Veterinary Project

Manages and provides information about individual animals, along with setting animal kinds and the adding of pre-existing conditions to avoid duplication.

«I_Animal» Animal

- name: String
- age: int
- weight: double
- gender: String
- preexistingConditions: String[]
- animalKind: AnimalKind

- Animal(name: String, age: int, weight: double, gender: String)
- Animal(name: String, age: int, weight: double, gender: String, preexistingConditions: String[])
- addPreexistingCondition(conditions: String[]): void
- isRepeated(array: String[], str: String): boolean
- getPreexistingCondition(): String[]
- setAnimalKind(kind: String): boolean
- setWeight(weight: double): void
- getName(): String
- getAge(): int
- getWeight(): double
- getAnimalKind(): AnimalKind
- getGender(): String

Represents a medical case with an owner, animal, and treatment

«I_Case» Case

- owner: Person
- animal: Animal
- treatment: Treatment

- Case()
- Case(owner: Person, animal: Animal, treatment: Treatment)
- getOwner(): Person
- getAnimal(): Animal
- getTreatment(): Treatment

1 animal 1

owner identify 1

verify

1 1

«I_Node» Node

- currentCase: Case
- next: Node
- previous: Node

- Node(currentCase: Case)
- getPrev(): Node
- getNext(): Node
- setPrev(prev: Node): void
- setNext(next: Node): void
- getCase(): Case
- displayUrgentCase(): String

Represents a node in a doubly linked list structure, storing a Case

0..* 1

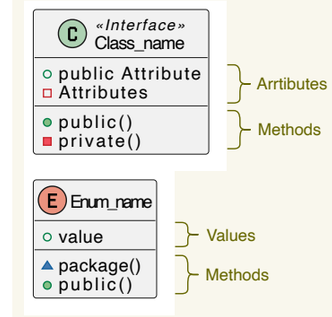
«I_List» List

- front: Node
- tail: Node
- size: int

- List()
- append(node: Node): boolean
- getHead(): Node
- getTail(): Node
- insert(index: int, node: Node): boolean
- insertHead(node: Node): void
- remove(index: int): boolean
- removeNode(node: Node): void
- swap(first_index: int, second_index: int): boolean
- getNodeAtIndex(nodeIndex: int)
- sort(): void
- findIndex(ownerName: String): int

Manages a doubly linked list with methods for appending, inserting, removing nodes, swapping node positions, sorting based on emergency cases, and finding the index of a node by owner name. It includes functions to get the head and tail nodes and retrieve a node at a specific index. The insertHead method adds a node to the beginning, considering whether the list is empty or not.

Notation Reference



Prepared by
Name :
Matriculation no :

01.02.2024

Enumerates only 5 different kinds of animal in Veterinary

classify

AnimalKind

- dog: String
- cat: String
- bird: String
- reptile: String
- rodent: String

- AnimalKind(value: String)
- toString(): String

Manages and provides a general person informations for ensuring a unique identifier for each person. This class will be inherited to Doctor and Assistant classes.

«I_Person» Person

- name: String
- age: int
- address: String

- Person()
- Person(name: String, age: int, address: String)
- getName(): String
- getAge(): int
- getAddress(): String

1

Treatment

- vaccination: String
- injury: String
- diagnostics: String
- emergency: String

- Treatment(value: String)
- toString(): String

Enumerates only 4 different treatment types in Veterinary

identify

authorize

identify

«I_Assistant» Assistant

//extends Person

- Assistant(name: String, age: int, address: String)

Manages and provides general information about an assistant, leveraging its inheritance from the Person class.

0..*

operate

queue 1

1

«I_Doctor» Doctor

//extends Person

- authorizedFor: AnimalKind[]
- operationTime: int
- startingTime: int
- finishedTime: int
- handleCase: Node

- Doctor(name: String, age: int, address: String)
- Doctor(name: String, age: int, address: String, authorizedFor: AnimalKind[])
- getAuthorizedFor(): AnimalKind[]
- addAuthorizedAnimal(animalKind: String): boolean
- isValidAnimalKind(animalKind: String): boolean
- getName(): String
- getAge(): int
- getAddress(): String
- toString(): String

Manages and provides a general information about a doctor by leveraging its inheritance from the Person class, along with specific authorizations for treating animal kinds. It includes attributes related to ongoing treatments, such as parameters for operation time and case details.

0..* operate 1

«I_Veterinary» Veterinary

- doctors: Doctor[]
- assistants: Assistant[]
- queue: List

- Veterinary(doctors: Doctor[], assistants: Assistant[])
- Veterinary()
- getDoctors(): Doctor[]
- getAssistants(): Assistant[]
- addCase(newCase: Case): boolean
- haveVeterinaryAuthorize(newCase: Case): boolean
- getCaseList(): List
- removeCase(patientName: String): boolean
- printMostUrgentCases(k: int): void
- calculateTreatmentTime(treatmentType: String): int
- execute(): ArrayList<String>
- isDoctorAuthorized(doctors: Doctor, node: Node): boolean
- concludeCase(doctor: Doctor): String
- registerDoctorCase(time: int, currentCase: Node, doctor: Doctor): void
- findNextCase(doctor: Doctor): Node

The class manages a veterinary system with doctors, assistants, and a patient queue. It provides access to the lists of doctors, assistants, and the patient queue. The class includes methods to add and remove cases, print urgent cases, and execute treatments. The execute method simulates the treatment process, considering factors such as doctor availability, authority, and treatment times. The calculation of treatment time depends on the numbers of doctors and assistants. Internal methods handle case authorization, doctor authorization, finding the next case for a doctor, and concluding each case.