Event Calendar

SJSU SAN JOSÉ STATE UNIVERSITY

Team:

Lianshi Gan

Yuehongxiao Ma

Zhao Liu

This slide deck is submitted in fulfillment of the final project for CS 151 Object-Oriented Design, Section 6, Dr. Yeung, Fall 2018.

Pain Points of the Problem

- Context of the Problem
  - Bad self-management of schedule
  - Inconvenient for paper calendar
  - Accidentally miss important events

Current Solutions

- Google Calendar
- iCloud Calendar
- OutLook

My Solution

- My solution is better in the follow aspects:
  - Display different color of event by priority
  - Based on Java, easily adopt most platform

Possible Ideas

- I have tried several ideas:
  - Allow user to set 5 different types of event
  - User could set the reminder time
  - Allow user to search their event by name

Useful Source Code / Libraries

- In my protyping effort, I found the following resource / source code in public domain extremely helpful:
  - Calendar project from github
  - https://github.com/gjiang3/Calendar

Rapid Prototyping

- In my prototyping effort, we focused on the following main ideas / features:
  - #1 Allow user to check day and month by GUI
  - #2 Allow user to create events for specific hour
  - #3 Allow user to remove existing events
- At the end, I am really pleased with the result of my prototyping with feature #2. With that, I am confident that my idea about creating event will work.

Use Case #1

- For my paper analysis, I worked out the details for the following use cases:

Use Case #1 Pick a Day/Check the days of the week

- 1. The system automatic point the current day by open
- 2. User could click on any day in the same month to check the day of week on selected day

Use Case #1

Variation #1.1

- 1. User could click on next/prev to select month if he/she want to check days in the other month
- 2. User click on the day in selected month to pick the day they want to check

Use Case #2

Use Case #2 Create a new Event

1. User pick a day from calendar to choose the event day.
2. User could click on “Create” button to create a event
3. In pop-up window, user could enter event title amd pick the event start/end time
4. User click on save button to see the event on the time table of selected day

Use Case #2

- Variation #2.1
  1. user try to pick the start time late than end time in step 3
  2. System will display Error: Schedule conflicts
- Variation #2.2
  1. user try to create a event with same time of existing event
  2. System will display Error: Schedule conflicts
- Variation #2.3
  1. user creates a event without entering title
  2. The event will show as “untitled event” on the calendar

Use Case #3

Use Case #3 Cancel a existing event

1. User select the day of event that need to be removed
2. User click on “Cancel” button to open the pop-up window of remove function
3. User pick the start/end time to select the time period of events that need to be removed
4. User click on “save” button to remove event

Use Case #3

- Variation #3.1
  1. user try to pick the start time late than end time in step 3
  2. System will display Error: Schedule conflicts
- Variation #3.2
  1. user pick a time period that contains more than one event
  2. System will remove all events within selected time period

Use Case #4

Use Case #4 Saving all events

1. User want to save all created events in a txt file
2. User click on “Quit” button to exit the program
3. System will automatically output the event list into a txt file under the same directory

Use Case #4

- Variation #4.1

1. user close the program without creating any event
2. System will still create a empty file

- Variation #4.2

1. user open the program with existing saved file
2. System will read the file and input all events

CRC 1

CalendarFrame	
• Create a GUI frame to contain MonthCalendar and Agenda • Implement ChangeListener to display the changes	• Events • Agenda • MonthCalendar • CalendarController

CRC 2

MonthCalendar	
• Create the GUI for Calendar • Create Create/Cancel/Quit buttons • Accept user input • Tigger create/cancel event • Tigger quit program	• CalcendarController • CreateEvent • CancelEvent • Date

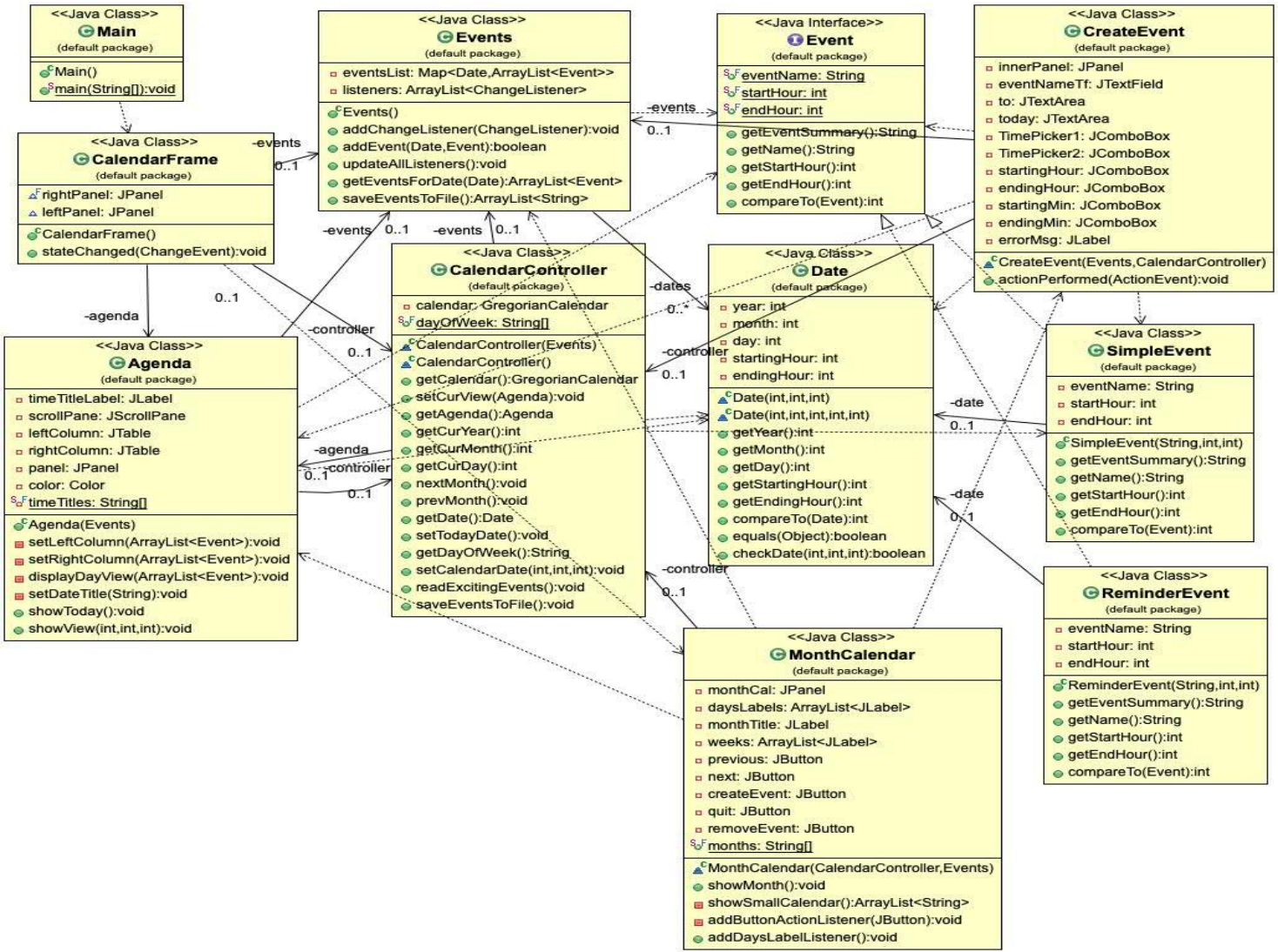
CRC 3

CreateEvent	
• Display GUI for input event information • Create a event • Passing event information to controller	• CalcendarController • SimpleEvent • Events

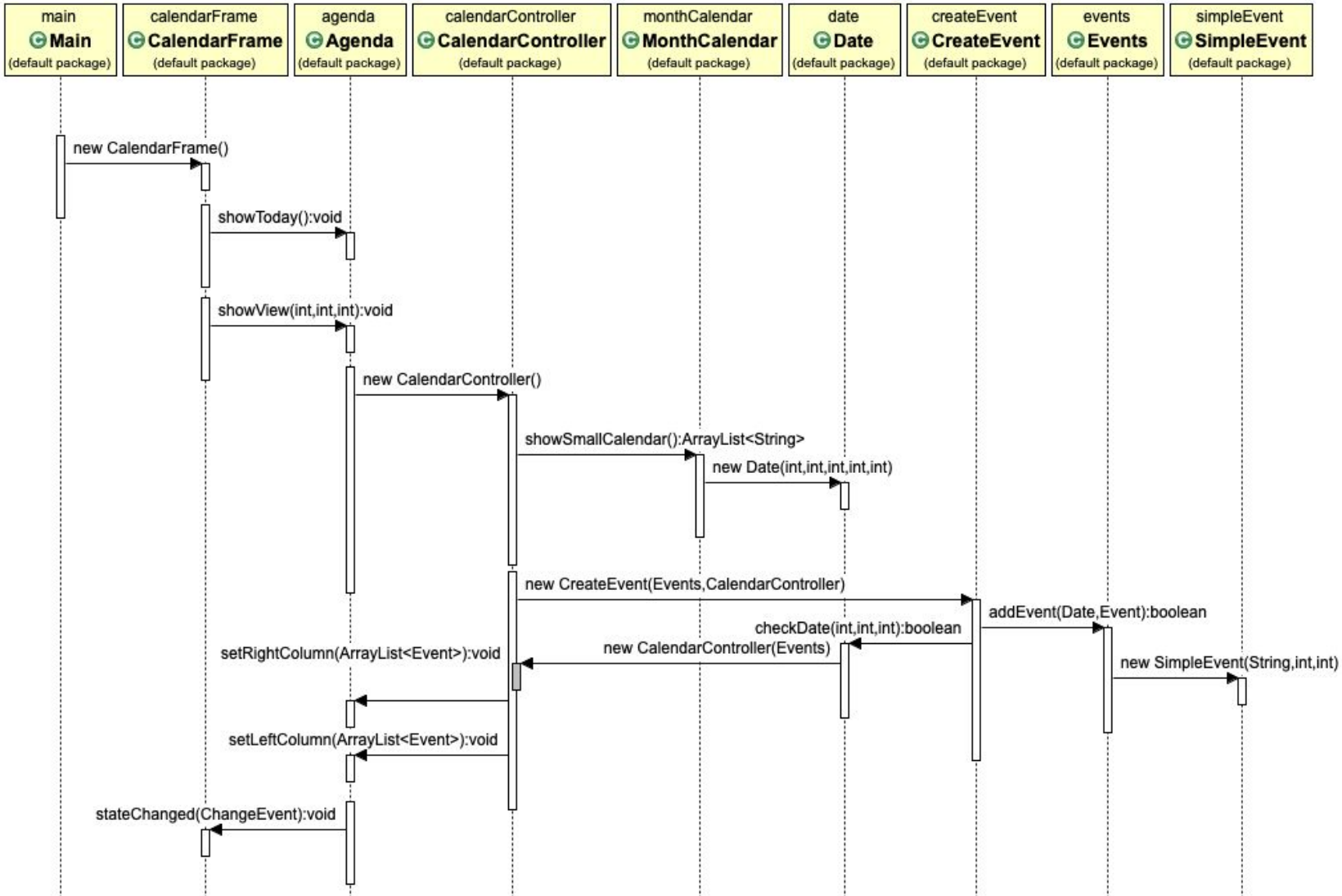
CRC 4

Events	
• Manage Event array-list • Manage Event day • Manage Event time • Update Event information	• Event • SimpleEvent • Date

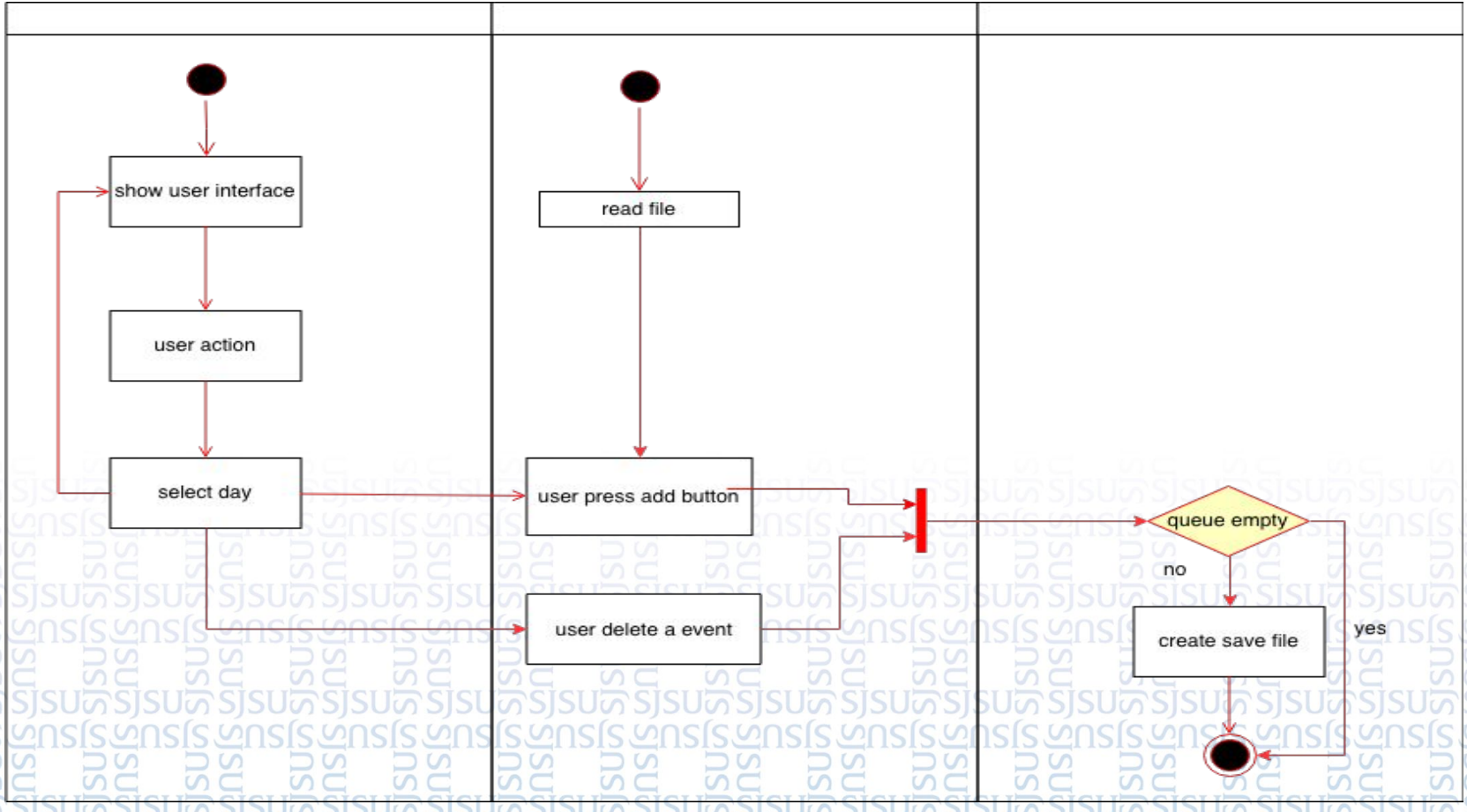
UML Class Modeling



UML Sequence Diagram



UML State Diagram



JavaDocs

PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class CalendarFrame

```
java.lang.Object
    java.awt.Component
        java.awt.Container
            java.awt.Window
                java.awt.Frame
                    javax.swing.JFrame
                        CalendarFrame
```

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.event.ChangeListener, javax.swing.RootPaneContainer, javax.swing.WindowConstants

---

```
public class CalendarFrame
extends javax.swing.JFrame
implements javax.swing.event.ChangeListener
```

CalendarFrame is the JFrame that holds all GUI components. It will display the main interface to the users.

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JFrame

javax.swing.JFrame.AccessibleJFrame

Nested classes/interfaces inherited from class java.awt.Frame

java.awt.Frame.AccessibleAWTFrame

Nested classes/interfaces inherited from class java.awt.Window

java.awt.Window.AccessibleAWTWindow, java.awt.Window.Type

Nested classes/interfaces inherited from class java.awt.Container

java.awt.Container.AccessibleAWTContainer

Nested classes/interfaces inherited from class java.awt.Component

JavaDocs

PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class CreateEvent

```
java.lang.Object
    java.awt.Component
        java.awt.Container
            java.awt.Window
                java.awt.Frame
                    javax.swing.JFrame
                        CreateEvent
```

All Implemented Interfaces:

java.awt.event.ActionListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.Windov

---

public class **CreateEvent**
extends javax.swing.JFrame
implements java.awt.event.ActionListener

CreateEvent class displays the small pop out window to asks the user for event details and creates a new event in

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JFrame

javax.swing.JFrame.AccessibleJFrame

Nested classes/interfaces inherited from class java.awt.Frame

java.awt.Frame.AccessibleAWTFrame

Nested classes/interfaces inherited from class java.awt.Window

java.awt.Window.AccessibleAWTWindow, java.awt.Window.Type

Nested classes/interfaces inherited from class java.awt.Container

java.awt.Container.AccessibleAWTContainer

Nested classes/interfaces inherited from class java.awt.Component

JavaDocs

PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class Events

java.lang.Object
Events

public class **Events**
extends java.lang.Object

Contains all the events that are currently scheduled in the calendar

Constructor Summary

Constructors

Constructor and Description
Events()

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	addChangeListener(javax.swing.event.ChangeListener cl) Adds a change listener to list so that it will be notified of any changes in the treemap
boolean	addEvent(Date date, SimpleEvent dayEvents) Adds a new event to the calendar
boolean	cancelEvent(Date date, SimpleEvent dayEvents) Remove an event to the calendar
java.util.ArrayList<SimpleEvent>	getEventsForDate(Date date) Retrieves all events in sorted order by start hour for a given date
java.util.ArrayList<java.lang.String>	saveEventsToFile() Save all events information as strings to a result list
void	updateAllListeners() Update all listeners with new information if state of treemap changes

5C Criteria for Class Interface Design

- I choose Class SimpleEvent():

```java
public class SimpleEvent implements Event, Comparable<Event>
{
    private String eventName;
    private int eventStartHour;
    private int eventEndHour;
    private String eventType;

    /**...*/
    public SimpleEvent(String name, int start, int end){...}

    public String getEventType(){return eventType;}

    public void setEventType(String type){eventType = type;}

    public void setEventName(String _eventName) { eventName = _eventName; }

    public void setEventStartHour(int _eventStartHour) { eventStartHour = _eventStartHour; }

    public void setEventEndHour(int _eventEndHour) { eventEndHour = _eventEndHour; }

    /**...*/
    public String getEventSummary()
    {...}

    /**...*/
    public String getEventName() { return eventName; }

    /**...*/
    public int getEventStartHour() { return eventStartHour; }

    /**...*/
    public int getEventEndHour() { return eventEndHour; }

    @Override
    public int compareTo(Event other) {...}
```

5C Criteria for Class Interface Design

- Cohesion
  The class only describe a single simple event object, it only has methods related to one single simple event object. Methods: setName, setStartHour, setEndHour, getEName, getStartHour, getEndHour,toString,compareTo. No other method related to the operation outside of a event.

5C Criteria for Class Interface Design

- Completeness
- The class support operations to get the information of the single event, to get the event start time, to get the event end time, and to compare to another event object. The class is reusable for other kinds of events with more detailed implements, like reminder event.
-

5C Criteria for Class Interface Design

- Convenience

  The method and variables are simple, clear, easy enough for other people to use because they only support the operations related to a simple event.

5C Criteria for Class Interface Design

- Clarity

  The class is well-structured, all the variables and methods are only related to a single event object. And the variable name is clear and direct. Example: eventName, startHour

5C Criteria for Class Interface Design

- Consistency

  The class related to the interface all have a consistent format on naming conventions, behavior and methods. For similar methods, the interface has a direct naming way to differentiate methods by its usage.

  Example:

  setName: set a new event name

  setStartHour: set a new event starting time in hour

  setEndHour: set a new event ending time in hour

Unit Tests

- Write unit tests for the classes described in 4a.
- I have written unit tests for these classes
  - `Insert a snipplet for one of the unit test`

Encapsulation

- I have followed the principle of encapsulation closely when I design our classes
  - Here is class Date. It is well encapsulated because all the variables is private and hidden from other classes.
  - Variables can only be accessed by accessor method declared.

```java
public class Date implements Comparable<Date>
{
    private int year;
    private int month;
    private int day;

    /**...*/
    Date(int m, int d, int y )
    {
        year = y;
        month = m;
        day = d;
    }

    /**...*/
    public int getYear() { return year; }

    /**...*/
    public int getMonth() { return month; }

    /**...*/
    public int getDay() { return day; }

    @Override
    public int compareTo(Date other) {...}

    @Override
    public boolean equals(Object obj) {...}

}
```

Encapsulation

- Here is another example class SimpleEvent. It is well encapsulated because
  - all the variables is private and hidden from other classes.
  - Variables can only be accessed by accessor method declared.

```java
import java.util.ArrayList;
import java.util.List;

/**
 * SimpleEvent holds the data for one simple event in calendar
 */
public class SimpleEvent implements Event, Comparable<Event>
{
    private String eventName;
    private int startHour;
    private int endHour;
    private Date date;

    /**...*/
    public SimpleEvent(String name, int start, int end){...}

    /**...*/
    public String getEventSummary()
    {...}

    /**...*/
    public String getName() { return eventName; }

    /**...*/
    public int getStartHour() { return startHour; }

    /**...*/
    public int getEndHour() { return endHour; }

    @Override
    public int compareTo(Event other) {...}

}
```

Loose-coupling

- Show a design that closely follows the principle of loose-coupling.
- Also, I follow the principle of loose-coupling whenever it is possible.
  - Here is an example. (`insert a code snipplet here`.)
  - Class A is loosely coupled with Class B because (…)

Loose-coupling

```java
public interface Event extends Comparable<Event>{
    String eventName = new String();
    int startHour = -1;
    int endHour = -1;
    Date date = null;

    String getEventSummary();
    String getName();
    int getStartHour();
    int getEndHour();
    Date getDate();
    int compareTo(Event other);
}
```

```java
public class ReminderEvent extends TimerTask implements Event, Comparable<Event>
{
    private String eventName;
    private int startHour;
    private int endHour;
    private Date date;

    /**...*/
    public ReminderEvent(String name, int start, int end, Date d){...}

    /**...*/
    public String getEventSummary()
    {...}

    /**...*/
    public String getName() { return eventName; }

    /**...*/
    public int getStartHour() { return startHour; }

    /**...*/
    public int getEndHour() { return endHour; }

    public Date getDate(){return date;}

    @Override
    public int compareTo(Event other) {...}

    public void run(){...}

    public void showReminder() {...}
}
```

Loose-coupling

```java
public interface Event extends Comparable<Event>{
    String eventName = new String();
    int startHour = -1;
    int endHour = -1;
    Date date = null;

    String getEventSummary();
    String getName();
    int getStartHour();
    int getEndHour();
    Date getDate();
    int compareTo(Event other);
}
```

```java
public class SimpleEvent implements Event, Comparable<Event>
{
    private String eventName;
    private int startHour;
    private int endHour;
    private Date date;

    /**...*/
    public SimpleEvent(String name, int start, int end, Date d){...}

    /**...*/
    public String getEventSummary()
    {...}

    /**...*/
    public String getName() { return eventName; }

    /**...*/
    public int getStartHour() { return startHour; }

    /**...*/
    public int getEndHour() { return endHour; }

    public Date getDate(){return date;}

    @Override
    public int compareTo(Event other) {...}

}
```

Loose-coupling

- Also, I follow the principle of loose-coupling whenever it is possible.

```
Event newEvent = null;
if(eventTypePicker.getSelectedItem().equals("Basic")){
    newEvent = new SimpleEvent(eventName, eventStartHour, eventEndHour,controller.getDate());
}
else if(eventTypePicker.getSelectedItem().equals("Reminder")){
    newEvent = new ReminderEvent(eventName, eventStartHour, eventEndHour,controller.getDate());
}

if (events.addEvent(eventDate, newEvent)) {
    this.setVisible(false);
    this.dispose();
    controller.getAgenda().showView(controller.getCurYear(), controller.getCurMonth(), controller.getCurDay());
} else {
    errorMsg.setText("Error: Schedule conflicts.");
}
```

Class SimpleEvent is loosely coupled with Class ReminderEvent because they comes from same interface but they are two independent classes that have different implementations.

Polymorphism

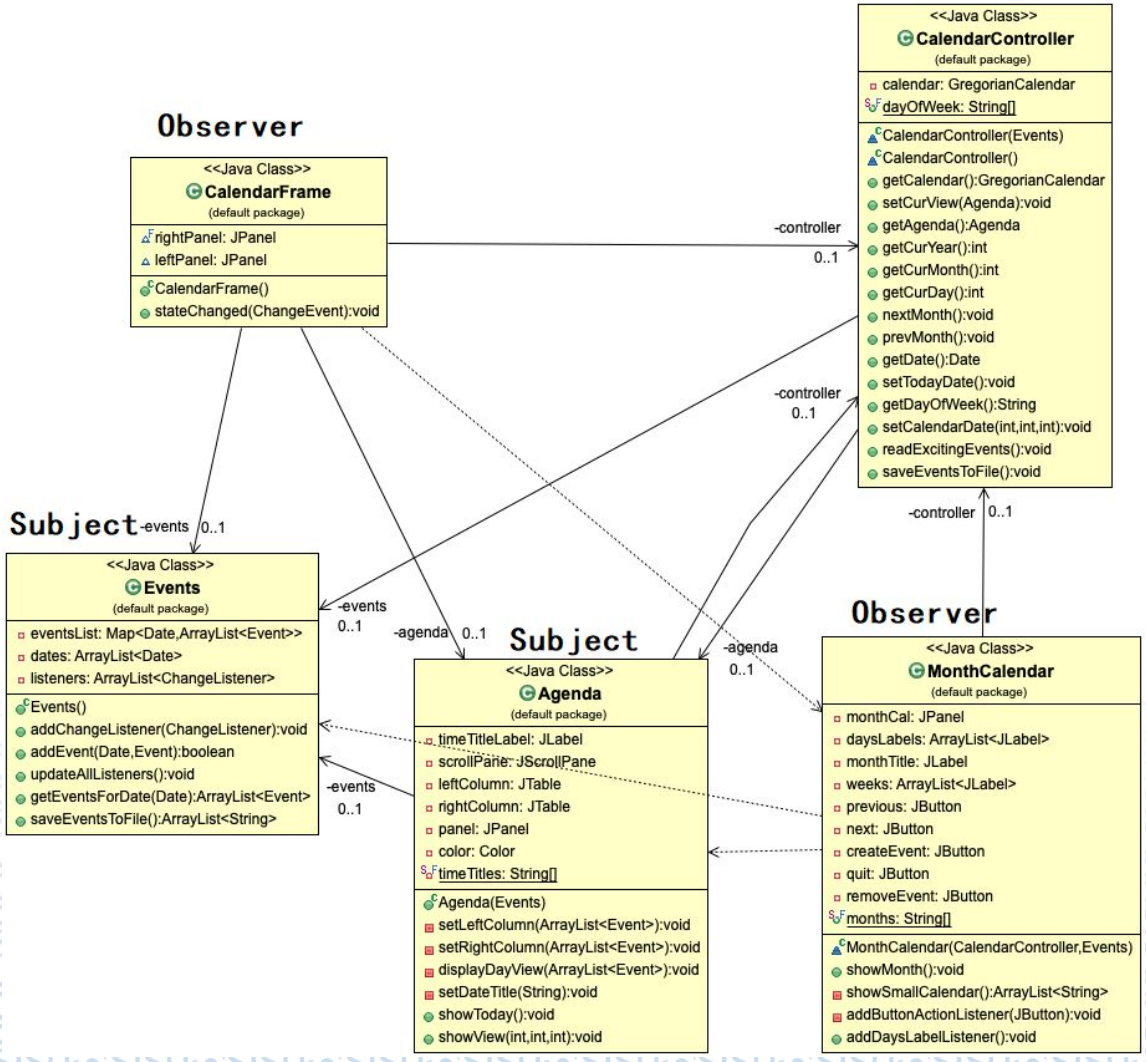
```java
public class SimpleEvent implements Event, Comparable<Event>
{
    private String eventName;
    private int startHour;
    private int endHour;
    private Date date;

    /**...*/
    public SimpleEvent(String name, int start, int end, Date d){...}

    /**...*/
    public String getEventSummary()
    {...}

    /**...*/
    public String getName() { return eventName; }

    /**...*/
    public int getStartHour() { return startHour; }

    /**...*/
    public int getEndHour() { return endHour; }

    public Date getDate(){return date;}

    @Override
    public int compareTo(Event other) {...}

}
```

```java
public class ReminderEvent extends TimerTask implements Event, Comparable<Event>
{
    private String eventName;
    private int startHour;
    private int endHour;
    private Date date;

    /**...*/
    public ReminderEvent(String name, int start, int end, Date d){...}

    /**...*/
    public String getEventSummary()
    {...}

    /**...*/
    public String getName() { return eventName; }

    /**...*/
    public int getStartHour() { return startHour; }

    /**...*/
    public int getEndHour() { return endHour; }

    public Date getDate(){return date;}

    @Override
    public int compareTo(Event other) {...}

    public void run(){...}

    public void showReminder() {...}
}
```

Polymorphism

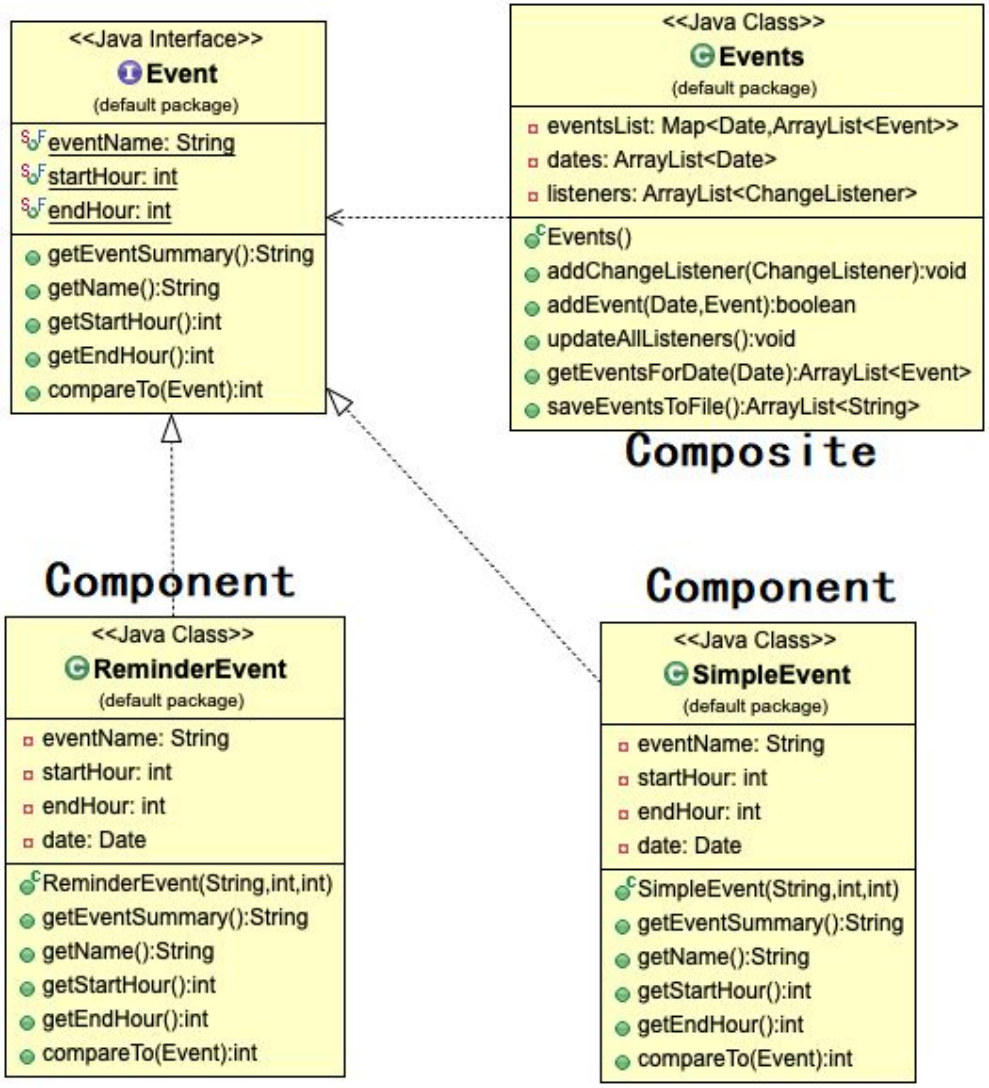
- The design here shows polymorphism design because
- SimpleEvent implements comparable and event interface
- ReminderEvent implements comparable and event interface, and also is a timer task.

Behavioral Design Pattern



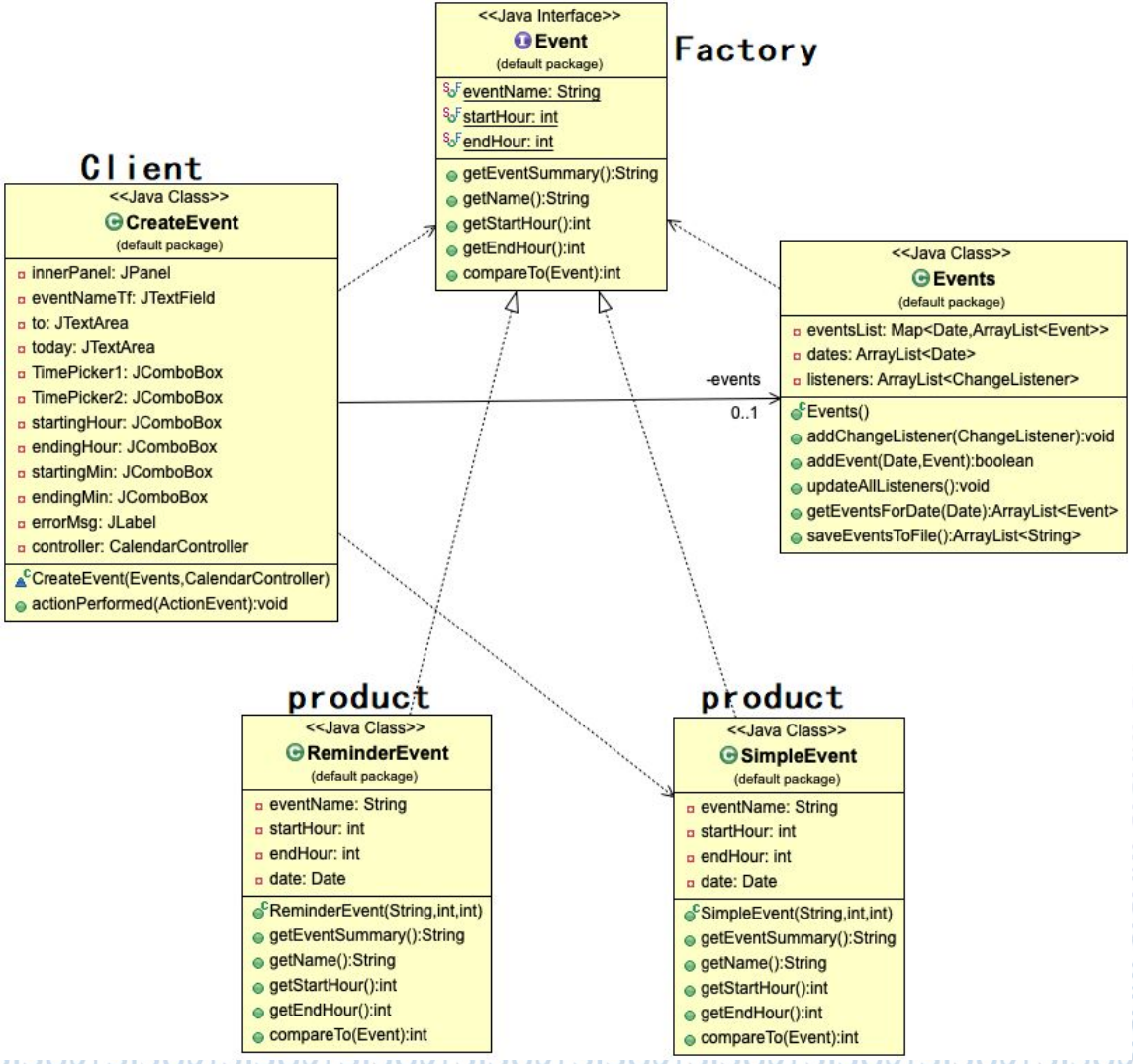
- Observer Pattern

Structural Design Pattern



- Composite Pattern

Creational Design Pattern



- Factory pattern

Overriding the `equals()` Method

In the Date class to compare date object.

```
/**
 * Check if a date object is equal to another date object
 */
@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    } else if (this == obj) {
        return true;
    } else {
        return false;
    }
}
```

Serialization

- Add serialization feature to a class and demonstrate the saving of object instances into a file stream and loading from the file stream into object instances.
- Insert a code snipplet implementing serialization features.

Reflection Methods

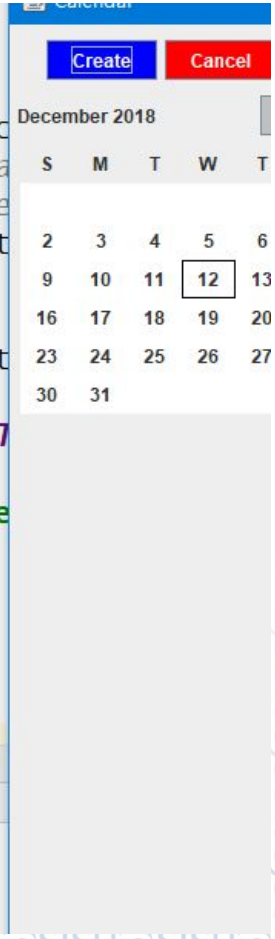
-

```
public Agenda(Events events) throws IOException {
    dateTitle = new JLabel();
    //colorSimpleEvent = new Color(100,200,150); // the event color
    colorSimpleEvent = new Color( i: 152,  i1: 217,  i2: 233); // the eve
    colorReminderEvent = new Color( i: 100, i1: 200, i2: 150);
    panel = new JPanel(new BorderLayout());
    scrollPane = new JScrollPane(panel);
    controller = new CalendarController();
    this.events = events;
    this.setLayout(new BorderLayout());

    //use reflection class to call method
    try{
        Method method = this.getClass().getMethod( s: "showToday");
        method.invoke( o: this);
        //showToday();
    }catch (Exception e) {
        e.printStackTrace();
    }

}
```

Running Thread

```
 *
 */
public class SimpleCalendar {
    public static void main(String[] args) throws IOExc
        CalendarFrame cf = new CalendarFrame(); // crea
        cf.setSize(new Dimension( i: 700,  i1: 700)); // se
        Dimension dim = Toolkit.getDefaultToolkit().get
        Thread thread = new Thread() {
            public void run() {
                cf.setLocation( i: dim.width / 2 - cf.get
                cf.setResizable(true);  // resize
                cf.setDefaultCloseOperation(JFrame.EXIT
                cf.setVisible(true); // visible
                System.out.println("Test Show Main Thre
            }
        };
        thread.start();
    }
}
```

```
SimpleCalendar
"D:\Program Files\Java\jdk1.8.0_171\bin\java.exe" ...
Test Show Main Thread
```



Thread Synchronization

```
"D:\Program Files\Java\jdk1.8.0_171\bin\java
Producer produced-0
Test Show Main Thread
Consumer consumed-0
Producer produced-1
Producer produced-2
Consumer consumed-1
Producer produced-3
```

```java
import ...
/**
 * Thread tester to demostrate thread synchronization
 * @author Lianshi Gan, Zhao Liu, Yuehongxiao Ma
 *
 */
public class ThreadTester {

    ThreadTester() throws InterruptedException{
        // Object of a class that has both produce()
        // and consume() methods
        final PC pc = new PC();

        // Create producer thread
        Thread t1 = new Thread(new Runnable()
        {
```

```
"D:\Program Files\Java\jdk1.8.0_171\bin\java.exe" ...
Show thread synchronization tester
Producer produced-0
Test Show Main Thread
Producer produced-1
Consumer consumed-0
Consumer consumed-1
Producer produced-2
Producer produced-3
Consumer consumed-2
```

Final Project Demo

- Spend about 5 minutes to show your final project program demo to your fellow classmates.