الجامعة السورية الخاصة
SYRIAN PRIVATE UNIVERSITY

Week 9 | كلية الهندسة المعلوماتية | مقرر تصميم نظم البرمجيات

Design Patterns
The Composite Pattern

د. رياض سنبل

Composite Pattern

Problem

- A Quiz Maker System that supports different types of questions (multiple choices, True-False, Fill The Gap, etc).
- Each question may be an individual question or a group of questions (user can choose how to group questions; based on the topic or the type or the difficulty, etc).
- The generated quiz can include any nested number of groups, each consisting of any types of questions based on the needs of the user.

The Solution

```java
public interface Question {
    void ask();

}
```

```java
public class QuizGroup implements Question {
    private String groupName;
    private List<Question> questions;

    public QuizGroup(String groupName) {
        this.groupName = groupName;
        this.questions = new ArrayList<>();
    }

    public void addQuestion(Question question) {
        questions.add(question);
    }

    public void ask() {
        System.out.println("[*]" + groupName);
        for (Question question : questions) {
            question.ask();
        }
        System.out.println("");
    }
```

```java
public class SingleChoiceQuestion implements Question {
    private String questionText;
    private List<String> choices;
    private int correctChoice;

    public SingleChoiceQuestion(String questionText, List<String> choices, int cor
        this.questionText = questionText;
        this.choices = choices;
        this.correctChoice = correctChoice;
    }

    public void ask() {
        System.out.println("[Choose] " + questionText);
        for (int i = 0; i < choices.size(); i++) {
            System.out.println((i + 1) + ". " + choices.get
        }
    }
```

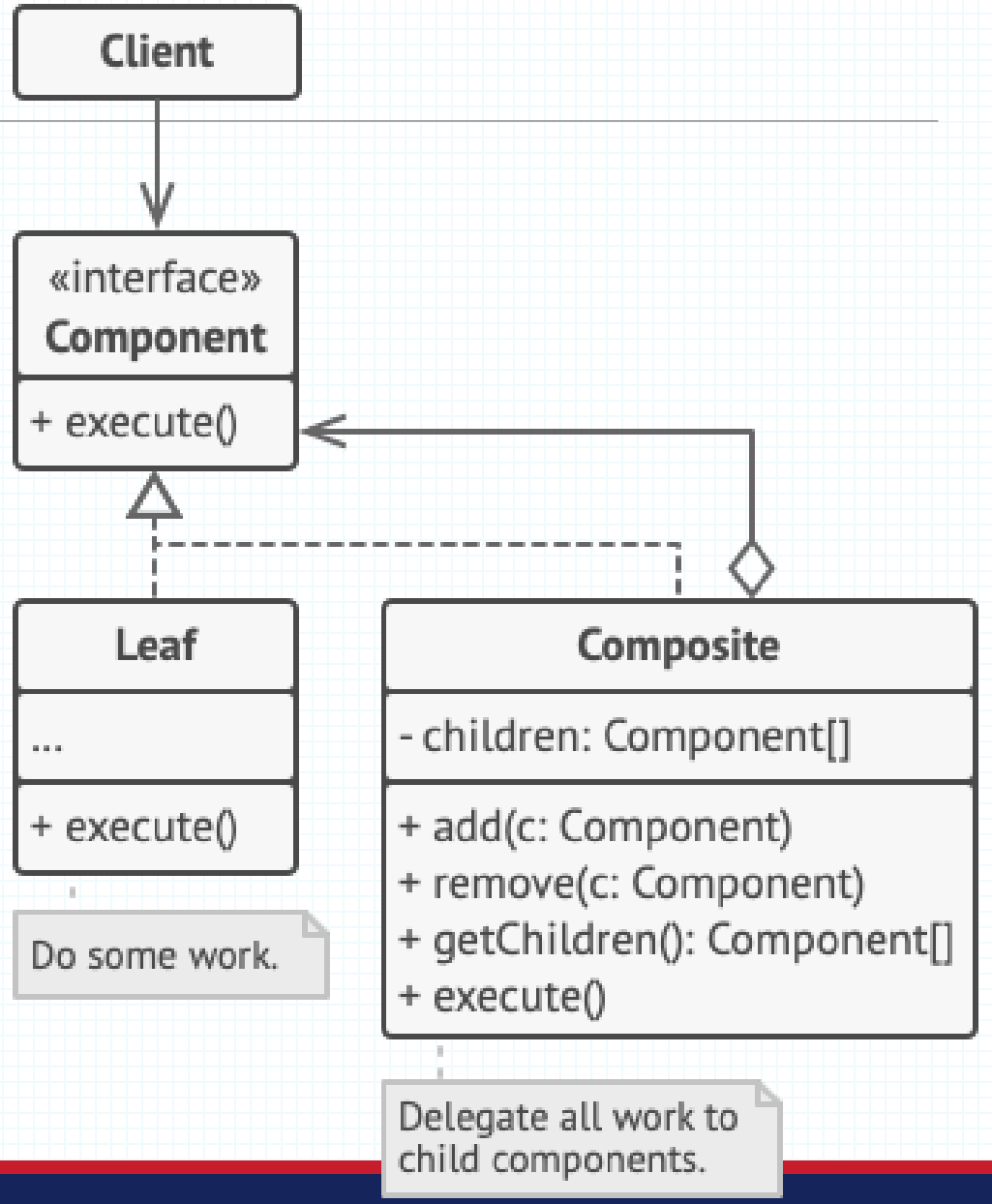
```java
public class TrueFalseQuestion implements Question {
    private String questionText;
    private boolean correctAnswer;

    public TrueFalseQuestion(String questionText, boolean correctAnswer) {
        this.questionText = questionText;
        this.correctAnswer = correctAnswer;
    }

    public void ask() {
        System.out.println("[True or False?] " + questionText);
    }
```

```java
public class QuizManager {
    public static void GenerateQuiz(Question quiz) {
        quiz.ask();
    }
}
```

```java
public class QuizExample {
    public static void main(String[] args) {

        Question singleChoiceQuestion = new SingleChoiceQuestion("What is the capital of France?",
                List.of("Paris", "Berlin", "Madrid"), 1);
        Question trueFalseQuestion = new TrueFalseQuestion("The Nile River is the longest river in
        QuizGroup quizGroup1 = new QuizGroup("Geography Quiz");
        quizGroup1.addQuestion(singleChoiceQuestion);
        quizGroup1.addQuestion(trueFalseQuestion);


        Question trueFalseQuestion1 = new TrueFalseQuestion("1+5=4?", true);
        Question trueFalseQuestion2 = new TrueFalseQuestion("15+8-2*8=25?", true);
        // Create a group of questions
        QuizGroup quizGroup2 = new QuizGroup("Math Quiz");
        quizGroup2.addQuestion(trueFalseQuestion1);
        quizGroup2.addQuestion(trueFalseQuestion2);

        QuizGroup WholeExam = new QuizGroup("Admission Exam");
        WholeExam.addQuestion(quizGroup1);
        WholeExam.addQuestion(quizGroup2);

        // Conduct the quiz
        QuizManager.GenerateQuiz(WholeExam);
    }
}
```

Composite Pattern

- Composite is a structural design pattern that lets you compose objects into tree structures and then work with these structures as if they were individual objects.



Design Patterns

- **Creational Patterns** *(abstracting the object-instantiation process)*
  - Factory Method
  - Abstract Factory
  - Singleton
  - Builder
  - Prototype
- **Structural Patterns** *(how objects/classes can be combined)*
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Facade
  - Flyweight
  - Proxy
- **Behavioral Patterns** *(communication between objects)*
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Observer
  - State
  - Strategy
  - Chain of Responsibility
  - Visitor
  - Template Method

Use Case

DYNAMIC WORKFLOW SYSTEM

Dynamic Workflow System

- *You are developing a dynamic workflow system for a content management application. Users should be able to define custom workflows for processing different types of content. Each workflow step may involve various actions such as validation, transformation, and storage.*
- *Design a solution that allows users to create and execute dynamic workflows while maintaining flexibility for future modifications.*

```python
class WorkflowStep:
    def __init__(self, name, actions):
        self.name = name
        self.actions = actions

    def execute(self, content):
        print(f"Executing step: {self.name}")
        for action in self.actions:
            action.execute(content)
```

```python
class Workflow:
    def __init__(self, steps):
        self.steps = steps

    def execute(self, content):
        for step in self.steps:
            step.execute(content)

class WorkflowEngine:
    def execute_workflow(self, workflow, content):
        workflow.execute(content)
```

```python
class Action:
    def execute(self, content):
        pass

class ValidationAction(Action):
    def execute(self, content):
        print("Validation action performed")

class TransformationAction(Action):
    def execute(self, content):
        print("Transformation action performed")

class StorageAction(Action):
    def execute(self, content):
        print("Storage action performed")
```