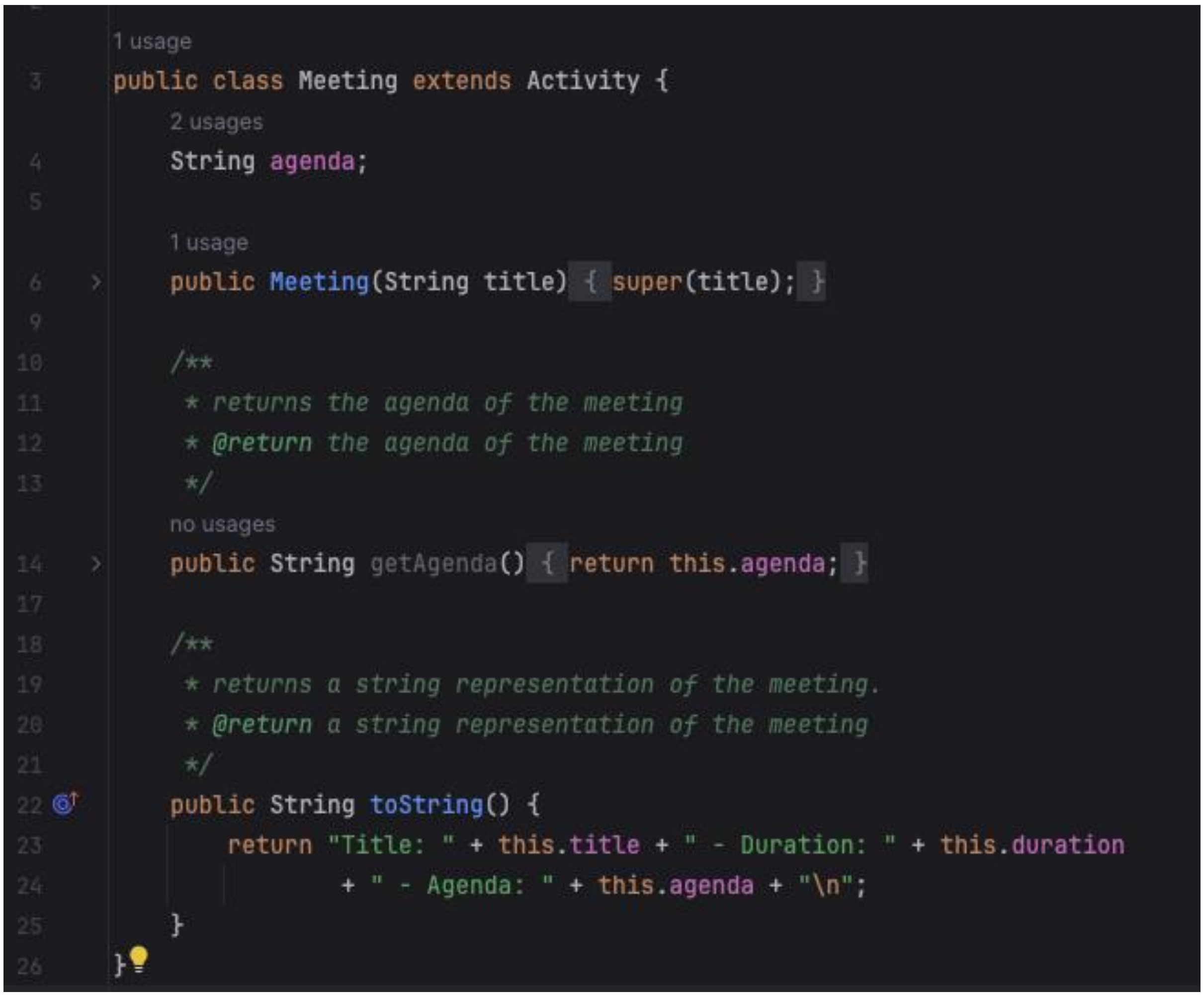
Task 1:
My solution for this task can be found in Activity.java.

Task 2:

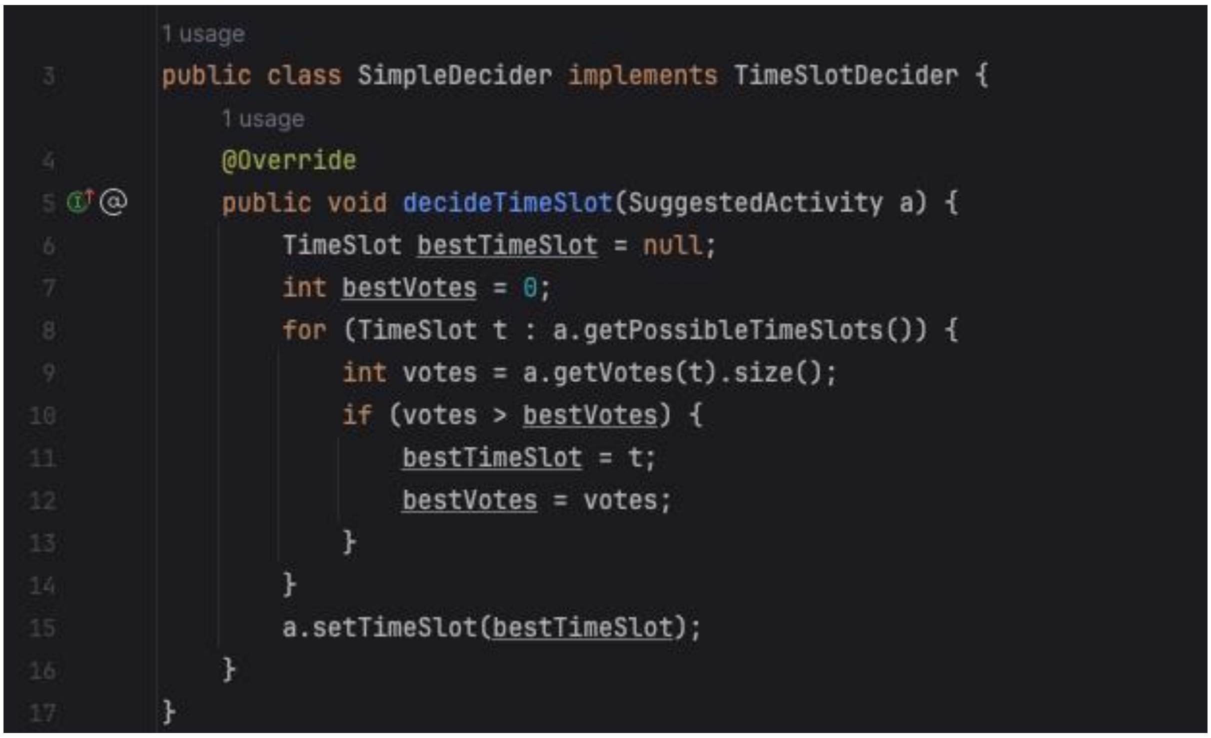


```java
1 usage
public class Meeting extends Activity {
    2 usages
    String agenda;

    1 usage
    public Meeting(String title) { super(title); }

    /**
     * returns the agenda of the meeting
     * @return the agenda of the meeting
     */
    no usages
    public String getAgenda() { return this.agenda; }

    /**
     * returns a string representation of the meeting.
     * @return a string representation of the meeting
     */
    public String toString() {
        return "Title: " + this.title + " - Duration: " + this.duration
                + " - Agenda: " + this.agenda + "\n";
    }
}
```

Memory is allocated for 3 instances when a meeting is created. The three are: agenda which is directly in the meeting class, Title and duration which are inherited from the activity class.

Task 3:
My solution for this task can be found in SuggestedActivity.java.

Task 4:

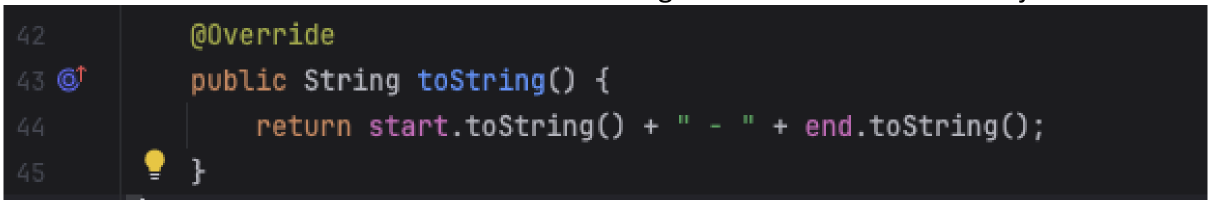


```
1 usage
public class SimpleDecider implements TimeSlotDecider {
    1 usage
    @Override
    public void decideTimeSlot(SuggestedActivity a) {
        TimeSlot bestTimeSlot = null;
        int bestVotes = 0;
        for (TimeSlot t : a.getPossibleTimeSlots()) {
            int votes = a.getVotes(t).size();
            if (votes > bestVotes) {
                bestTimeSlot = t;
                bestVotes = votes;
            }
        }
        a.setTimeSlot(bestTimeSlot);
    }
}
```

I have implemented this decideTimeSlot method such that it finds the timeSlot that has the most votes in cases of multiple timeSlots with equal amounts of votes it simply selects the first timeSlot it finds with that amount of votes.

Task 5:
My solution to this task can be found in Participant.java

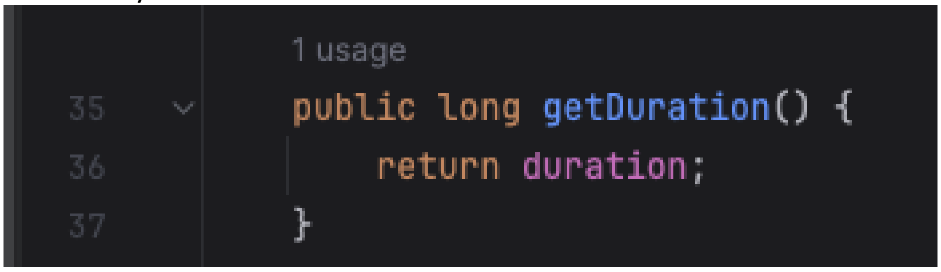
Task 6:
In the TimeSlot class I have overwritten the toString method inherited from object



```
    @Override
    public String toString() {
        return start.toString() + " - " + end.toString();
    }
```

I have done this to be able to easily print out timeSlots in my application.

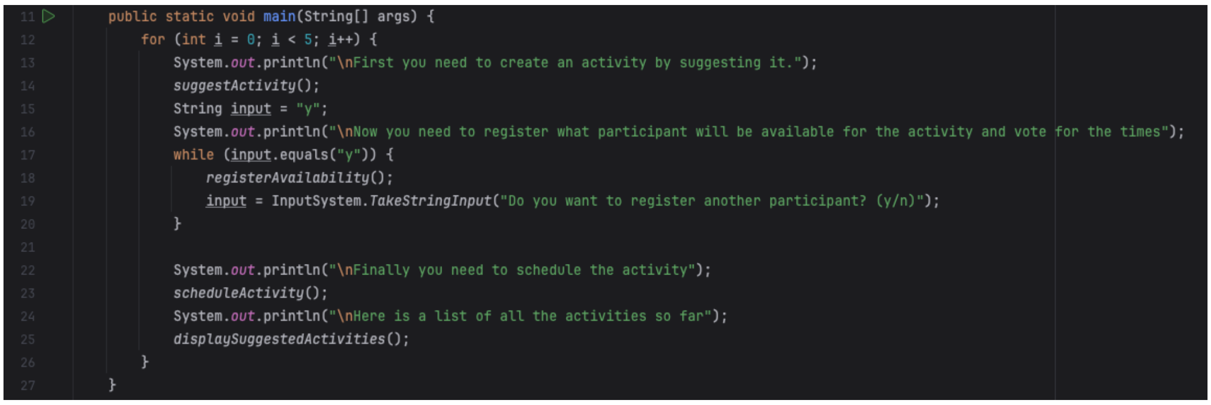
Furthermore I also implemented a getDurationMethod this was the easiest way to let the activity know the duration when the polling is closed and a time slot has been selected for the activity.



```
    1 usage
    public long getDuration() {
        return duration;
    }
```

Task 7:
The entire code for the ActivityOrganizer class can be found in ActivityOrganizer.java

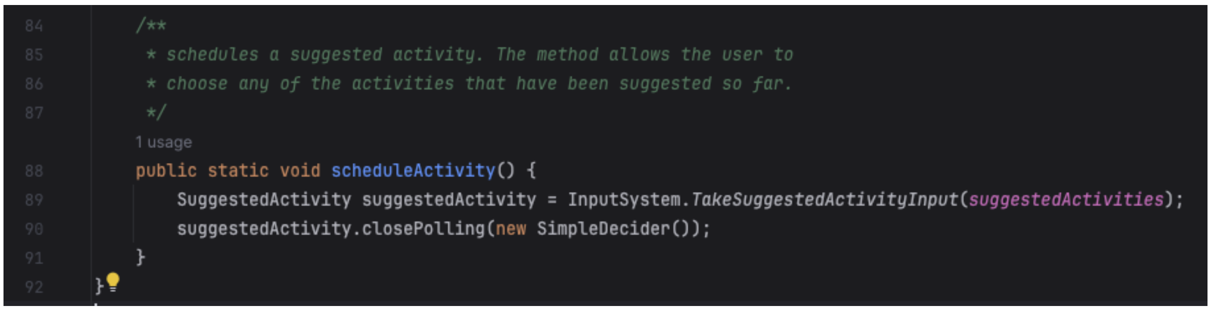
Task 8:



```java
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            System.out.println("\nFirst you need to create an activity by suggesting it.");
            suggestActivity();
            String input = "y";
            System.out.println("\nNow you need to register what participant will be available for the activity and vote for the times");
            while (input.equals("y")) {
                registerAvailability();
                input = InputSystem.TakeStringInput("Do you want to register another participant? (y/n)");
            }

            System.out.println("\nFinally you need to schedule the activity");
            scheduleActivity();
            System.out.println("\nHere is a list of all the activities so far");
            displaySuggestedActivities();
        }
    }
```

This main method takes the user through five iterations of suggesting an activity and the chosing or creating participants to vote. Then finally it schedules the activity and then prints a list of all the activities created so far.

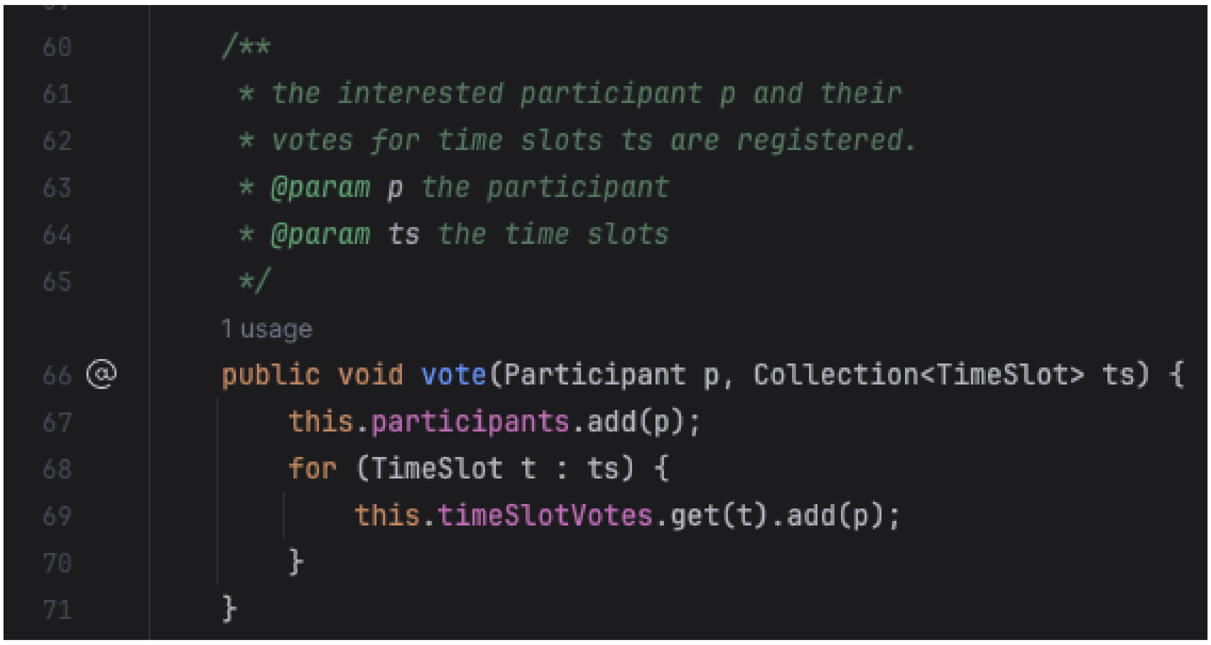
Task 9:



```java
    /**
     * schedules a suggested activity. The method allows the user to
     * choose any of the activities that have been suggested so far.
     */
    1 usage
    public static void scheduleActivity() {
        SuggestedActivity suggestedActivity = InputSystem.TakeSuggestedActivityInput(suggestedActivities);
        suggestedActivity.closePolling(new SimpleDecider());
    }
}
```

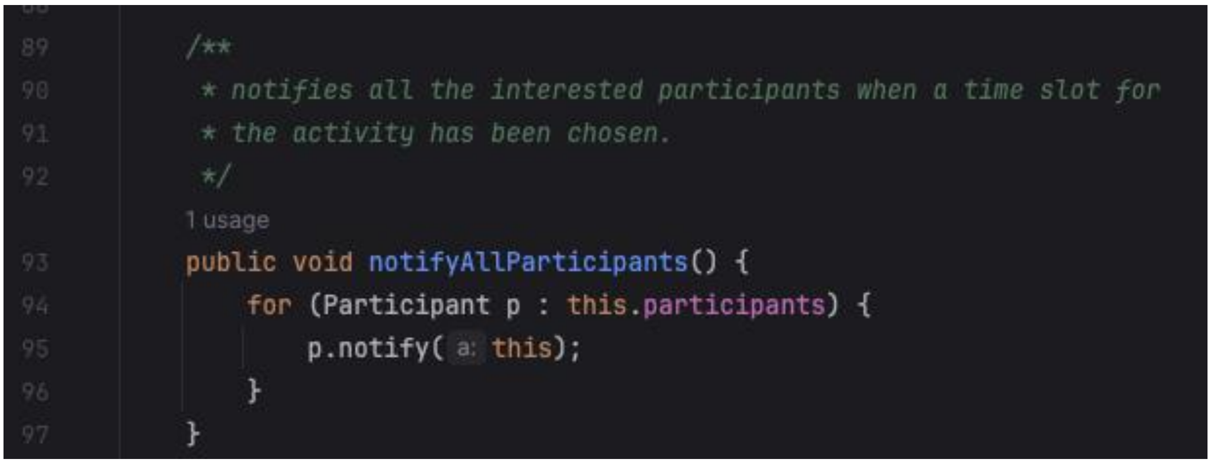
SuggestedActivity is defined with early binding as it is evaluated in compile time as the class SuggestedActivity. The SimpleDecider on the other hand uses late binding since it is bound to a SimpleDecider in run Time.

Task 10:



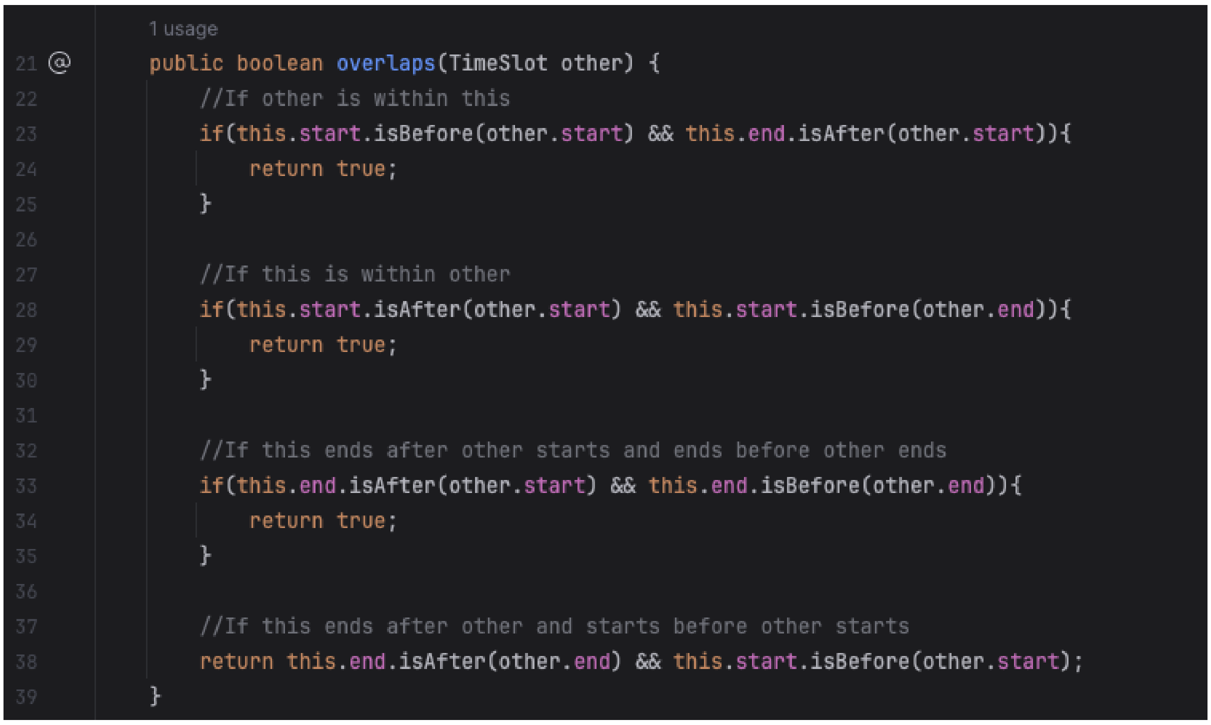
```
    /**
     * the interested participant p and their
     * votes for time slots ts are registered.
     * @param p the participant
     * @param ts the time slots
     */
    1 usage
    public void vote(Participant p, Collection<TimeSlot> ts) {
        this.participants.add(p);
        for (TimeSlot t : ts) {
            this.timeSlotVotes.get(t).add(p);
        }
    }
```

In the suggestedActivity class the observer pattern is almost fulfilled firstly in the vote method the Participant is registered in participants, later they are then notified with the notifyAllParticipants method. If the participants here are considered observers, they then register for events by voting, it is as of now not possible to unregister from the event again. This could easily be added such that the participants could unregister from the event and then fulfill the observer pattern.



```
    /**
     * notifies all the interested participants when a time slot for
     * the activity has been chosen.
     */
    1 usage
    public void notifyAllParticipants() {
        for (Participant p : this.participants) {
            p.notify( a: this);
        }
    }
```

Task 11:



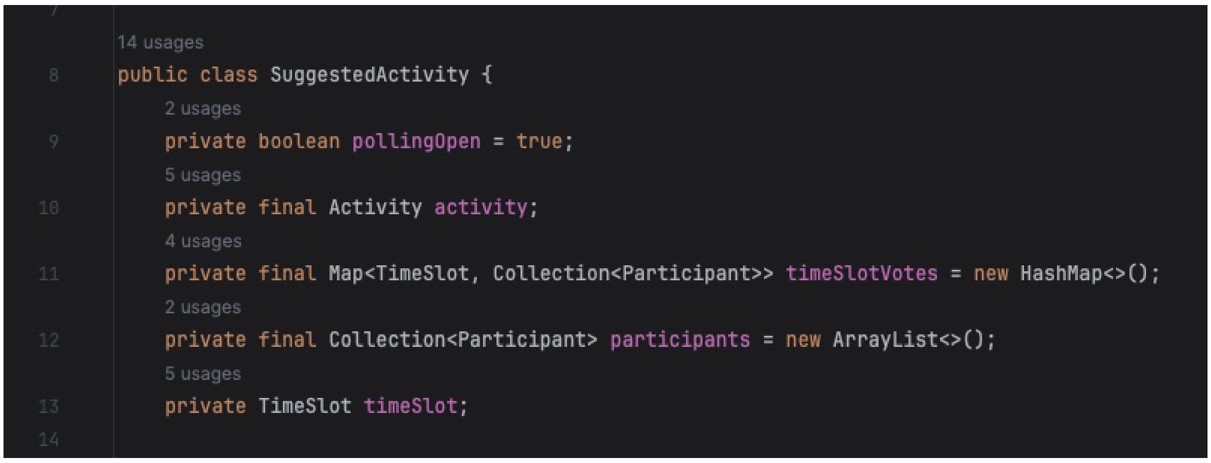
```java
    1 usage
    public boolean overlaps(TimeSlot other) {
        //If other is within this
        if(this.start.isBefore(other.start) && this.end.isAfter(other.start)){
            return true;
        }

        //If this is within other
        if(this.start.isAfter(other.start) && this.start.isBefore(other.end)){
            return true;
        }

        //If this ends after other starts and ends before other ends
        if(this.end.isAfter(other.start) && this.end.isBefore(other.end)){
            return true;
        }

        //If this ends after other and starts before other starts
        return this.end.isAfter(other.end) && this.start.isBefore(other.start);
    }
```

The four cases where the activity overlaps with another activity in the calendar are shown in the figure above. Lets call them t1 and t2 the two timeslots we compare for simplicity. The four cases are if:

t2 is within t1,
t1 is within t2.
t1 start after t2 starts and ends after t2 ends
t1 ends after t2 ends and starts before t2 starts

All these 4 cases are covered in the image above and thus it should ensure no overlap, there is one case though where this check isn't sufficient, and this is the case where t2 is equal to t1 in both start and end. Then these checks would return false due to the isBefore and isAfter methods returning false for cases that are equal (because it is not before or after as it is the same).

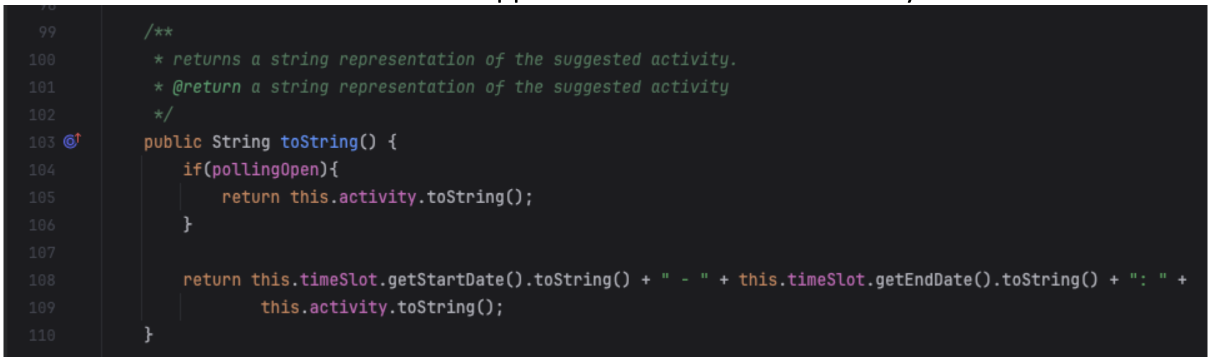
Task 12



```
14 usages
public class SuggestedActivity {
    2 usages
    private boolean pollingOpen = true;
    5 usages
    private final Activity activity;
    4 usages
    private final Map<TimeSlot, Collection<Participant>> timeSlotVotes = new HashMap<>();
    2 usages
    private final Collection<Participant> participants = new ArrayList<>();
    5 usages
    private TimeSlot timeSlot;
```

All variables in the SuggestedActivity are encapsulated, all are private and thus accessed (and changed) with methods within the class. This also adds a level of abstraction to the class since the user of this class has no need to know how the data is saved and only needs to retrieve data and have it stored.

Furthermore abstraction is also used on the Activity parameter as it is an abstract class the actual class of the parameter is evaluated in runTime when e.g I run .toString on activity that is method is overwritten by the runtime classes method. Since only meeting is an inheritor this will in the current version of the application be the class of activity.



```
    /**
     * returns a string representation of the suggested activity.
     * @return a string representation of the suggested activity
     */
    public String toString() {
        if(pollingOpen){
            return this.activity.toString();
        }

        return this.timeSlot.getStartDate().toString() + " - " + this.timeSlot.getEndDate().toString() + ": " +
                this.activity.toString();
    }
```