

luisFelippeM

fix the long method code smell using extract method refactoring technique

```
public class StudyGoal extends Registry{ private String goal; private List<String> goalRequirements; private Boolean isCompleted; private
LocalDateTime createdDate; private Double goalCompletion; private StudyObjective studyObjective; private StudyPlan studyPlan; private String
summary;
```

```
public StudyGoal(String name, StudyObjective objective, StudyPlan plan) {
    this.name = name;
    this.studyObjective = objective;
    this.studyPlan = plan;
    goalRequirements = new ArrayList<>();
}

public void editActiveCompleted(boolean active, boolean completed){
    this.isActive = active;
    this.isCompleted = completed;
}

public String setGoalSummary(){
    StringBuilder summary = new StringBuilder();
    summary.append("Goal Summary:\n").append("\n\n");
    if(this.isActive){
        summary.append("Active Goal:\n").append(goal).append("\n\n");
    }
    if(this.isCompleted){
        summary.append("Completed Goal:\n").append(goal).append("\n\n");
    }
    if(this.goalRequirements != null){
        summary.append("Requirements:\n");
        for(String requirement : this.goalRequirements){
            summary.append(requirement).append(", ");
        }
    }
    if(this.studyPlan != null){
        summary.append("Plan:\n");
        summary.append(this.studyPlan.toString());
    }
    if(this.studyObjective != null){
        summary.append("Objective:\n");
        summary.append(this.studyObjective.toString());
    }
    this.summary = summary.toString();
    return summary.toString();
}
```

```
public void addRequirement(String requirement){
```

Can you explain the benefits?

```

public void addRequirement(String requirement){
    this.goalRequirements.add(requirement);
}

public void resetRequirements(){
    this.goalRequirements.clear();
}

public boolean isCompleted() {
    return isCompleted;
}

public void toggleIsCompleted(){
    this.isCompleted = !this.isCompleted;
}


public LocalDateTime getLimitDate() {
    return createdDate;
}

public void setLimitDate(LocalDateTime limitDate) {
    this.createdDate = limitDate;
}

public void addDaysLimitDate(int days){
    this.createdDate = this.createdDate.plusDays(days);
}

public void setGoal(String goal) {
    this.goal = goal;
}
}

```

 GitHub Copilot

Let's fix the long method `setGoalSummary()` by breaking it down into smaller, focused methods.

Plan:

1. Extract goal status summary into separate method
2. Extract requirements summary into separate method
3. Extract plan and objective summary into separate method
4. Combine these in main method

Implementation:

```

public class StudyGoal extends Registry {
    // ...existing code...

    public String setGoalSummary() {
        StringBuilder summary = new StringBuilder();
        summary.append("Goal Summary:\n").append("\n\n");

        appendGoalStatus(summary);
        appendRequirements(summary);
        appendPlanAndObjective(summary);

        this.summary = summary.toString();
        return summary.toString();
    }

    private void appendGoalStatus(StringBuilder summary) {
        if (this.isActive) {
            summary.append("Active Goal:\n").append(goal).append("\n\n");
        }
        if (this.isCompleted) {
            summary.append("Completed Goal:\n").append(goal).append("\n\n");
        }
    }

    private void appendRequirements(StringBuilder summary) {
        if (this.goalRequirements != null) {
            summary.append("Requirements:\n");
            for (String requirement : this.goalRequirements) {
                summary.append(requirement).append(", ");
            }
        }
    }

    private void appendPlanAndObjective(StringBuilder summary) {
        if (this.studyPlan != null) {
            summary.append("Plan:\n");
            summary.append(this.studyPlan.toString());
        }
        if (this.studyObjective != null) {
            summary.append("Objective:\n");
            summary.append(this.studyObjective.toString());
        }
    }

    // ...existing code...
}

```

The refactoring:

1. Creates smaller, focused methods with single responsibilities
2. Improves readability and maintainability
3. Makes the code easier to test
4. Maintains the same functionality while being more organized
5. Each extracted method handles a specific part of the summary generation

