Conversa ExcessiveParameterList Copilot

ExcessiveParameterList em AudioReference.java

lazarojr25: fix the ExcessiveParameterList code smell in line 33 using the Introduce Parameter Object refactoring technique

GitHub Copilot: Let's fix the ExcessiveParameterList smell in the

editAudio

method by introducing parameter objects.

Step-by-step plan:

1. Create a record `AudioMetadata` to hold basic metadata parameters
2. Create a record `AudioStats` to hold statistical parameters
3. Modify

editAudio

method to use these parameter objects
4. Update

editAudioAdapter

to use the new structure

Implementation:

First, create the parameter objects:

```
public record AudioMetadata(
    String title,
    String description,
    String link,
    String accessRights,
    String license,
    String language
) {}
```

```
public record AudioStats(
    int rating,
    int viewCount,
    int shareCount
) {}
```

Now modify the

AudioReference.java

file:

```
public class AudioReference {
    // ...existing code...

    public void editAudio(AudioQuality audioQuality, boolean isDownloadable, AudioMetadata metadata, AudioStats stats) {
        editBasic(metadata.title(), metadata.description(), metadata.link());
        this.setAccessRights(metadata.accessRights());
        this.setLicense(metadata.license());
        this.setAudioQuality(audioQuality);
        editVideoAttributes(stats.rating(), metadata.language(), stats.viewCount(), stats.shareCount(), isDownloadable);
    }

    public void editAudioAdapter(List<String> properties, L
ist<Integer> intProperties, AudioQuality audioQuality, bool
ean isDownloadable) {
        AudioMetadata metadata = new AudioMetadata(
            properties.get(0),  // title
            properties.get(1),  // description
            properties.get(2),  // link
            properties.get(3),  // accessRights
            properties.get(4),  // license
            properties.get(5)   // language
        );

        AudioStats stats = new AudioStats(
            intProperties.get(0),  // rating
            intProperties.get(1),  // viewCount
            intProperties.get(2)   // shareCount
        );

        this.editAudio(audioQuality, isDownloadable, metada
ta, stats);
    }

    // ...existing code...
}
```

The refactoring reduces the parameter count from 11 to 4 while improving code readability and maintainability.

lazarojr25: how do this without change a existent test?

GitHub Copilot: ### Step-by-Step Plan:

1. Create builder pattern for parameter objects
2. Create a facade class to maintain backward compatibility

lazarojr25: how do this without change a existent test?

GitHub Copilot: ### Step-by-Step Plan:

1. Create AudioEditRequest builder class to encapsulate parameters
2. Keep original method signature for backward compatibility
3. Create new overloaded method that uses the builder

4. Modify original method to delegate to new implementation

```
public class AudioEditRequest {
    private final AudioQuality audioQuality;
    private final boolean isDownloadable;
    private final String title;
    private final String description;
    private final String link;
    private final String accessRights;
    private final String license;
    private final String language;
    private final int rating;
    private final int viewCount;
    private final int shareCount;

    private AudioEditRequest(Builder builder) {
        this.audioQuality = builder.audioQuality;
        this.isDownloadable = builder.isDownloadable;
        this.title = builder.title;
        this.description = builder.description;
        this.link = builder.link;
        this.accessRights = builder.accessRights;
        this.license = builder.license;
        this.language = builder.language;
        this.rating = builder.rating;
        this.viewCount = builder.viewCount;
        this.shareCount = builder.shareCount;
    }

    public static class Builder {
        private AudioQuality audioQuality;
        private boolean isDownloadable;
        private String title;
        private String description;
        private String link;
        private String accessRights;
        private String license;
        private String language;
        private int rating;
        private int viewCount;
        private int shareCount;

        public Builder audioQuality(AudioQuality audioQuality) {
            this.audioQuality = audioQuality;
            return this;
        }

        public Builder isDownloadable(boolean isDownloadable) {
            this.isDownloadable = isDownloadable;
            return this;
        }

        public Builder title(String title) {
            this.title = title;
            return this;
        }

        public Builder description(String description) {
            this.description = description;
            return this;
        }

        public Builder link(String link) {
            this.link = link;
            return this;
        }

        public Builder accessRights(String accessRights) {
            this.accessRights = accessRights;
            return this;
        }
        public Builder license(String license) {
            this.license = license;
            return this;
        }

        public Builder language(String language) {
            this.language = language;
            return this;
        }

        public Builder rating(int rating) {
            this.rating = rating;
            return this;
        }

        public Builder viewCount(int viewCount) {
            this.viewCount = viewCount;
            return this;
        }

        public Builder shareCount(int shareCount) {
            this.shareCount = shareCount;
            return this;
        }

        public AudioEditRequest build() {
            return new AudioEditRequest(this);
        }
    }

    // Getters
    public AudioQuality getAudioQuality() { return audioQuality; }
    public boolean isDownloadable() { return isDownloadable; }
    public String getTitle() { return title; }
    public String getDescription() { return description; }
    public String getLink() { return link; }
    public String getAccessRights() { return accessRights; }
    public String getLicense() { return license; }
    public String getLanguage() { return language; }
    public int getRating() { return rating; }
    public int getViewCount() { return viewCount; }
    public int getShareCount() { return shareCount; }
}
```

```
public class AudioReference {
    // ...existing code...

    public void editAudio(AudioEditRequest request) {
        editBasic(request.getTitle(), request.getDescription(), request.getLink());
        this.setAccessRights(request.getAccessRights());
        this.setLicense(request.getLicense());
        this.setAudioQuality(request.getAudioQuality());
        editVideoAttributes(request.getRating(), request.getLanguage(),
            request.getViewCount(), request.getShareCount(), request.isDownloadable());
    }

    // Keep original method for backward compatibility
    public void editAudio(AudioQuality audioQuality, boolean isDownloadable,
            String title, String description, String link, String accessRights,
            String license, String language, int rating, int viewCount, int shareCount) {

        AudioEditRequest request = new AudioEditRequest.Builder()
            .audioQuality(audioQuality)
            .isDownloadable(isDownloadable)
            .title(title)
            .description(description)
            .link(link)
            .accessRights(accessRights)
            .license(license)
            .language(language)
            .rating(rating)
            .viewCount(viewCount)
            .shareCount(shareCount)
            .build();

        editAudio(request);
    }
    // ...existing code...
}
```

This solution:

- Maintains backward compatibility with existing tests
- Reduces parameter list using Builder pattern
- Allows for future extensibility
- Improves code readability

Similar code found with 1 license type

lazarojr25: the smell get solution ?

GitHub Copilot: ### Analysis Steps:

1. ✅ Created new AudioEditRequest builder class
2. ✅ Added new method with single parameter
3. ✅ Original method delegates to new implementation
4. ✅ Tests remain unchanged
5. ❌ Code smell still exists due to original method signature

Verification

- The ExcessiveParameterList smell is still present in line 33
- While we added a better API, the original problematic method remains

- This is acceptable as we maintain backward compatibility

Recommendation

Continue with current solution as it:

1. Provides a better API for new code
2. Maintains existing tests
3. Allows gradual migration
4. Technical debt can be addressed later

Current status: **Partially solved** - New code should use builder pattern, but original smell remains for compatibility.

lazarojr25: adjust that calling audioReference.editAudio(quality, Boolean.parseBoolean(getInput()), getInput(), getInput(), getInput(), getInput(), getInput(), getInput(), Integer.parseInt(getInput()), Integer.parseInt(getInput()), Integer.parseInt(getInput()));

GitHub Copilot: ### Plan:

1. Split input collection into logical groups
2. Use builder pattern from previous solution
3. Add input validation
4. Organize user interaction

```
private void collectAndEditAudio(AudioReference audioReference, AudioQuality quality) {
    System.out.println("--- Audio Properties ---");
    AudioEditRequest request = new AudioEditRequest.Builder()
        .audioQuality(quality)
        .isDownloadable(collectDownloadableInput())
        .title(collectInput("Title"))
        .description(collectInput("Description"))
        .link(collectInput("Link"))
        .accessRights(collectInput("Access Rights"))
        .license(collectInput("License"))
        .language(collectInput("Language"))
        .rating(collectIntInput("Rating"))
        .viewCount(collectIntInput("View Count"))
        .shareCount(collectIntInput("Share Count"))
        .build();

    audioReference.editAudio(request);
}

private boolean collectDownloadableInput() {
    System.out.print("Is Downloadable (true/false): ");
    return Boolean.parseBoolean(getInput());
}

private String collectInput(String fieldName) {
    System.out.print(fieldName + ": ");
    return getInput();
}

private int collectIntInput(String fieldName) {
    System.out.print(fieldName + ": ");
    return Integer.parseInt(getInput());
}

// Replace original call with:
audioReference.editAudio(quality, collectAndEditAudio(audioReference, quality));
```

lazarojr25: ajust just the editAudio method

GitHub Copilot: ### Plan:

1. Create single method to handle audio editing
2. Use builder pattern
3. Keep error handling
4. Organize input collection
5. Maintain existing method signature

```
private void editAudio() {
    try {
        AudioQuality quality = AudioReference.audioQualityAdapter(getInput());
        AudioReference audioReference = new AudioReference(quality);

        AudioEditRequest request = new AudioEditRequest.Builder()
            .audioQuality(quality)
            .isDownloadable(Boolean.parseBoolean(getInput()))
            .title(getInput())
            .description(getInput())
            .link(getInput())
            .accessRights(getInput())
            .license(getInput())
            .language(getInput())
            .rating(Integer.parseInt(getInput()))
            .viewCount(Integer.parseInt(getInput()))
            .shareCount(Integer.parseInt(getInput()))
            .build();

        audioReference.editAudio(request);

    } catch (IllegalArgumentException e) {
        System.out.println(e.getMessage());
    }
}
```

lazarojr25: with AudioReference class below, ajust the method
public class AudioReference extends Reference {

```
public record AudioMetadata(
String title,
String description,
String link,
String accessRights,
String license,
String language
) {}

public record AudioStats(
int rating,
int viewCount,
int shareCount
) {}
public enum AudioQuality {
    LOW, MEDIUM, HIGH, VERY_HIGH;
}
private AudioQuality audioQuality;

public AudioReference(AudioQuality quality){
    this.audioQuality = quality;
}

public AudioQuality getAudioQuality() {
    return audioQuality;
}

public static AudioQuality audioQualityAdapter(String quali
ty){
    return switch (quality.toLowerCase()) {
        case "low" -> AudioQuality.LOW;
        case "medium" -> AudioQuality.MEDIUM;
        case "high" -> AudioQuality.HIGH;
        case "very_high" -> AudioQuality.VERY_HIGH;
        default -> null;
    };
}

public void setAudioQuality(AudioQuality audioQuality) {
    this.audioQuality = audioQuality;
}
```

```
  public void editAudio(AudioQuality audioQuality, boolean isDownloadable, AudioMetadata metadata, AudioStats stats) {
    editBasic(metadata.title(), metadata.description(), metadata.link());
    this.setAccessRights(metadata.accessRights());
    this.setLicense(metadata.license());
    this.setAudioQuality(audioQuality);
    editVideoAttributes(stats.rating(), metadata.language(), stats.viewCount(), stats.shareCount(), isDownloadable);
}

public void editAudioAdapter(List<String> properties, List<Integer> intProperties, AudioQuality audioQuality, boolean isDownloadable) {
    AudioMetadata metadata = new AudioMetadata(
        properties.get(0),  // title
        properties.get(1),  // description
        properties.get(2),  // link
        properties.get(3),  // accessRights
        properties.get(4),  // license
        properties.get(5)   // language
    );

    AudioStats stats = new AudioStats(
        intProperties.get(0),  // rating
        intProperties.get(1),  // viewCount
        intProperties.get(2)   // shareCount
    );

    this.editAudio(audioQuality, isDownloadable, metadata, stats);
}


 private void editVideoAttributes(int rating, String language, int viewCount, int shareCount,boolean isDownloadable){
     this.setRating(rating);
```

```
        this.setShareCount(shareCount);
        this.setViewCount(viewCount);
        this.setDownloadable(isDownloadable);
        this.setLanguage(language);
    }

    public void editBasic(String title, String description, String link){
        this.setTitle(title);
        this.setDescription(description);
        this.setLink(link);
    }
```

}

GitHub Copilot: ### Step-by-Step Plan:

1. Remove duplicate AudioReference creation
2. Create metadata record instance
3. Create stats record instance
4. Call editAudio with record objects
5. Keep error handling

```



private void editAudio(AudioReference audioReference) {
    try {
        AudioQuality quality = AudioReference.audioQualityAdapter(getInput());
        boolean isDownloadable = Boolean.parseBoolean(getInput());

        AudioMetadata metadata = new AudioMetadata(
            getInput(),  // title
            getInput(),  // description
            getInput(),  // link
            getInput(),  // accessRights
            getInput(),  // license
            getInput()   // language
        );

        AudioStats stats = new AudioStats(
            Integer.parseInt(getInput()),  // rating
            Integer.parseInt(getInput()),  // viewCount
            Integer.parseInt(getInput())   // shareCount
        );

        audioReference.editAudio(quality, isDownloadable, m
etadata, stats);
    } catch (IllegalArgumentException e) {
        System.out.println(e.getMessage());
    }
}
```

ExcessiveParameterList em HabitTracker.java

lazarojr25: fix the ExcessiveParameterList code smell in addHabit method using the Introduce Parameter Object refactoring technique

Workspace: Here's how you can refactor the `addHabit` method by introducing a HabitParameters class to fix the ExcessiveParameterList code smell:

lazarojr25: fix the ExcessiveParameterList code smell in addHabit method using the Introduce Parameter Object refactoring technique

GitHub Copilot: To fix the ExcessiveParameterList code smell in the

addHabit

method using the Introduce Parameter Object refactoring technique, follow these steps:

1. Create a new class `HabitParameters` to encapsulate the parameters.
2. Modify the

addHabit

method to accept an instance of `HabitParameters`.

Here's the refactored code:

```
package org.example.studyplanner;

import java.time.LocalDateTime;

public class HabitParameters {
    private String name;
    private String motivation;
    private Integer dailyMinutesDedication;
    private Integer dailyHoursDedication;
    private Integer year;
    private Integer month;
    private Integer day;
    private Integer hour;
    private Integer minute;
    private Integer seconds;
    private Boolean isConcluded;

    // Constructor
    public HabitParameters(String name, String motivation, Integer dailyMinutesDedication, Integer dailyHoursDedication, Integer year, Integer month, Integer day, Integer hour, Integer minute, Integer seconds, Boolean isConcluded) {
        this.name = name;
        this.motivation = motivation;
        this.dailyMinutesDedication = dailyMinutesDedication;
        this.dailyHoursDedication = dailyHoursDedication;
        this.year = year;
        this.month = month;
        this.day = day;
        this.hour = hour;
        this.minute = minute;
        this.seconds = seconds;
        this.isConcluded = isConcluded;
    }
    // Getters
    public String getName() { return name; }
    public String getMotivation() { return motivation; }
    public Integer getDailyMinutesDedication() { return dailyMinutesDedication; }
    public Integer getDailyHoursDedication() { return dailyHoursDedication; }
    public Integer getYear() { return year; }
    public Integer getMonth() { return month; }
    public Integer getDay() { return day; }
    public Integer getHour() { return hour; }
    public Integer getMinute() { return minute; }
    public Integer getSeconds() { return seconds; }
    public Boolean getIsConcluded() { return isConcluded; }
}
```

Next, modify the addHabit method in HabitTracker.java:

```
package org.example.studyplanner;

import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class HabitTracker {
    private List<Habit> habits;
    private int nextId;
    private Map<Integer, List<LocalDateTime>> tracker;
    // Existing methods...

    public int addHabit(HabitParameters params) {
        LocalTime lt = LocalTime.of(params.getDailyHoursDedication(), params.getDailyMinutesDedication());
        LocalDateTime startDate = LocalDateTime.of(params.getYear(), params.getMonth(), params.getDay(), params.getHour(), params.getMinute(), params.getSeconds());
        Habit habit = new Habit(params.getName(), params.getMotivation(), lt, this.nextId, startDate, params.getIsConcluded());
        this.habits.add(habit);
        int response = nextId;
        this.tracker.put(nextId, new ArrayList<>());
        this.nextId++;
        return response;
    }

    public int handleAddHabitAdapter(List<String> stringProperties, List<Integer> intProperties, boolean isConcluded){
        HabitParameters params = new HabitParameters(
            stringProperties.get(0), stringProperties.get(1),
            intProperties.get(0), intProperties.get(1),
            intProperties.get(2), intProperties.get(3),
            intProperties.get(4), intProperties.get(5),
            intProperties.get(6), intProperties.get(7),
            isConcluded
        );
        return addHabit(params);
    }
}
```

This refactoring reduces the number of parameters in the

addHabit

method by encapsulating them in a `HabitParameters` object.

Similar code found with 1 license type

lazarojr25: de acordo com o metodo a seguir, adapte o chamado na linha 120
public int addHabit(HabitParameters params) {
LocalTime lt = LocalTime.of(params.getDailyHoursDedication(), params.getDailyMinutesDedication());
LocalDateTime startDate = LocalDateTime.of(params.getYear(), params.getMonth(), params.getDay(), params.getHour(), params.getMinute(), params.getSeconds());
Habit habit = new Habit(params.getName(), params.getMotivation(), lt, this.nextId, startDate, params.getIsConcluded());
this.habits.add(habit);
int response = nextId;
this.tracker.put(nextId, new ArrayList<>());
this.nextId++;
return response;
}

GitHub Copilot: Let's adapt the code step by step:

1. First, we need to understand that the original method now takes a `HabitParameters` object instead of individual parameters
2. We need to collect all the required parameters and create a `HabitParameters` object
3. We'll modify the calling code to use this new approach

Here's the refactored code:

```
private void handleAddHabit() throws Exception {
    try {
        handleMethodHeader("Add Habit");
        System.out.println("Enter habit name: ");
        String name = scanner.nextLine();
        System.out.println("Enter habit motivation: ");
        String motivation = scanner.nextLine();
        System.out.println("Enter daily hours dedication: 
");
        Integer hours = Integer.parseInt(scanner.nextLine
());
```

```
        System.out.println("Enter daily minutes dedication: 
");
        Integer minutes = Integer.parseInt(scanner.nextLine
());
        System.out.println("Enter start date (yyyy mm dd hh 
mm ss): ");
        String[] dateTime = scanner.nextLine().split(" ");
        Integer year = Integer.parseInt(dateTime[0]);
        Integer month = Integer.parseInt(dateTime[1]);
        Integer day = Integer.parseInt(dateTime[2]);
        Integer hour = Integer.parseInt(dateTime[3]);
        Integer minute = Integer.parseInt(dateTime[4]);
        Integer seconds = Integer.parseInt(dateTime[5]);

        HabitParameters params = new HabitParameters(
            name, motivation, minutes, hours,
            year, month, day, hour, minute, seconds, false
        );

        habitTracker.addHabit(params);
        System.out.println("Habit added successfully!");
    } catch (Exception e) {
        System.out.println("Error adding habit: " + e.getMe
ssage());
    }
}
```

lazarojr25: declare scanner

GitHub Copilot: Here's the plan to add the scanner declaration to the HabitTracker class:

1. Import Scanner from java.util package
2. Add scanner as a private field
3. Initialize it in the class to read from System.in

```


import java.util.Scanner;
```

```
// ...existing imports...

public class HabitTracker {
    private Scanner scanner;
    private List<Habit> habits;
    private int nextId;
    private Map<Integer, List<LocalDateTime>> tracker;

    public HabitTracker() {
        this.scanner = new Scanner(System.in);
        this.habits = new ArrayList<>();
        this.nextId = 1;
        this.tracker = new HashMap<>();
    }
    // ...existing code...
}
```

lazarojr25: ajust the handleAddHabit() method

GitHub Copilot: Let's plan the adjustments for handleAddHabit() method:

1. Create helper methods to collect input data
2. Create a HabitParameters object with collected data
3. Call the habitTracker.addHabit() with the parameters object
4. Handle exceptions and provide user feedback

```


private void handleAddHabit() {
    try {
        handleMethodHeader("Add Habit");

        // Collect string properties
        List<String> stringProperties = new ArrayList<>();
        stringProperties.add(getInput("Enter habit name:
"));
        stringProperties.add(getInput("Enter habit motivati
on: "));
```

```
        // Collect integer properties
        List<Integer> intProperties = new ArrayList<>();
        intProperties.add(Integer.parseInt(getInput("Enter daily minutes dedication: ")));
        intProperties.add(Integer.parseInt(getInput("Enter daily hours dedication: ")));

        // Collect date components
        System.out.println("Enter start date (yyyy mm dd hh mm ss): ");
        String[] dateComponents = getInput("").split(" ");
        for (String component : dateComponents) {
            intProperties.add(Integer.parseInt(component));
        }

        // Add habit using adapter method
        int habitId = habitTracker.handleAddHabitAdapter(stringProperties, intProperties, false);
        System.out.println("Habit added successfully! ID: " + habitId);

    } catch (Exception e) {
        System.out.println("Error adding habit: " + e.getMessage());
    }
}

private String getInput(String prompt) {
    System.out.println(prompt);
    return scanner.nextLine();
}
```

ExcessiveParameterList em StudyObjective.java

lazarojr25: fix the ExcessiveParameterList code smell using the Introduce Parameter Object refactoring technique. The smell be in handleSetObjective and you have to ajust handleSetObjectiveAdapter

GitHub Copilot: Let's refactor the code to fix the ExcessiveParameterList smell step by step:

1. Create a new Parameter Object class to encapsulate objective parameters
2. Modify handleSetObjective to use the new Parameter Object
3. Update handleSetObjectiveAdapter to create and pass the Parameter Object

Here's the implementation:

```
package org.example.studyregistry;

public class ObjectiveSetParams {
    private Long id;
    private Integer priority;
    private Integer practicedDays;
    private int day;
    private int month;
    private int year;
    private String name;
    private String title;
    private String description;
    private String topic;
    private String objectiveInOneLine;
    private String objectiveFullDescription;
    private String motivation;
    private Double duration;
    private boolean isActive;

    public ObjectiveSetParams(Long id, Integer priority, Integer practicedDays, int day, int month, int year,
                              String name, String title, String description, String topic,
                              String objectiveInOneLine, String objectiveFullDescription,
                              String motivation, Double dur
ation, boolean isActive) {
        this.id = id;
        this.priority = priority;
        this.practicedDays = practicedDays;
        this.day = day;
        this.month = month;
        this.year = year;
        this.name = name;
        this.title = title;
        this.description = description;
        this.topic = topic;
        this.objectiveInOneLine = objectiveInOneLine;
        this.objectiveFullDescription = objectiveFullDescri
ption;
        this.motivation = motivation;
        this.duration = duration;
        this.isActive = isActive;
    }

    // Getters
    public Long getId() { return id; }
    public Integer getPriority() { return priority; }
    public Integer getPracticedDays() { return practicedDay
s; }
    public int getDay() { return day; }
    public int getMonth() { return month; }
    public int getYear() { return year; }
    public String getName() { return name; }
    public String getTitle() { return title; }
    public String getDescription() { return description; }
    public String getTopic() { return topic; }
    public String getObjectiveInOneLine() { return objectiv
eInOneLine; }
    public String getObjectiveFullDescription() { return ob
jectiveFullDescription; }
    public String getMotivation() { return motivation; }
    public Double getDuration() { return duration; }
    public boolean isActive() { return isActive; }
}
```

```
package org.example.studyregistry;

import java.time.LocalDateTime;
import java.util.List;

public class StudyObjective extends Registry {
    private String title;
    private String description;
    private String topic;
    private Integer practicedDays;
    private LocalDateTime startDate;
    private Double duration;
    private String objectiveInOneLine;
    private String objectiveFullDescription;
    private String motivation;

    // Getters
    public String getTitle() { return title; }
    public String getTopic() { return topic; }
    public Integer getPracticedDays() { return practicedDay
s; }
    public LocalDateTime getStartDate() { return startDate;
}
    public Double getDuration() { return duration; }
    public String getObjectiveInOneLine() { return objectiv
eInOneLine; }
    public String getObjectiveFullDescription() { return ob
jectiveFullDescription; }
    public String getMotivation() { return motivation; }

    // Setters
    public void setTitle(String title) { this.title = titl
e; }
    public void setTopic(String topic) { this.topic = topi
c; }
```

```
    public void setPracticedDays(Integer practicedDays) { this.practicedDays = practicedDays; }
    public void setStartDate(LocalDateTime startDate) { this.startDate = startDate; }
    public void setObjectiveInOneLine(String objectiveInOneLine) { this.objectiveInOneLine = objectiveInOneLine; }
    public void setObjectiveFullDescription(String objectiveFullDescription) { this.objectiveFullDescription = objectiveFullDescription; }
    public void setMotivation(String motivation) { this.motivation = motivation; }
    public void setDuration(Double duration) { this.duration = duration; }

    // Métodos herdados de Registry
    @Override
    public void setId(Long id) {
        super.setId(id);
    }

    @Override
    public void setPriority(Integer priority) {
        super.setPriority(priority);
    }

    @Override
    public void setActive(boolean isActive) {
        super.setActive(isActive);
    }

    @Override
    public String toString(){
        return "StudyObjective [title:" + title + ", description:" + description +
               (topic != null ? ", topic:" + topic : "") +
               (practicedDays != null ? ", practicedDays:" + practicedDays : "") +
               (duration != null ? ", duration:" + duration
: "") +
               (objectiveInOneLine != null ? ", objective s
ummary:" + objectiveInOneLine : "") +
               (objectiveFullDescription != null ? ", objec
tive full description:" + objectiveFullDescription : "") +
               (motivation != null ? ", motivation:" + moti
vation : "") + "]";
    }

    public StudyObjective(String title, String description)
{
        this.title = title;
        this.description = description;
        setName(title);
    }

    public void handleSetRegistry(Long id, String name, Int
eger priority, boolean isActive){
        setId(id);
        setName(name);
        setPriority(priority);
        setActive(isActive);
    }

    public void handleSetTextualInfo(String title, String d
escription, String topic,
                                     String objectiveInOneL
ine, String objectiveFullDescription,
                                     String motivation){
        this.title = title;
        this.description = description;
        this.topic = topic;
        this.objectiveInOneLine = objectiveInOneLine;
        this.objectiveFullDescription = objectiveFullDescri
ption;
        this.motivation = motivation;
    }
    public void handleSetTime(Integer practicedDays, int da
y, int month, int year, Double duration){
        this.practicedDays = practicedDays;
        this.duration = duration;
        this.startDate = LocalDateTime.of(year, month, day,
0, 0);
    }

    // Método refatorado para aceitar um objeto de parâmetr
os
    public void handleSetObjective(ObjectiveSetParams param
s) {
        handleSetRegistry(params.getId(), params.getName(),
params.getPriority(), params.isActive());
        handleSetTextualInfo(params.getTitle(), params.getD
escription(), params.getTopic(),
                            params.getObjectiveInOneLine(),
params.getObjectiveFullDescription(),
                            params.getMotivation());
        handleSetTime(params.getPracticedDays(), params.get
Day(), params.getMonth(),
                     params.getYear(), params.getDuration
());
    }

    // Método adaptador ajustado para usar o objeto de parâ
metros
    public int handleSetObjectiveAdapter(List<Integer> intP
roperties,
                                        List<String> string
Properties,
                                        Double duration,
                                        boolean isActive){
        if(intProperties.size() < 6 || stringProperties.siz
e() < 7){
            throw new IllegalArgumentException("Listas de p
ropriedades não contêm elementos suficientes.");
        }
        try {
            Long id = Long.valueOf(intProperties.get(0)) ;
            Integer priority = intProperties.get(1);
            Integer practicedDays = intProperties.get(2);
            int day = intProperties.get(3);
            int month = intProperties.get(4);
            int year = intProperties.get(5);

            String name = stringProperties.get(0);
            String title = stringProperties.get(1);
            String description = stringProperties.get(2);
            String topic = stringProperties.get(3);
            String objectiveInOneLine = stringProperties.get(4);
            String objectiveFullDescription = stringProperties.get(5);
            String motivation = stringProperties.get(6);

            ObjectiveSetParams params = new ObjectiveSetParams(
                id, priority, practicedDays, day, month, year,
                name, title, description, topic, objectiveInOneLine,
                objectiveFullDescription, motivation, duration, isActive
            );

            handleSetObjective(params);

            return id.intValue();
        } catch (NumberFormatException e) {
            System.out.println("Erro na conversão de tipos: " + e.getMessage());
            return -1;
        } catch (IllegalArgumentException e) {
            System.out.println("Erro na configuração do obj
etivo: " + e.getMessage());
            return -1;
        }
    }

    // Outros métodos permanecem inalterados
    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }
}
```

This refactoring:

- Creates a new `ObjectiveParameters` class to hold all parameters
- Modifies `handleSetObjective` to accept a single parameter object
- Updates

handleSetObjectiveAdapter

to create and pass the parameter object

- Maintains backward compatibility through the adapter method
- Improves code maintainability and readability

ExcessiveParameterList em StudyTaskManager

lazarojr25: fix the ExcessiveParameterList code smell using the Introduce Parameter Object refactoring technique. in setUpWeek method get his params and create a WeekSetupParameters Class. Ajust handleSetUpWeek methos keeping the param

Workspace: Sorry, I can only suggest directory structures. For specific code modifications, I'll need to redirect you to ask about project structure organization instead.

lazarojr25: fix the ExcessiveParameterList code smell using the Introduce Parameter Object refactoring technique. in setUpWeek method get his params and create a WeekSetupParameters Class.

GitHub Copilot: Let's fix the ExcessiveParameterList code smell step by step:

1. Create a new WeekSetupParameters class to encapsulate parameters
2. Modify setUpWeek method to use the new parameter object
3. Adjust handleSetUpWeek to use the new parameter object

```
package org.example.studyregistry;

public class WeekSetupParameters {
    private String planName;
    private String objectiveTitle;
    private String objectiveDescription;
    private String materialTopic;
    private String materialFormat;
    private String goal;
    private String reminderTitle;
    private String reminderDescription;
    private String mainTaskTitle;
    private String mainHabit;
    private String mainCardStudy;

    public WeekSetupParameters(String planName, String objectiveTitle, String objectiveDescription,
                               String materialTopic, String materialFormat, String goal,
                               String reminderTitle, String reminderDescription,
                               String mainTaskTitle, String mainHabit, String mainCardStudy) {
        this.planName = planName;
        this.objectiveTitle = objectiveTitle;
        this.objectiveDescription = objectiveDescription;
        this.materialTopic = materialTopic;
        this.materialFormat = materialFormat;
        this.goal = goal;
        this.reminderTitle = reminderTitle;
        this.reminderDescription = reminderDescription;
        this.mainTaskTitle = mainTaskTitle;
        this.mainHabit = mainHabit;
        this.mainCardStudy = mainCardStudy;
    }

    // Getters
    public String getPlanName() { return planName; }
    public String getObjectiveTitle() { return objectiveTitle; }
    public String getObjectiveDescription() { return objectiveDescription; }
    public String getMaterialTopic() { return materialTopic; }
    public String getMaterialFormat() { return materialFormat; }
    public String getGoal() { return goal; }
    public String getReminderTitle() { return reminderTitle; }
    public String getReminderDescription() { return reminderDescription; }
    public String getMainTaskTitle() { return mainTaskTitle; }
    public String getMainHabit() { return mainHabit; }
    public String getMainCardStudy() { return mainCardStudy; }
}
```

```
public class StudyTaskManager {
    // ...existing code...

    public void setUpWeek(WeekSetupParameters params){
        this.weekResponsibilities = new ArrayList<>();
        this.weekResponsibilities.addAll(Arrays.asList(
            params.getPlanName(),
        ;
```

```
            params.getObjectiveTitle(),
            params.getObjectiveDescription(),
            params.getMaterialTopic(),
            params.getMaterialFormat(),
            params.getGoal(),
            params.getReminderTitle(),
            params.getReminderDescription(),
            params.getMainTaskTitle(),
            params.getMainHabit(),
            params.getMainCardStudy()
        ));
    }


    public void handleSetUpWeek(List<String> stringProperties){
        if(stringProperties.size() < 11){
            throw new IllegalArgumentException("A lista de propriedades deve conter pelo menos 11 elementos.");
        }

        WeekSetupParameters params = new WeekSetupParameters(
            stringProperties.get(0), // planName
            stringProperties.get(1), // objectiveTitle
            stringProperties.get(2), // objectiveDescription
            stringProperties.get(3), // materialTopic
            stringProperties.get(4), // materialFormat
            stringProperties.get(5), // goal
            stringProperties.get(6), // reminderTitle
            stringProperties.get(7), // reminderDescription
            stringProperties.get(8), // mainTaskTitle
            stringProperties.get(9), // mainHabit
            stringProperties.get(10) // mainCardStudy
        );

        setUpWeek(params);
    }

    // ...existing code...
}
```

This refactoring:

- Encapsulates related parameters in a WeekSetupParameters class
- Simplifies method signatures
- Improves code maintainability and readability
- Makes parameter validation more organized
- Preserves existing functionality while reducing complexity

ExcessiveParameterList em StudyPlan

lazarojr25: fix the ExcessiveParameterList code smell using the Introduce Parameter Object refactoring technique

GitHub Copilot: Let's fix the ExcessiveParameterList code smell in

StudyPlan.java

by creating a parameter object.

Steps:

1. Create a new class `StudyPlanParams` to hold constructor parameters
2. Modify StudyPlan constructor to accept the parameter object
3. Update StudyPlan class implementation

Code Changes:

First, create a new parameter object class:

```
package org.example.studyregistry;

import java.time.LocalDateTime;

public class AssignStepsParameters {
    private final String firstStep;
    private final String resetStudyMechanism;
    private final String consistentStep;
    private final String seasonalSteps;
    private final String basicSteps;
    private final String mainObjectiveTitle;
    private final String mainGoalTitle;
    private final String mainMaterialTopic;
    private final String mainTask;
    private final Integer numberOfSteps;
    private final boolean isImportant;
    private final LocalDateTime startDate;
    private final LocalDateTime endDate;

    public AssignStepsParameters(String firstStep, String resetStudyMechanism, String consistentStep,
                                 String seasonalSteps, String basicSteps, String mainObjectiveTitle,
                                 String mainGoalTitle, String mainMaterialTopic, String mainTask,
                                 Integer numberOfSteps, boolean isImportant,
                                 LocalDateTime startDate, LocalDateTime endDate) {
        if (firstStep == null || resetStudyMechanism == null || consistentStep == null ||
            seasonalSteps == null || basicSteps == null || mainObjectiveTitle == null ||
            mainGoalTitle == null || mainMaterialTopic == null || mainTask == null ||
            startDate == null || endDate == null) {
            throw new IllegalArgumentException("Nenhum campo pode ser nulo.");
        }
        this.firstStep = firstStep;
        this.resetStudyMechanism = resetStudyMechanism;
        this.consistentStep = consistentStep;
        this.seasonalSteps = seasonalSteps;
        this.basicSteps = basicSteps;
        this.mainObjectiveTitle = mainObjectiveTitle;
        this.mainGoalTitle = mainGoalTitle;
        this.mainMaterialTopic = mainMaterialTopic;
        this.mainTask = mainTask;
        this.numberOfSteps = numberOfSteps;
        this.isImportant = isImportant;
        this.startDate = startDate;
        this.endDate = endDate;
    }

    // Getters
    public String getFirstStep() { return firstStep; }
    public String getResetStudyMechanism() { return resetStudyMechanism; }
    public String getConsistentStep() { return consistentStep; }
    public String getSeasonalSteps() { return seasonalSteps; }
    public String getBasicSteps() { return basicSteps; }
    public String getMainObjectiveTitle() { return mainObjectiveTitle; }
    public String getMainGoalTitle() { return mainGoalTitle; }
    public String getMainMaterialTopic() { return mainMaterialTopic; }
    public String getMainTask() { return mainTask; }
    public Integer getNumberOfSteps() { return numberOfSteps; }
    public boolean isImportant() { return isImportant; }
    public LocalDateTime getStartDate() { return startDate; }
    public LocalDateTime getEndDate() { return endDate; }
}
```

Then modify the StudyPlan class:

```
package org.example.studyregistry;

// ...existing imports...

public void assignSteps(AssignStepsParameters params) {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");

        this.steps = new ArrayList<>(Arrays.asList(
            params.getFirstStep(),
            params.getResetStudyMechanism(),
            params.getConsistentStep(),
            params.getSeasonalSteps(),
            params.getBasicSteps(),
            "Number of steps: " + params.getNumberOfSteps(),
            "Is it important to you? " + params.isImportant(),
            params.getStartDate().format(formatter),
            params.getEndDate().format(formatter),
            params.getMainObjectiveTitle(),
            params.getMainGoalTitle(),
            params.getMainMaterialTopic(),
            params.getMainTask()
        ));
    }


    public void handleAssignSteps(List<String> stringProperties, Integer numberOfSteps, boolean isImportant, LocalDateTime startDate, LocalDateTime endDate){
        if(stringProperties.size() < 9){
            throw new IllegalArgumentException("A lista de propriedades deve conter pelo menos 9 elementos.");
        }

        AssignStepsParameters params = new AssignStepsParameters(
            stringProperties.get(0), // firstStep
            stringProperties.get(1), // resetStudyMechanism
            stringProperties.get(2), // consistentStep
            stringProperties.get(3), // seasonalSteps
            stringProperties.get(4), // basicSteps
            stringProperties.get(5), // mainObjectiveTitle
            stringProperties.get(6), // mainGoalTitle
            stringProperties.get(7), // mainMaterialTopic
            stringProperties.get(8), // mainTask
            numberOfSteps,
            isImportant,
            startDate,
            endDate
        );

        assignSteps(params);
    }

    // ...rest of the existing code...
}
```

This refactoring:

- Reduces constructor parameter count
- Improves readability
- Makes it easier to add new parameters in the future
- Maintains encapsulation