

luisFelippeM



fix the long method code smell using extract method refactoring technique

```
public class TodoTracker { private List<ToDo> toDos = new ArrayList<>(); private Map<Integer, List<LocalDateTime>> tracker; private Integer nextId; private static TodoTracker instance;
```

```
private TodoTracker() {
    this.tracker = new HashMap<>();
    this.toDos = new ArrayList<>();
    this.nextId = 1;
}

public static TodoTracker getInstance() {
    if (instance == null) {
        instance = new TodoTracker();
    }
    return instance;
}

@Override
public String toString() {
    StringBuilder str = new StringBuilder();
    for (ToDo toDo : toDos) {
        String todoInfo = toDo.toString();
        str.append(todoInfo);
        str.append("\n");
        Integer id = toDo.getId();
        List<LocalDateTime> todosDate = this.tracker.get(id);
        if(todosDate == null){
            str.append("No tracks found\n");
        }else{
            for (LocalDateTime ldt : todosDate) {
                String pattern = "yyyy-MM-dd HH:mm:ss";
                DateTimeFormatter formatter = DateTimeFormatter.ofPattern(pattern);
                String formattedDate = formatter.format(ldt);
                str.append(formattedDate);
                str.append("\n");
            }
        }
    }
    String response = str.toString();
    if(response.isEmpty()){
        return "No ToDos found";
    }
    return response;
}
```



```

public void addToDoExecutionTime(Integer id){
    List<LocalDateTime> et = tracker.computeIfAbsent(id, k -> new ArrayList<>());
    LocalDateTime now = LocalDateTime.now();
    et.add(now);
}

public List<ToDo> getTodos() {
    return todos;
}

public ToDo getToDoById(Integer id) {
    for (ToDo toDo : todos) {
        if (toDo.getId() == id) {
            return toDo;
        }
    }
    return null;
}


public Integer addToDo(String title, String description, Integer priority) {
    ToDo toAdd = new ToDo(nextId, title, description, priority);
    nextId++;
    this.todos.add(toAdd);
    return toAdd.getId();
}

public void removeToDo(Integer id) {
    todos.removeIf(toDo -> toDo.getId() == id);
}

public List<ToDo> sortTodosByPriority() {
    List<ToDo> sortedTodos = new ArrayList<>(todos);
    sortedTodos.sort(Comparator.comparingInt(ToDo::getPriority));
    return sortedTodos;
}

public List<String> searchInTodos(String search) {
    List<String> todos = new ArrayList<>();
    for (ToDo toDo : todos) {
        if (toDo.getTitle().toLowerCase().contains(search.toLowerCase()) || toDo.getDescription().toLowerCase().contains(search.toLowerCase())) {
            todos.add(toDo.toString());
        }
    }
    return todos;
}
}

```

 GitHub Copilot

Let's refactor the long `toString()` method by breaking it down into smaller, more focused methods.

Plan:

1. Extract date formatting logic into a separate method
2. Extract tracking info building logic into a separate method
3. Extract todo list building logic into a separate method
4. Simplify the main `toString()` method

Here's the refactored code:

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

    @Override
    public String toString() {
        String todoListStr = buildTodoListString();
        return todoListStr.isEmpty() ? "No Todos found" : todoListStr;
    }

    private String buildTodoListString() {
        StringBuilder str = new StringBuilder();
        for (ToDo todo : todos) {
            str.append(buildTodoWithTrackingInfo(todo));
        }
        return str.toString();
    }

    private String buildTodoWithTrackingInfo(ToDo todo) {
        StringBuilder str = new StringBuilder();
        str.append(todo.toString()).append("\n");
        str.append(getTrackingInfo(todo.getId()));
        return str.toString();
    }

    private String getTrackingInfo(Integer id) {
        List<LocalDateTime> todosDate = this.tracker.get(id);
        if (todosDate == null) {
            return "No tracks found\n";
        }
        return formatTrackingDates(todosDate);
    }

    private String formatTrackingDates(List<LocalDateTime> dates) {
        StringBuilder str = new StringBuilder();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        for (LocalDateTime dt : dates) {
            str.append(dt.format(formatter)).append("\n");
        }
        return str.toString();
    }
}
```

Can you explain the changes made?

```

private String formatTrackingDates(List<LocalDateTime> dates) {
    StringBuilder str = new StringBuilder();
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
    for (LocalDateTime ldt : dates) {
        str.append(formatter.format(ldt)).append("\n");
    }
    return str.toString();
}

// ...existing code...
}

```

The refactoring breaks down the complex `toString()` method into smaller, single-responsibility methods:

- `buildTodoListString()`: Handles the main todo list building
- `buildTodoWithTrackingInfo()`: Builds string for a single todo with its tracking info
- `getTrackingInfo()`: Gets tracking information for a specific todo
- `formatTrackingDates()`: Formats the tracking dates

This improves:

- Code readability
- Maintainability
- Testability
- Reusability of the extracted methods

🔄 ⬆️ ↩️ 📄

LuisFelippeM



```
private String buildToDoWithTrackingInfo(ToDo toDo) { StringBuilder str = new StringBuilder(); str.append(toDo.toString()).append("\n"); str.append(getTrackingInfo(toDo.getId())); return str.toString(); }
```

this is now a feature envy code smell, solve it

```
public class ToDo implements PlannerMaterial {
    private Integer id;
    private String title;
    private String description;
    private int priority;

    public ToDo(Integer id, String title, String description, int priority) {
        validateFields(id, title, priority);
        this.id = id;
        this.title = title;
        this.description = description;
        this.priority = priority;
    }

    private void validateFields(Integer id, String title, int priority) {
        if (id == null || id < 0) {
            throw new IllegalArgumentException("ID must be positive");
        }
        if (title == null || title.trim().isEmpty()) {
            throw new IllegalArgumentException("Title cannot be empty");
        }
        if (priority < 1 || priority > 5) {
            throw new IllegalArgumentException("Priority must be between 1 and 5");
        }
    }

    public void updateTitle(String newTitle) {
        if (newTitle == null || newTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Title cannot be empty");
        }
        this.title = newTitle;
    }

    public void updateDescription(String newDescription) {
        this.description = newDescription != null ? newDescription : "";
    }

    public void increasePriority() {
        if (priority < 5) priority++;
    }
}
```



```
public void decreasePriority() {
    if (priority > 1) priority--;
}

public boolean isHighPriority() {
    return priority >= 4;
}

public int getId() {
    return id;
}

public void setId(Integer id) {
    if (id == null || id < 0) {
        throw new IllegalArgumentException("ID must be positive");
    }
    this.id = id;
}

public String getTitle() {
    return title;
}

public String getDescription() {
    return description;
}

public int getPriority() {
    return priority;
}

@Override
public String toString() {
    return MessageFormat.format("[{0}] Priority:{3} To Do {0}: {1}, {2}]",
        id, title, description, priority);
}
```

Step by Step Plan

1. Analyze the feature envy: `buildTodoWithTrackingInfo` is using more features of `ToDo` than its own class
2. Move the tracking info formatting logic into the `ToDo` class
3. Add tracking-related methods to `ToDo`
4. Update `ToDoTracker` to use the new methods

Implementation

```
public class ToDo implements PlannerMaterial {
    // ...existing code...

    public String getFormattedTodoWithTracking(List<LocalDateTime> trackingDates) {
        StringBuilder str = new StringBuilder();
        str.append(this.toString()).append("\n");

        if (trackingDates == null || trackingDates.isEmpty()) {
            return str.append("No tracks found\n").toString();
        }

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        for (LocalDateTime date : trackingDates) {
            str.append(formatter.format(date)).append("\n");
        }
        return str.toString();
    }
}
```

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

    private String buildTodoWithTrackingInfo(ToDo toDo) {
        return toDo.getFormattedTodoWithTracking(tracker.get(toDo.getId()));
    }

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

This refactoring:

- Moves formatting responsibility to the `ToDo` class
- Reduces coupling between `ToDoTracker` and `ToDo`
- Makes the code more maintainable and cohesive