Software Modeling & Analysis

Functional Digital Watch

OOPT Stage 2050, 2060 – Construct, Test

Title

용사여, 일어나시계

Date

2019-05-26

Team 4

201511246 김상재

201511272 양재민

201511292 전도현

201710515 최연지

Index

Activity 2051. Implements Class & Method Definitions

Type	Class
Name	WatchSystem
Purpose	시계의 메인 시스템으로 사용자의 반응에 따라 적절한 기능을 수행하는 클래스
Overview(Class)	
Cross Reference	Functions: All Use Cases: All
Exceptional Courses of Events	N/A

Type	Method
Name	realTimeTask
Purpose	WatchSystem에서 각 기능이 10ms 마다 반복적으로 실행되어야 하는 기능들을 실행
Overview(Method)	
Cross Reference	Functions: R1.1, R1.2, R2.1, R3.1, R4.1, R5.1, R6.1, R7.1, R8.1, R8.2, R9.2
Exceptional Courses of Events	N/A

Type	Class
Name	ModeDB
Purpose	시계의 정보들이 저장되는 Database
Overview(Class)	시계에서 기능이 삭제될 경우 기능에 대한 정보가 저장되고 시계에서 기능이 생길 경우 기능에 대한 정보를 가져올 수 있는 클래스이다.
Cross Reference	Functions: R2.3, R2.4
Exceptional Courses of Events	

Type	Class
Name	Bell
Purpose	Timer와 Alarm에서의 기능에 대한 Bell을 재생하거나 정지시키는 Class이다.
Overview(Class)	Timer와 Alarm에서의 기능에 대한 Bell을 재생하거나 정지시키는 Class이다.
Cross Reference	Functions: R5.6, R5.7, R6.4, R6.6, R6.7
Exceptional Courses of Events	N/A

Type	Class
Name	Mode Setting
Purpose	시계의 기능들을 설정하는 Class이다.
Overview(Class)	시계의 기능들을 삽입하거나 삭제하는 Class이다.
Cross Reference	Functions: R2.1, R2.2, R2.3, R2.4
Exceptional Courses of Events	N/A

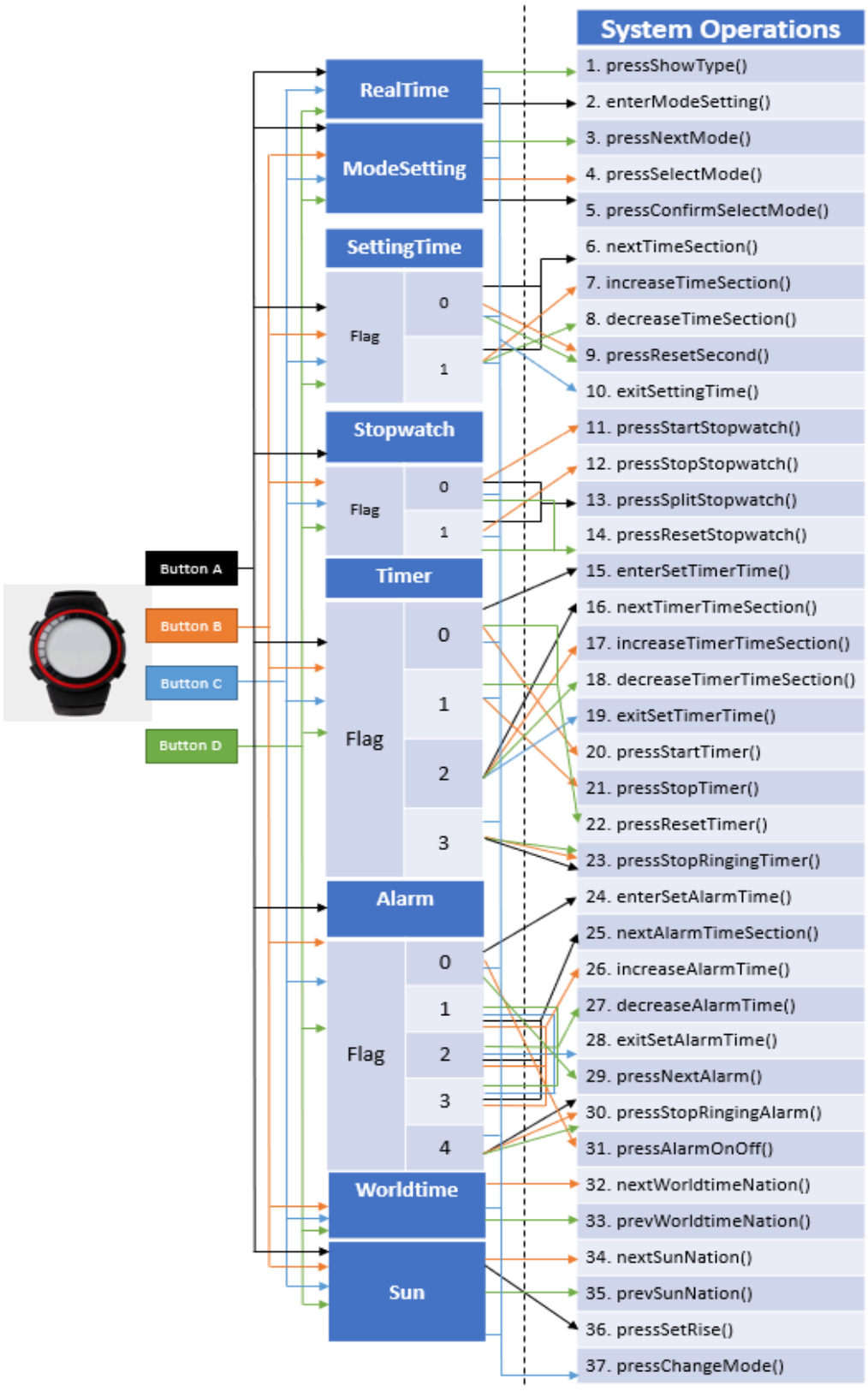
Type	Method
Name	confirmSelectMode
Purpose	사용자가 선택한 시계의 기능들을 적용하는 Method이다.
Overview(Method)	사용자가 선택한 시계의 기능들을 적용하는 Method이다. 이전에 시스템에 적용되어 있던 기능이 있을 경우 그대로 적용시키고, 만일 기능이 없으면 ModeDB에서 기능의 정보들을 가져와 생성한다. 이전 시스템에는 있으나 적용되는 기능이 아닐 경우 ModeDB에 기능의 정보를 저장한다.
Cross Reference	Functions: R2.2, R2.3, R2.4
Exceptional Courses of Events	N/A

Type	Class
Name	TimeThread
Purpose	시계의 반복연산을 위한 thread class
Overview(Class)	매 10ms 마다 WatchSystem의 realTimeTask를 실행하여 시계의 GUI를 출력하고 각 기능의 연산을 실행한다.
Cross Reference	R9.2
Exceptional Courses of Events	N/A

Type	Class
Name	WorldTime
Purpose	각 국가의 시간을 출력한다.
Overview(Class)	각 국가의 시간을 UTC/GMT +- 값을 이용하여 계산하여 출력해준다. 각 국가의 시간 기준은 각 국가의 수도로 하였다.
Cross Reference	R1.2, R7.1, R7.2, R7.3, R9.2
Exceptional Courses of Events	N/A

Type	Class
Name	Sun
Purpose	각 국가의 일출/일몰 시각을 출력한다.
Overview(Class)	각 국가의 위도와 경도 그리고 시간값을 이용하여 일출/일몰 시각을 출력한다.
Cross Reference	R1.2, R8.1, R8.2, R8.3, R8.4, R9.2
Exceptional Courses of Events	N/A

Activity 2052. Implements Windows



Name	Press A Button
Responsibilities	사용자가 A 버튼을 누른다.
Type	GUI
Cross Reference	Operations: 2, 5, 6, 13, 15, 16, 23, 24, 25, 30, 36
Notes	N.1 모드 설정에 진입 N.2 항목을 순서대로 넘김 N.3 스탑워치 작동 중 누른 시각을 기록 및 출력. N.4 타이머 설정 모드로 진입 N.5 Alarm 설정 모드로 진입 N.6 Sun Set 을 Sun Rise 로 Sun Rise 를 Sun Set 으로 변경 N.7 울리는 벨을 끈다. N.8 Select Mode 에서 설정된 모드들을 실제 모드로 설정
Pre-Conditions	N.1 ShowRealTime 모드여야 한다. N.3 스탑워치가 실행중이어야 한다. N.4 타이머 모드여야 한다. N.6 Sun 모드여야 한다. N.7 벨이 울리고 있어야 한다. N.8 Show Mode Setting 이 실행되어 있어야 한다.
Post-Conditions	N.1 사용자가 모드들을 설정할 수 있는 설정으로 들어간다. N.2 다음 Section 이 blink 된다. N.3 화면에 버튼을 누른 시각이 출력한다. N.4, N.5 설정 화면을 출력한다. N.6 변경된 시간 출력 형식으로 시간이 출력한다. N.7 벨이 울리지 않는다. N.8 Real Time 모드를 화면에 출력

Name	Press B Button
Responsibilities	사용자가 B 버튼을 누른다.
Type	GUI
Cross Reference	Operations: 4, 7, 9, 11, 12, 17, 20, 21, 23, 26, 30, 31, 32, 34
Notes	N.1 Mode list 에 사용자가 선택한 Mode 를 추가한다. N.2 선택한 섹션의 값을 1 증가시킨다. N.3 초를 0 으로 Reset 한다. N.4 스탑워치의 시간이 흐르도록 한다 N.5 스탑워치를 중지시킨다. N.6 타이머를 시작한다. N.7 타이머를 정지한다. N.8 벨을 끈다 N.9 현재 출력된 알람을 On 하거나 Off 한다. N.10 다음 Nation 을 가리킴

Pre-Conditions	N.1 Show Mode Setting 이 실행되어 있어야한다. N.2 N.3 변경 항목이 선택되어 있어야 한다. N.4 현재 모드가 Stopwatch 모드여야 한다. N.5 스탑워치가 실행 중이어야한다. N.6. Timer 가 정지되어있어야 한다. N.7 Timer 의 시간이 흘러가고 있어야한다. N.8 벨이 울리고 있어야한다. N.9 Alarm 모드여야 한다. N.10 World Time 모드 또는 Sun Rise/Set 모드여야 한다.
Post-Conditions	N.1 선택한 Mode 가 추가된 화면을 출력해준다. N.2 항목의 값이 증가했음을 화면에 출력해준다. N.3 초가 0 으로 바뀐 것 을 화면에 출력해준다. N.4 Stopwatch 가 시작되었음을 출력한다. N.5 Stopwatch 가 중지 된 것을 출력한다. N.6 Timer 가 시작되었음을 화면에 출력한다. N.7 Timer 가 정지되었음을 화면에 출력한다. N.8 벨이 울리지 않는다. N.9 알람을 On 에서 Off로 Off 에서 On으로 변경된 것을 화면에 출력한다 N.10 다음 Nation 의 World Time 또는 Sun Rise/Set 을 출력

Name	**Press C Button**
Responsibilities	사용자가 C 버튼을 누른다.
Type	GUI
Cross Reference	Operations: 10, 19, 28, 37
Notes	N.1 시간 설정, 타이머 설정, 알람 설정을 나간다. N.2 다음 모드를 실행시킨다.
Pre-Conditions	N.1 설정 모드여야 한다.
Post-Conditions	N.1 다음모드, Timer, 알람 모드로 이동 N.2 다음 모드를 실행시킨다.

Name	**Press D Button**
Responsibilities	사용자가 D 버튼을 누른다.
Type	GUI
Cross Reference	Operations: 1, 3, 8, 9, 14, 18, 22, 23, 27, 29, 30, 33, 35
Notes	N.1 24-hour 을 12-hour 로 12-hour 을 24-hour 로 출력 형식을 변경 N.2 다음 모드를 보여줌 N.3 선택된 섹션의 값을 1 감소 시킴. N.4 초를 0 으로 Reset 한다.

	N.5 StopWatch 를 reset 한다. N.6 타이머의 시간을 초기화한다. N.7 벨을 끈다. N.8 다음 알람을 가리킨다. N.9 이전 Nation 을 가리킨다.
Pre-Conditions	N.1 ShowRealTime 모드여야 한다. N.2 ModeSetting 모드여야 한다. N.3, N.4 변경 항목이 선택되어 있어야 한다. N.5 스탑워치가 정지한 상태여야 한다. N.6 Timer 가 정지되어 있어야 한다. N.7 벨이 울리고 있어야 한다. N.8 Alarm 모드여야 한다. N.9 Worldtime 모드 또는 Sun Rise/Set 모드여야 한다.
Post-Conditions	N.1 변경된 시간 출력 형식으로 시간이 출력 됨. N.2 다음 모드가 출력된다. N.3 항목의 값이 감소했음을 화면에 출력시킨다. N.4 초가 0 초로 바뀌었음을 화면에 출력시킨다. N.5 Stopwatch 가 초기화 된 화면을 출력한다. N.6 Timer 의 Time 이 User 가 설정한 값으로 초기화 됨을 출력한다. N.7 벨이 울리지 않는다. N.8 다음 알람에 대한 정보를 출력한다. N.9 이전 Nation 의 Worldtime 또는 Sun Rise/Set 을 출력한다.

d

Activity 2055. Write Unit Test Code

1. RealTimeTest

```
public class RealTimeTest {

    @Test
    public void setSecond() {
        RealTime realTime = new RealTime();
        realTime.setSecond(31); // 0 -> 31
        assertEquals(31,realTime.requestRealTime().get(Calendar.SECOND));
        assertEquals(0, realTime.requestRealTime().get(Calendar.MILLISECOND));
    }

    @Test
    public void nextSection() {
        RealTime realTime = new RealTime();
        realTime.setCurrSection(5); // [currSection] 0: Second -> 5: Year / Last Section
        realTime.nextSection(); // [currSection] 5: Year -> 0: Second
        realTime.nextSection(); // [currSection] 0: Second -> 1: Minute
        assertEquals(1, realTime.getCurrSection()); // [currSection] 1: Minute
    }

    @Test
public void increaseTime() {
    RealTime realTime = new RealTime();

    // 1: Max Minute Increase Test
    realTime.nextSection(); // 0: Second -> 1: Minute
    realTime.setRealTime(Calendar.MINUTE, 59);

    realTime.increaseTime(); // 59 -> 0
    realTime.increaseTime(); // 0 -> 1

    assertEquals(1, realTime.requestRealTime().get(Calendar.MINUTE));
    assertEquals(0, realTime.requestRealTime().get(Calendar.HOUR));

    // 2: Max Hour Increase Test
    realTime.nextSection(); // 1: Minute -> 2: HOUR
    realTime.setRealTime(Calendar.HOUR_OF_DAY, 23);

    realTime.increaseTime(); // 23 -> 0
    realTime.increaseTime(); // 0 -> 1

    assertEquals(1, realTime.requestRealTime().get(Calendar.HOUR));
    assertEquals(1, realTime.requestRealTime().get(Calendar.DATE));

    // 3: Max Date Increase Test
    realTime.nextSection(); // 2: HOUR -> 3: DATE
    realTime.setRealTime(Calendar.MONTH, Calendar.FEBRUARY);
    realTime.setRealTime(Calendar.DATE, realTime.requestRealTime().getLeastMaximum(Calendar.DATE));

    realTime.increaseTime(); // 28 -> 1
    realTime.increaseTime(); // 1 -> 2

    assertEquals(2, realTime.requestRealTime().get(Calendar.DATE));
    assertEquals(Calendar.FEBRUARY, realTime.requestRealTime().get(Calendar.MONTH));

    // 4: Max Month Increase Test
    realTime.nextSection(); // 3: DATE -> 4: MONTH
    realTime.setRealTime(Calendar.MONTH, Calendar.DECEMBER);

    realTime.increaseTime(); // 12 (DECEMBER) -> 1 (JANUARY)
    realTime.increaseTime(); // 1 (JANUARY) -> 2 (FEBRUARY)

    assertEquals(Calendar.FEBRUARY, realTime.requestRealTime().get(Calendar.MONTH));
    assertEquals(1970, realTime.requestRealTime().get(Calendar.YEAR));
}
```

```java
@Test
public void decreaseTime() {
    RealTime realTime = new RealTime();

    // 1: Min Minute Decrease Test
    realTime.nextSection(); // 0: Second -> 1: Minute
    realTime.setRealTime(Calendar.MINUTE, 0);

    realTime.decreaseTime(); // 0 -> 59
    realTime.decreaseTime(); // 59 -> 58

    assertEquals(58, realTime.requestRealTime().get(Calendar.MINUTE));
    assertEquals(0, realTime.requestRealTime().get(Calendar.HOUR_OF_DAY));

    // 2: Min Hour Decrease Test
    realTime.nextSection(); // 1: Minute -> 2: HOUR
    realTime.setRealTime(Calendar.HOUR_OF_DAY, 0);

    realTime.decreaseTime(); // 0 -> 23
    realTime.decreaseTime(); // 23 -> 22

    assertEquals(22, realTime.requestRealTime().get(Calendar.HOUR_OF_DAY));
    assertEquals(1, realTime.requestRealTime().get(Calendar.DATE));

    // 3: Min Date Decrease Test
    realTime.nextSection(); // 2: HOUR -> 3: DATE
    realTime.setRealTime(Calendar.MONTH, Calendar.FEBRUARY);
    realTime.setRealTime(Calendar.DATE, 1);

    realTime.decreaseTime(); // 1 -> 28
    realTime.decreaseTime(); // 28 -> 27

    assertEquals(27, realTime.requestRealTime().get(Calendar.DATE));
    assertEquals(Calendar.FEBRUARY, realTime.requestRealTime().get(Calendar.MONTH));

    // 4: Min Month Decrease Test
    realTime.nextSection(); // 3: DATE -> 4: MONTH
    realTime.setRealTime(Calendar.MONTH, Calendar.JANUARY);

    realTime.decreaseTime(); // 1 (JANUARY) -> 12 (DECEMBER)
    realTime.decreaseTime(); // 12 (DECEMBER) -> 11 (NOVEMBER)

    assertEquals(Calendar.NOVEMBER, realTime.requestRealTime().get(Calendar.MONTH));
    assertEquals(1970, realTime.requestRealTime().get(Calendar.YEAR));

    // 5: Min Year Decrease Test
    realTime.nextSection(); // 4: MONTH -> 5: YEAR
    realTime.setRealTime(Calendar.YEAR, 1971);

    realTime.decreaseTime(); // 1971 -> 1970
    realTime.decreaseTime(); // 1970 -> 1970 => Not Changed

    assertEquals(1970, realTime.requestRealTime().get(Calendar.YEAR));
}


@Test
public void calculateTime() {
    RealTime realTime = new RealTime();
    realTime.requestRealTime().set(2019, 5, 22, 15, 30, 20);
    for(int i = 0; i < 20; i++)
        realTime.calculateTime();

    assertEquals(10 * 20, realTime.requestRealTime().get(Calendar.MILLISECOND));
    assertEquals(20, realTime.requestRealTime().get(Calendar.SECOND));
    assertEquals(30, realTime.requestRealTime().get(Calendar.MINUTE));
    assertEquals(15, realTime.requestRealTime().get(Calendar.HOUR_OF_DAY));
    assertEquals(22, realTime.requestRealTime().get(Calendar.DATE));
    assertEquals(5, realTime.requestRealTime().get(Calendar.MONTH));
    assertEquals(2019, realTime.requestRealTime().get(Calendar.YEAR));
}
@Test
public void requestChangeType(){
    RealTime realTime = new RealTime();
    assertEquals(true, realTime.isIs24H());
    realTime.requestChangeType();
    assertEquals(false, realTime.isIs24H());
}
@Test
public void setCurrSection(){
    RealTime realTime = new RealTime();
    assertEquals(0, realTime.getCurrSection());
    realTime.setCurrSection(3);
    assertEquals(3, realTime.getCurrSection());
}
```

2. ModeSettingTest

```
public class ModeSettingTest {

    @Test
    public void requestNextMode() {
        ModeSetting ms = new ModeSetting();
        // 1: Max currIndex Test
        ms.setCurrIndex(5);
        ms.requestNextMode(); // 5 -> 0
        ms.requestNextMode(); // 0 -> 1

        assertEquals(1, ms.getCurrIndex());

        // 2: Ignore Selected Mode
        ArrayList<String> selectedMode = new ArrayList<~>();
        selectedMode.add("Timer");
        selectedMode.add("TimeSetting");

        ms.setNewMode(selectedMode);
        ms.setCurrIndex(0); // 0: Stopwatch

        // if ms.requestNextMode then it may be 0-> (1) -> 2 -> 3 -> 4 -> (5) -> 0
        ms.requestNextMode(); // 0: Stopwatch -> (1: Timer) -> 2: Alarm
        assertEquals(2, ms.getCurrIndex());
        ms.requestNextMode(); // 2: Alarm -> 3: Worldtime
        assertEquals(3, ms.getCurrIndex());
        ms.requestNextMode(); // 3: Worldtime -> 4: Sun
        assertEquals(4, ms.getCurrIndex());
        ms.requestNextMode(); // 4: Sun -> (5: TimeSetting) -> 0: Stopwatch
        assertEquals(0, ms.getCurrIndex());
    }

    @Test
    public void requestSelectMode() {
        ModeSetting ms = new ModeSetting();

        // Make newMode's size 4
        ArrayList<String> selectedMode = new ArrayList<~>();
        selectedMode.add("SettingTime");
        selectedMode.add("Stopwatch");
        selectedMode.add("Timer");
        selectedMode.add("Alarm");

        ms.setNewMode((ArrayList<String>)selectedMode.clone()); // Avoid Call-By-Reference

        ms.setCurrIndex(4); // 4: Sun
        ms.requestSelectMode();
        // {"SettingTime", "Stopwatch", "Timer, "Alarm"}
        //                  !=
        // {"Stopwatch", "Timer", "Alarm, "Sun"}
        assertNotEquals(selectedMode, ms.getNewMode());
    }
```

```java
    @Test
    public void confirmSelectMode() throws UnsupportedAudioFileException, IOException, LineUnavailableException {
        WatchSystem sys = new WatchSystem();
        ModeSetting ms = (ModeSetting)sys.getMenu(0);
        sys.enterModeSetting();
        //ms.requestModeSetting();

        assertThat(ms.getPrevModeObject().get(0), instanceOf(SettingTime.class));
        assertThat(ms.getPrevModeObject().get(1), instanceOf(Stopwatch.class));
        assertThat(ms.getPrevModeObject().get(2), instanceOf(Timer.class));
        assertThat(ms.getPrevModeObject().get(3), instanceOf(Alarm.class));

        // Make newMode's size 4
        ArrayList<String> selectedMode = new ArrayList<~>();
        selectedMode.add("Timer");
        selectedMode.add("Stopwatch");
        selectedMode.add("SettingTime");

        ms.setNewMode((ArrayList<String>)selectedMode.clone()); // Avoid Call-By-Reference

        // {SettingTime, Timer, Stopwatch} -> {Timer, Stopwatch, SettingTime}
        sys.pressConfirmSelectMode();
        ArrayList confirmMode = (ArrayList)ms.getSys().getMenu().clone();
        assertThat(confirmMode.get(2), instanceOf(Timer.class));
        assertThat(confirmMode.get(3), instanceOf(Stopwatch.class));
        assertThat(confirmMode.get(4), instanceOf(SettingTime.class));

        // Test Save Data
        Timer timer = new Timer();
        timer.getRsvTime().clear();

        Calendar temp = Calendar.getInstance();
        temp.clear();
        temp.set( year: 2010, Calendar.FEBRUARY,  date: 3,  hourOfDay: 20, minute: 30, second: 40);
        timer.setRsvTime((Calendar)temp.clone());

        sys.setMenu(2, timer);
        assertEquals(timer.getRsvTime(), temp);

        sys.enterModeSetting();
        selectedMode.clear();
        ms.setNewMode((ArrayList<String>)selectedMode.clone()); // Delete Timer
        sys.pressConfirmSelectMode();
        assertEquals(0, sys.getMaxCnt()); // Completely Deleted

        // Timer data is saved
        assertEquals(((Calendar)ms.getDb().loadData(1).get(1)), temp);

        // Test Load data
        sys.enterModeSetting();
        selectedMode.clear();
        selectedMode.add("Timer"); // Create Timer
        ms.setNewMode((ArrayList<String>)selectedMode.clone());

        sys.pressConfirmSelectMode();
        confirmMode = (ArrayList)sys.getMenu().clone();

        assertEquals(temp, ((Timer)confirmMode.get(2)).getRsvTime());
    }
}
```

3. StopwatchTest

```
public class StopwatchTest {

    @Test
    public void realTimeTaskStopwatch() {
        Stopwatch stp = new Stopwatch();
        stp.requestStartStopwatch(); // [status] 0: Stopped -> 1: Continued

        Calendar temp = Calendar.getInstance();
        temp.clear();

        assertEquals(temp, stp.getStpTime());

        for(int i = 0; i < 10002; i++)
            stp.realTimeTaskStopwatch();

        assertEquals(-32400000 + 10 * 10002, stp.getStpTime().getTimeInMillis());
        assertEquals(20, stp.getStpTime().get(Calendar.MILLISECOND));
        assertEquals(40, stp.getStpTime().get(Calendar.SECOND));
        assertEquals(1, stp.getStpTime().get(Calendar.MINUTE));
    }

    @Test
    public void requestStartStopwatch() {
        Stopwatch stp = new Stopwatch();
        stp.requestStartStopwatch(); // [status] 0: Stopped -> 1: Continued
        assertEquals(1, stp.getStatus()); // [status] 1: Continued
    }

    @Test
    public void requestStopStopwatch() {
        Stopwatch stp = new Stopwatch();
        stp.requestStartStopwatch(); // [status] 0: Stopped -> 1: Continued
        stp.requestStopStopwatch(); // [status] 1: Continued -> 0: Stopped
        assertEquals(0, stp.getStatus()); // [status] 0: Stopped
    }

    @Test
    public void requestSplitStopwatch() {
        Stopwatch stp = new Stopwatch();
        stp.requestStartStopwatch(); // [status] 0: Stopped -> 1: Continued

        for(int i = 0; i < 10002; i++)
            stp.realTimeTaskStopwatch();

        stp.requestSplitStopwatch();

        assertEquals(-32400000 + 10 * 10002, stp.getSplitTime().getTimeInMillis());
        assertEquals(20, stp.getSplitTime().get(Calendar.MILLISECOND));
        assertEquals(40, stp.getSplitTime().get(Calendar.SECOND));
        assertEquals(1, stp.getSplitTime().get(Calendar.MINUTE));


        for(int i = 0; i < 10002; i++)
            stp.realTimeTaskStopwatch();

        stp.requestStopStopwatch(); // [status] 1: Continued -> 0: Stopped
        stp.requestSplitStopwatch(); // Not Changed

        assertNotEquals(-32400000 + 10 * 10002 + 10 * 10002, stp.getSplitTime().getTimeInMillis());
        assertEquals(20, stp.getSplitTime().get(Calendar.MILLISECOND));
        assertEquals(40, stp.getSplitTime().get(Calendar.SECOND));
        assertEquals(1, stp.getSplitTime().get(Calendar.MINUTE));
    }

    @Test
    public void requestResetStopwatch() {
        Stopwatch stp = new Stopwatch();
        stp.requestStartStopwatch(); // [status] 0: Stopped -> 1: Continued

        for(int i = 0; i < 10; i++)
            stp.realTimeTaskStopwatch();
        stp.requestStopStopwatch(); // [status] 1: Continued -> 0: Stopped
        stp.requestResetStopwatch();

        assertEquals(-32400000, stp.getStpTime().getTimeInMillis());
        assertEquals(-32400000, stp.getSplitTime().getTimeInMillis());
    }
```

4. TimerTest

```
public class TimerTest {

    @Test
    public void requestTimerTime() {
        Timer timer = new Timer();
        timer.requestTimerTime(); // [status] 0: Stopped -> 2: Setting
        assertEquals(2, timer.getStatus());
    }

    @Test
    public void requestNextTimerTimeSection() {
        Timer timer = new Timer();
        timer.setCurrSection(2); // [currSection] 0: Second -> 2: Hour / Last Section
        timer.requestNextTimerTimeSection(); // [currSection] 2: Hour -> 0: Second
        timer.requestNextTimerTimeSection(); // [currSection] 0: Second -> 1: Minute
        assertEquals(1, timer.getCurrSection()); // [currSection] 1: Minute
    }

    @Test
    public void requestIncreaseTimerTimeSection() {
        Timer timer = new Timer();

        // 1: Max Second Increase Test
        timer.getRsvTime().set(Calendar.SECOND, 59);

        timer.requestIncreaseTimerTimeSection(); // 59 -> 0
        timer.requestIncreaseTimerTimeSection(); // 0 -> 1

        assertEquals(0, timer.getRsvTime().get(Calendar.MILLISECOND));
        assertEquals(1, timer.getRsvTime().get(Calendar.SECOND));
        assertEquals(0, timer.getRsvTime().get(Calendar.MINUTE));

        // 2: Max Minute Increase Test
        timer.setCurrSection(1);
        timer.getRsvTime().set(Calendar.MINUTE, 59);

        timer.requestIncreaseTimerTimeSection(); // 59 -> 0
        timer.requestIncreaseTimerTimeSection(); // 0 -> 1

        assertEquals(1, timer.getRsvTime().get(Calendar.SECOND));
        assertEquals(1, timer.getRsvTime().get(Calendar.MINUTE));
        assertEquals(0, timer.getRsvTime().get(Calendar.HOUR_OF_DAY));

        // 3: Max Minute Increase Test
        timer.setCurrSection(2);
        timer.getRsvTime().set(Calendar.HOUR_OF_DAY, 23);

        timer.requestIncreaseTimerTimeSection(); // 23 -> 24(0)
        timer.requestIncreaseTimerTimeSection(); // 0 -> 1

        assertEquals(1, timer.getRsvTime().get(Calendar.SECOND));
        assertEquals(1, timer.getRsvTime().get(Calendar.MINUTE));
        assertEquals(1, timer.getRsvTime().get(Calendar.HOUR_OF_DAY));
        assertEquals(1, timer.getRsvTime().get(Calendar.DATE));
    }
```

```java
    @Test
    public void requestDecreaseTimerTimeSection() {
        Timer timer = new Timer();
        Calendar testTime = Calendar.getInstance();
        testTime.set( year: 1970, Calendar.JANUARY,  date: 1,  hourOfDay: 22,  minute: 37,  second: 0);
        testTime.set(Calendar.MILLISECOND, 0);
        timer.setRsvTime((Calendar)testTime.clone()); // Avoid Call-by-Reference

        // 1: Min Second Decrease Test
        timer.requestDecreaseTimerTimeSection(); // 0 -> 59
        timer.requestDecreaseTimerTimeSection(); // 59 -> 58

        assertEquals(0, timer.getRsvTime().get(Calendar.MILLISECOND));
        assertEquals(58, timer.getRsvTime().get(Calendar.SECOND));
        assertEquals(37, timer.getRsvTime().get(Calendar.MINUTE));
        assertEquals(22, timer.getRsvTime().get(Calendar.HOUR_OF_DAY));

        // 2: Min Minute Decrease Test
        timer.setCurrSection(1);
        timer.getRsvTime().set(Calendar.MINUTE, 0);
        timer.requestDecreaseTimerTimeSection(); // 0 -> 59
        timer.requestDecreaseTimerTimeSection(); // 59 -> 58

        assertEquals(0, timer.getRsvTime().get(Calendar.MILLISECOND));
        assertEquals(58, timer.getRsvTime().get(Calendar.SECOND));
        assertEquals(58, timer.getRsvTime().get(Calendar.MINUTE));
        assertEquals(22, timer.getRsvTime().get(Calendar.HOUR_OF_DAY));

        // 3: Max Minute Increase Test
        timer.setCurrSection(2);
        timer.getRsvTime().set(Calendar.HOUR_OF_DAY, 0);
        timer.requestDecreaseTimerTimeSection(); // 0 -> 23
        timer.requestDecreaseTimerTimeSection(); // 23 -> 22

        assertEquals(0, timer.getRsvTime().get(Calendar.MILLISECOND));
        assertEquals(58, timer.getRsvTime().get(Calendar.SECOND));
        assertEquals(58, timer.getRsvTime().get(Calendar.MINUTE));
        assertEquals(22, timer.getRsvTime().get(Calendar.HOUR_OF_DAY));
        assertEquals(1, timer.getRsvTime().get(Calendar.DATE));
    }
    @Test
    public void changeStatus() {
        Timer timer = new Timer();
        timer.changeStatus(1); // [status] 0: Stopped -> 1: Continued
        assertEquals(1, timer.getStatus()); // [status] 1: Continued

        timer.changeStatus(-1); // [status] 1: Continued -> -1: Not valid
        assertEquals(1, timer.getStatus()); // [status] 1: Continued
    }

    @Test
    public void requestResetTimer() {
        Timer timer = new Timer();
        Calendar testTime = Calendar.getInstance();
        testTime.set( year: 1970, Calendar.JANUARY,  date: 1,  hourOfDay: 22,  minute: 34,  second: 30);
        testTime.set(Calendar.MILLISECOND, 0);

        // [rstTime] 1970. Calendar.JANUARY. 1. 00. 00. 000 -> 1970. Calendar.JANUARY. 1. 22. 34. 30. 000
        timer.setRsvTime((Calendar)testTime.clone()); // Avoid Call-By-Reference

        // [timerTime] 1970. Calendar.JANUARY. 1. 00. 00. 000 -> 1970. Calendar.JANUARY. 1. 22. 34. 30. 000
        timer.requestResetTimer();

        assertEquals(timer.getRsvTime(), timer.getTimerTime()); // [rsvTime] == [timerTime]
    }

    @Test
    public void ringOff() {
        Timer timer = new Timer();
        timer.setStatus(3); // [status] 0: Stopped -> 3: Ringing
        timer.ringOff(); // [status] 3: Ringing -> 0: Stopped
        assertEquals(0, timer.getStatus()); // [status] 0: Stopped
    }
```

```
    @Test
    public void realTimeTimerTask() {
        Timer timer = new Timer();
        timer.getRsvTime().set(1970, 1-1, 1, 10, 30, 12);
        timer.getRsvTime().set(Calendar.MILLISECOND, 0);

        timer.requestResetTimer();
        assertEquals(timer.getRsvTime().getTimeInMillis(), timer.getTimerTime().getTimeInMillis());

        timer.changeStatus(1); // [Status] 0: Stopped -> 1: Continued
        timer.realTimeTimerTask();
        assertEquals(timer.getRsvTime().getTimeInMillis() - 10, timer.getTimerTime().getTimeInMillis());

        // Decrease timerTime to 00:00:00.000
        while(timer.getTimerTime().getTimeInMillis() > -32400000)
            timer.realTimeTimerTask();

        // [status] 1: Continued -> 3: Ringing
        assertEquals(3, timer.getStatus()); // [status] 3: Ringing
    }

    /*
    @Test
    public void showTimer() {}
    */

    @Test
    public void requestExitSetTimerTime() {
        Timer timer = new Timer();

        timer.changeStatus(2); // [status] 0: Stopped -> 2: Setting
        timer.requestNextTimerTimeSection(); // [currSection] 0: Second -> 1: Minute

        // [status] 2: Setting -> 0: Stopped / [currSection] 1: Minute -> 0: Second
        timer.requestExitSetTimerTime();

        assertEquals(0, timer.getStatus()); // [status] 0: Stopped
        assertEquals(0, timer.getCurrSection()); // [currSection] 0: Second
    }
}
```

5. AlarmTest

```
public class AlarmTest {

    @Test
    public void realTimeTaskAlarm() throws UnsupportedAudioFileException, IOException, LineUnavailableException{

    }

    @Test
    public void requestSettingAlarm() throws UnsupportedAudioFileException, IOException, LineUnavailableException {
        RealTime realTime = new RealTime();
        Alarm alarm = new Alarm(realTime);
        alarm.requestSettingAlarm();
        assertEquals(1, alarm.getStatus());
    }

    @Test
    public void requestAlarmNextSection() throws UnsupportedAudioFileException, IOException, LineUnavailableException{
        RealTime realTime = new RealTime();
        Alarm alarm = new Alarm(realTime);

        alarm.requestSettingAlarm();
        assertEquals(0, alarm.getCurrSection());    // 0 : Minute
        assertEquals(1, alarm.getStatus());         // Alarm Setting

        alarm.requestAlarmNextSection();
        assertEquals(1, alarm.getCurrSection());    // 1 : Hour

        alarm.requestAlarmNextSection();
        assertEquals(2, alarm.getCurrSection());    // 2 : Frequency_Second
        assertEquals(2, alarm.getStatus());         // Alarm Frequency

        alarm.requestAlarmNextSection();
        assertEquals(3, alarm.getCurrSection());    // 3 : Frequency_Minute

        alarm.requestAlarmNextSection();
        assertEquals(4, alarm.getCurrSection());    // 4 : Count

        alarm.requestAlarmNextSection();
        assertEquals(5, alarm.getCurrSection());    // 5 : Bell

        alarm.requestAlarmNextSection();
        assertEquals(0, alarm.getCurrSection());     // 0 : Minute
        assertEquals(1, alarm.getStatus());
    }
```

```java
@Test
public void increaseSection() throws UnsupportedAudioFileException, IOException, LineUnavailableException{
    RealTime realTime = new RealTime();
    Alarm alarm = new Alarm(realTime);
    assertEquals(0, alarm.getCurrSection()); // 0: Minutes

    alarm.requestSettingAlarm();
    alarm.increaseSection();
    assertEquals(1, alarm.getReservated(alarm.getCurrAlarm()).get(Calendar.MINUTE));

    // 1: Max Minute Increase Test
    alarm.getReservated(alarm.getCurrAlarm()).set(Calendar.MINUTE, 59);
    alarm.increaseSection();
    assertEquals(0, alarm.getReservated(alarm.getCurrAlarm()).get(Calendar.MINUTE));
    assertEquals(0, alarm.getReservated(alarm.getCurrAlarm()).get(Calendar.HOUR_OF_DAY));

    // 2: Max Hour Increase Test
    alarm.getReservated(alarm.getCurrAlarm()).set(Calendar.HOUR_OF_DAY, 23);
    alarm.getReservated(alarm.getCurrAlarm()).set(Calendar.MINUTE, 59);

    alarm.requestAlarmNextSection();
    alarm.increaseSection();

    assertEquals(0, alarm.getReservated(alarm.getCurrAlarm()).get(Calendar.HOUR_OF_DAY));
    assertEquals(59, alarm.getReservated(alarm.getCurrAlarm()).get(Calendar.MINUTE));
    assertEquals(1, alarm.getReservated(alarm.getCurrAlarm()).get(Calendar.DATE));

    // 3: Max Frequency_Second Increase Test
    alarm.requestAlarmNextSection();
    alarm.getFrequency(alarm.getCurrAlarm()).set(Calendar.SECOND, 59);
    alarm.getFrequency(alarm.getCurrAlarm()).set(Calendar.MINUTE, 8);
    alarm.increaseSection();
    alarm.increaseSection();

    assertEquals(0, alarm.getFrequency(alarm.getCurrAlarm()).get(Calendar.MILLISECOND));
    assertEquals(1, alarm.getFrequency(alarm.getCurrAlarm()).get(Calendar.SECOND));
    assertEquals(8, alarm.getFrequency(alarm.getCurrAlarm()).get(Calendar.MINUTE));

    // 4: Max Frequency_Minute Increase Test
    alarm.requestAlarmNextSection();

    // 5: Max Repeat Increase Test
    alarm.requestAlarmNextSection();
    alarm.setRepeat(alarm.getCurrAlarm(), 5);
    alarm.increaseSection();

    assertEquals(0, alarm.getRepeat(alarm.getCurrAlarm()));

    // 6: Max Bell Increase Test
    alarm.requestAlarmNextSection();
    alarm.increaseSection();    // Alarm1.wav
    alarm.increaseSection();    // Alarm2.wav
    alarm.increaseSection();    // Alarm3.wav
    alarm.increaseSection();    // Alarm4.wav
    alarm.increaseSection();    // Alarm5.wav
}
```

```java
    @Test
    public void decreaseSection() throws UnsupportedAudioFileException, IOException, LineUnavailableException{
        RealTime realTime = new RealTime();
        Alarm alarm = new Alarm(realTime);
        assertEquals(0, alarm.getCurrSection()); // 0: Minutes

        // 1: Min Minute Decrease Test
        alarm.requestSettingAlarm();
        alarm.decreaseSection();
        assertEquals(59, alarm.getReservated(alarm.getCurrAlarm()).get(Calendar.MINUTE));
        assertEquals(0, alarm.getReservated(alarm.getCurrAlarm()).get(Calendar.HOUR_OF_DAY));

        // 2: Min Hour Decrease Test
        alarm.requestAlarmNextSection();
        alarm.decreaseSection();
        assertEquals(23, alarm.getReservated(alarm.getCurrAlarm()).get(Calendar.HOUR_OF_DAY));
        assertEquals(1, alarm.getReservated(alarm.getCurrAlarm()).get(Calendar.DATE));

        // 3: Min Frequency_Second Decrease Test
        alarm.requestAlarmNextSection();
        alarm.decreaseSection();
        assertEquals(59, alarm.getFrequency(alarm.getCurrAlarm()).get(Calendar.SECOND));
        assertEquals(0, alarm.getFrequency(alarm.getCurrAlarm()).get(Calendar.MINUTE));

        // 4: Min Frequency_Minute Decrease Test
        alarm.requestAlarmNextSection();
        alarm.decreaseSection();
        assertEquals(9, alarm.getFrequency(alarm.getCurrAlarm()).get(Calendar.MINUTE));

        // 5: Min Repeat Decrease Test
        alarm.requestAlarmNextSection();
        alarm.decreaseSection();
        alarm.decreaseSection();
        assertEquals(5, alarm.getRepeat(alarm.getCurrAlarm()));

        // 6: Max Bell Increase Test
        alarm.requestAlarmNextSection();
        alarm.decreaseSection();    // Alarm4.wav
        alarm.decreaseSection();    // Alarm3.wav
        alarm.decreaseSection();    // Alarm2.wav
        alarm.decreaseSection();    // Alarm1.wav
        alarm.decreaseSection();    // Alarm4.wav
    }
    @Test
    public void requestNextAlarm() throws UnsupportedAudioFileException, IOException, LineUnavailableException{
        RealTime realTime = new RealTime();
        Alarm alarm = new Alarm(realTime);
        alarm.setCurrAlarm(3); // [currAlarm] Point last alarm
        alarm.requestNextAlarm(); // [currAlarm] 3 -> 0
        alarm.requestNextAlarm(); // [currAlarm] 0 -> 1
        assertEquals(1, alarm.getCurrAlarm());
    }

    @Test
    public void requestStopRinging() throws UnsupportedAudioFileException, IOException, LineUnavailableException{
        RealTime realTime = new RealTime();
        Alarm alarm = new Alarm(realTime);
        // Alarm Ringing Index not -1
        alarm.getBell(1).play(5);
        assertTrue(alarm.isRinging());
        // Alarm Ringing Index -1
        alarm.requestStopRinging();
        assertFalse(alarm.isRinging());
    }

    @Test
    public void requestAlarmOnOff() throws UnsupportedAudioFileException, IOException, LineUnavailableException{
        RealTime realTime = new RealTime();
        Alarm alarm = new Alarm(realTime);
        alarm.requestAlarmOnOff();
        assertTrue(alarm.getAlarmCurrAlarmStatus());

        alarm.requestAlarmOnOff();
        assertFalse(alarm.getAlarmCurrAlarmStatus());
    }
}
```

```
    @Test
    public void ringProperly() throws UnsupportedAudioFileException, IOException, LineUnavailableException{
        RealTime realTime = new RealTime();
        realTime.requestRealTime().set(2000, Calendar.JANUARY, 1, 1, 1, 30);
        Alarm alarm = new Alarm(realTime);
        Calendar tmp = Calendar.getInstance();
        Calendar tmpFre = Calendar.getInstance();
        tmp.clear();
        tmp.set( year: 2000, Calendar.JANUARY,  date: 1,  hourOfDay: 1,  minute: 1,  second: 30);
        Calendar compare = (Calendar)tmp.clone();
        // Frequency is set by 1 min 30 sec
        tmpFre.set( year: 1970, Calendar.JANUARY,  date: 0,  hourOfDay: 0,  minute: 1,  second: 30);
        // Reputation is 2
        alarm.setRepeat(0, 2);
        alarm.setFrequency(0, tmpFre);
        alarm.requestAlarmOnOff();
        alarm.setAlarm(tmp);

        assertEquals(realTime.requestRealTime().getTime(), alarm.getAlarm().getTime());

        // First Reputation
        for(int i = 0; i < 90; i++) {
            alarm.realTimeTaskAlarm();
            realTime.requestRealTime().add(Calendar.SECOND, 1);
        }
        assertEquals(realTime.requestRealTime().getTime(), alarm.getAlarm().getTime());

        // Second Reputation
        for(int i = 0 ; i<90; i++) {
            alarm.realTimeTaskAlarm();
            realTime.requestRealTime().add(Calendar.SECOND, 1);
        }
        assertEquals(realTime.requestRealTime().getTime(), alarm.getAlarm().getTime());

        // Reset To Init Alarm
        alarm.realTimeTaskAlarm();
        assertEquals(compare.getTime(), alarm.getAlarm().getTime());
    }
}
```

6. WorldtimeTest

```java
public class WorldtimeTest {

    @Test
    public void nextNation() {
        RealTime realTime = new RealTime();
        Worldtime worldtime = new Worldtime(realTime);

        // Test NextCity at Last City
        worldtime.setCurrNation(worldtime.getMaxNation() - 1); // [currNation] 19 -> max-1
        worldtime.nextNation(); // [currNation] max-1 -> 0
        worldtime.nextNation(); // [currNation] 0 -> 1

        assertEquals(1, worldtime.getCurrNation()); // [currNation] 1
    }

    @Test
    public void prevNation() {
        RealTime realTime = new RealTime();
        Worldtime worldtime = new Worldtime(realTime);

        // Test Prev City at First City
        worldtime.setCurrNation(0); // [currNation] 19 -> 0
        worldtime.prevNation(); // [currNation] 0 -> max-1
        worldtime.prevNation(); // [currNation] max-1 -> max-2

        assertEquals(worldtime.getMaxNation() - 2, worldtime.getCurrNation()); // [currNation] max-2
    }

    @Test
    public void realTimeTaskWorldtime() {
        RealTime realTime = new RealTime();
        Worldtime worldtime = new Worldtime(realTime);
        realTime.requestRealTime().set(2019, Calendar.MAY,23,18,20,0); // Seoul(GMT+9)
        assertEquals(realTime.requestRealTime().getTimeZone().getID(),worldtime.getNationTimeZone(worldtime.getCurrNation())); // Seoul(GMT+9) == Seoul(GMT+9)
        worldtime.setCurrNation(4); // Seoul(GMT+9) -> Paris(GMT+2)
        worldtime.prevNation(); // Paris(GMT+2) -> London(GMT)

        // Increase Time
        for(int i = 0; i < 100; i++){
            realTime.calculateTime();
            worldtime.realTimeTaskWorldtime();
        }

        assertNotEquals(worldtime.getWorldTime().get(Calendar.HOUR_OF_DAY), worldtime.getCurrTime().get(Calendar.HOUR_OF_DAY)); // Seoul(GMT+9) != London(GMT)
        assertEquals(worldtime.getWorldTime().get(Calendar.MILLISECOND), worldtime.getCurrTime().get(Calendar.MILLISECOND)); // Seoul(GMT+9) == London(GMT)
        assertEquals(worldtime.getWorldTime().get(Calendar.SECOND), worldtime.getCurrTime().get(Calendar.SECOND)); // Seoul(GMT+9) == London(GMT)
        assertEquals(worldtime.getWorldTime().get(Calendar.MINUTE), worldtime.getCurrTime().get(Calendar.MINUTE)); // Seoul(GMT+9) == London(GMT)
    }

    /*
    @Test
    public void showWorldTime() { }
    */
}
```

7. SunTest

```
public class SunTest {

    @Test
    public void realTimeTaskSun() {
        RealTime realTime = new RealTime();
        realTime.requestRealTime().set(2019, 5,23,03,59,30);
        Sun sun = new Sun(realTime);

        // Before realTimeTaskSun
        assertEquals(23, sun.getSun(0).get(Calendar.DATE)); // Today(23)'s Sun Rise
        assertEquals(23, sun.getSun(1).get(Calendar.DATE)); // Today(23)'s Sun Set

        realTime.requestRealTime().set(2019, 5,23,12,59,30);
        sun.realTimeTaskSun(); // [Changed] this.currTime.getTimeInMillis() >= this.sun[0].getTimeInMillis()
        assertEquals(24, sun.getSun(0).get(Calendar.DATE)); // Tomorrow(23->24)'s Sun Rise
        assertEquals(23, sun.getSun(1).get(Calendar.DATE)); // Today(23)'s Sun Set

        realTime.requestRealTime().set(2019, 5,23,20,59,30);
        sun.realTimeTaskSun(); // [Changed] this.currTime.getTimeInMillis() >= this.sun[1].getTimeInMillis()
        assertEquals(24, sun.getSun(0).get(Calendar.DATE)); // Tomorrow(24)'s Sun Rise
        assertEquals(24, sun.getSun(1).get(Calendar.DATE)); // Tomorrow(23 -> 24)'s Sun Set
    }

    @Test
    public void requestSetRise() {
        RealTime realTime = new RealTime();
        Sun sun = new Sun(realTime);

        assertEquals(0, sun.getCurrMode()); // [currMode] 0: Sun Rise
        sun.requestSetRise(); // [currMode] 0: Sun Rise -> 1: Sun Set
        assertEquals(1, sun.getCurrMode()); // [currMode] 1: Sun Set
        sun.requestSetRise(); // [currMode] 1: Sun Set -> 0: Sun Rise
        assertEquals(0, sun.getCurrMode()); // [currMode] 0: Sun Rise
    }

    @Test
    public void requestNextNation() {
        RealTime realTime = new RealTime();
        Sun sun = new Sun(realTime);

        // Test requestNextCity at Last City
        sun.setCurrNation(sun.getMaxNation() - 1); // [currNation] 19 -> max-1
        sun.requestNextNation(); // [currNation] max-1 -> 0
        sun.requestNextNation(); // [currNation] 0 -> 1
        assertEquals(1, sun.getCurrNation()); // [currNation] 1
    }
```

```java
    @Test
    public void requestPrevNation() {
        RealTime realTime = new RealTime();
        Sun sun = new Sun(realTime);

        // Test requestPrevCity at First City
        sun.setCurrNation(0); // [currNation] 19 -> 0
        sun.requestPrevNation(); // [currNation] 0 -> max-1
        sun.requestPrevNation(); // [currNation] max-1 -> max-2

        assertEquals(sun.getMaxNation() - 2, sun.getCurrNation()); // [currNation] max-2
    }

    @Test
    public void initSun() {
        RealTime realTime = new RealTime();
        Sun sun = null;

        // 1. Today's sun rise and Today's sun set
        realTime.requestRealTime().set(2019, Calendar.MAY, 23, 03, 59, 30);
        sun = new Sun(realTime);

        assertEquals(23, sun.getSun(0).get(Calendar.DATE)); // Today(23)'s Sun Rise
        assertEquals(23, sun.getSun(1).get(Calendar.DATE)); // Today(23)'s Sun Set

        // 2. Tomorrow's sun rise and Today's sun set
        realTime.requestRealTime().set(2019, Calendar.MAY, 23, 12, 59, 30);
        sun = new Sun(realTime);

        assertEquals(24, sun.getSun(0).get(Calendar.DATE)); // Tomorrow(24)'s Sun Rise
        assertEquals(23, sun.getSun(1).get(Calendar.DATE)); // Today(23)'s Sun Set

        // 3. Tomorrow's sun rise and Tomorrow's sun set
        realTime.requestRealTime().set(2019, Calendar.MAY, 23, 20, 59, 30);
        sun = new Sun(realTime);

        assertEquals(24, sun.getSun(0).get(Calendar.DATE)); // Tomorrow(24)'s Sun Rise
        assertEquals(24, sun.getSun(1).get(Calendar.DATE)); // Tomorrow(24)'s Sun Set
    }
}
```

Activity 2061. Unit Testing

All in Develop_T4_SMA.test (1): 41 total, 41 passed 33.60 s

Collapse | Expand

"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" -ea -Didea.test.cyclic.buffer.size=1048576 "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2019.1\lib\idea_rt.jar=65292:C:\Program Files\JetBrains\IntelliJ IDEA 2019.1\bin" -Dfile.encoding=UTF-8 -classpath "C:\Program Files\JetBrains\IntelliJ IDEA 2019.1\lib\idea_rt.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2019.1\plugins\junit\lib\junit-rt.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2019.1\plugins\junit\lib\junit5-rt.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\charsets.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\deploy.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\access-bridge-64.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\cldrdata.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\dnsns.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\jaccess.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\jfxrt.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\localedata.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\nashorn.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\sunec.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\sunjce_provider.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\sunmscapi.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\sunpkcs11.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\zipfs.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\javaws.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\jce.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\jfr.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\jfxswt.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\jsse.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\management-agent.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\plugin.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\resources.jar;C:\Program Files\Java\jdk1.8.0_191\jre\lib\rt.jar;C:\Users\Public\GitHub\Develop_SMA_T4\out\test\classes;C:\Users\Public\GitHub\Develop_SMA_T4\out\production\classes;C:\Users\Publ
Jeon\.gradle\caches\modules-2\files-2.1\com.luckycatlabs\SunriseSunsetCalculator\1.2\ee2cf7467f62325deb4e4e533425bcacbc8f07fd\SunriseSunsetCalculator-1.2.jar;C:\Users\DoHyeon Jeon\.gradle\caches\modules-2\files-2.1\junit\junit\4.12\2973d150c0dc1fefe998f834810d68f278ea58ec\junit-4.12.jar;C:\Users\DoHyeon Jeon\.gradle\caches\modules-2\files-2.1\org.hamcrest\hamcrest-core\1.3\42a25dc3219429f0e5d060061f71acb49bf010a0\hamcrest-core-1.3.jar" com.intellij.rt.execution.junit.JUnitStarter -ideVersion5 @w@C:\Users\DOHYEO~1\AppData\Local\Temp\idea_working_dirs_junit.tmp @C:\Users\DOHYEO~1\AppData\Local\Temp\idea_junit.tmp -socket65291 Process finished with exit code 0

AlarmTest	31.85 s
ModeSettingTest	1.38 s
RealTimeTest	5 ms
StopwatchTest	22 ms
SunTest	43 ms
TimerTest	302 ms
WorldtimeTest	2 ms

AlarmTest		40.99 s
AlarmTest.requestExitAlarmSetting	passed	497 ms
AlarmTest.decreaseSection	passed	20.14 s
AlarmTest.ringProperly	passed	58 ms
AlarmTest.requestAlarmNextSection	passed	40 ms
AlarmTest.requestNextAlarm	passed	60 ms
AlarmTest.requestSettingAlarm	passed	33 ms
AlarmTest.requestAlarmOnOff	passed	37 ms
AlarmTest.requestStopRinging	passed	92 ms
AlarmTest.increaseSection	passed	20.04 s
ModeSettingTest		885 ms
ModeSettingTest.requestSelectMode	passed	3 ms
ModeSettingTest.requestNextMode	passed	0 ms
ModeSettingTest.confirmSelectMode	passed	882 ms
RealTimeTest		4 ms
RealTimeTest.nextSection	passed	0 ms
RealTimeTest.setCurrSection	passed	0 ms
RealTimeTest.requestChangeType	passed	1 ms
RealTimeTest.calculateTime	passed	0 ms
RealTimeTest.decreaseTime	passed	1 ms
RealTimeTest.setSecond	passed	0 ms
RealTimeTest.increaseTime	passed	2 ms
StopwatchTest		36 ms
StopwatchTest.requestStartStopwatch	passed	1 ms
StopwatchTest.requestResetStopwatch	passed	0 ms
StopwatchTest.requestStopStopwatch	passed	1 ms
StopwatchTest.requestSplitStopwatch	passed	28 ms
StopwatchTest.realTimeTaskStopwatch	passed	6 ms
SunTest		62 ms
SunTest.realTimeTaskSun	passed	28 ms
SunTest.requestNextNation	passed	13 ms
SunTest.requestSetRise	passed	2 ms
SunTest.requestPrevNation	passed	8 ms
SunTest.initSun	passed	11 ms
TimerTest		412 ms
TimerTest.requestExitSetTimerTime	passed	12 ms
TimerTest.requestIncreaseTimerTimeSection	passed	11 ms
TimerTest.requestTimerTime	passed	10 ms
TimerTest.requestResetTimer	passed	11 ms
TimerTest.realTimeTimerTask	passed	333 ms
TimerTest.changeStatus	passed	10 ms
TimerTest.requestDecreaseTimerTimeSection	passed	7 ms
TimerTest.requestNextTimerTimeSection	passed	10 ms
TimerTest.ringOff	passed	8 ms
Ring off operation activated		
WorldtimeTest		5 ms
WorldtimeTest.nextNation	passed	2 ms
WorldtimeTest.prevNation	passed	0 ms
WorldtimeTest.realTimeTaskWorldtime	passed	3 ms

Activity 2063. System Testing

No.	Name	Test Description
	RealTime Test	
1-1	nextSection	다음 Section을 호출하였을 때, 마지막 Section의 경우 맨 처음으로 돌아가는지 확인
1-2	calculateTime	설정된 시간이 정확하게 들어갔는 지, 그리고 매번 함수를 호출하였을 시, 10ms마다 증가한 값이 저장되어 있는 지 확인
1-3	setSecond	설정된 초가 정확하게 들어갔는 지 확인
1-4	increaseTime	해당 함수를 실행시켰을 때 값이 증가하는 지, 해당 section에서 가장 큰 수일 때 실행시키면, 다음 section이 증가하고 해당 section이 가장 작은 수가 되는지 확인
1-5	decreaseTime	해당 함수를 실행시켰을 때 값이 감소하는 지, 해당 section에서 가장 작은 수일 때 실행시키면, 다음 section이 감소하고 해당 section이 가장 큰 수가 되는지 확인
1-6	requestChangeType	해당 함수를 실행시켰을 때 24-hour 에서 12 hour 로 바뀌는 지 확인.
1-7	setCurrSection	해당 함수를 실행시켰을 때 맞는 section에 있는지 확인.
	Stopwatch Test	
2-1	requestStartStopwatch	정지된 상태에서 실행시켰을 때, 해당 모드의 상태가 변화하는 지 확인
2-2	requestStopStopwatch	실행중인 상태에서 실행시켰을 때, 해당 모드의 상태가 변화하는지 확인
2-3	requestResetStopwatch	정지된 상태에서 실행시켰을 때, 초기값으로 바뀌는 지 확인
2-4	requestSplitStopwatch	실행중인 상태에서 실행시켰을 때, Split된 시간의 정보가 원하는 값으로 저장이 되었는지 확인, 정지된 상태에서 실행시켰을 경우, 저장이 안되는 지 확인
2-5	realTimeTaskStopwatch	원하는 만큼의 시간이 증가하여 저장되어 있는지 확인
	Timer Test	
3-1	requestExitSetTimerTime	실행시켰을 때, 상태와 Section이 초기화되는 지 확인
3-2	requestIncreaseTimerTimeSection	해당 함수를 실행시켰을 때 값이 증가하는 지, 해

		당 section에서 가장 큰 수일 때 실행시키면, 다음 section이 증가하고 해당 section이 가장 작은 수가 되는지 확인
3-3	requestTimerTime	실행시켰을 때, 상태정보가 바뀌는 지 확인
3-4	requestResetTimer	임의로 설정된 시간이 설정된 초기 시간으로 바뀌는 지 확인
3-5	realTimeTimerTask	10ms 단위로 줄어들어 초기시간에 도달하는 지 확인, 이 후 벨이 울리는 상태로 돌입하는 지 확인
3-6	changeStatus	상태의 최대, 최소값을 벗어나는지 확인
3-7	requestDecreaseTimerTimeSection	해당 함수를 실행시켰을 때 값이 감소하는 지, 해당 section에서 가장 작은 수일 때 실행시키면, 다음 section이 감소하고 해당 section이 가장 큰 수가 되는지 확인
3-8	requestNextTimerTimeSection	각 Section이 잘 바뀌는 지 확인(0~2)
3-9	ringOff	벨이 울리는 상태에 있을 때, 초기 상태로 바뀌는 지 확인
	Alarm Test	
4-1	requestExitAlarmSetting	알람 시간 설정상태에서 상태와 Section이 초기화되는 지 확인
4-2	decreaseSection	해당 함수를 실행시켰을 때 값이 감소하는 지, 해당 section에서 가장 작은 수일 때 실행시키면, 다음 section이 감소하고 해당 section이 가장 큰 수가 되는지 확인
4-3	ringProperly	임의로 지정된 시간이 임의로 설정된 시간에 도달했을 때 울리는 상태에 도달하는 지 확인, 이 후 주기와 반복에 맞추어 지정된 시간이 추가되는 지 확인, 반복 횟수가 0보다 작아지면 다시 처음에 설정된 시간으로 바뀌는 지 확인
4-4	realTimeTaskAlarm	설정된 시간에 도달했을 때, 벨이 울리는 상태로 바뀌는 지 확인
4-5	requestAlarmNextSection	Section을 하나씩 추가하면서 특정 Section에 도달하면 상태가 같이 바뀌는 지 확인
4-6	requestNextAlarm	총 4개의 알람이 존재하여 마지막 알람에서 해당 함수를 실행시켰을 때, 맨 처음 알람으로 바뀌는 지 확인
4-7	requestSettingAlarm	알람 시간 설정 상태로 바뀌는 지 확인
4-8	requestAlarmOnOff	알람을 울리는 설정이 켜지는 지, 다시 꺼지는 지 확인
4-9	requestStopRinging	알람이 울리고 있을 시, 꺼지는 지 확인

4-10	increaseSection	해당 함수를 실행시켰을 때 값이 증가하는 지, 해당 section에서 가장 큰 수일 때 실행시키면, 다음 section이 증가하고 해당 section이 가장 작은 수가 되는지 확인
	Worldtime Test	
5-1	nextNation	해당 함수를 실행시켰을 때 값이 증가하는 지, 해당 section에서 가장 큰 수일 때 실행시키면, 다음 section이 증가하고 해당 section이 가장 작은 수가 되는지 확인
5-2	prevNation	해당 함수를 실행시켰을 때 값이 감소하는 지, 해당 section에서 가장 작은 수일 때 실행시키면, 다음 section이 감소하고 해당 section이 가장 큰 수가 되는지 확인
5-3	realTimeTaskWorldTime	설정된 세계 시간이 '시' 말고 다른 section들은 동일한지 확인, 시간이 흐를 때에도 동일한 지 확인
	Sun Test	
6-1	realTimeTaskSun	시간이 해당 날의 일출, 일몰을 넘어갔을 경우, 그 다음 날의 일출, 일몰을 보여주는 지 확인
6-2	requestNextNation	해당 함수를 실행시켰을 때 값이 증가하는 지, 해당 section에서 가장 큰 수일 때 실행시키면, 다음 section이 증가하고 해당 section이 가장 작은 수가 되는지 확인
6-3	requestSetRise	일출에서 일몰로, 일몰에서 일출로 바뀌는 지 확인
6-4	requsetPrevNation	해당 함수를 실행시켰을 때 값이 감소하는 지, 해당 section에서 가장 작은 수일 때 실행시키면, 다음 section이 감소하고 해당 section이 가장 큰 수가 되는지 확인
6-5	initSun	실제 시간만을 기준으로 정확히 호출되는 지 확인
	ModeSetting Test	
7-1	requestSelectMode	모드를 선택하였을 때, 모드의 리스트에 저장되어 있는 값이 바뀌었는지 확인
7-2	requestNextMode	해당 함수를 실행시켰을 때 값이 증가하는 지, 해당 section에서 가장 큰 수일 때 실행시키면, 다음 section이 증가하고 해당 section이 가장 작은 수가 되는지 확인
7-3	confirmSelectMode	설정된 모드들이 정확하게 저장되었는지 확인

Activity 2067. Design Traceability Analysis