```java

    /**
     * This function replaces the set of divisions by the stated divisions. The function guarantees that the inverse membership is handled correctly.
     * @param divisionRepository The division repository needed to save the altered divisions.
     * @param eventRepository The event repository needed to save the altered events.
     * @param divs The new list of divisions for the user.
     */
    public void replaceDivisions(DivisionRepository divisionRepository, EventRepository eventRepository, List<Division> divs)
    {
        logger.debug("[{}] Replacing division set", this);

        List<Division> finalDivisions = DivisionHelper.getExpandedSetOfDivisions(divs, divisionRepository);
        List<Division> oldDivisions = divisions;

        if((finalDivisions == null || finalDivisions.isEmpty()) && (oldDivisions == null || oldDivisions.isEmpty()))
        {
            logger.debug("[{}] Division sets before and after are both empty", this);
            divisions = new ArrayList<>();
        } else if(finalDivisions == null || finalDivisions.isEmpty())
        {
            logger.debug("[{}] Division set after is empty, before is not. Removing membership subscription from old divisions", this);
            oldDivisions.stream().forEach(div -> div.removeMember(this));
            divisionRepository.save(oldDivisions);

            //Updating events, affected by division change
            oldDivisions.parallelStream().forEach(div -> {
                List<Event> changedEvents = eventRepository.findByInvitedDivision(div);
                changedEvents.parallelStream().forEach(event -> event.updateInvitedUser(divisionRepository));
                eventRepository.save(changedEvents);
            });
            divisions = new ArrayList<>();
        } else if(oldDivisions == null || oldDivisions.isEmpty())
        {
            logger.debug("[{}] Division set before is empty, after is not. Adding membership subscription to new divisions", this);
            finalDivisions.stream().forEach(div -> div.addMember(this));
            divisionRepository.save(finalDivisions);

            //Updating events, affected by division change
            finalDivisions.parallelStream().forEach(div -> {
                List<Event> changedEvents = eventRepository.findByInvitedDivision(div);
                changedEvents.parallelStream().forEach(event -> event.updateInvitedUser(divisionRepository));
                eventRepository.save(changedEvents);
            });
            divisions = finalDivisions;
        } else
        {
            logger.debug("[{}] Division set after and before are not empty. Applying changed membership subscriptions", this);
            List<Division> intersect = finalDivisions.stream().filter(oldDivisions::contains).collect(Collectors.toList()); //These items are already in the list, and do not need to be modified

            //Collecting changed division for batch save
            List<Division> changedDivisions = Collections.synchronizedList(new ArrayList<>());

            //Removing membership from removed divisions
            oldDivisions.parallelStream()
                    .filter(div -> !intersect.contains(div))
                    .forEach(div -> {
                        div.removeMember(this);
                        changedDivisions.add(div);
                    });

            //Adding membership to added divisions
            finalDivisions.parallelStream()
                    .filter(div -> !intersect.contains(div))
                    .forEach(div -> {
                        div.addMember(this);
                        changedDivisions.add(div);
                    });

            divisionRepository.save(changedDivisions);

            //Updating events, affected by division change
            changedDivisions.parallelStream().distinct().forEach(div -> {
                List<Event> changedEvents = eventRepository.findByInvitedDivision(div);
                changedEvents.parallelStream().distinct().forEach(event -> event.updateInvitedUser(divisionRepository));
                eventRepository.save(changedEvents);
            });
            divisions = finalDivisions;
        }
    }
```