객체지향개발론및실습

Laboratory 1. Strategy 패턴: 묵찌바 게임

1. 목적

- 알고리즘의 군을 정의하고 캡슐화해주며 서로 언제든지 바꿀 수 있도록 해주는 전략 패턴을 실습해본다.
- 전략 패턴은 has-a 관계를 이용하며, OCP와 DIP 원칙에 충실한 행위 패턴이다.

2. 개요

- 사용자와 컴퓨터 간의 묵찌바를 하는 게임을 개발하고자 한다.
- 컴퓨터가 묵찌바를 제시하는 다양한 알고리즘을 만들어 필요하면 동적으로 바꾸고자 한다.
- 이 실습에서는 무조건 랜덤으로 묵찌바를 결정하는 전략과 지난 번 상대방이 낸 손을 이용하여 묵찌바를 결정하는 두 가지 전략을 만들어 보고, 과제로 이 두 전략 외에 새로운 전략을 만들어 본다.

3. 묵찌바 게임

- 보통 GUI는 MVC(Model-View-Controller) 개념을 이용하여 전체 응용을 구성한다. 즉, 응용 로직, 화면 처리, 사용자 입력 처리를 분리하여 구현한다. 이와 관련 자세한 이론은 추후에 학습한다. 이 실습에서는 뷰 클래스와 모델 클래스로 나누어 응용을 구현한다. 구현한 뷰 클래스는 화면 처리와 사용자 입력 처리를 모두 담당하며, 모델 객체를 뷰의 멤버 변수로 생성하여 사용하고 있다.

3.0.1 GameView

- 뷰 클래스에는 기본적으로 화면 구성을 위한 코드가 포함되어야 한다.
  `constructGamePaneView`, **`constructButtonPaneView`**가 여기에 해당한다.
- 이 응용은 항상 사용자가 묵찌바 3개의 손 중 하나를 선택함으로써 응용이 진행된다. **`userSelected`**는 사용자가 관련 **`RadioButton`**을 선택함에 따라 이미지를 변경하여 준다.
- 사용자가 묵찌바 3개의 손 중에 하나를 선택하고 "선택" 버튼을 누르면 컴퓨터가 손을 결정하고, 두 손을 비교하여 게임을 진행한다.
- 묵찌바 게임은 최초에는 가위바위보로 선공 여부를 결정해야 한다.
- 사용자가 선택 버튼을 클릭하면 컴퓨터의 다음 손을 결정한 후 현재 진행 단계에 따라 **`playGawiBawiBo`**와 **`playMookJiBa`**를 이용하여 게임의 결과를 화면에 보여주는 역할을 한다

3.0.2 GameModel

- 뷰에서 사용자로부터 받은 정보를 바탕으로 게임을 진행하기 위한 로직을 구현하고 있다.
- 사용자의 손과 컴퓨터 플레이어를 나타내는 객체를 멤버로 유지하며, 가위바위보를 해야 하는지 묵찌바를 해야 하는지 구분하기 위한 변수 **`playingMookJiBa`**와 묵찌바에서 현재 누가 공격하고 있는지를 나타내는 변수 **`isUserAttack`**을 유지하고 있다.

- 필요한 로직은 가위바위보 게임 로직과 묵찌바 로직이다. 이 클래스에서도 **playGawiBawiBo**와 **playMookJiBa** 메소드에 이 손을 비교하여 게임 결과를 판단하는 로직이 구현되어 있다.

3.0.3 GameResult

- 묵찌바 게임과 가위바위보 게임의 결과를 나타내는 열거형 타입
- 열거형: **USERWIN**, **COMPUTERWIN**, **DRAW**
- 묵찌바 게임에서 **DRAW**의 의미는 비긴 것이 아니라 계속 해야 하는 경우를 나타낸다.

3.1 HandType 열거형 클래스

- 묵찌바 게임에서 손의 모양 정보를 정의하는 열거형 타입
- 열거형: **MOOK**, **JI**, **BA**
- 멤버변수
  - **Image image**: 묵찌바 이미지를 유지한다. (JavaFX)
  - 파이썬은 묵찌바 이미지 파일 이름만 유지하고, **getImage** 메소드를 호출하면 이미지 객체를 생성하여 반환한다.
  - C++는 추가로 유지하는 것이 없고, **getImage** 메소드를 호출하면 이미지 객체를 생성하여 반환한다.
- 메소드
  - **HandType winValueOf()**
    - 사후조건: 현재 값을 가위바위보 게임에서 이길 수 있는 열거형 값을 반환한다. 즉, MOOK이면 BA, JI이면 MOOK, BA이면 JI를 반환하여야 한다.
    - 묵찌바를 하더라도 처음에는 가위바위보를 해서 공격할 측을 결정해야 한다.
    - C++는 별도 일반 함수로 이 함수를 제공한다.
- **static** 메소드
  - **HandType valueOf(int)**
    - 사후조건: 주어진 **ordinal**에 해당하는 열거형 상수를 반환한다.
    - C++는 **getHandType**이라는 동일 기능을 하는 별도 일반 함수를 제공한다.

3.2 ComputerPlayer 클래스

- 묵찌바 게임에서 컴퓨터 역할을 하는 클래스
- 멤버변수
  - **HandType hand**: 이번 턴에서 컴퓨터가 낸 손 정보
  - **PlayingStrategy strategy**: 묵찌바를 결정할 전략 객체
- 메소드
  - **public void setStrategy(PlayingStrategy)**: 묵찌바 전략을 설정, 바꿀 때 사용하는 관계 주입 메소드
  - **public HandType getHand()**: 이번 턴의 묵찌바 값을 반환함. **nextHand()** 메소드를 호출한 후에 그다음 **nextHand()** 메소드를 호출할 때까지 이 메소드를 사용하여 현재 손 값을 조회할 수 있음
  - **public HandType nextHand()**: 전략을 이용하여 컴퓨터가 이번에 선택한 손 정보를 반환함

3.3 PlayingStrategy 인터페이스

- 컴퓨터 플레이어가 다음 손 모양을 결정하기 위한 알고리즘을 구현하는 클래스들이 공통적으로 가져야 하는 interface
- 연산
  - **`HandType computeNextHand()`**: 다음 묵찌바 값을 결정하여 주는 메소드

3.4 RandomStrategy 클래스

- 무조건 랜덤으로 다음에 낼 손을 결정하는 전략을 구현하는 클래스로서, **`PlayingStrategy`** 인터페이스를 구현한다.
- 메소드
  - **`public HandType computeNextHand()`**
    - 사후조건: 랜덤 수를 생성하여 그 값에 따라 다음 낼 손을 생성하여 반환함.
      랜덤 수는 **`ThreadLocalRandom.current()`**를 이용하여 생성하고,
      반환값은 **`HandType`**의 **`valueOf(int)`** 메소드를 활용한다.
    - 파이썬과 C++는 **`QRandomGenerator`**를 이용한다.

3.5 LastHandBasedStrategy 클래스

- 지난 묵찌바의 결과가 사용자 결정에 많은 영향을 준다는 가정하에 만든 전략이다.
  - 지난번에 제시한 손과 다른 손 중 하나를 낼 확률이 높다고 가정하여 만든 전략이다.
  - 구체적으로 20% 확률로 지난 번에 낸 손과 같은 손을 낸다고 가정하고, 80% 확률로 지난 번에 낸 손과 다른 손을 낸다고 가정하며, 둘 중에 어느 것을 낼지는 확률이 같다고 가정한다.
  - 랜덤 전략은 수비와 공격을 구분하지 않는다. 이 전략에서는 수비와 공격을 구분하여 낼 손을 결정해야 한다.
- 이 전략을 구현하기 위해서는 사용자가 지난 번에 낸 손과 현재 게임 진행 상황(공격 또는 수비)을 전략 객체가 알아야 한다. 이를 위해 **`PlayingStrategy`**에 선언된 메소드를 다음과 같이 바꾸어야 한다.
- 메소드
  - **`public HandType computeNextHand(`**
    `HandType lastUserHand, boolean isUserAttack)`
    - 사후조건: 지난 사용자가 낸 손과 현재 게임 상황(공격, 수비)을 반영하여 전략에 맞는 컴퓨터 손을 계산하여 반환하여야 한다.
- 이처럼 **`PlayingStrategy`**를 수정하면 다른 전략 클래스인 **`RandomStrategy`**에도 영향을 준다. 그뿐만 아니라 **`ComputerPlayer`** 클래스, **`GameModel`**도 모두 수정해야 한다. 이것은 그렇게 바람직한 것은 아니다.
- 향후 다른 전략을 개발하고자 할 경우 필요한 정보가 매우 다양할 수 있다. 이를 대비하여 효과적으로 추상화하는 것은 쉽지 않다는 것을 잘 보여주는 예제이다. 이 경우 향후 변화를 대비하여 전략의 클라이언트 자체를 전달하는 것이 효과적인 방법일 수 있다.
- 이론 슬라이드 21, 22 참조
- **`computeNextHand`**는 그대로 유지한 상태에서 **`public void setLastUserHand(HandType)`**와 같은 메소드를 **`PlayingStrategy`**에 추가할 수 있지만 프로그램에 전체적 수정이 필요한 것은 마찬가지이다.

4. 실습내용

4.1 자바

- el.koreatech.ac.kr에서 2022-OOD-Lab01-Java.zip을 다운받아 압축파일을 **import**한다.
- **GameModel** 클래스의 **playGawiBawiBo**와 **playMookJiBa** 메소드를 완성한다.
- **RandomStrategy**를 완성한다.
- **LastHandBasedStrategy**를 완성하여 **RandomStrategy** 대신에 사용하도록 프로그램을 수정한다.

4.2 파이썬

- el.koreatech.ac.kr에서 2022-OOD-Lab01-PY.zip을 다운받아 압축파일을 해제함
- 프로그램의 실행은 python3 gameview.py를 이용하면 된다.
- **GameModel** 클래스의 **playGawiBawiBo**와 **playMookJiBa** 메소드를 완성한다.
- **RandomStrategy**를 완성한다.
- **LastHandBasedStrategy**를 완성하여 **RandomStrategy** 대신에 사용하도록 프로그램을 수정한다.

4.3 C++

- el.koreatech.ac.kr에서 2022-OOD-Lab01-CPP.zip을 다운받아 압축파일을 해제함
- Qt Creator에서 프로젝트 열기를 하거나 프로젝트 파일(.pro)을 클릭하여 Qt Creator를 실행한다.
- **GameModel** 클래스의 **playGawiBawiBo**와 **playMookJiBa** 메소드를 완성한다.
- **RandomStrategy**를 완성한다.
- **LastHandBasedStrategy**를 완성하여 **RandomStrategy** 대신에 사용하도록 프로그램을 수정한다.

5. Qt6 사용 팁

- Qt Creator를 이용하여 응용을 개발할 때 이미지나 사운드 파일 등을 프로그램에서 사용하고 싶으면 먼저 “Add New”를 이용하여 Qt–Qt Resource File을 선택한다. Resource 파일명을 입력하고 Finish를 선택하면 qrc 확장 파일이 하나 생성된다.
- 그다음 Add Prefix를 선택한 후 Prefix에 ‘/’를 입력한다.
- 그다음 Add Files를 이용해 사용할 이미지 파일이나 사운드 파일을 등록한다.
- 예를 들어 프로젝트 폴더에 “/image/a.jpg” 파일을 등록하였다면 프로그램 내에서 해당 파일을 지정할 때 “:/image/a.jpg”을 이용하면 된다.

6. 숙제

- 실습에 제시된 2개의 전략 외에 새로운 전략 알고리즘을 구현하여 제출해야 한다. 새 알고리즘은 공격할 때와 수비할 때 서로 다른 전략을 사용해야 한다. 제출할 때 새 알고리즘을 고안한 배경을 상세히 전략을 구현한 파일의 프로그램 주석에 설명한다. 전략이 다양한 정보가 필요할 때 이것을 어떻게 효과적으로 추상화하는 것이 좋은지 생각해보고, 구현할 때 해당 고민을 바탕으로 구현한다.