

fix the feature envy code smell using move method refactoring technique

```
public class StudyCardsController { private FlashCard flashCard = new FlashCard("FlashCard"); private LeitnerSystem leitnerSystem = new LeitnerSystem("LeitnerSystem"); private CardManager manager = CardManager.getCardManager(); private Map<String, Runnable> actions = new HashMap<>();
```

```
    public StudyCardsController() {
        assignActions();
    }

    public StudyCardsController(LeitnerSystem leitnerSystem) {
        assignActions();
        this.leitnerSystem = leitnerSystem;
    }

    void assignActions(){
        actions.put("1", this::handleViewCards);
        actions.put("2", this::handleCreateCard);
        actions.put("3", this::handleRemoveCard);
        actions.put("4", this::handleRandomFlashCard);
        actions.put("5", this::handleInsertCardInBox);
        actions.put("6", this::handleRemoveCardFromBox);
        actions.put("7", this::handleUpgradeCardFromBox);
        actions.put("8", this::handleDowngradeCardFromBox);
        actions.put("9", this::handleViewBoxes);
        actions.put("10", this::handleGetRandomCardFromBox);
    }

    public void handleViewCards(){
        Map<Integer, Card> cards = manager.getCardsMap();
        List<Integer> keys = new ArrayList<>(cards.keySet());
        StringBuilder response = new StringBuilder();
        for(Integer key : keys){
            Card card = cards.get(key);
            response.append("[id: ").append(key).append("] Question: ").append(card.getQuestion()).append(",
        }
        System.out.println(response.toString().isEmpty() ? "No cards" : response.toString());
    }

    public void handleRemoveCard(){
        System.out.println("Type card id:");
        int id = Integer.parseInt(getInput());
        manager.removeCard(id);
    }
}
```

```

public void handleCreateCard(){
    System.out.println("Type the question: \n");
    String question = getInput();
    System.out.println("Type the answer: \n");
    String answer = getInput();
    manager.addCard(question, answer);
}

public void handleRandomFlashCard() {
    System.out.println("Random flash card:");
    Integer id = flashCard.randomFlashCard();
    System.out.println(manager.formatCard(id));
}

public void handleInsertCardInBox(){
    System.out.println("Type card id:");
    int id = Integer.parseInt(getInput());
    System.out.println("Type box(0-4):");
    int box = Integer.parseInt(getInput());
    leitnerSystem.addCardToBox(id, box);
}

public void handleRemoveCardFromBox(){
    System.out.println("Type card id:");
    int id = Integer.parseInt(getInput());
    System.out.println("Type box(0-4):");
    int box = Integer.parseInt(getInput());
    leitnerSystem.removeCardFromBox(id, box);
}

public void handleViewBoxes(){
    System.out.println(leitnerSystem.toString());
}

public void handleUpgradeCardFromBox() {
    try{
        System.out.println("Type card id:");
        int id = Integer.parseInt(getInput());
        System.out.println("Type box(0-4):");
        int box = Integer.parseInt(getInput());
        leitnerSystem.upgradeCard(id, box);
    }catch (Exception e){
        System.out.println(e.getMessage());
    }
}
}

```

```

public void handleDowngradeCardFromBox() {
    try{
        System.out.println("Type card id:");
        int id = Integer.parseInt(getInput());
        System.out.println("Type box(0-4):");
        int box = Integer.parseInt(getInput());
        leitnerSystem.downgradeCard(id, box);
    }catch (Exception e){
        System.out.println(e.getMessage());
    }
}

public String getRandomCardFromBox() {
    String response = "";
    response += leitnerSystem.getMethodName();
    List<Box> boxes = leitnerSystem.getBoxes();
    response += leitnerSystem.getRandomCard(boxes);
    return response;
}

public void handleGetRandomCardFromBox() {
    try{
        String response = getRandomCardFromBox();
        System.out.println(response);
    }catch (Exception e){
        System.out.println(e.getMessage());
    }
}

public void handleCardsInput(){
    try{
        while(true){
            controllerOptions();
            String response = validateInput(actions);
            if(response == null) {return;}
            actions.get(response).run();
        }
    }catch (Exception e){
        System.out.println(e.getMessage());
    }
}
}

```

```

public static void controllerOptions(){
    System.out.println("""
        0 - return
        1 - view cards
        2 - create card
        3 - delete card
        4 - (FlashCard) Get random card
        5 - (Leitner) Insert card in box
        6 - (Leitner) Remove card from box
        7 - (Leitner) Upgrade card from box
        8 - (Leitner) Downgrade card from box
        9 - (Leitner) View boxes
        10- (Leitner) Get random card from box
        """);
}

```

```

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 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);
    sourceBox.validateCard(cardId);

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

```

Based on the code, I notice there's a feature envy code smell in the `StudyCardsController` class where `getRandomCardFromBox()` method is heavily using `LeitnerSystem` data and methods. Let's move this method to the `LeitnerSystem` class.

Here's how to refactor the code:

1. First, move the `getRandomCardFromBox()` method to `LeitnerSystem`:

```

public class LeitnerSystem extends StudyMethod {
    // ... existing code ...

    public String getRandomCardFromBox() {
        String response = "";
        response += getMethodName();
        response += getRandomCard(getBoxes());
        return response;
    }
}

```

2. Then modify the `StudyCardsController` to use the moved method:

```

public class StudyCardsController {
    // ... existing code ...

    public void handleGetRandomCardFromBox() {
        try {
            String response = leitnerSystem.getRandomCardFromBox();
            System.out.println(response);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    // Remove the old getRandomCardFromBox() method since it's now moved
}

```

This refactoring:

1. Moves the method closer to the data it operates on
2. Reduces coupling between `StudyCardsController` and `LeitnerSystem`'s internal structure
3. Makes the code more maintainable by keeping related functionality together
4. Follows the Single Responsibility Principle better

The controller now only needs to know about calling the method and handling its response, while the implementation details live in the `LeitnerSystem` class where they belong.

Cannot resolve method 'getRandomCardFromBox' in 'StudyCardsController'

```
@Test @Order(1) @DisplayName("Get Random Card From Box Test") void getRandomCardFromBox() { String response =
studyCardsController.getRandomCardFromBox(); if(response == null){ fail(); } assertTrue(response.contains("Test LeitnerSystem"));
if(response.contains("Test Random Leitner Question 1")){ assertTrue(response.contains("Test Random Leitner Answer 1")); } else {
assertTrue(response.contains("Test Random Leitner Answer 2")); }
```

```
}
```

this is the test class and i cant change it

Step-by-Step Plan

1. Keep `getRandomCardFromBox()` in `StudyCardsController` to satisfy the test
2. Move the implementation to `LeitnerSystem`
3. Make `StudyCardsController` delegate to `LeitnerSystem`
4. Update both classes to maintain functionality

Implementation

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

    public String getRandomCardFromBox() {
        return leitnerSystem.getRandomCardFromBoxFormatted();
    }

    public void handleGetRandomCardFromBox() {
        try {
            String response = getRandomCardFromBox();
            System.out.println(response);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
    // ...existing code...
}
```

```
public class LeitnerSystem extends StudyMethod {
    // ...existing code...

    public String getRandomCardFromBoxFormatted() {
        String response = "";
        response += getMethodName();
        response += getRandomCard(getBoxes());
        return response;
    }
    // ...existing code...
}
```

This refactoring:

1. Keeps test compatibility
2. Moves business logic to `LeitnerSystem`
3. Maintains loose coupling
4. Preserves existing functionality