

luisFelippeM

fix the long method code smell using the extract method refactoring technique

```
public class LeitnerSystem extends StudyMethod{ List<Box> boxes = null; public LeitnerSystem(String methodName) { super(methodName);  
boxes = new ArrayList<>(Arrays.asList(new Box(), new Box(), new Box(), new Box(), new Box())); }
```

```
@Override  
public String getMethodName() {  
    return this.methodName;  
}  
  
@Override  
void setMethodName(String methodName) {  
    this.methodName = methodName;  
}  
  
@Override  
public String toString(){  
    StringBuilder response = new StringBuilder();  
    int index = 0;  
    for(Box box : boxes){  
        response.append("Box ").append(index).append(": ").append(box.toString()).append("\n");  
        index++;  
    }  
    return response.toString();  
}  
  
public void clearBoxes(){  
    boxes.clear();  
    boxes = new ArrayList<>(Arrays.asList(new Box(), new Box(), new Box(), new Box(), new Box()));  
}  
  
public List<Box> getBoxes() {  
    return boxes;  
}  
  
public String getRandomCard(List<Box> otherBoxes){  
    if(otherBoxes == null){  
        return null;  
    }  
    if(otherBoxes.isEmpty()){  
        return null;  
    }  
    Box allBoxes = new Box();  
    for(Box box : otherBoxes){  
        allBoxes.addCards(box.getCards());  
    }  
}
```

What other refactorings can we apply?

```

    }
    Integer randomCard = allBoxes.getRandomCard();
    if(randomCard == null){
        return "No card found";
    }
    CardManager manager = CardManager.getCardManager();
    Card card = manager.getCard(randomCard);
    String response = "["+ randomCard + "]" ";
    response += "The random question was: " + card.getQuestion() + " | ";
    response += "The answer is: " + card.getAnswer();
    return response;
}

public void addCardToBox(Integer id, Integer boxId) {
    this.boxes.get(boxId).addCard(id);
}

public void removeCardFromBox(Integer id, Integer boxId) {
    this.boxes.get(boxId).removeCard(id);
}

public Card takeCardFromBox(Integer boxId){
    Integer cardId = boxes.get(boxId).getRandomCard();
    return this.cardManager.getCard(cardId);
}

public void boxIdValidation(Integer boxId) throws Exception {
    if(boxId == null || boxId > (boxes.size()-1) || boxId <= 0){
        throw new Exception("Invalid box ID");
    }
}

public void upgradeCard(Integer cardId, Integer boxId) throws Exception {
    boxIdValidation(boxId);

    Box refBox = boxes.get(boxId);
    if(refBox.hasCard(cardId)){
        throw new Exception("No card Found");
    }
    refBox.removeCard(cardId);
    boxes.get(Math.min(boxId + 1, 4)).addCard(cardId);
}

public void downgradeCard(Integer cardId, Integer boxId) throws Exception {
    boxIdValidation(boxId);


    Box refBox = boxes.get(boxId);
    if(refBox.hasCard(cardId)){

```

What other refactorings can we apply?

```
}  
    refBox.removeCard(cardId);  
    boxes.get(Math.max(boxId - 1, 0)).addCard(cardId);  
}
```

```
}
```

 GitHub Copilot

Let's refactor this code by extracting methods to improve readability and maintainability.

Steps:

1. Extract box initialization logic
2. Extract random card response formatting
3. Split card movement validation and execution
4. Create utility method for box boundary checks

Here's the refactored code:

```
public class LeitnerSystem extends StudyMethod {
    List<Box> boxes = null;

    public LeitnerSystem(String methodName) {
        super(methodName);
        initializeBoxes();
    }

    private void initializeBoxes() {
        boxes = new ArrayList<>(Arrays.asList(new Box(), new Box(), new Box(), new Box(), new Box()));
    }

    @Override
    public String getMethodName() {
        return this.methodName;
    }

    @Override
    void setMethodName(String methodName) {
        this.methodName = methodName;
    }

    @Override
    public String toString() {
        StringBuilder response = new StringBuilder();
        int index = 0;
        for (Box box : boxes) {
            response.append("Box ").append(index).append(": ").append(box.toString()).append("\n");
            index++;
        }
        return response.toString();
    }

    public void clearBoxes() {
        boxes.clear();
        initializeBoxes();
    }

    public List<Box> getBoxes() {
        return boxes;
    }

    public String getRandomCard(List<Box> otherBoxes) {
        if (otherBoxes == null || otherBoxes.isEmpty()) {
            return null;
        }
    }
}
```

```

        Integer randomCardId = getRandomCardFromBoxes(otherBoxes);
        if (randomCardId == null) {
            return "No card found";
        }

        return formatCardResponse(randomCardId);
    }

    private Integer getRandomCardFromBoxes(List<Box> otherBoxes) {
        Box allBoxes = new Box();
        for (Box box : otherBoxes) {
            allBoxes.addCards(box.getCards());
        }
        return allBoxes.getRandomCard();
    }

    private String formatCardResponse(Integer cardId) {
        CardManager manager = CardManager.getCardManager();
        Card card = manager.getCard(cardId);
        return String.format("[%d] The random question was: %s | The answer is: %s",
            cardId, card.getQuestion(), card.getAnswer());
    }

    public void addCardToBox(Integer id, Integer boxId) {
        this.bboxes.get(boxId).addCard(id);
    }

    public void removeCardFromBox(Integer id, Integer boxId) {
        this.bboxes.get(boxId).removeCard(id);
    }

    public Card takeCardFromBox(Integer boxId) {
        Integer cardId = bboxes.get(boxId).getRandomCard();
        return this.cardManager.getCard(cardId);
    }

    private void validateBoxId(Integer boxId) throws Exception {
        if (boxId == null || boxId > (bboxes.size() - 1) || boxId <= 0) {
            throw new Exception("Invalid box ID");
        }
    }

    private void validateCardInBox(Box box, Integer cardId) throws Exception {
        if (box.hasCard(cardId)) {
            throw new Exception("No card Found");
        }
    }
}

```

```

private int getTargetBoxIndex(Integer currentBox, boolean isUpgrade) {
    return isUpgrade ?
        Math.min(currentBox + 1, 4) :
        Math.max(currentBox - 1, 0);
}

public void upgradeCard(Integer cardId, Integer boxId) throws Exception {
    moveCard(cardId, boxId, true);
}

public void downgradeCard(Integer cardId, Integer boxId) throws Exception {
    moveCard(cardId, boxId, false);
}

private void moveCard(Integer cardId, Integer boxId, boolean isUpgrade) throws Exception {
    validateBoxId(boxId);
    Box sourceBox = boxes.get(boxId);
    validateCardInBox(sourceBox, cardId);

    sourceBox.removeCard(cardId);
    int targetBoxIndex = getTargetBoxIndex(boxId, isUpgrade);
    boxes.get(targetBoxIndex).addCard(cardId);
}
}

```

Key improvements:

- Extracted box initialization into separate method
- Created utility methods for card movement logic
- Separated card response formatting
- Added helper method for box boundary calculations
- Improved validation handling
- Combined upgrade/downgrade logic into single method with boolean flag
- Improved naming for better readability

🔄 📄 💬 📌

What other refactorings can we apply?