Class Diagram

Last edited by **Priya Ashok Sardhara** 6 hours ago

Class Diagram

Full Class diagram

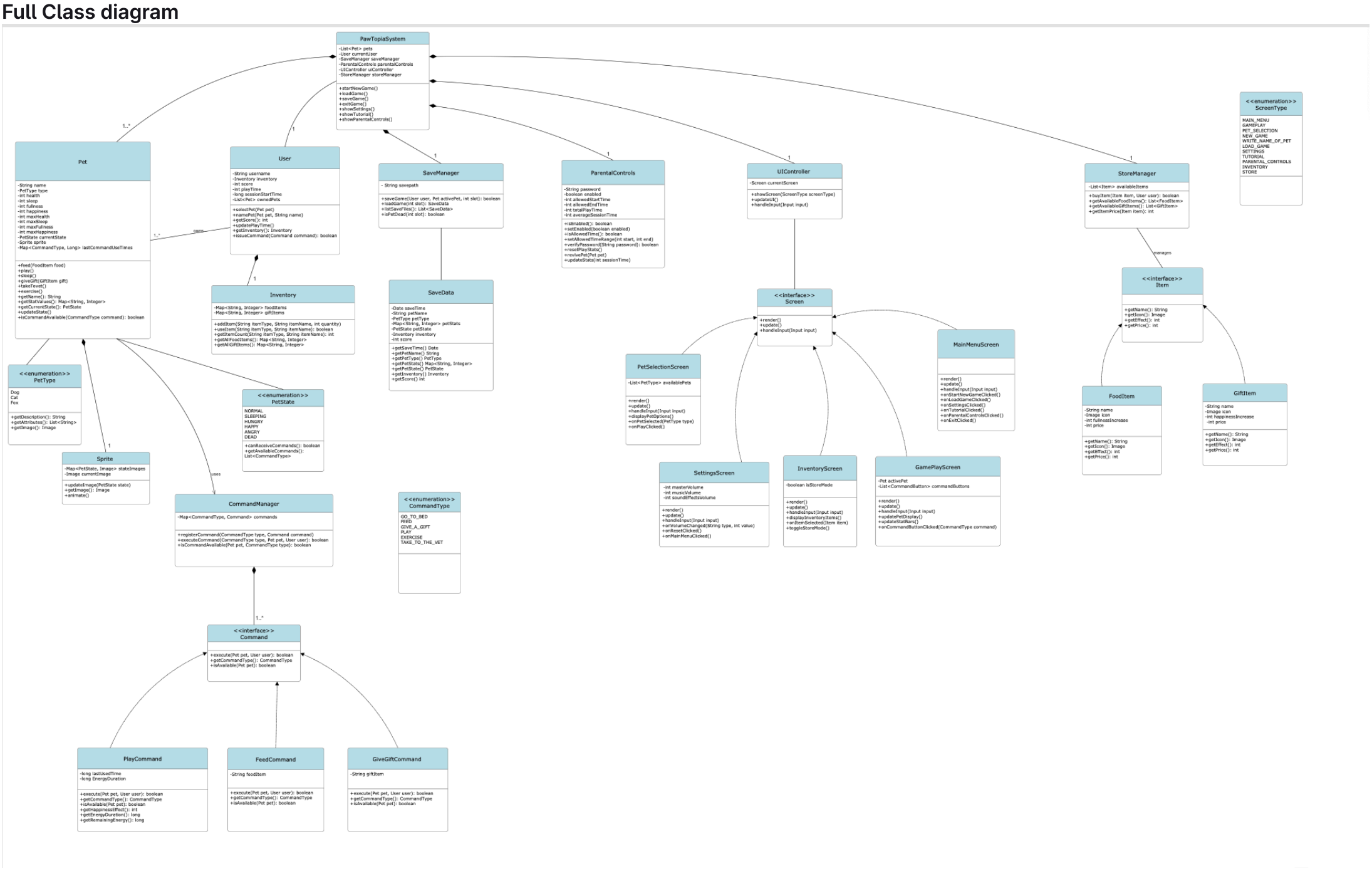


Image 1

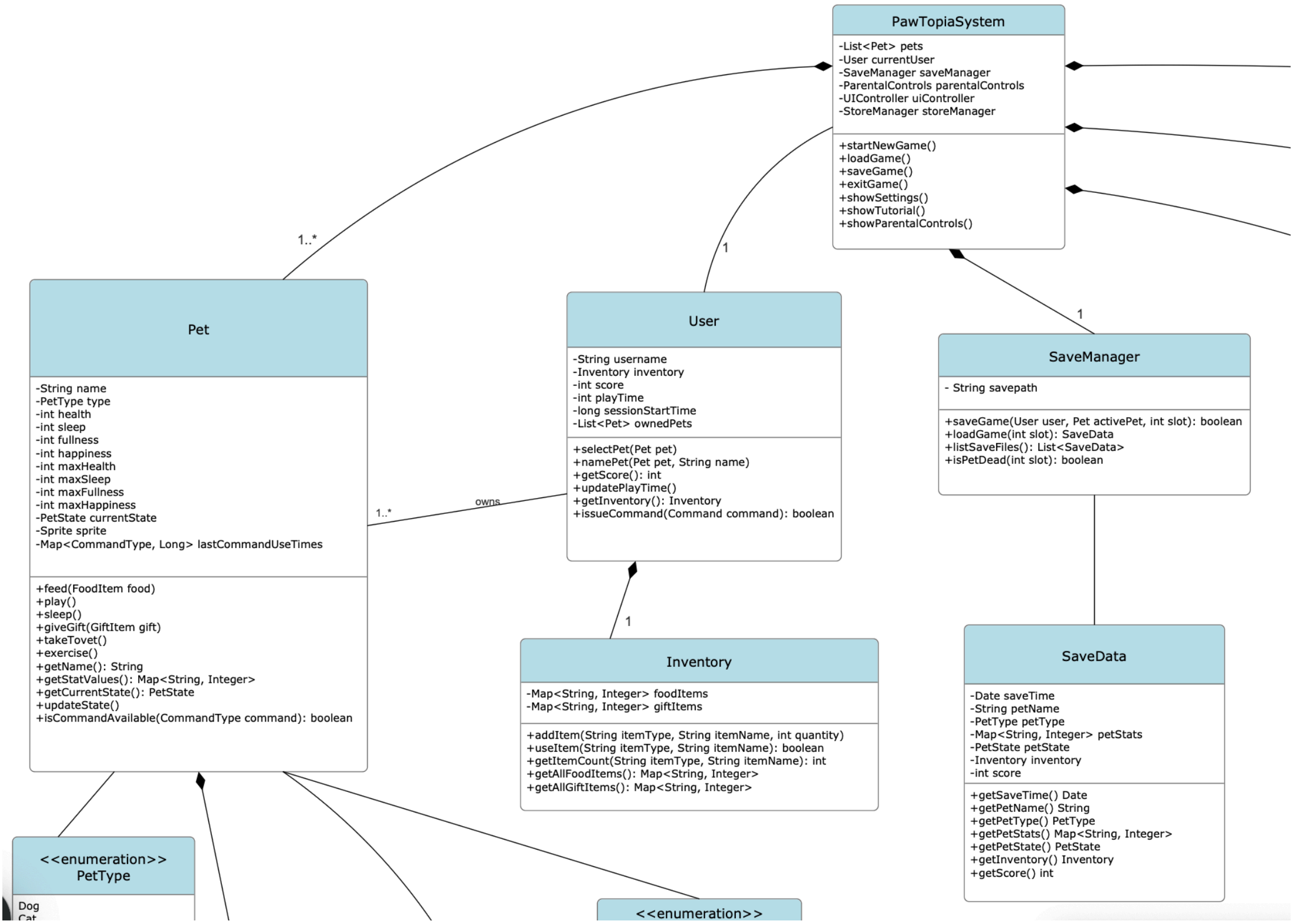


Image 2

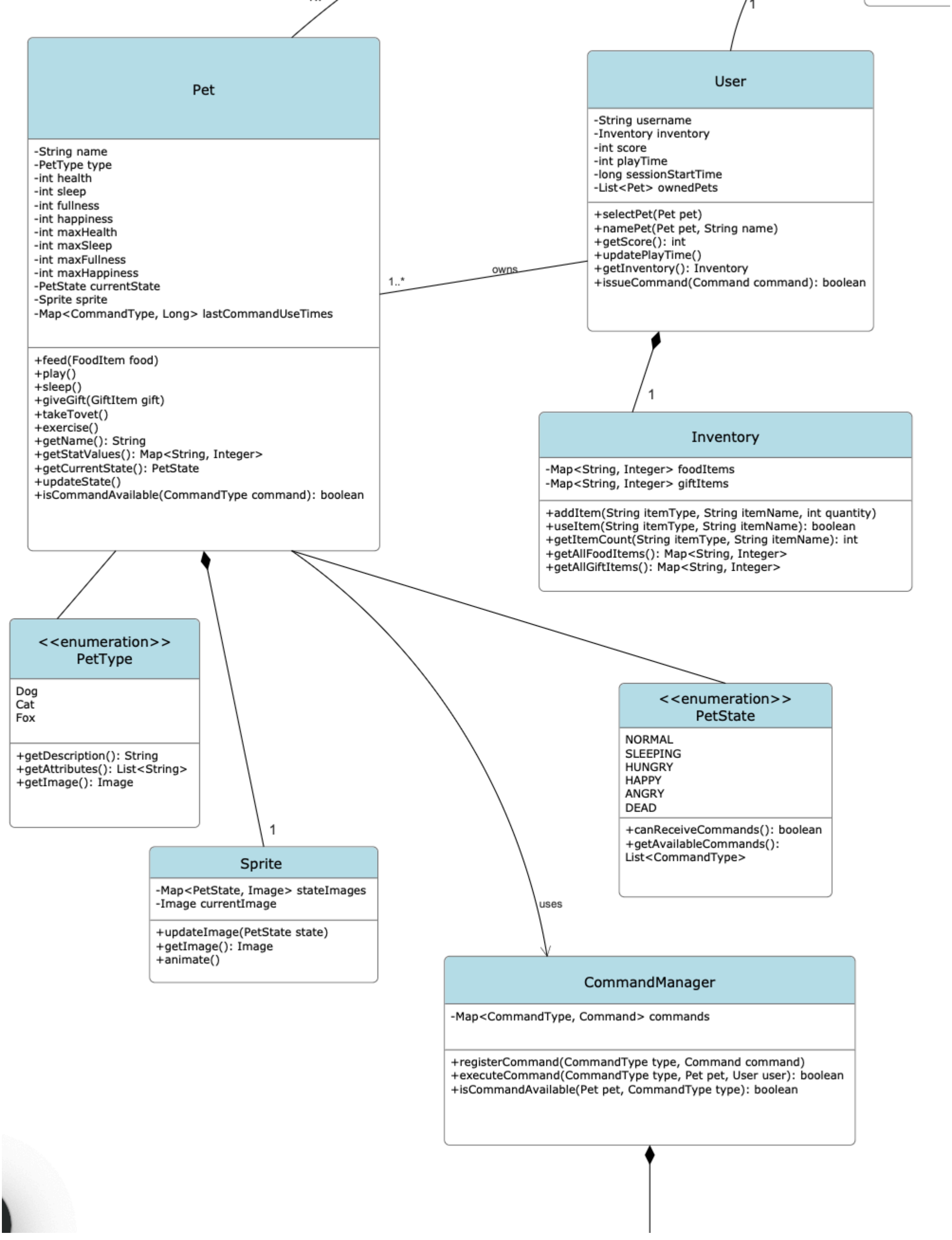

Pet
-String name
-PetType type
-int health
-int sleep
-int fullness
-int happiness
-int maxHealth
-int maxSleep
-int maxFullness
-int maxHappiness
-PetState currentState
-Sprite sprite
-Map<CommandType, Long> lastCommandUseTimes
+feed(FoodItem food)
+play()
+sleep()
+giveGift(GiftItem gift)
+takeTovet()
+exercise()
+getName(): String
+getStatValues(): Map<String, Integer>
+getCurrentState(): PetState
+updateState()
+isCommandAvailable(CommandType command): boolean
User
-String username
-Inventory inventory
-int score
-int playTime
-long sessionStartTime
-List<Pet> ownedPets
+selectPet(Pet pet)
+namePet(Pet pet, String name)
+getScore(): int
+updatePlayTime()
+getInventory(): Inventory
+issueCommand(Command command): boolean
owns
1..*
1
Inventory
-Map<String, Integer> foodItems
-Map<String, Integer> giftItems
+addItem(String itemType, String itemName, int quantity)
+useItem(String itemType, String itemName): boolean
+getItemCount(String itemType, String itemName): int
+getAllFoodItems(): Map<String, Integer>
+getAllGiftItems(): Map<String, Integer>
<<enumeration>>
PetType
Dog
Cat
Fox
+getDescription(): String
+getAttributes(): List<String>
+getImage(): Image
1
Sprite
-Map<PetState, Image> stateImages
-Image currentImage
+updateImage(PetState state)
+getImage(): Image
+animate()
<<enumeration>>
PetState
NORMAL
SLEEPING
HUNGRY
HAPPY
ANGRY
DEAD
+canReceiveCommands(): boolean
+getAvailableCommands():
List<CommandType>
uses
CommandManager
-Map<CommandType, Command> commands
+registerCommand(CommandType type, Command command)
+executeCommand(CommandType type, Pet pet, User user): boolean
+isCommandAvailable(Pet pet, CommandType type): boolean

Image 3

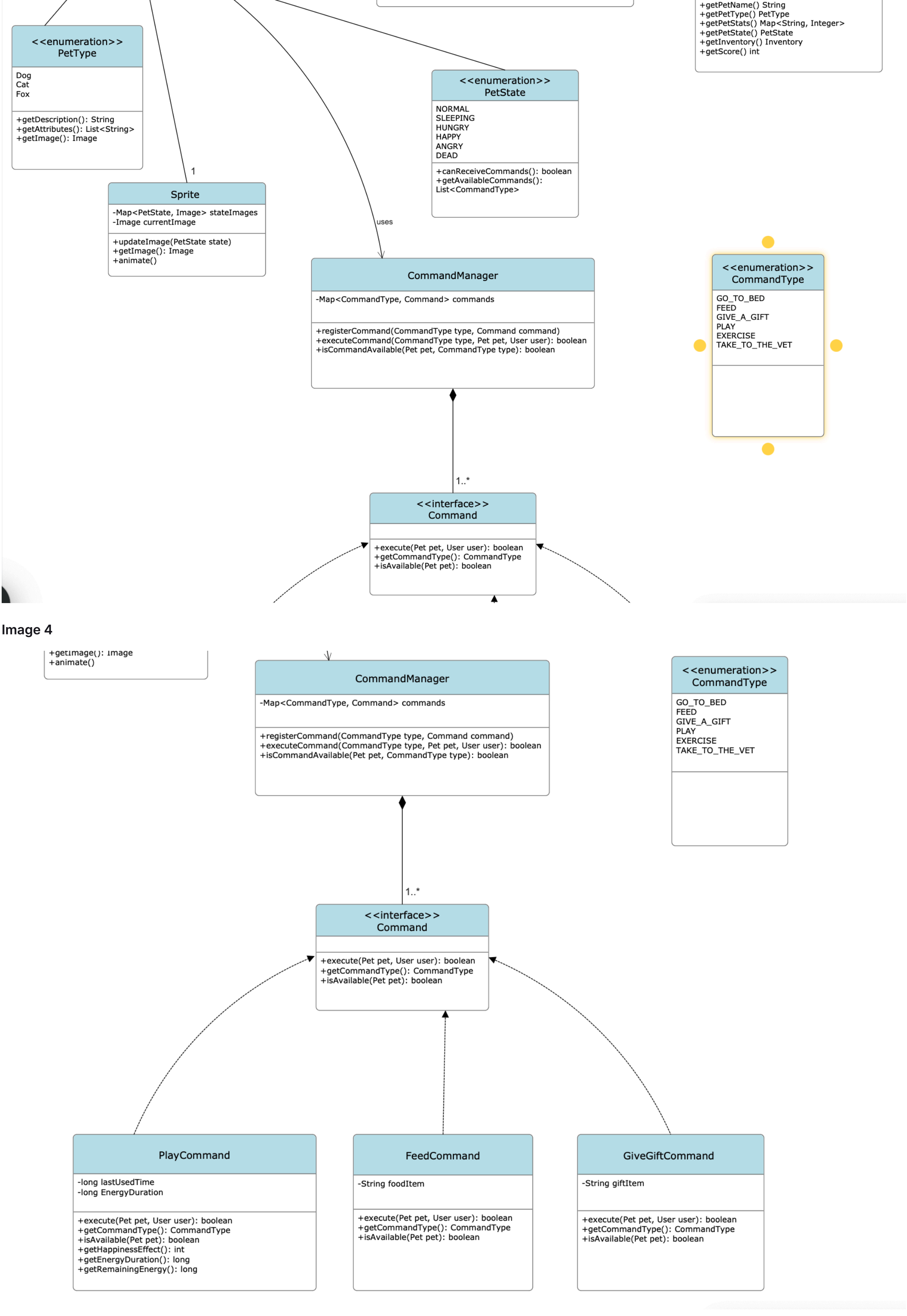

<<enumeration>>
PetType
Dog
Cat
Fox
+getDescription(): String
+getAttributes(): List<String>
+getImage(): Image
1
Sprite
-Map<PetState, Image> stateImages
-Image currentImage
+updateImage(PetState state)
+getImage(): Image
+animate()
<<enumeration>>
PetState
NORMAL
SLEEPING
HUNGRY
HAPPY
ANGRY
DEAD
+canReceiveCommands(): boolean
+getAvailableCommands():
List<CommandType>
+getPetName() String
+getPetType() PetType
+getPetStats() Map<String, Integer>
+getPetState() PetState
+getInventory() Inventory
+getScore() int
uses
CommandManager
-Map<CommandType, Command> commands
+registerCommand(CommandType type, Command command)
+executeCommand(CommandType type, Pet pet, User user): boolean
+isCommandAvailable(Pet pet, CommandType type): boolean
<<enumeration>>
CommandType
GO_TO_BED
FEED
GIVE_A_GIFT
PLAY
EXERCISE
TAKE_TO_THE_VET
1..*
<<interface>>
Command
+execute(Pet pet, User user): boolean
+getCommandType(): CommandType
+isAvailable(Pet pet): boolean

Image 4

+getImage(): Image
+animate()
CommandManager
-Map<CommandType, Command> commands
+registerCommand(CommandType type, Command command)
+executeCommand(CommandType type, Pet pet, User user): boolean
+isCommandAvailable(Pet pet, CommandType type): boolean
<<enumeration>>
CommandType
GO_TO_BED
FEED
GIVE_A_GIFT
PLAY
EXERCISE
TAKE_TO_THE_VET
1..*
<<interface>>
Command
+execute(Pet pet, User user): boolean
+getCommandType(): CommandType
+isAvailable(Pet pet): boolean
PlayCommand
-long lastUsedTime
-long EnergyDuration
+execute(Pet pet, User user): boolean
+getCommandType(): CommandType
+isAvailable(Pet pet): boolean
+getHappinessEffect(): int
+getEnergyDuration(): long
+getRemainingEnergy(): long
FeedCommand
-String foodItem
+execute(Pet pet, User user): boolean
+getCommandType(): CommandType
+isAvailable(Pet pet): boolean
GiveGiftCommand
-String giftItem
+execute(Pet pet, User user): boolean
+getCommandType(): CommandType
+isAvailable(Pet pet): boolean

Image 5

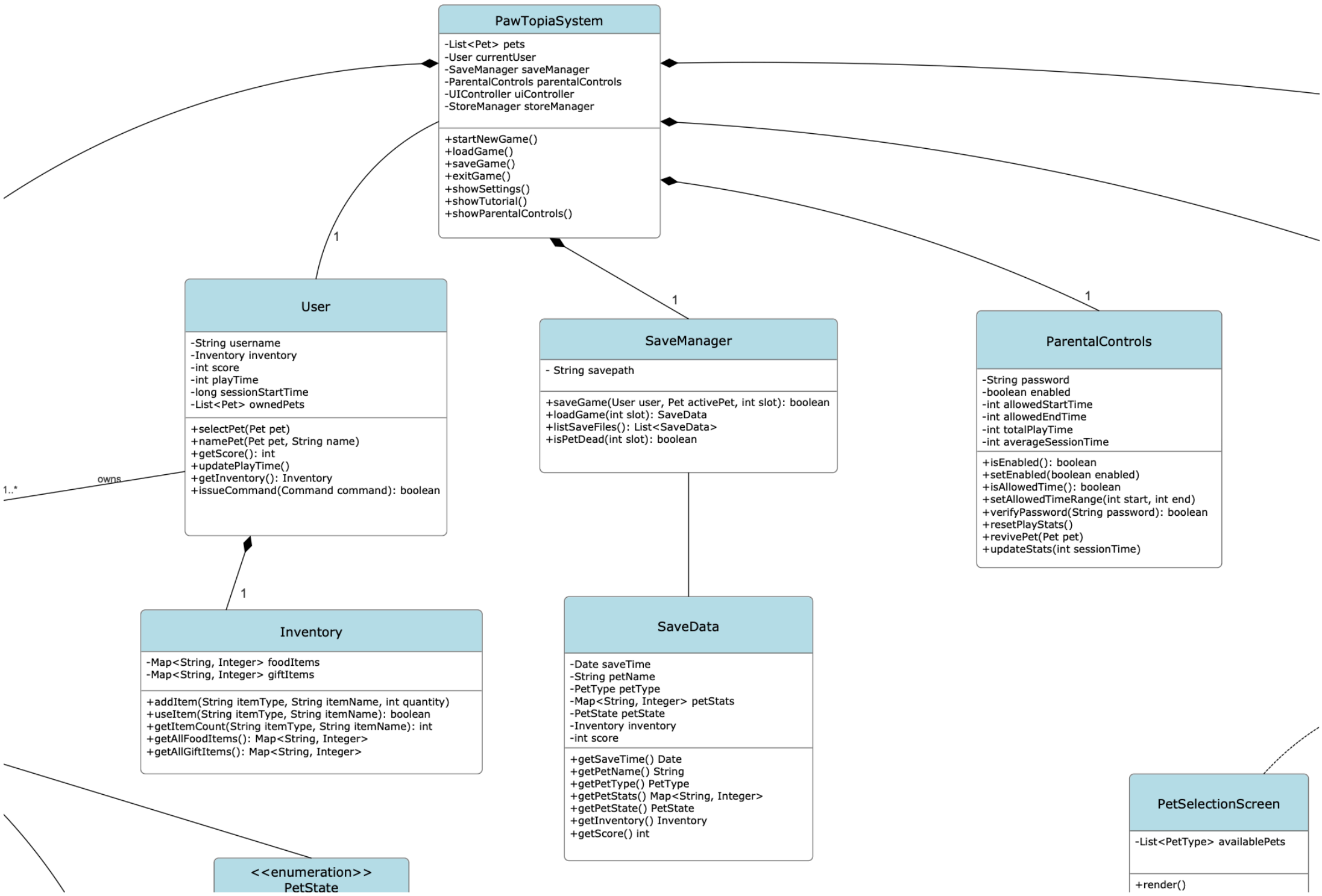


Image 6

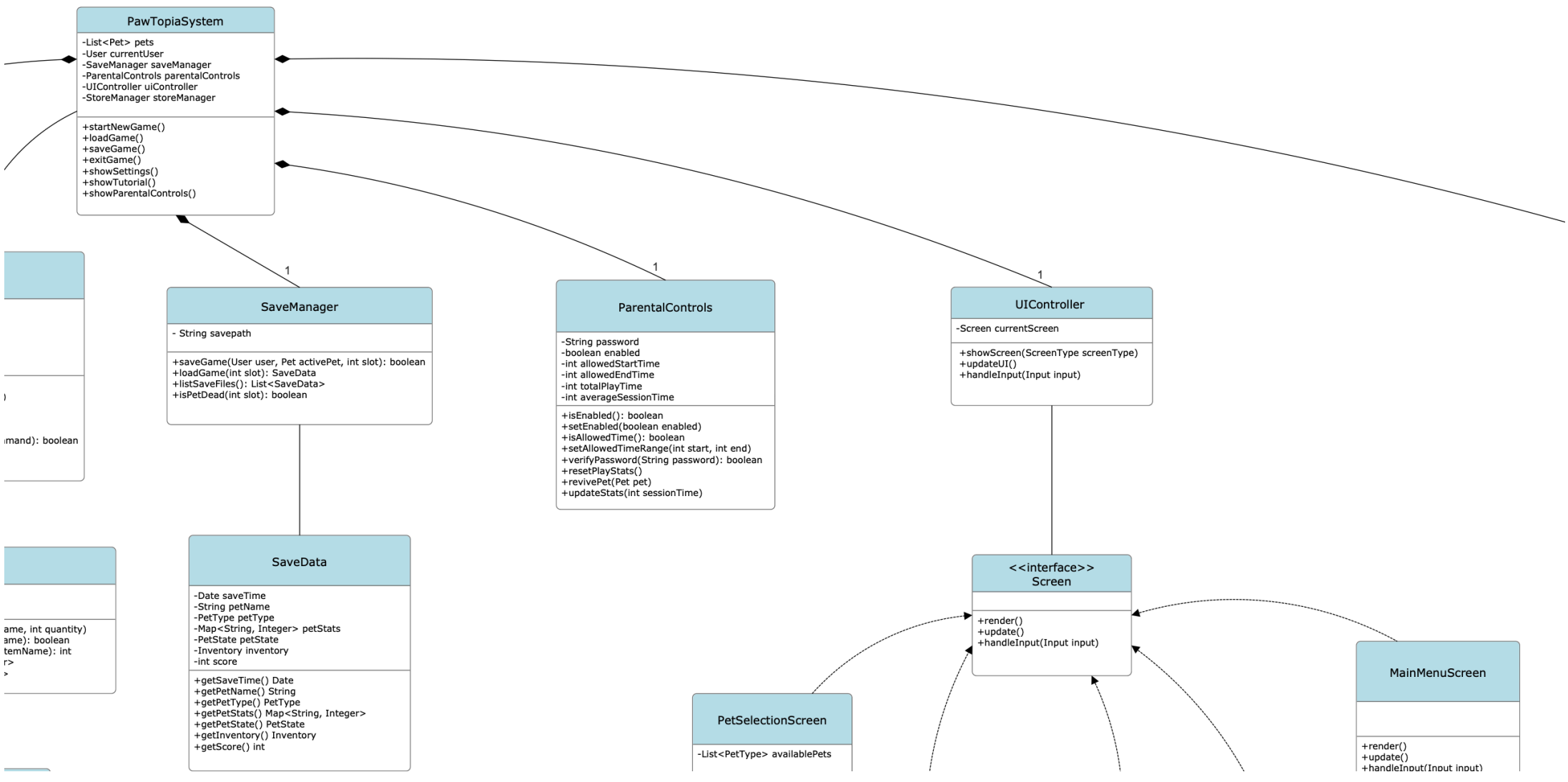


Image 7

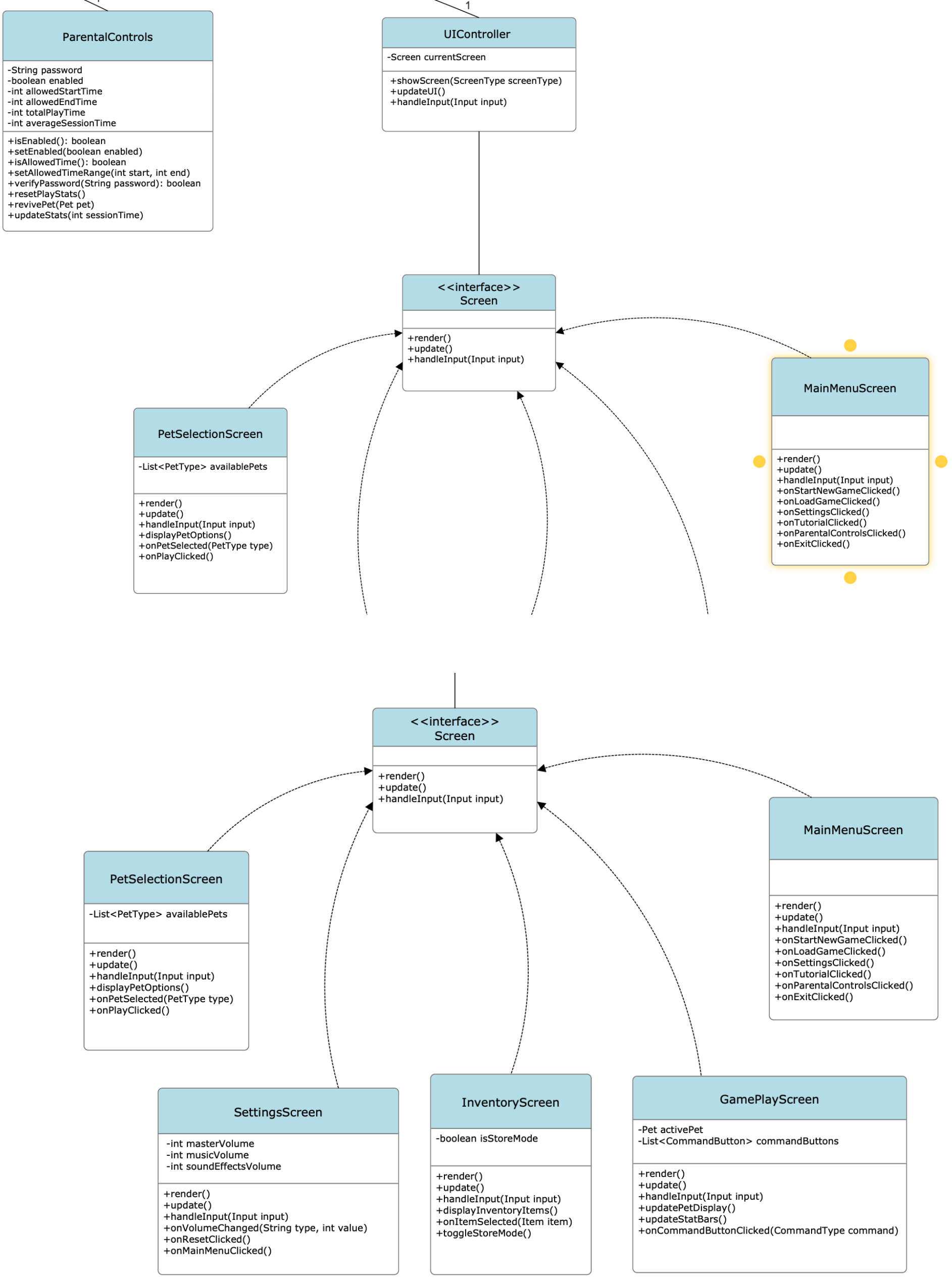


Image 8

Image 9

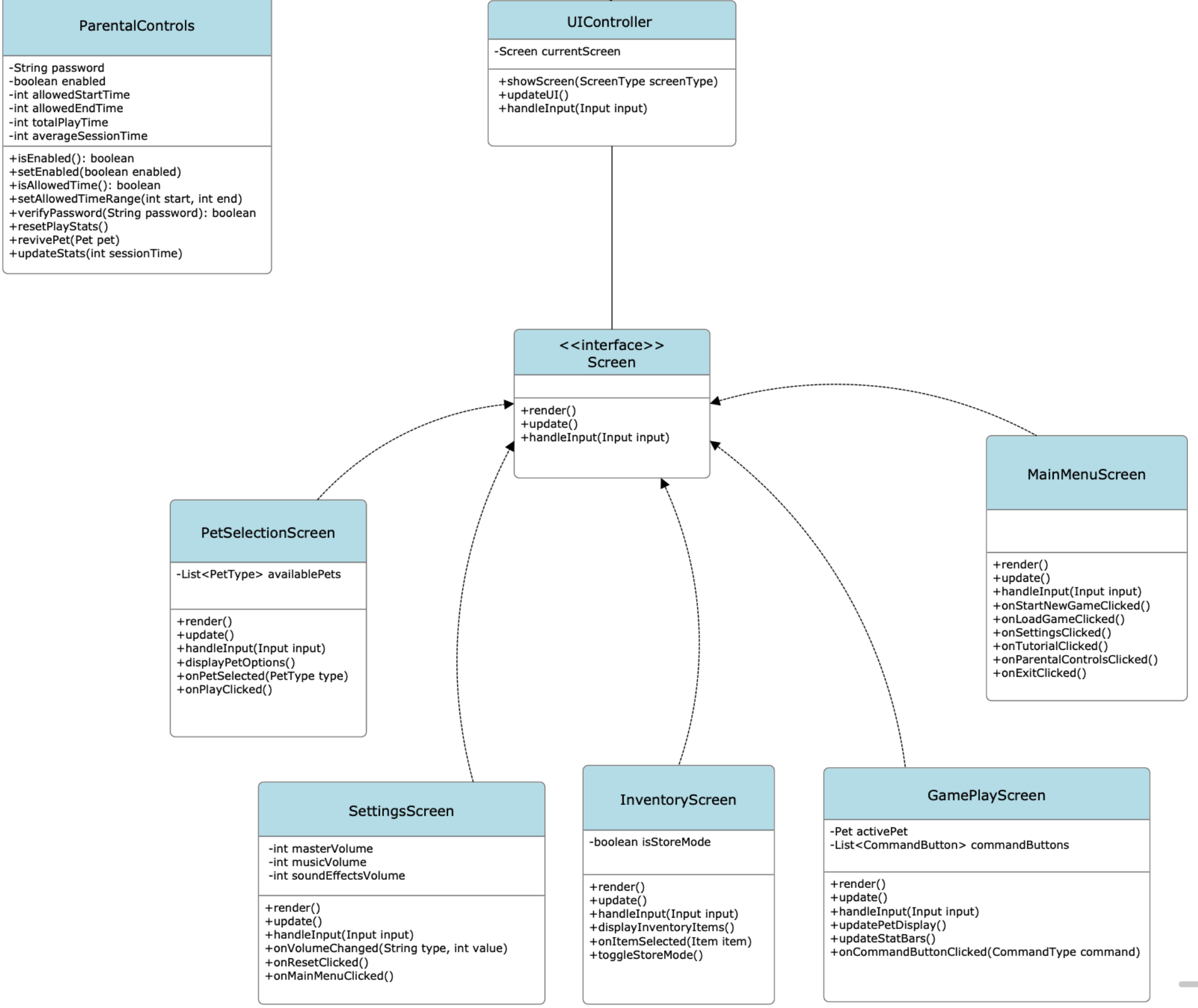

ParentalControls
-String password
-boolean enabled
-int allowedStartTime
-int allowedEndTime
-int totalPlayTime
-int averageSessionTime
+isEnabled(): boolean
+setEnabled(boolean enabled)
+isAllowedTime(): boolean
+setAllowedTimeRange(int start, int end)
+verifyPassword(String password): boolean
+resetPlayStats()
+revivePet(Pet pet)
+updateStats(int sessionTime)
UIController
-Screen currentScreen
+showScreen(ScreenType screenType)
+updateUI()
+handleInput(Input input)
<<interface>>
Screen
+render()
+update()
+handleInput(Input input)
PetSelectionScreen
-List<PetType> availablePets
+render()
+update()
+handleInput(Input input)
+displayPetOptions()
+onPetSelected(PetType type)
+onPlayClicked()
MainMenuScreen
+render()
+update()
+handleInput(Input input)
+onStartNewGameClicked()
+onLoadGameClicked()
+onSettingsClicked()
+onTutorialClicked()
+onParentalControlsClicked()
+onExitClicked()
SettingsScreen
-int masterVolume
-int musicVolume
-int soundEffectsVolume
+render()
+update()
+handleInput(Input input)
+onVolumeChanged(String type, int value)
+onResetClicked()
+onMainMenuClicked()
InventoryScreen
-boolean isStoreMode
+render()
+update()
+handleInput(Input input)
+displayInventoryItems()
+onItemSelected(Item item)
+toggleStoreMode()
GamePlayScreen
-Pet activePet
-List<CommandButton> commandButtons
+render()
+update()
+handleInput(Input input)
+updatePetDisplay()
+updateStatBars()
+onCommandButtonClicked(CommandType command)

Image 10

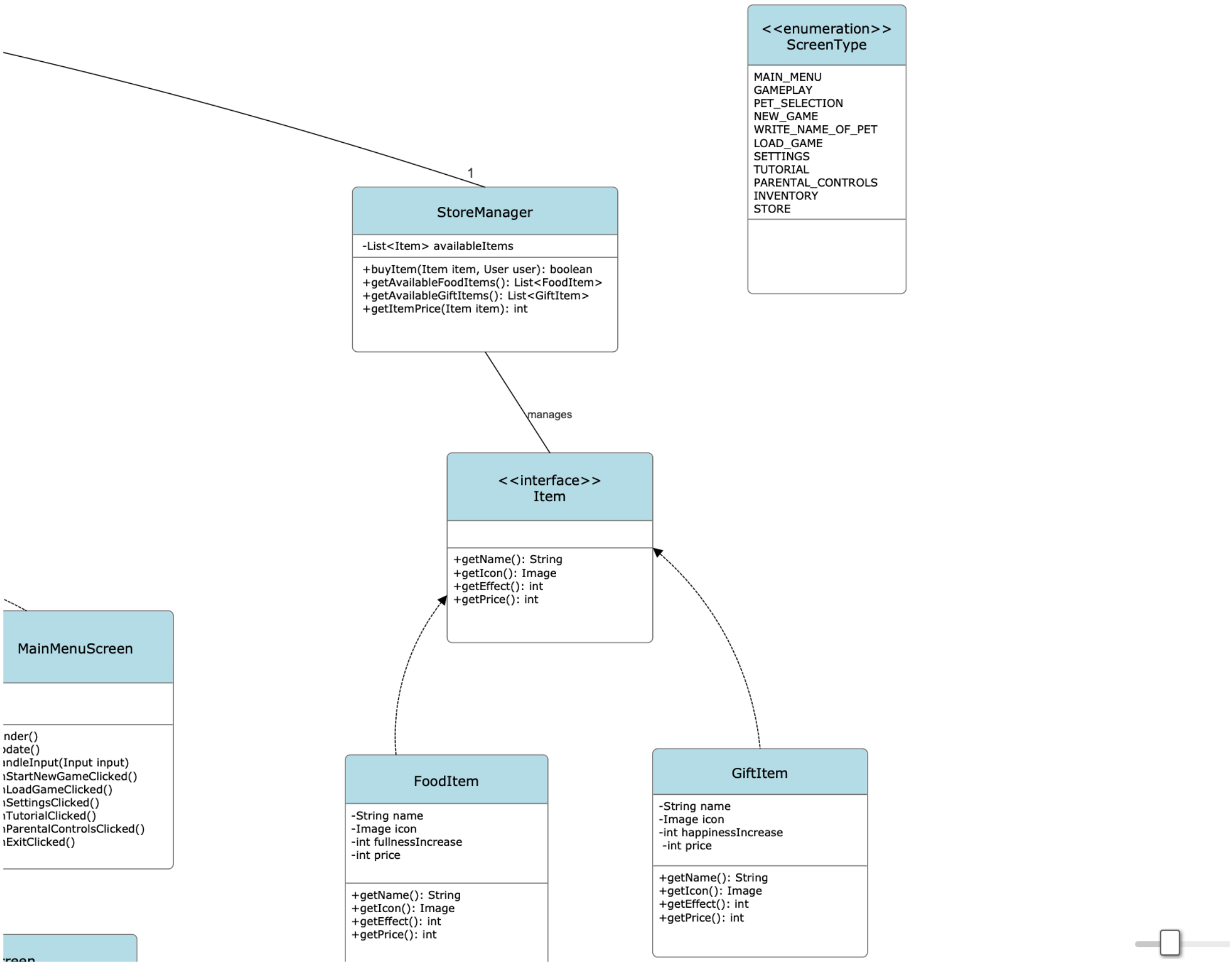


Image 11

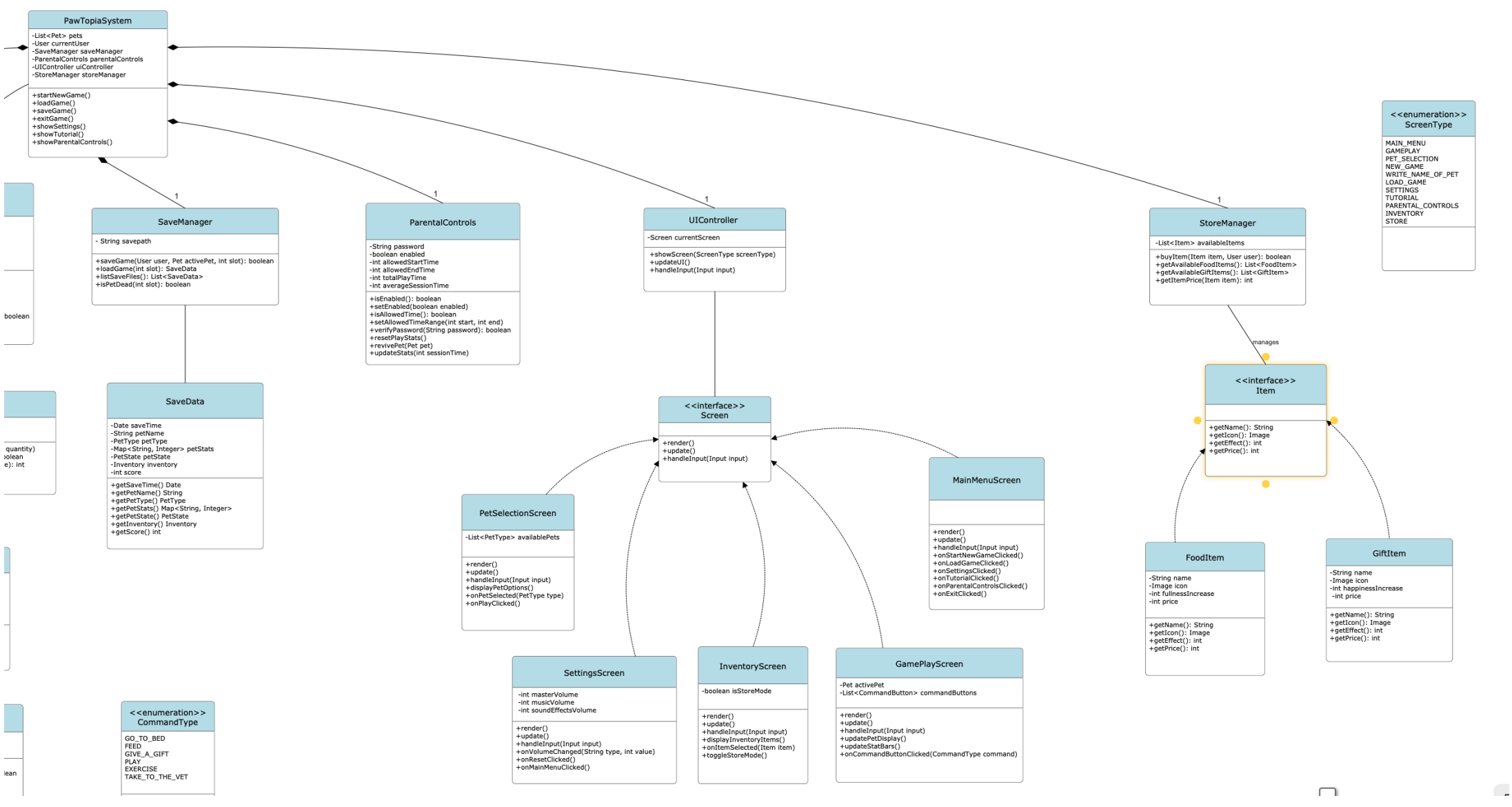


Image 12

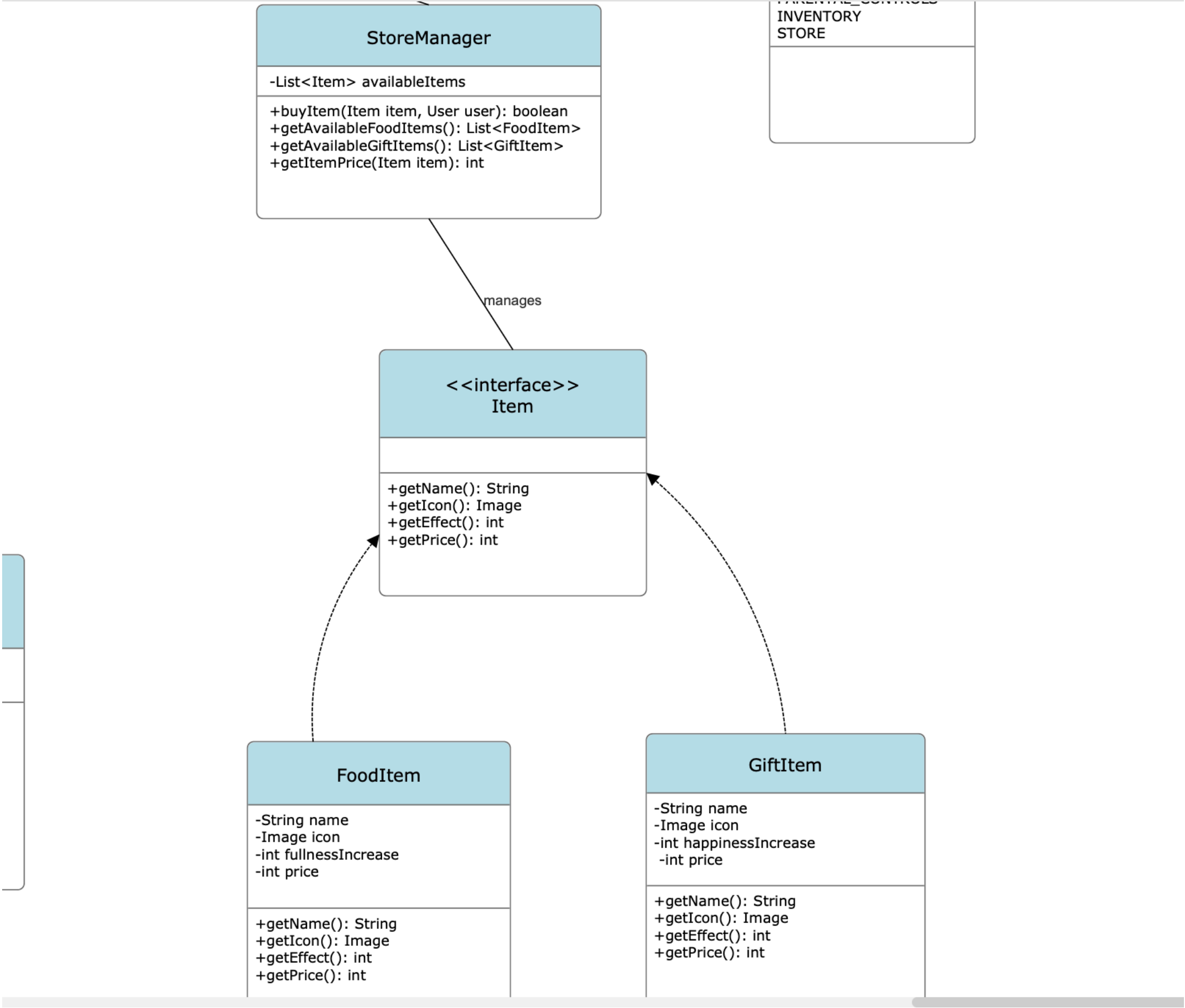
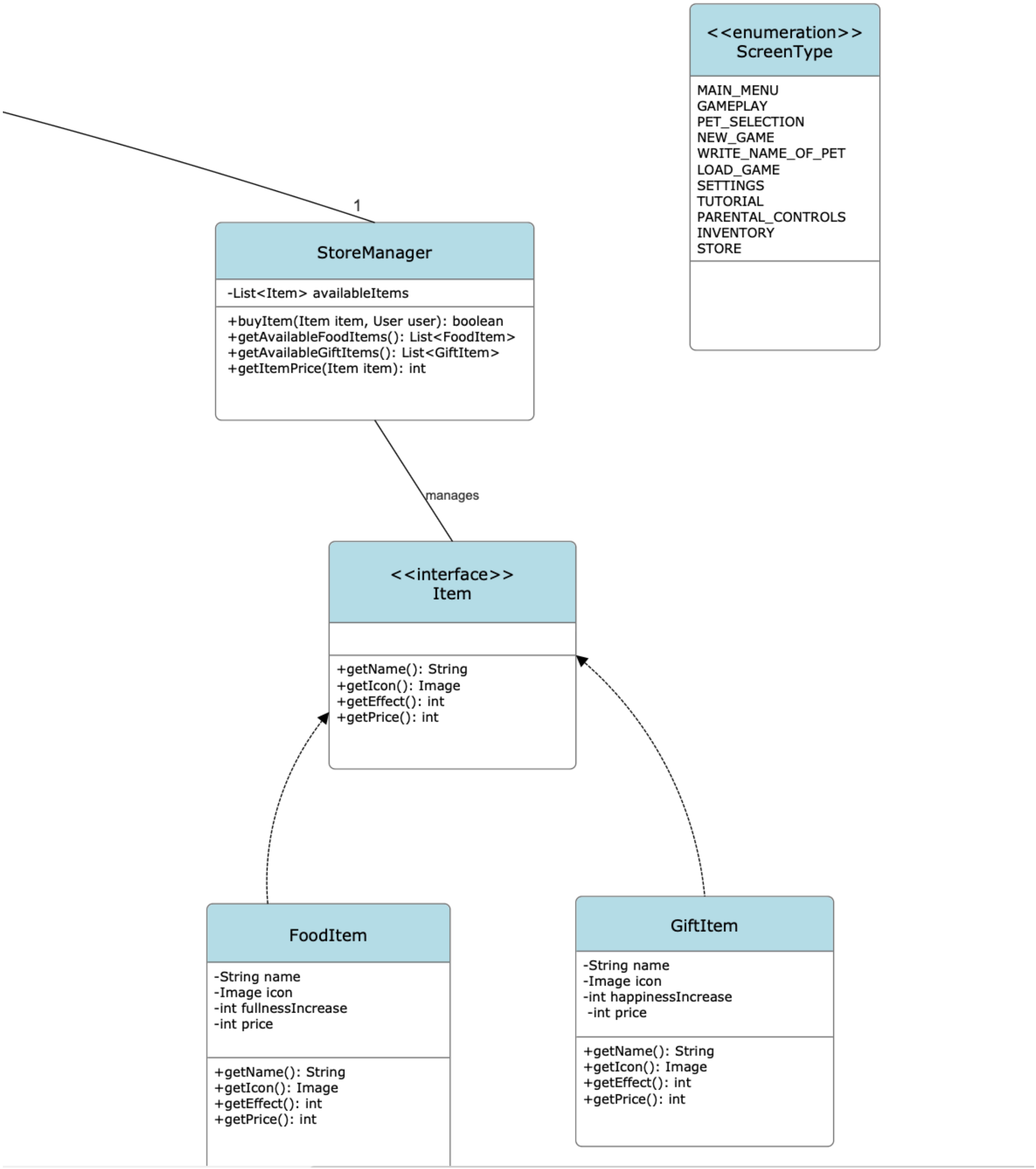

StoreManager
-List<Item> availableItems
+buyItem(Item item, User user): boolean
+getAvailableFoodItems(): List<FoodItem>
+getAvailableGiftItems(): List<GiftItem>
+getItemPrice(Item item): int
INVENTORY
STORE
manages
<<interface>>
Item
+getName(): String
+getIcon(): Image
+getEffect(): int
+getPrice(): int
FoodItem
-String name
-Image icon
-int fullnessIncrease
-int price
+getName(): String
+getIcon(): Image
+getEffect(): int
+getPrice(): int
GiftItem
-String name
-Image icon
-int happinessIncrease
-int price
+getName(): String
+getIcon(): Image
+getEffect(): int
+getPrice(): int

Image 13



Class Diagram Description and Design Decisions

The class diagram represents the foundational architecture of our project, defining the core business logic and interactions between key components. The design follows an object-oriented approach, ensuring modularity, reusability, and maintainability.

1. Core Classes and Their Responsibilities

The main classes in our diagram define the fundamental components of the system. These include:

GameManager

- This class acts as the central controller of the application, managing game state transitions, player progress, and interactions with other components.
- It includes methods for starting a new game, loading a saved game, and updating game logic.
- The class interacts with multiple other classes, ensuring a structured flow of operations.

Player

- Represents the user playing the game and stores essential attributes such as player name, progress, and inventory.
- Methods include modifying player statistics, handling user inputs, and interacting with the game environment.
- The class has associations with Inventory and GameState, ensuring player data is consistently updated.

GameState

- Maintains the current status of the game, including player progress, achievements, and ongoing interactions.
- Ensures data persistence when the game is saved or loaded.
- Works closely with the GameManager to update changes dynamically.

Inventory

- A structured collection of items the player has acquired throughout the game.
- Uses a composition relationship with Player, ensuring every player has a unique inventory.
- Methods allow adding, removing, and retrieving items, making inventory management efficient.

2. Game Logic and Interaction

The following classes define the core gameplay elements:

Pet

- Represents the virtual pet that the player interacts with.
- Includes attributes such as happiness, hunger, energy, and health.
- Methods allow feeding, playing, exercising, and treating the pet, with logic that updates its state based on actions taken.

ActionHandler

- Implements the core game logic, handling player actions and their impact on the game environment.
- Uses methods to process player commands and update relevant game objects accordingly.

GameUI

- Represents the interface layer responsible for rendering the game's graphical elements.
- Provides interaction between the player and the game through various UI elements.
- The class does not contain business logic but serves as a bridge between the user and the backend.

3. Utility and Persistence

Additional classes provide essential support functionalities:

SettingsManager

- Stores and retrieves user preferences, including audio settings, difficulty level, and parental controls.
- Ensures that settings persist across sessions.

SaveLoadManager

- Handles game data persistence, allowing players to save and load their progress.
- Uses file I/O operations to store structured game data.

ParentalControl

- Implements restrictions for younger players.
- Provides functionality for limiting gameplay duration and monitoring player activity.

4. Class Relationships and Design Decisions

- **Associations and Multiplicity:** Classes are linked appropriately using associations. For example, a **Player** owns an **Inventory**, while **GameManager** manages multiple **GameState** objects.
- **Generalization and Inheritance:** Some classes share a parent-child relationship, optimizing code reuse and extending functionality.
- **Encapsulation and Visibility:** Attributes and methods follow appropriate visibility constraints (Private, Public, Protected) to ensure data integrity and prevent unintended modifications.

Conclusion

This structured design ensures **scalability and modularity**, allowing future enhancements without major redesigns. The use of **enumerations, well-defined classes, and object-oriented principles** ensures a maintainable and extensible system.