HuSCII – Group 2
TCSS 360A – Spring '15
Jingzhu Guo, Duy Huynh, Ian McPeek, Putthida Samrith
Project State

User stories (in priority order from most to least important).

1. As a Park Manager I want to submit a new job request.
    a. Code: This has been implemented fully; *ParkManager.java* has a method called *addJob()* which takes the data of a job taken from the console and passes it to *JobController.java*, which then creates a *Job* to be added in a collection containing all jobs.
    b. Test: Nothing can break; it is just adding a *Job* to a collection.
    c. Console: This user story can be carried out in console. The Business Rules associated with adding jobs can be verified via console. The console prevents illegal job data from being passed

2. As a Volunteer I want to view upcoming jobs I can sign up for.
    a. Code: This has been implemented fully; In *VolunteerConsole.java* the method *viewUpcomingJobs()* uses the method of the same name from the *JobController.java* class to retrieve a list of all jobs that haven't been already completed, and displays it to console.
    b. Test: This is tested in the *JobControllerTest.java* class to make sure upcoming jobs doesn't return any past jobs, and the *isCompleted()* method in *BusinessRules.java* is tested to check whether a job has been completed
    c. Console: This is enforced by a business rule to safely display only the jobs that are indeed upcoming.

3. As a Volunteer I want to volunteer for a job.
    a. Code: This has been fully implemented. From the console, the user can choose a job from a List of upcoming jobs. The job that volunteer chooses will be checked if it violates any business rules. If it doesn't, volunteer will then be added to that job by using addVolunteer() from *Job* class.
    b. Test: This has been fully tested. We test when there is no job to sign up, when job is full in all work categories, when volunteer already signed up for that job, when the work category that volunteer wants to sign up is full (BR3), and when volunteer already signed up for another job that occurs at the same time (BR7).
    c. Console: This user story can be carried out in *VolunteerConsole* class. Volunteer can sign up for a job if it doesn't violate any business rules.

4. As a Volunteer I want to view the jobs I am signed up for.
    a. Code: In *VolunteerConsole* class, there is a method called viewSignedUpJobs() which called getUpcomingJobs() from *JobController* class. We iterate through the list of all upcoming jobs, and check if any of those jobs contain volunteer's email.
    b. Test: We implemented two possible cases; when volunteer has signed up jobs, and when volunteer doesn't have any jobs.
    c. Console: This user story can be carried out in *VolunteerConsole* class. Volunteer is able to see the list of signed-up jobs if it has; otherwise, it will display a message stating there's no signed up job.

5. As a Park Manager I want to view upcoming jobs in the parks that I manage.
    a. Code: This has been fully implemented; viewMyJobs() method in *ParkManagerConsole* class called getMyJobs() from *ParkManager* class that returns only upcoming jobs for that park.
    b. Test: This has been fully tested. We test two possible cases; when park manager doesn't have any jobs, and when park manager has job. We also test to make sure that it will return only upcoming jobs in *JobControllerTest* class.
    c. Console:

6. As a Park Manager I want to view the Volunteers for a job in the parks that I manage.
    a. Code: There is a method called viewVolunteer() in *ParkManagerConsole* class which retrieved all the parks that park manager managed. This method also called getUserList() from *UserController* class to return all volunteers under that park name.
    b. Test: This has been fully tested. We test getManagedParks() to make sure that it retrieve only this park manager's managed parks in a list. We test when park manager doesn't have any jobs, so there is no volunteer. We also test when there are volunteers in that park job, and when there is no volunteer sign up for that job.
    c. Console: This user story has been carried out in *ParkManagerConsole* class.

7. As an Administrator, I want to search volunteers by last name.
    a. Code: In *AdministratorConsole* class, there is a method called hasLastName() which verifies if the input last name exists in the system. This method also called getVolunteers() in *UserController* class which checks each user for last name and "volunteer" role.
    b. Test: We test everything that could break. This included when there is no volunteer in the file, when there is only one name in the file, when there are multiple duplicated last names in the file, and when last name not found.
    c. Console: This user story can be carried out in *AdministratorConsole* class which will display volunteer's last name on the console or message when no volunteer found.

Business rules

1. A job may not be added if the total number of pending jobs is currently 30.
    a. Code: This has already been implemented in *BusinessRule* Class. The checkMaxJobs() method, which takes the list of all jobs in the system, compares it with the number of maximum jobs (30) that can be added. We check for boolean if the maximum job limit has been reached.

   b. Test: We test everything that could break. This included testing when the number of pending jobs in the system is more than 30, less than 30, and exactly 30.
   c. Console: In *ParkManagerConsole* class, when park manager wants to submit a job (User Story #1), we first check if it violates this rule. If it does, park manager will not be able to submit a job.

2. A job may not be added if the total number of pending jobs during that week (3 days on either side of the job days) is currently 5. In other words, during any consecutive 7 day period there can be no more than 5 jobs.
   a. Code: Implemented via *checkJobWeek()* in the *BusinessRules* class, which takes a list of jobs, retrieves all job dates and compares it to the date requested by the console. Returns a boolean: if false, the date requested is invalid.
   b. Test: Testing for this Business Rule is done in the BusinessRules class. We tested scenarios when the jobs for a week has been maxed out. Also tested for when there are multiple jobs that span more than a day that would count towards the 5 job max.
   c. Console: In *ParkManagerConsole* class, when park manager wants to submit a job and if there are too many jobs for that week, the console will prevent the user from submitting. Instead, the console will continue to prompt the user for valid dates.

3. A Volunteer may not sign up for a work category on a job if the maximum number of Volunteers for that work category has already been reached.
   a. Code: In *Job* class, the method addVolunteer() did handle this rule. Volunteer can only be added to the system only if the number of current volunteer needed for that job is less than the maximum number of volunteer needed.
   b. Test: We test everything that could break. This included when the number of current volunteer in light, medium, and heavy work categories is less than the number of maximum volunteer needed in each work category.
   c. Console: In *VolunteerConsole* class, volunteer has a choice to choose which work category he/she wants to sign up for. If it violates this rule, volunteer cannot sign up for a job.

4. A job may not be scheduled that lasts more than two days.
   a. Code: This has already been implemented. In *BusinessRule* class, the checkJobDuration() method takes the data of a job, and calculate the difference between end date and start date. We check for boolean to see if the job length is less than two days.
   b. Test: We test everything that could break. This included when the job length is less than two days, more than two days, and exactly at two days.
   c. Console: In *ParkManagerConsole* class, park manager has the ability to enter start and end date of a job when submitting a job (User Story #1). A job cannot be added if it violates this rule.

5. A job may not be added that is in the past or more than three months in the future.
    a. Code: In *BusinessRule* class, there is a method called isCompleted() that checks if a job is in a past. There is also a method called futureDate() that checks if a job is more than 3 months in the future. The method valiDate() checks for boolean if those two methods are true or false.
    b. Test: We test every possible date that could break this rule. This included the past date, yesterday date, today date, tomorrow date, date within the 3 months range, date exactly three months from today, date one day before 3 months from today, and date after 3 months from today.
    c. Console: In ParkManagerConsole, when park manager submits a job (User Story #1), he/she has the ability to enter the start and end date. We will check if the date violates this rule or not. If it does, park manager is not able to submit a job.

6. A Volunteer may not sign up for a job that has passed.
    a. Code: In *VolunteerConsole* Class, signMeUp() method called viewUpComingJob() method in which it takes the list of all upcoming jobs in the system by using getUpcomingJobs() from *JobController* class.
    b. Test: We test three possible cases; when there are no jobs in the file, when there are all past jobs in the file, and when there are both past and upcoming jobs. We successfully tested those three cases.
    c. Console: Volunteer is not able to see any past jobs when signs up for a job, because it displays only upcoming jobs for signing up.

7. A Volunteer may not sign up for two jobs on the same day.
    a. Code: In *BusinessRule* class, there is a method called checkTwoJobsSameDay() which takes the role of the user, list of all jobs in the system, and the date of a job that volunteer signs up for. This method loops through all the jobs that volunteer signs up for, and compare those dates with the date of a job that volunteer will sign up for.
    b. Test: We test everything that could break. This included when that volunteer already signed up for another job at the same time, and when there is no conflict date.
    c. Console: In *VolunteerConsole* class, when volunteer signs up for a job (User Story #3) in signMeUp() method, volunteer will not be able to sign up if it violates this rule.

8. A Park Manager can create jobs only for those parks that he/she manages.
    a. Code: In *ParkManagerConsole* class, a method submitJob() called getManagedParks() from *UserController* Class which retrieves this park manager's managed parks in a list.
    b. Test: We test everything that could break. We tested if the park manager's email matches the park name that he/she managed. We also tested when park manager doesn't have any parks.

c. Console: When park manager wants to submit a job (User Story #1), it will display only parks that park manager manages, and he/she needs to choose which park to create a job.