Group 2

Project Deliverable 2

1. Class Diagram

A computer screen shot of a diagram

Description automatically generated

**Describe Associations and Multiplicities**

Add a description of the associations:

* **Game ↔ Player**:
  + Each game requires at least one player but can support multiple players.
* **GroupOfCards ↔ Card**:
  + A GroupOfCards (like a deck) contains many cards.
* **Game ↔ GroupOfCards**:
  + The game uses a single instance of GroupOfCards to manage the deck.

|  |
| --- |
| **Group Members Task done** |
| Anmol Verma Updated War.java for round and scoring logic |
| Designed UML Class Diagram and relationships |
| Wrote the design document, explained OO principles |
| Performed testing and debugging |

**Comments on Object-Oriented Principles**

**Encapsulation**

* **Explanation**:
  + All attributes in classes like Player, War, and GroupOfCards are marked private.
  + Public getter and setter methods are provided for controlled access.
* **Example**:
  + private int score; in Player class is accessed via:

java

Copy code

public int getScore() {

return score;

}

public void setScore(int score) {

this.score = score;

}

**Cohesion**

* **Explanation**:
  + Each class performs a single, well-defined role.
  + Examples:
    - Player represents the player and tracks their attributes like name and score.
    - War manages game logic, rounds, and winner declaration.

**Coupling**

* **Explanation**:
  + Classes are loosely coupled.
  + War interacts with Player and GroupOfCards but doesn’t depend on their internal implementation.
* **Example**:
  + War relies on the Player interface to interact with players, rather than their internal details.

**Inheritance**

* **Explanation**:
  + Abstract classes like Card and Game allow reuse of common functionality.
  + Concrete subclasses like PlayingCard and War implement specific details.
* **Example**:
  + PlayingCard extends Card to add suit and value:

java

Copy code

public class PlayingCard extends Card {

private String suit;

private int value;

}

**Aggregation**

* **Explanation**:
  + Game aggregates multiple Player objects to simulate a multiplayer game.
* **Example**:
  + ArrayList<Player> in Game represents players, but Game doesn’t own them.

**Composition**

* **Explanation**:
  + GroupOfCards is composed of Card objects, meaning the cards depend on the deck.
* **Example**:
  + If GroupOfCards is destroyed, all associated Card objects are also removed.

**Flexibility/Maintainability**

* **Explanation**:
  + The design allows easy updates, such as adding new games or modifying rules.
* **Example**:
  + Adding a new card game involves creating a new class that extends Game without altering existing classes.