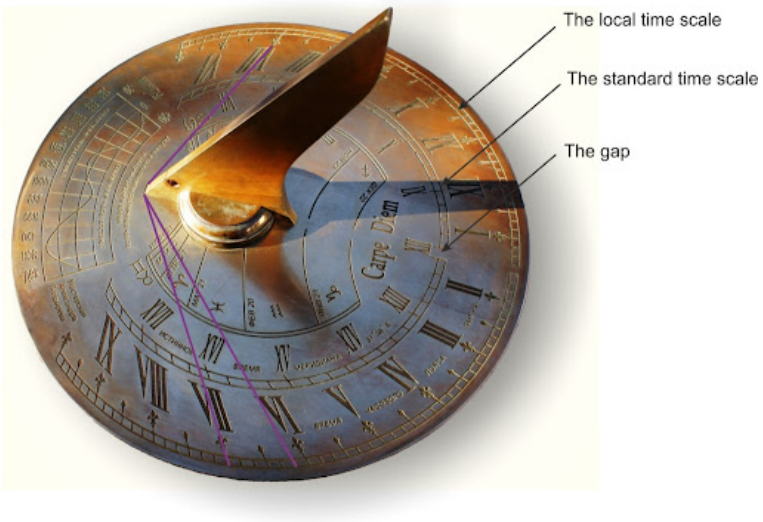

Figure 1: The construction of a typical horizontal sundial

2.2 Water clock

The Greeks began using the Clepsydra, or water clock around 325 BCE. The clepsydra works on the simple principle of the flow of water either into or out of, a container. The water would drip at a nearly constant rate from a small hole near the base of the container. Markings on the side of the container measured the hours. As you can imagine, this was not a very accurate method of calculating time, as water tends to drip faster when the container is full, due to pressure, while it will drip more slowly as the container empties.

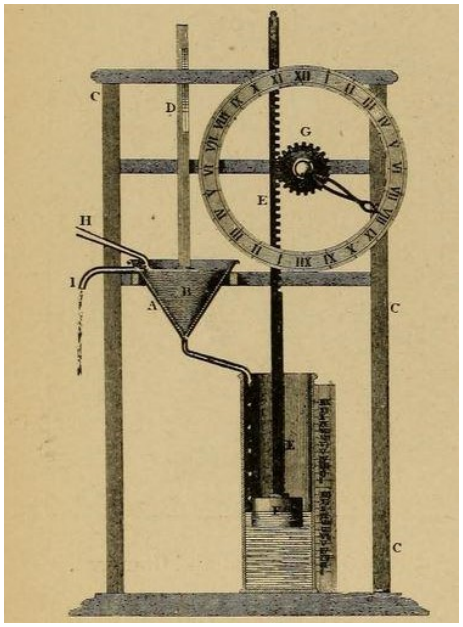

Figure 2: On the left is Water clock- A form of Clepsydra, and On the right is Candle clock stock illustration

2.3 Candle clocks

During the third-century Common Era (CE), the Chinese further developed the clepsydra to drive various mechanisms, which illustrated astronomical phenomena. Su Sung built an elaborate clock tower, which stood over 30 feet in height, with doors that opened to reveal manikins that rang bells or gongs, or held tablets indicating the hour.

2.4 Hourglass

Hourglasses were the first dependable, reusable, reasonably accurate, and easily constructed time-measurement devices. From the 15th century onwards, hourglasses were used primarily to tell time while at sea. An hourglass comprises two glass bulbs connected vertically by a narrow neck that allows a regulated trickle of material, usually sand, from the upper bulb to the lower one. Hourglasses are still in use today.

Figure 3: On the left is Hourglass illustration, and On the right is Bracket Clock in Georgian-London

2.5 Monastery clocks and clock Towers

Church life and specifically monks calling others to prayer made timekeeping devices a necessity in daily life. The earliest medieval European clockmakers were Christian monks. The first recorded clock was built by the future Pope Sylvester II around the year 996.

Figure 4: Cathedral of Our Lady of Strasbourg (Cathédrale Notre Dame de Strasbourg)

3 Developing analog clock application

According to the requirements from the Objective, I divide the project into 2 main parts. The first one is the time picker module which allows the user to set up the time for the clock, and another is the clock display module. Before starting the project, I have to convert the requirements of the Objective to user-friendly interfaces that enable users to understand how to use the product. The result of UI/UX design is shown from the Figure 5 to the Figure 7.

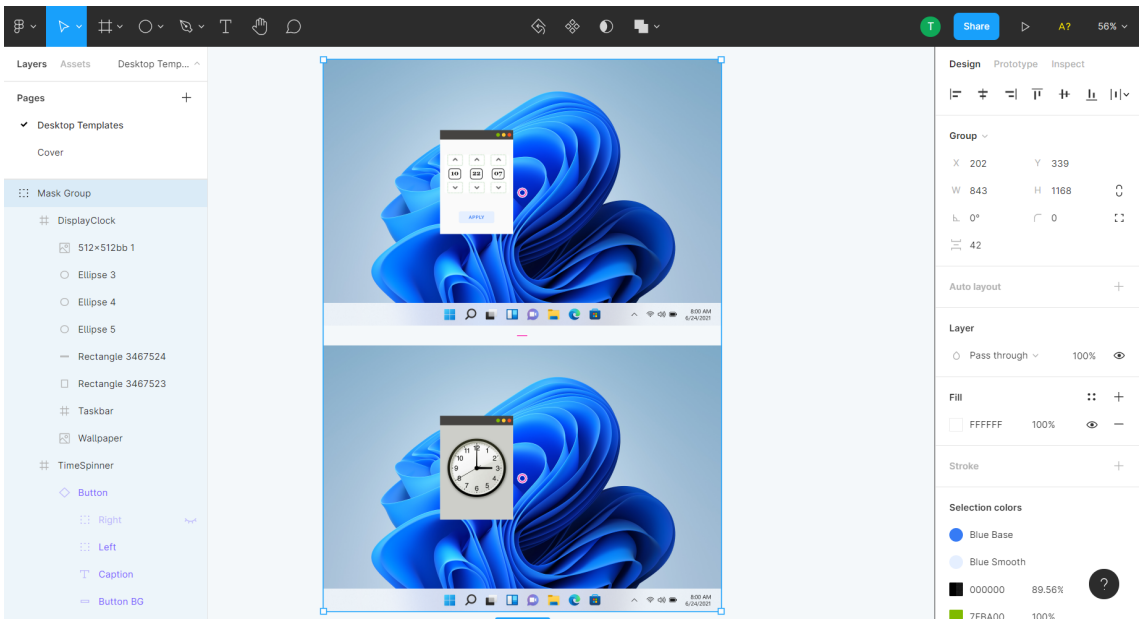

Figure 5: Design UI for Analog clock application using Figma

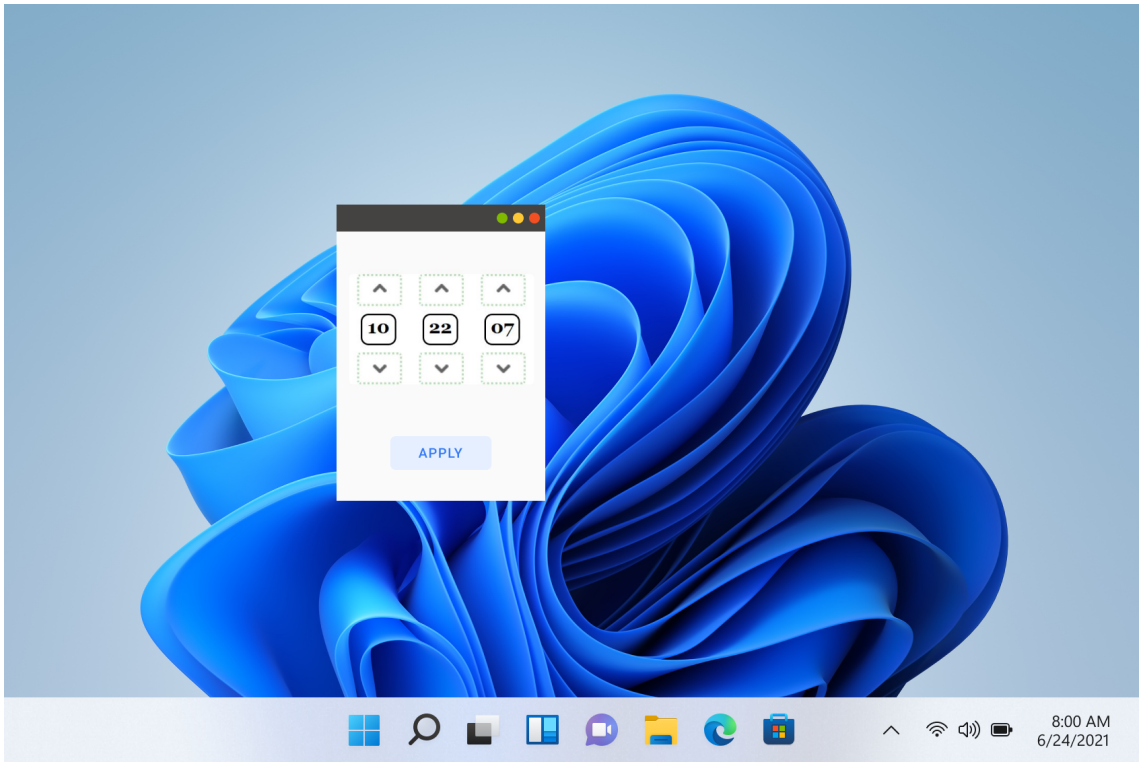

Figure 6: Design UI layer 1

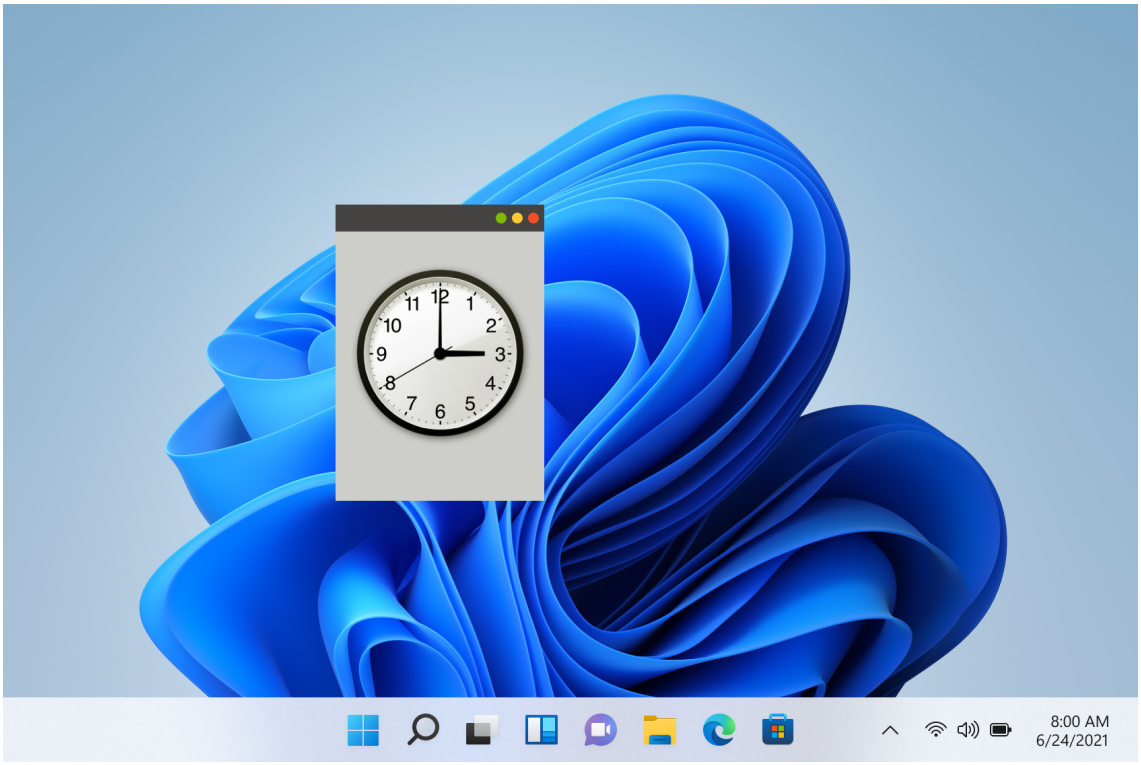

Figure 7: Design UI layer 2

3.1 Environment and Tool

The first time to build a Java project, I set up the latest Java version, but a problem appears that those versions from Java SE 11 do not support applets. So I reinstall the older version is Java SE 8. To compile and pack up Java source code, I use the IntelliJ IDEA of JetBrains.

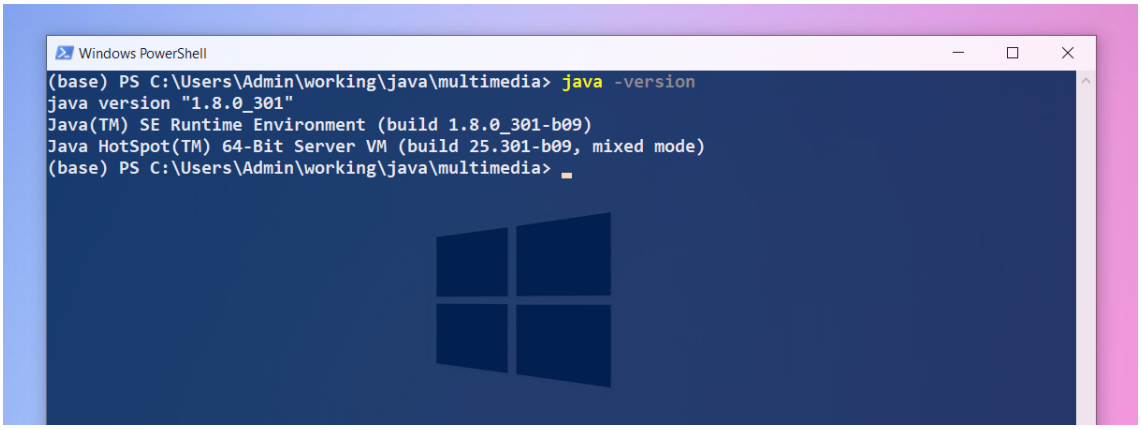

Figure 8: Check version of Java

3.2 Time picker

3.2.1 Challenges and Solution

After considering the requirement, if I set the textbox which users can type from their keyboard, I have to face a challenge to verify what they type.
I have many years as a user to experiment with many widget clocks on the mobile phone as well as the desktop clock on different operating systems. So the best way to users set up the time is to give them a spinner. Additionally, the strength of the spinner is it has a start point and a finish point.

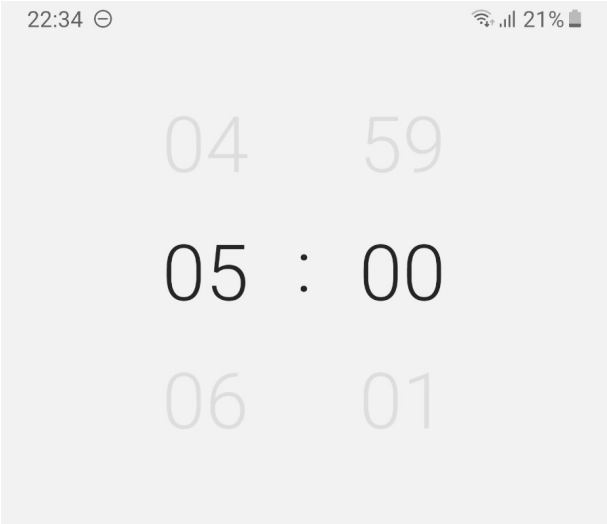

Figure 9: The clock spinner on the mobile application

3.2.2 Implementing the TimeSpinner Class

Inheriting the Spinner class from the JavaFX framework because default of JavaFx did not support Time-spinning. Spinner is similar to ComboBox or List, it allows user to choose from a set of values. Like the editable ComboBox, the Spinner also allows the user to add values. Unlike the ComboBox, the Spinner does not have a drop-down list, it does not display a list of possible values, it only displays one current value at a time. It is often used as an alternative to ComboBox or List when the set of possible values is very large.

```
// Import the Spinner class
import javafx.scene.control.Spinner;
// Import the LocalTime class
import java.time.LocalTime;
public class TimeSpinner extends Spinner<LocalTime> {
    ...
}
```

In the spinner, I use the **enum** class to switch the edit mode between “hour”, “minute”, and “second”. An **enum** is a special class that represents a group of constants (unchangeable variables, like final variables).
To indicate the type of the wrapped Object for the TimeSpinner Class, I adopt the LocalTime Class from java. This class is an immutable class that represents time with a default format of “hour-minute-second”.
At each mode of **enum**, I set three behaviors for object includes *increment*, *decrement*, and *select*. When users click on the increment/decrement buttons on the user interface that will call the method corresponding in the TimeSpinner Class. For the *decrement* method, I reuse the *increment* method with a negative step value. Besides, I use *getEditor()* to select the appropriate portion in a spinner’s editor. It returns the component that displays and potentially changes the model’s value.

```
// Method to increment the hour value
    LocalTime increment(LocalTime time, int steps) {
        return time.plusHours(steps);
    }
// Method to decrement the hour value
    LocalTime decrement(LocalTime time, int steps) {
        return increment(time, -steps);
    }
// Method to select the spinner’s edtior
```

```
    void select(TimeSpinner spinner) {
        int index = spinner.getEditor().getText().indexOf(":");
        spinner.getEditor().selectRange(0, index);
    }
```

As the common point between the ComboBox and the Spinner is editable by the keyboard, so the best approach for selecting the individual parts of the spinner's editor is to check the *caretPosition* in the editor and increment/decrement the appropriate portion as required. Additionally, I set a *TextFormatter* Class on the editor to control the allowed input. It's possible to have a formatter with just filter or value converter. If value converter is not provided however, setting a value will result in an *IllegalStateException* and the value is always null.

```
// Import the TextFormatter class
import javafx.scene.control.TextFormatter;

TextFormatter<LocalTime> textFormatter = new
    TextFormatter<LocalTime>(localTimeConverter, LocalTime.now(), c -> {
    String newText = c.getControlNewText();
    if (newText.matches("[0-9]{0,2}:[0-9]{0,2}:[0-9]{0,2}")) {
        return c ;
    }
    return null ;
}
```

3.3 Clock display

3.3.1 Challenges and Solution

For the project requirements, I just implement either the Analog style or the Digital style. So I decide to choose the Analog clock because I have many things to do with it.
One of the problems is to calculate the coordinates formed by the hand of the clock. Consider on a unit circle, it has 60 intervals of the minute-hand and second-hand corresponding with 360 degrees or 2π, so each interval is $\frac{\pi}{30}$. Similarly, each interval of hour-hand is $\frac{\pi}{6}$, but it will be more complex because I want to simulate it as a lifelike clock, so I add an offset from the minute value to create a distortion.

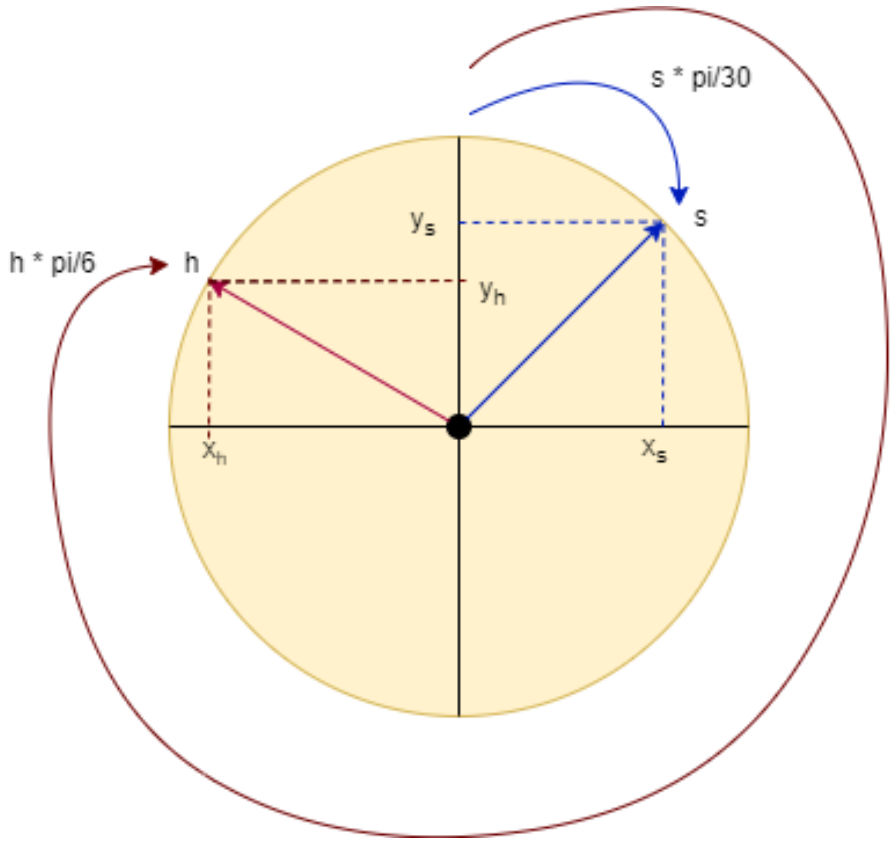

Figure 10: Illustrate the angle formed by the hands of the clock

3.3.2 Implementing the DisplayClock Class

Drawing the hands of the clock by the below coordinates:
$x_{second} = A_{second}\cos(second \times \frac{\pi}{30} - \frac{\pi}{2}) + x_{center}$
$y_{second} = A_{second}\sin(second \times \frac{\pi}{30} - \frac{\pi}{2}) + y_{center}$
$x_{minute} = A_{minute}\cos(minute \times \frac{\pi}{30} - \frac{\pi}{2}) + x_{center}$
$y_{minute} = A_{minute}\sin(minute \times \frac{\pi}{30} - \frac{\pi}{2}) + y_{center}$
$x_{hour} = A_{hour}\cos(hour \times \frac{\pi}{6} + minute \times \frac{\pi}{360} - \frac{\pi}{2}) + x_{center}$
$y_{hour} = A_{hour}\sin(hour \times \frac{\pi}{6} + minute \times \frac{\pi}{360} - \frac{\pi}{2}) + y_{center}$
In which, A is the amplitude of the hands of the clock and $A_{hour} < A_{minute} < A_{second}$.
To create this module I need to adopt the *Thread* and *Graphics* Classes of Java. Besides, I have to use the *JPanel* Class of Java Swing [1] to develop Graphical User Interface based application.

```
// Import the Graphics class
import java.awt.Graphics;
// Import the JPanel class from Java Swing
import javax.swing.JPanel;

/* To create the DisplayClock Class
    by inheriting from JPanel Class and Threads
*/
public class DisplayClock extends JPanel implements Runnable {
    ...
}
```

Threads allows a program to operate more efficiently by doing multiple things at the same time [2]. Additionally, it can be used to perform complicated tasks in the background without interrupting the main program. There are two ways to create a thread object, the first one is using *Runnable* interface and another is to extend from the *Thread* class. I use the *Threads* to

execute the behavior of the hands of the clock, the pseudo-code of this concept as shown in the Algorithm (1).

Algorithm 1 Pseudo code of threading concept for the Analog clock

```
thread ← new Thread
thread.start()
while thread do
  calculating the hands coordinate of the clock.
  delay 1000 millisecond.
  repaint the clock.
  if second < 59 then
    increase second
  else
    second ← 0
    if minute < 59 then
      increase minute
    else
      minute ← 0
      if hour < 11 then
        increase hour
      else
        hour ← 0
      end if
    end if
  end if
end while
thread ← ∅
return
```

In this experiment, I implement some methods and submethods in the DisplayClock Class, which include: drawClockFace, drawNumberClock, drawHands.

```
private void drawClockFace(Graphics g) {
    /*
    To draw the face of the clock.
    It has tick marks cooresponding with 1 second or 1 minute.
    The arguments include: graphic object.
    */
    ...
}

private void drawNumberClock(Graphics g) {
    /*
    To draw the number to the face clock.
    The arguments include: graphic object.
    */
    ...
}

private void drawHands(Graphics g,
                       int hour,
                       int minute,
                       int second,
                       Color colorSecond,
                       Color colorMHour) {
    /*
    To draw the hands of the clock.
```

```
        The arguments include: graphic object, the time values, the hands color.
        */
        ...
    }
```

3.4 Ensemble modules

In this section, I build a Main Class to ensemble the TimeSpinner Class and DisplayClock Class. As I described in the Figure 5, I need an "Apply" button as a confirmed statement from the users, so I adopt the *Button* Class from the Java Swing. Additionally, I want to set an action for this button, when users click on it the time spinner window will be closed and a clock will be displayed. To make this event, I have to use the *ActionEvent* and *EventHandler* Classes of Java Swing. After that, I add this button on the same dialog window with the time spinner.

```
    //Import the Button Class
    import javafx.scene.control.Button;
    //Import classes to execute the button action
    import javafx.event.ActionEvent;
    import javafx.event.EventHandler;

    //Create a Class to run the TimeSpinner Class
    public static class RunTimeSpinner extends Application {
        TimeSpinner spinner = new TimeSpinner();
        Button btn = new Button("Apply");
        ...
    }
```

For getting the time value from the spinner, I just use the attribute *spinner.getValue()* and convert its format cooresponding with hour, minute, and second. Besides, I pass the time value into the *Main* method by using the static variable.
In the *Main* method, I recall the *RunTimeSpinner* by using the attribute *Application.launch()*. When the time picker is closed by users, the other window will be created by inheriting from *JFrame()* of Java Swing which displays the analog clock. The flowchart for this procedure is shown in the Figure 11.

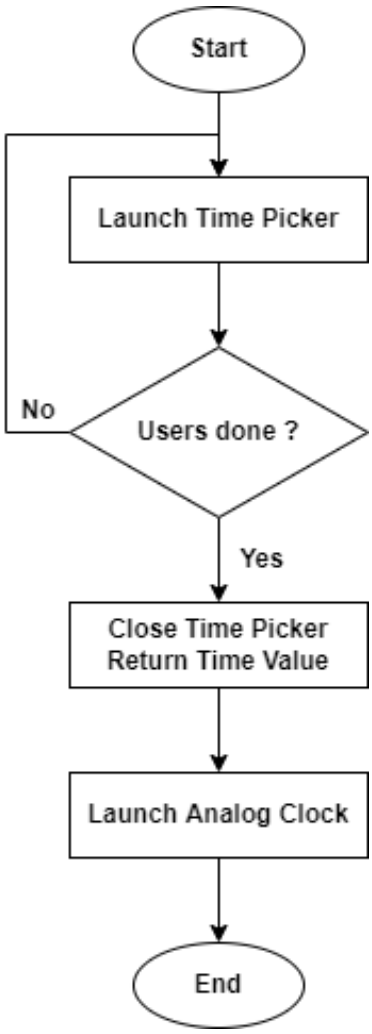

Figure 11: Proposal clock application process

3.5 Generate executable file

In this section, I create a JAR package which is executed by one mouse click to run all modules of the project. I use the IntelliJ IDEA, and I will show step by step from the Figure 12 to the Figure 17 to make it.

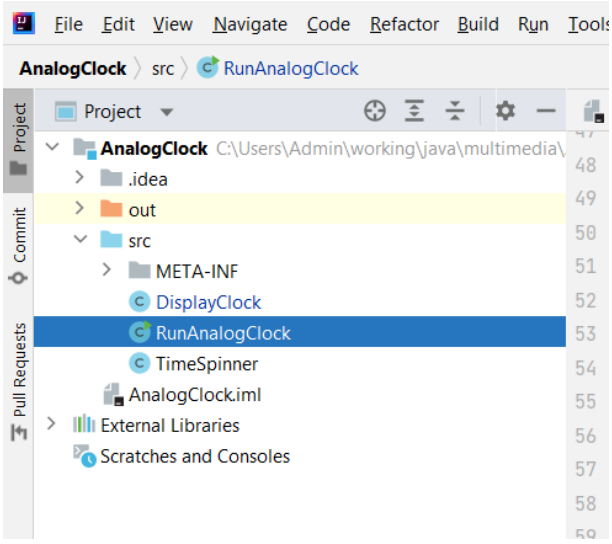

Figure 12: Step 1- Create a project by Intellij IDEA follow this structure

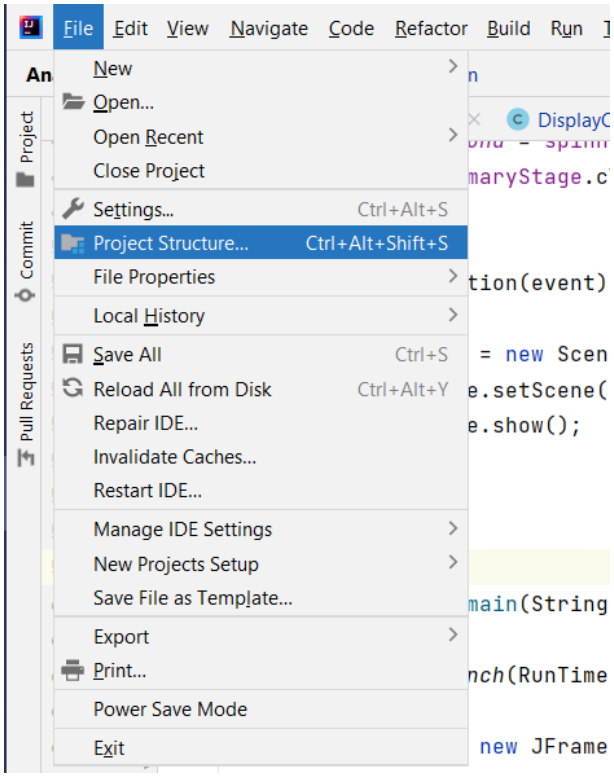

Figure 13: Step 2- Select File -> Project Structure

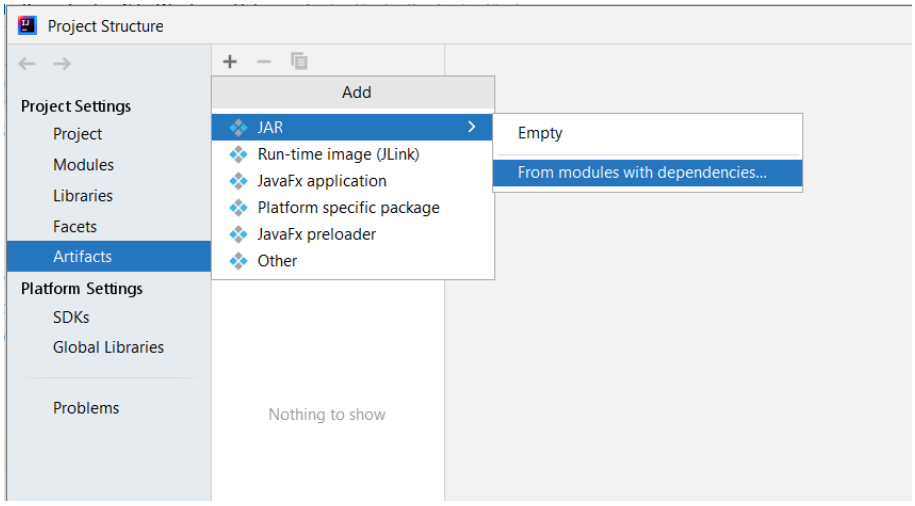

Figure 14: Step 3- In the Project Structure dialog box, select Artifacts and add new artifact

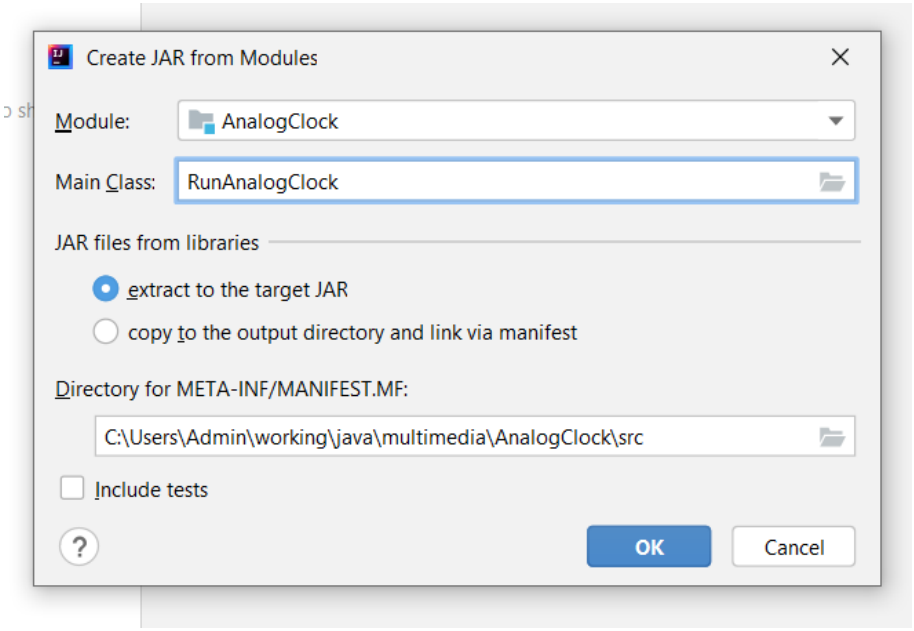

Figure 15: Step 4- Indicate the Main Class and select OK

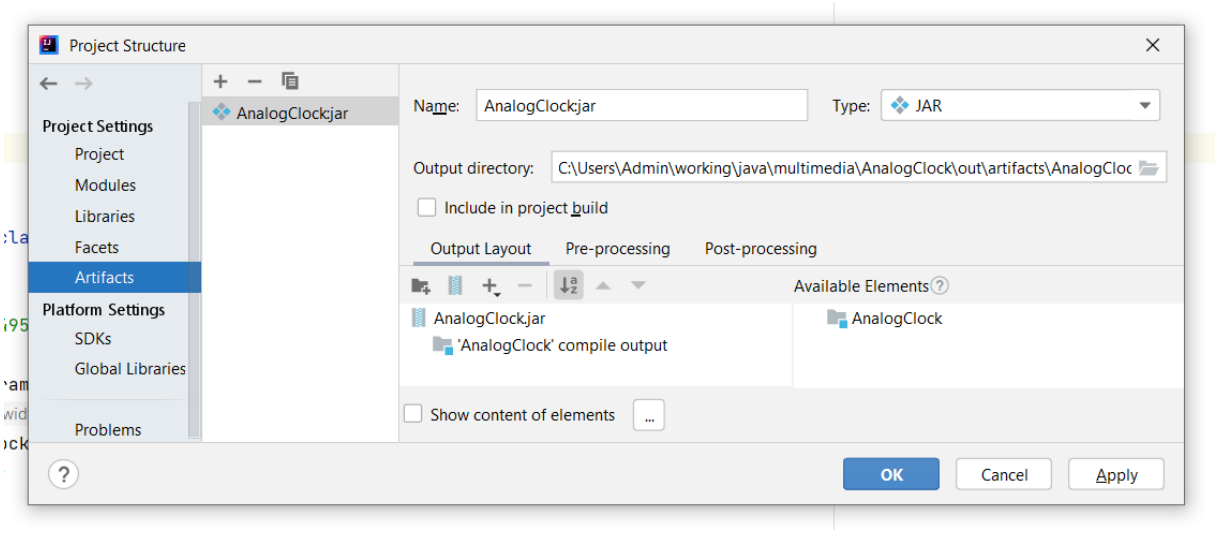

Figure 16: Step 5- Check the summary and select Apply

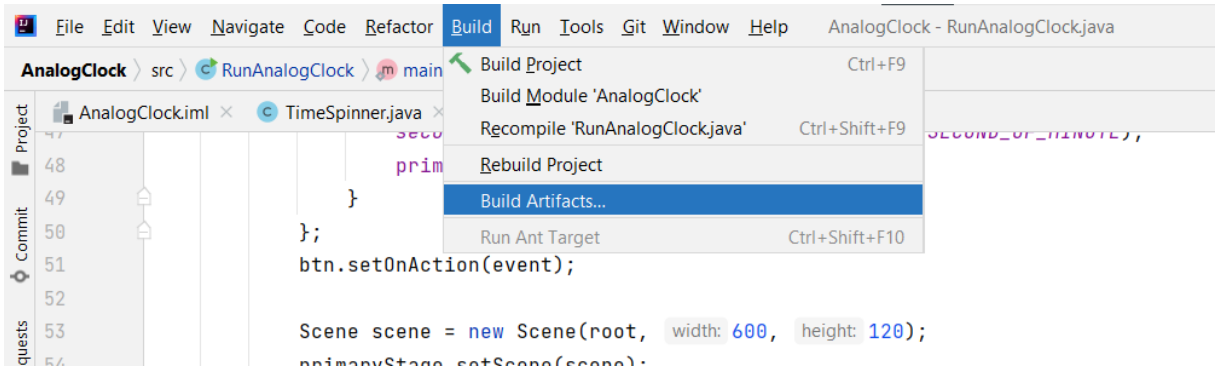

Figure 17: Step 6- Select Build -> Build Artifacts...

4 Testing application

4.1 System testing

- Performance: It makes sure that the software application performs as the requirements of the users, without depicting any defects or issues.
- Recovery: Ensures the recovery of the application is the expectations and in an accurate condition.
- Interface: System testing also focuses on the interface of the product and ensures that all requirements are met accurately and no issues occur when two components of the system are integrated together.
- Usability: This is another important aspect that is covered by system testing. It ensures the optimal user experience of the system.

Table 1: System testing

Testing conditions	**Expected results**	**Reality results**
Launch the Time picker.	Execute the application by one mouse click.	**Achieved**
Time picker editable.	Increment/Decrement button as well as input keyboard.	**Achieved**
Launch and Close the Clock interface.	Display correct time and closable	**Achieved**

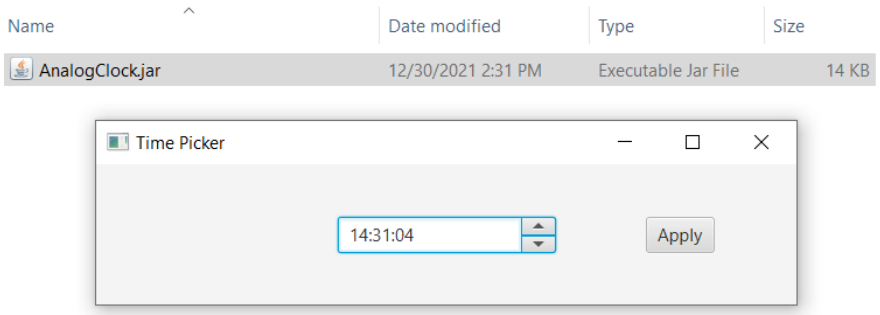

Figure 18: Launch application by JAR package

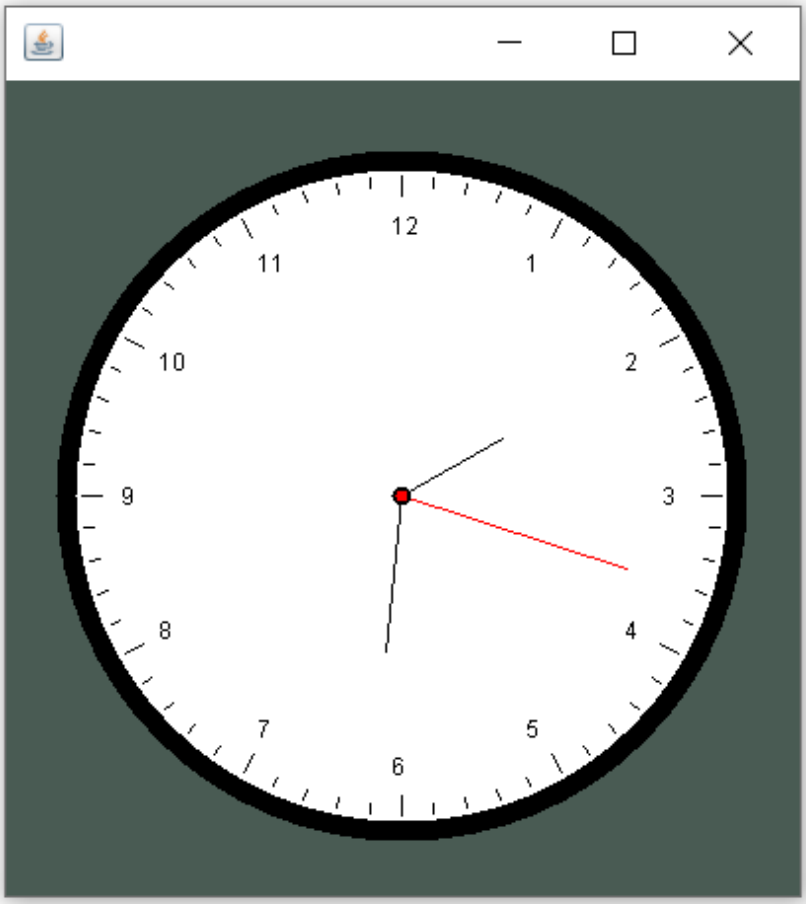

Figure 19: Analog clock application

4.2 Unit testing

This is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. I just check the basic function of methods to draw the graphics application and the result is passed.

References

[1] javapoint. "Java swing tutorial- javapoint." (), [Online]. Available: https://www.javatpoint.com/java-swing.

[2] R. Cadenhead, *Sams Teach Yourself Java in 21 Days (Covering Java 7 and Android)*, 6th. Sams publishing, 2012, ISBN: 0672335743.