Lab: Object-Oriented Programming
Card Dealing Simulation

Duration: 2 hours
Language: C#
Level: Beginner OO

Learning Objectives

By the end of this lab, students will:

1. Design and implement multiple related classes to represent playing cards and players.
2. Use arrays to store collections of objects.
3. Create and use object relationships (e.g., a player *has* cards, a manager *manages* a deck).
4. Apply encapsulation and object interaction through methods.

Scenario

You are creating a **card dealing simulation** for a small game system.
The system should:

- Represent **individual playing cards** (name, value, whether it's been dealt).
- Manage a **deck of cards** that can be shuffled and dealt.
- Represent **players** who receive cards and can see their total value.

Your job is to design, implement, and test this program using classes, arrays, and methods.

Cards and their values are:

```
string[] names = { "Ace Spades", "2 Spades", "3 Spades", "4 Spades", "5 Spades", "6
Spades", "7 Spades", "8 Spades", "9 Spades", "10 Spades", "Jack Spades", "Queen
Spades", "King Spades", "Ace Hearts", "2 Hearts", "3 Hearts", "4 Hearts", "5 Hearts",
"6 Hearts", "7 Hearts", "8 Hearts", "9 Hearts", "10 Hearts", "Jack Hearts", "Queen
Hearts", "King Hearts", "Ace Clubs", "2 Clubs", "3 Clubs", "4 Clubs", "5 Clubs", "6
Clubs", "7 Clubs", "8 Clubs", "9 Clubs", "10 Clubs", "Jack Clubs", "Queen Clubs",
"KingClubs", "Ace Diamonds", "2 Diamonds", "3 Diamonds", "4 Diamonds", "5 Diamonds",
"6 Diamonds", "7 Diamonds", "8 Diamonds", "9 Diamonds", "10 Diamonds", "Jack
Diamonds", "Queen Diamonds", "King Diamonds" };

int[] values = { 10, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10, 10, 2, 3, 4, 5, 6, 7, 8,
9, 10, 10, 10, 10, 10, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10, 10, 2, 3, 4, 5, 6, 7,
8, 9, 10, 10, 10, 10 };
```

Step 1: Identify the Classes

Your program should include the following classes:

- **Card** – represents a single playing card (e.g. *Ace of Spades*).
- **CardManager** – creates and manages all cards in a 52-card deck.
- **Player** – represents a player who can receive and hold multiple cards.

Task:

1. For each class, decide which data it needs (fields or properties).
2. Decide what each class should *do* (methods).
3. Note which classes will interact with each other (e.g. a Player uses a Card).

Think about

- What properties does a Card need? (Think: name, value, whether it's been dealt)
- What methods should CardManager provide to interact with the deck?
- How will a Player know which cards they've been dealt?

Step 2: Design the Card Class

The Card class should represent a single card.
Each card needs to store:

- Its **name** (like *Ace of Hearts*),
- Its **value** (e.g. 10),
- A flag to track whether it's been **dealt**.

It should also have a **constructor** and a method to return the card's details as text.

Task:
Write down the fields and methods you think the Card class needs.

Step 3: Design the CardManager Class

The CardManager manages a **deck of 52 cards**.
It should:

- Create all 52 cards when the program starts.
- Keep them in an array.
- Be able to **deal** a random card that hasn't been dealt yet.

Task:

1. Plan how you'll store all 52 cards (think arrays).
2. Decide how to pick a random undealt card.
3. Write pseudocode for a dealCard() method.

Step 4: Design the Player Class

Each player:

- Has a name.
- Can receive multiple cards (stored in an array).
- Should be able to show all the cards they've received and their **total score**.

Task:

1. Decide what data each Player object needs.
2. Describe the process for adding a card to a player.
3. Plan how to calculate and display the total of all their cards.

Step 5: Create a Main Program to Test Your Classes

Your main program should:

1. Create a CardManager object (the deck).
2. Create one or two Player objects.
3. Deal several cards to each player (e.g. 2–3 each).
4. Display each player’s cards and total value.

Task:
Write pseudocode describing this process — no need for actual code yet.

Step 6: Test and Extend

Once you’ve implemented your classes:

1. Test that all 52 cards are created correctly.
2. Test that cards are dealt randomly.
3. Test that players’ totals update correctly.

Optional Challenges:

- Prevent the same card from being dealt twice.
- Allow multiple rounds of dealing.
- Add a method to “reset” the deck.
- Add a scoring rule (e.g., count Aces as 1 or 11).