System Requirements Specification Index

For

Ecosystem Simulation System

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

TABLE OF CONTENTS

Ecosystem Simulation System
System Requirements Specification

1 Project Abstract

EcoLearn Foundation, a non-profit dedicated to environmental education, requires an interactive Ecosystem Simulation program to promote ecological awareness among students and the general public. The simulation will model interactions between different organisms in a virtual environment, demonstrating key ecological concepts such as food chains and environmental adaptation. By visualizing these relationships, EcoLearn aims to foster a deeper understanding of ecosystem dynamics and the importance of biodiversity conservation.

2 Business Requirements:

Screen Name	Console input screen
Problem Statement	1. System needs to simulate different types of organisms (plants, herbivores, carnivores) 2. System must support basic ecosystem operations such as organism creation and interaction 3. Console implementation must demonstrate object-oriented programming concepts like Classes and objects, Inheritance and polymorphism

3 Constraints

3.1 Class Requirements

1. `Organism` Class (Base Class):
    - Attributes: id, species_name, energy, is_alive

- Class Variables: organism_count
- Methods: display_info(), consume_energy(), interact()
- Example: Cannot be instantiated directly

2. `Plant` Class (inherits from `Organism`):
   - Additional attributes: growth_rate
   - Override methods: display_info(), interact()
   - New methods: photosynthesize()
   - Example: `Plant("P001", "Oak Tree", 100, True, 0.2)`
3. `Animal` Class (inherits from `Organism`):
   - Additional attributes: speed, diet_type, food_eaten
   - Override methods: display_info()
   - New methods: hunt()
   - Example: Cannot be instantiated directly
4. `Herbivore` Class (inherits from `Animal`):
   - Additional attributes: plant_preference
   - Override methods: hunt(), interact(), display_info()
   - Example: `Herbivore("H001", "Rabbit", 70, True, 3, "herbivore", "grass")`
5. `Carnivore` Class (inherits from `Animal`):
   - Additional attributes: hunting_efficiency
   - Override methods: hunt(), interact(), display_info()
   - Example: `Carnivore("C001", "Wolf", 80, True, 5, "carnivore", 0.7)`
6. `Environment` Class:
   - Attributes: name, weather_condition, organisms, day_count, next_id
   - Methods: add_organism(), remove_organism(), get_organisms_by_type(), find_organism_by_id(), get_population_count(), simulate_day(), get_next_id()
   - Example: `Environment("Forest Ecosystem", "sunny")`

3.2 Operation Constraints

1. Basic Interactions:
   - Plants generate energy through photosynthesis based on weather
     - Implement in `photosynthesize(weather_condition)` method

        - Weather impacts: sunny (1.0), cloudy (0.6), rainy (0.3)
    - Herbivores hunt plants with preference for specific species
        - Implement in `hunt(prey_organisms)` method
        - Should filter available plants by preference
    - Carnivores hunt herbivores with success based on hunting_efficiency
        - Implement in `hunt(prey_organisms)` method
        - Success chance = hunting_efficiency * (predator_speed / prey_speed)
    - Animals interact with prey through `interact(other_organism)` method
    - All organisms consume energy daily using `consume_energy(amount)` method
    - Organisms die when energy reaches zero (handled in energy setter)

2. Input Validation:
    - All organism IDs must be unique
    - Energy must be positive (10-100)
    - Growth rates must be between 0.1 and 0.5
    - Speed must be positive integer
    - Hunting efficiency must be between 0.3 and 0.7

3. Exception Handling:
    - Must handle OrganismNotFoundException when organism not found
    - Must handle InvalidInputException for invalid user inputs

4. Object-Oriented Requirements:
    - Must use protected attributes (single underscore prefix)
    - Must use private attributes for Environment (double underscore prefix)
    - Must implement inheritance hierarchy as specified
    - Must use polymorphism with method overriding

3.3 Output Constraints

1. Display Format:
    - Organism info must show: ID, species name, energy, status, type-specific attributes
    - Environment info must show: name, weather, day count, organism counts by type

2. Output Format:

- Must show in this order:
    - Show "== ECOSYSTEM SIMULATION =="
    - Show "Environment: {name}"
    - Show "Weather: {weather_condition}"
    - Show "Total Organisms: {count}"
    - Show "Population Breakdown:" followed by counts by type
    - Show "Simulation Day: {day_count}"

3.4 Property Implementation Constraints

1. Each class must implement proper property getters/setters:
    - Organism class: id, species_name, energy (with setter), is_alive
    - Plant class: growth_rate
    - Animal class: speed, diet_type, food_eaten
    - Herbivore class: plant_preference
    - Carnivore class: hunting_efficiency
    - Environment class: name, weather_condition (with setter), organisms, day_count

3.5 Constructor/Destructor Constraints

1. All classes must implement appropriate constructors that initialize attributes
2. The Organism class must increment organism_count in its constructor
3. The Organism class must decrement organism_count in its destructor
4. Environment class constructor must initialize private collections

4. Template code Structure:

1. Organism Classes:

- `Organism` (base class)
- `Plant` (derived from Organism)
- `Animal` (derived from Organism)
- `Herbivore` (derived from Animal)
- `Carnivore` (derived from Animal)

2. Environment Class:

- `Environment`

3. Exception Classes:

- `OrganismNotFoundException`
- `InvalidInputException`

4. Program Control:

- `main()` - main program function

5. EXECUTION STEPS TO FOLLOW:

1. Run the program
2. View the main menu
3. Select operations:
   - Option 1: Add Organism to Environment (numeric selection)
   - Option 2: Simulate One Day
   - Option 3: Simulate Multiple Days
   - Option 4: Display All Organisms
   - Option 0: Exit
4. Perform operations on the ecosystem simulation
5. View results after each operation
6. Exit program when finished