DATA CLASS

Card

Method to extract:

```
public String formatCard(Integer id) {
Card card = this.getCard(id);
return "[id: " + id + "] " + "Question: " +
card.getQuestion() + " Answer: " + card.getAnswer();
}
```

DataClass:

```
package org.example.studycards;

public class Card {
private String question;
private String answer;

public Card(String question, String answer) {
this.question = question;
this.answer = answer;
}

public String getQuestion() {
return question;
}

public void setQuestion(String question) {
this.question = question;
}

public String getAnswer() {
return answer;
}

public void setAnswer(String answer) {
this.answer = answer;
}

public void edit(String question, String answer) {
setQuestion(question);
setAnswer(answer);
}
}
```

fix the DataClass code smell using the ExtractMethod refactoring technique

Certainly, let's refactor the `Card` class using the Extract Method technique to improve code readability and maintainability.

Refactored Card Class:

Java

```java
package org.example.studycards;

public class Card {
    private String question;
    private String answer;

    public Card(String question, String answer) {
        this.question = question;
        this.answer = answer;
    }

    public String getQuestion() {
        return question;
    }

    public void setQuestion(String question) {
        this.question = question;
    }

    public String getAnswer() {
        return answer;
    }

    public void setAnswer(String answer) {
        this.answer = answer;
    }

    // Extracted Method: toString()
    @Override
    public String toString() {
        return "Question: " + question + ", Answer: " + answer;
    }

    public void edit(String question, String answer) {
        setQuestion(question);
        setAnswer(answer);
    }
}
```

Explanation:

1. **Extracted Method:** We extracted the logic of creating a string representation of the `Card` object into a new method called `toString()`. This method encapsulates the logic of how to format the card's information.
2. **Overridden toString()**: The `toString()` method is now overridden to provide a concise and informative string representation of the `Card` object.

Benefits of Refactoring:

- **Improved Readability:** The `formatCard()` method in the original code is now cleaner and easier to understand as it delegates the string formatting to the `Card` class's `toString()` method.
- **Increased Reusability:** The `toString()` method can be reused in other parts of the application where a string representation of the `Card` object is needed.
- **Better Maintainability:** Changes to the string formatting logic can now be made in one place (the `toString()` method) without affecting other parts of the code.

This refactoring improves the code's overall design and maintainability by following the principle of "Tell, Don't Ask." The `formatCard()` method now simply delegates the responsibility of creating the string representation to the `Card` object itself.

Now give the formatCard() new code

Certainly! Here's the updated `formatCard()` method after the refactoring:

Java

```java
public String formatCard(Integer id) {
    Card card = this.getCard(id);
    retu Mais id: " + id + "] " + card.toString();
}
```

Changes:

- The `card.getQuestion()` and `card.getAnswer()` calls are replaced with `card.toString()`. This leverages the extracted method in the `Card` class to obtain the formatted string represent Mais of the card.

This version is more concise and easier to read, as it delegates the string formatting responsibility to the `Card` class itself.