###### COSC 2767: OOP FW 2025

Project 1: Card Guessing Game

Software Design Report

###### Created by Daniel Allison

###### Student number 0585919

###### Nipissing University

### Introduction

#### Purpose

A console based game where the player is dealt two random playing cards and must guess which card has the higher value according to the game’s ranking rules.

#### Scope

The scope of this project includes source code which must account for core gameplay and command-line UI, a demonstration of the running results, and this design report.

Not in scope: multiplayer, networked play, GUI, persistence.

#### Audience

This design report was created with the following audience in mind: teaching assistants, graders, professors, future maintainers of the project.

### High Level Architecture

#### Component Overview

The game has been implemented with 2 high level components:

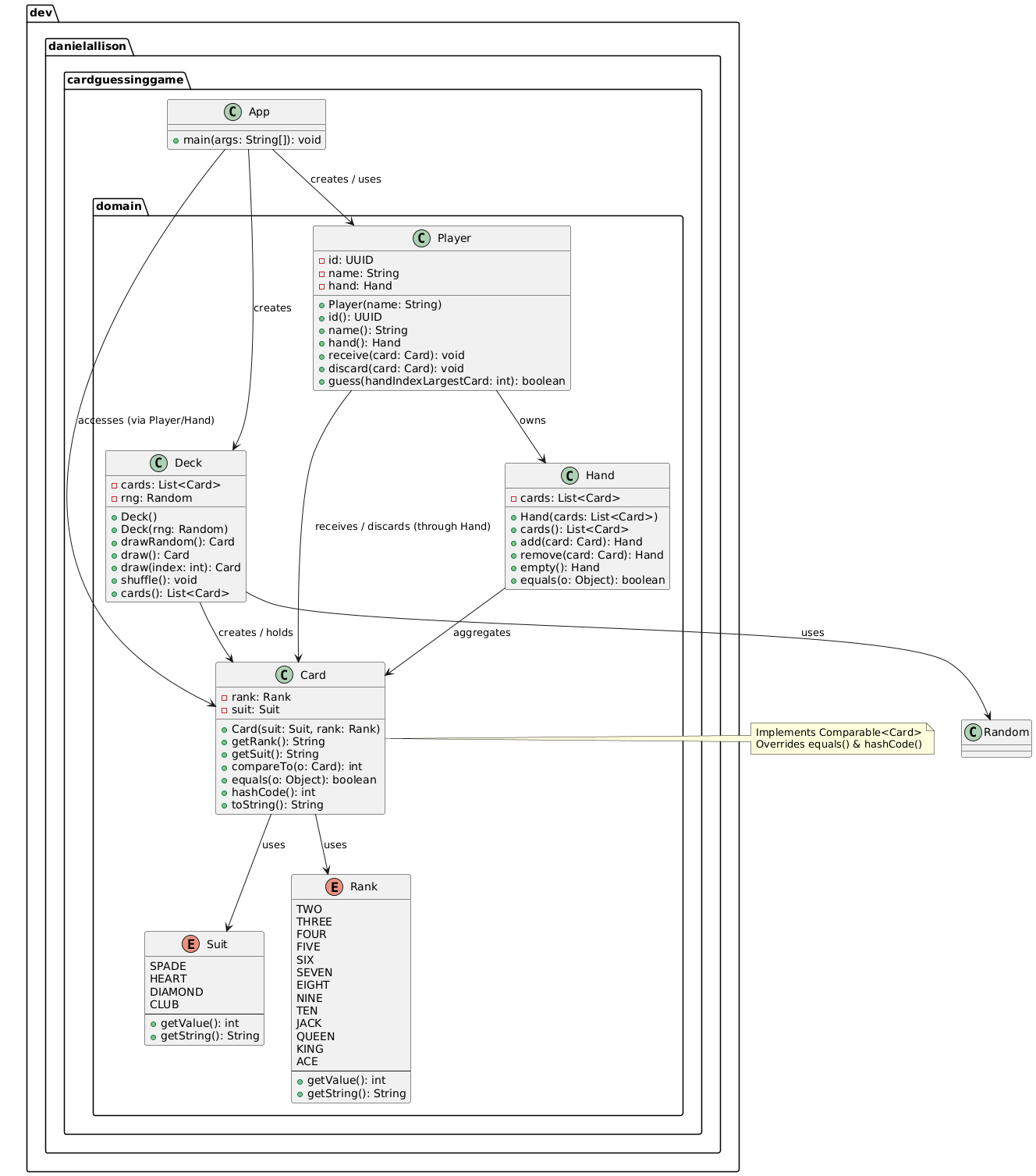
* *App* class
  + Contains console UI
  + Drives the game loop
  + Interacts with the user via STDIN/STDOUT
* *Domain* class
  + Contains the domain logic for playing card games
  + Includes *Card*, *Suit*, *Rank*, *Deck*, *Hand*, and *Player* classes

#### Design Decisions

* Keep the UI and domain separate
  + Allows unit testing of the domain without needing a console
  + Makes future UI replacements (ex. web server) easier
* Use *enums* for *Suit* and *Rank*
  + Guarantees a closed set of values
  + Provides natural ordering via *getValue()*
  + Makes *compareTo()* straightforward
* Immutable *Card* objects
  + Prevent accidental mutation of a card’s suit/rank after creation
  + Reinforces the idea that a deck should have 52 unique & unchanging cards
* *Player.guess(int)* returns a *boolean* rather than throwing exceptions
  + Creates a simple contract, the guess is either correct or incorrect
* Keep all code in one file as per expressed but undocumented preferences of professor

### Detailed Design

#### Class Diagram (UML)



#### Class Responsibilities

*Card*

* Represents a single playing card
* Provides ordering and equality
* Key public API methods:
  + *getRank(), getSuit(), compareTo(Card), equals(Object)*

*Suit*

* Defines the finite set of possible suits and exposes numeric values for comparison
* Key public API methods:
  + *getValue(), getString()*

*Rank:*

* Defines the finite set of possible ranks and exposes numeric values for comparison
* Key public API methods:
  + *getValue(), getString()*

*Deck*

* Hold a mutable collection of 52 distinct *Card* objects
* Supply random draws
* Key public API methods:
  + *drawRandom(), shuffle()*

*Hand*

* Models a player’s hand as a list of cards
* Key public API methods:
  + *add(Card), remove(Card), cards()*

*Player*

* Owns a hand
* Evaluates guesses
* Key public API methods:
  + *receive(Card), discard(Card), guess(int)*

*App*

* Orchestrates the game loop
* Handles console I/O
* Drives the domain objects
* Key public API methods:
  + *main(String[])*

#### Algorithms

* **Card Comparison** - *compareTo()* first compares *Suit.getValue()*. If suits are equal, it compares *Rank.getValue()*. Returns a signed integer consistent with *Comparable*.
* **Random Deal** – *Deck.drawRandom()* picks a random index via *rng*.*nextInt(cards.size())* and removes that card, guaranteeing no duplicates
* **Guess Evaluation** – *Player.guess(int idx)* iterates over the hand, skips the guessed card, and ensures the guessed card is strictly greater (compareResult > 0) than every other cards

### Implementation Details

Language – Java 24, Eclipse Temurin variety

Build system – Gradle wrapper (gradlew), with *run* task. DevContainer available for exact setup.

Dependencies - None

Code style - Google Java Format, applied using Spotless

Entry point – App.main

### Build & Execution Instructions

1. Prerequisites: JDK 17 or newer
2. Compile and run:
   1. *./gradlew clean build # compiles and runs tests*
   2. *./gradlew run # starts the interactive game*
3. Run tests only:
   1. *./gradlew test*
4. Package as a runnable JAR (optional)
   1. *./gradlew jar*
   2. *java -jar app/build/libs/app.jar*

### Extensibility & Potential Future Work

* Graphical UI
* Configurable Deck Size
* Scoring & Stats
* Multiplayer
* Exhaustible Deck
* Networked Multiplayer
* Use player name more often

### Object Oriented Programming Principles Utilized

**Encapsulation**

All fields in *Card, Deck, Hand, Player* are private. Access to state is provided through public getters. This keeps the internal state hidden and prevents accidental mutation from outside code.

**Abstraction**

The *Domain* classes *Card, Deck, Hand, Player* model the concepts of a player card game without exposing low level details such as how the random number generator works. The *App*class abstracts the user-interaction layer. This allows developers and code reviewers to think in terms of the concepts that make up playing cards, rather than primitive types.

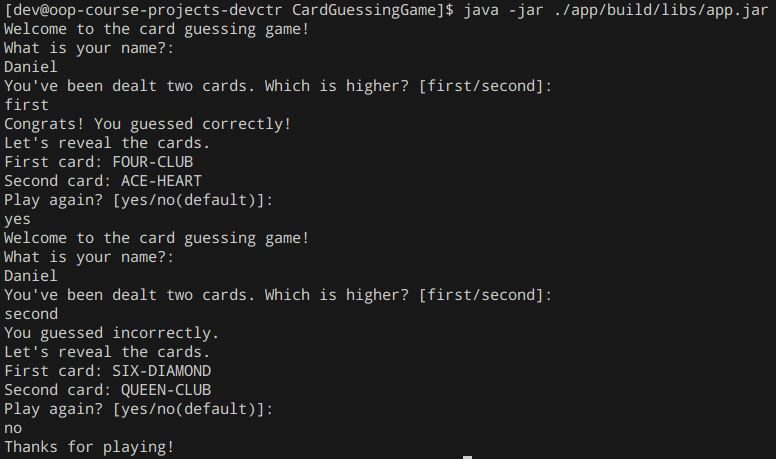
**Composition over Inheritance**

*Player* has a *Hand*. *Deck* has a *List<Card>*. *Hand* has a *List<Card>*. This allows functionality to be reused by assembling objects rather than building deep inheritance hiearchies. This keeps the class hierarchy flat and flexible.

**Polymorphism**

*Card* implements *Comparable<Card>* and overrides *compareTo*. Code that sorts or compares cards can treat any Card uniformly through the *Comparable* interface and use an familiar and idiomatic Java method. This also allows generic algorithms, like *Collections.sort*, to operate on cards without knowing their concrete type.

### Running Results



### Project Repository

This project is stored on GitHub at: <https://github.com/dan-a-gh/OOP-Course-Projects> in the CardGuessingGame directory.