CS 152 Midterm Exam (Spring 2025)

Willamette University

Name (Print): _____

Instructions:

- **Exam Format:** The exam is entirely on paper. Please write legibly. Plenty of space is provided for your answers, but don't feel that you have to fill all pages. Draw a line through any blank pages you do not want graded.
- **Allowed:** You may use your computer to access your digital notes, class slides, and the digital textbook as needed.
- **Not Allowed:** You may not use Generative AI (e.g., ChatGPT, CoPilot, etc.) or search engines.
- **Time allocation:** The time alloted for the exam is 60 minutes. It is recommended to allocate approximately 15 minutes per question and leaving the remaining time to look over your answers.
- Good luck!

Honor Statement: By signing below, I confirm that I completed this exam without the use of unauthorized resources, including generative AI.

Signature: _____

Rubric (100 points)

Criteria	Exceeds	Meets	Partial
Q1: OO Design (30 pts)			
Defined classes, attributes			
Data structure choices			
OO principles (encapsulation, etc.)			
Complexity analysis			
Q2: Pythonic Data Structure (30 pts)			
Part A: Conceptual design			
Part B: Pythonic elements			
Part C: Interface methods			
Part D: Implementation			
Q3: Data Structure Conversion (40 pts)			
`array_to_bag` function			
`bag_to_array` function			

This page intentionally left blank.

Question 1: OO Design & Data Structure (30 points)

Scenario: WUUp? – Willamette's Student Engagement Platform

WUUp? helps Willamette students manage:

- Events and meetups for campus activities.
- Bearcat Feed, a social media-like space for student posts.
- Study Groups, where students can join or create study sessions.

Your Task: Design an object-oriented (OO) system for WUUp?. Define key objects, their attributes, and methods, and be sure to justify your choices by discussing the OO principles you considered. In your response:

- Identify key classes.
- Identify state data (instance variables), including any data structures used.
- Explain behaviors of the objects (methods) and their time complexities.
- Describe how the objects interact (inheritance, polymorphism, association, composition) as applicable.

This page intentionally left blank. You may use this page to continue your answer from the previous page if needed.

Question 2: Pythonic Data Structure (30 points)

Scenario: Bistro Seating Availability System

A new digital sign outside The Bistro will display **real-time seating availability**. The Bistro staff need a data structure that will allow them to do the following.

- Manage available and occupied seats.
- Update the sign when a seat is taken or freed.
- Expand seating capacity when needed.

Your Task:

- Part A. Discuss a conceptual design of the **data structure** including:
  - What internal data structures will be used?
  - How will the sign know how many seats are available?
  - How will staff update seat availability?
- Part B. Explain how you might incorporate **Pythonic design elements**. Consider, but do not limit yourself to, the following:
  - `bracket operators`
  - `len operator`
  - `iter operator`
- Part C. Declare the method signatures in the interface `IBistroSeatingManager`. Be sure to use type hints and return types in the signatures.
- Part D. Write out a sample implementation `BistroSeatingManager` based on your interface.

Part A: Data Structure Conceptual Design

Part B: Pythonic Design Elements

Part C: Interface Definition

```
class IBistroSeatingManager(ABC):
    """Interface for managing seating availability at The Bistro. """
```

Part D: Sample Implementation

```
class BistroSeatingManager(IBistroSeatingManager):
    """Implementation for managing seating availability at The Bistro. """
```

Question 3: Data Structure Conversion (40 points)

Scenario: Converting Between Custom Data Structures

The class `Conversions` is responsible for converting between different data structures in our system.

Your Task: Complete the following class definition by implementing the two methods:

- `array_to_bag(array: IArray) -> IBag`
  Converts an `IArray` to an `IBag`, ignoring duplicate elements.
- `bag_to_array(bag: IBag) -> IArray`
  Converts an `IArray` to an `IBag`, ignoring duplicate elements.

Note: You must only use methods declared in `IBag` and `IArray`. You cannot interact with the internals of these data structures.

```
class Conversions:
    """Handles conversion between IArray and IBag."""

    @staticmethod
    def array_to_bag(array: IArray) -> IBag:
        """Converts an array to a bag, ignoring duplicates."""
```

```
    @staticmethod
    def bag_to_array(bag: IBag) -> IArray:
        """Converts a bag back into an array."""
```