System Requirements Specification Index

For

Wildlife Rehabilitation Management System

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

TABLE OF CONTENTS

1 PROJECT ABSTRACT

The Wildlife Rescue Alliance (WRA) requires a specialized management system to track and care for injured wildlife brought to their rehabilitation centre. The system will focus on proper resource management through effective use of constructors and destructors in an object-oriented paradigm. This will ensure that animals are properly registered upon intake, resources are allocated appropriately during their stay, and all resources are properly released and documented when animals are discharged. The system will demonstrate how proper initialization and cleanup operations are critical for managing limited rehabilitation resources and ensuring quality care for all wildlife patients.

2 BUSINESS REQUIREMENTS:

Screen Name	Console input screen
Problem Statement	1. System needs to track wildlife patients through their rehabilitation lifecycle 2. Classes must implement proper constructors for initialization of resources 3. Classes must implement proper destructors for cleanup of resources 4. System must support Animal intake and initial assessment, Treatment plan tracking, Resource allocation and release, Discharge and release tracking

3 CONSTRAINTS

3.1 CLASS REQUIREMENTS

1. `Animal` Class:

- Constructor must initialize: animal_id, species, condition, intake_date
- Must validate animal_id is non-empty string
- Destructor must handle tracking when animal is removed from system
- Class should maintain a count of all animals in the system

2. `Enclosure` Class:
   - Constructor must initialize: enclosure_id, enclosure_type, capacity
   - Must validate capacity is positive integer (greater than zero)
   - Destructor must handle cleanup when enclosure is removed
   - Class should maintain a count of all enclosures in the system
3. `RehabilitationCenter` Class:
   - Constructor must initialize: name, location, animals dict, enclosures dict
   - Must validate name is non-empty string
   - Destructor must handle cleanup of center resources
   - Should track system start time for operational statistics

3.2 Constructor/Destructor Constraints

1. Constructor Requirements:
   - Must validate all input parameters before initialization
   - Animal constructor must reject empty or non-string IDs
   - Enclosure constructor must reject non-integer or non-positive capacity
   - RehabilitationCenter constructor must reject empty or non-string names
   - Must initialize all required attributes with appropriate defaults
2. Destructor Requirements:
   - Animal destructor must check if animal was properly discharged
   - Enclosure destructor must check if animals still assigned
   - RehabilitationCenter destructor must clean up all collections
   - Classes must maintain proper counts through constructors/destructors

3.3 Property Implementation Constraints

1. Each class must implement proper property getters/setters:
   - Animal class: animal_id, species, condition, status, assigned_enclosure

- Enclosure class: enclosure_id, enclosure_type, capacity, animals, available_capacity
- RehabilitationCenter class: name, location, animal_count, enclosure_count

2. Property getters must provide read-only access to private attributes
3. Only assigned_enclosure should have a setter to maintain encapsulation

3.4 Error Handling Constraints

1. All constructors must validate input parameters and raise appropriate exceptions
2. User input in main() must be validated with try-except blocks
3. Edge cases must be handled gracefully (e.g., when assigned_enclosure is None)

4. Template code Structure:

1. Base Classes:

- `Animal` - base class for tracked wildlife
- `Enclosure` - class for managing animal housing
- `Treatment` - class for tracking medical care
- `Staff` - class for managing care providers
- `RehabilitationCenter` - main class coordinating the system

2. Helper Classes:

- `Logger` - utility class for uniform logging
- `ResourceManager` - utility class for managing system resources

3. Exception Classes:

- `ValidationError` - for parameter validation issues
- `ResourceError` - for resource allocation issues
- `InitializationError` - for constructor problems
- `CleanupError` - for destructor problems

4. Main Program:

- `main()` - main program function demonstrating the system

5. Execution Steps to follow:

1. Run the program
2. Observe the proper initialization of objects (constructor calls)
3. Observe the allocation of resources

4. Monitor the object lifecycle through operations
5. Observe the proper cleanup of objects (destructor calls)
6. View resource allocation and release
7. Verify proper error handling
8. Examine the final system state