Ulster University

# COM410 Programming in Practice

## A5.3 Case Study - Playing Cards

## Case Study – Playing Cards

- Playing cards provide an excellent example of an ADT that can support multiple applications
    - Suit (Hearts, Spades, Clubs, Diamonds)
    - Rank (2, 3, 4, …, Queen, King, Ace)
    - Value (2, 3, 4, …, 9, 10, 10, 10, 10, 11)
    - Rank Value (e.g. 0, 1, 2, …, 10, 11, 12)
    - Colour
    - Face up/face down
    - Collection of cards can be a deck (1 only?)
    - Collection of cards can be a hand (multiple?)
    - Cards behave differently in different situations (Snap, Poker, Bridge, etc.)
    - Standard operations – deal, shuffle, compare, etc.

# Scenario – Playing Card

- In your `DataStructures` project, implement a `Card` class to represent a single standard playing card.
  - A card is characterised by 2 values
    - Its rank (one of “2”, “3”, “4”, “5”, ”6”, “7”, “8”, “9”, “10”, “Jack”, “Queen”, “King”, “Ace”)
    - Its suit (one of “Clubs”, “Diamonds”, “Hearts”, “Spades”)
  - Include the following public methods on Card objects
    - A `Card()` constructor to generate a card with random rank and suit values
    - Accessor methods `getRank()` and `getSuit()` to return the rank and suit
    - A `toString()` method to return the card in the form (e.g.) “2 of Clubs”
    - A method `isBiggerThan()` that takes another Card object as a parameter and returns true if the card has a higher rank than the parameter card.
  - Implement a `CardTest` class to generate 2 new cards and display them biggest first

## Scenario – Playing Cards

- In your `DataStructures` project, create the class `Deck` to represent a full pack of 52 `Card` objects.
  - The deck should create one of each combination of suit and rank. You will also need to create an overloaded `Card` constructor that accepts rank and suit parameter values.
  - Provide a method `toString()` that returns a string representation of the deck, with each card printed on a separate line.
  - Provide a method `deal()` that returns a `Card` object and removes it from the deck
  - In the `CardTest` class, provide code to create a deck, print it, and deal and print 5 cards
  - Provide a method `shuffle()` that randomizes the order of the cards in the deck
  - In the `CardTest` class, demonstrate that the shuffle has had an effect by  creating a deck and printing it to the console.

## Challenge – Snap!

- Implement the class `Snap` to play the classic card game according to the following rules
  - The application should generate a shuffled deck of 52 cards and display them in shuffled order.
  - The rank of each consecutive pair of cards is compared and the message “SNAP!!!” is displayed when a match is found (i.e., for 52 cards, 51 comparisons are made).
  - After all comparisons are made, the application reports the number of snaps found.
- Extend the `Snap` class to check for **supersnaps** (where the cards are of the same colour as well as the same rank). Add new methods to the `Card` class to help you as required.
  - The message “SUPERSNAP!!!” should be displayed when a snap with similarly coloured cards is found.
  - The final output should report both the number of supersnaps and the number of snaps.