Bilkent University

Object-Oriented

Software Engineering

System Design Report

Department of Computer Engineering

Roomie

Project Group:

- *Deniz Alkışlar*
- *Ekinsu Bozdağ*
- *Eliz Tekcan*
- *Serhat Aras*
- *Selen Haysal*

Supervisor:

- *Bora Güngören*

Context of the Paper

Introduction 4

1.1. Purpose of the System 4

1.2. Design Goals 4

End-User 4

Developer 5

Tradeoffs 6

1.3. References 7

1.4. Overview 7

Software Architecture 7

2.1. Overview 7

2.2. Subsystem Decomposition 8

2.3 Architectural Design 9

2.4 Hardware / Software Mapping 10

2.5 Persistent Data Management 11

2.6 Access Control and Security 11

2.7 Boundary Conditions 11

Initialization 11

Termination 11

Error 11

Subsystem Services 12

3.1 Game(Model) 12

3.1.1. Game Class 13

3.1.2. Game Environment Class 14

3.1.3 Question Class 15

3.1.4. HouseItem Class 16

3.1.5. Room Class 16

3.1.6. House Class 17

3.1.7. Outdoor Class 17

3.1.8. FoodItem Class 18

3.1.9. Cafe Class 19

3.1.10. School Class 19

3.1.11. NightClub Class 20
3.1.12. Library Class 20
3.1.13. Events Class 21
3.1.14. Event Class 21
3.1.15. Options Class 22
3.1.16. Stats Class 23
3.1.17. Graph Class 23
3.1.18. Player Class 24
3.1.19. Backpack Class 25
3.1.20. Item Class 26
3.1.21. Item Collection Class 27
3.1.22. Randomizer Class 27
3.1.23. Enumaration Classes: 28
3.2 User Interface(View) 28
3.2.1 ActivityManager Class 29
3.2.2 LoginActivity Class 29
3.2.3 HouseActivity Class 30
3.2.4 OutsideActivity Class 31
3.2.5 Other Classes 32
3.3 Game Management(Controller) 32
3.3.1 GameManager Class 32
3.3.2 InputManager Class 33
3.3.3 SaveManager Class 34
3.3.4 SoundManager Class 35
3.4 Detaited System Design 36

Introduction

1.1. Purpose of the System

Roomie is a story-based, single player Android game designed to simulate the university life of a student. With Roomie, we aim to make users enjoy and understand how the university life can actually be, and how our choices matter with the different scenarios Roomie offers. The user-friendly design of Roomie makes it simpler to play. As the game starts, the user creates his/her unique character with entering its name and gender, with this feature we aim to make the game user-specific. Roomie is designed to keep the user occupied with the notifications and unexpected tasks during the day, even if the game is not on. This feature of Roomie makes it more challenging compared to other story-based games, and it helps Roomie to preserve its users interest. Roomie also includes a status bar with four different labels: health, money, sociality and grades. With this feature our aim is to keep the user engaged and more competitive while making choices since their choice will increase or decrease the levels of these statuses. Roomie also aims to display the user its interesting features in their first few visits, so that the user will preserve interest.

1.2. Design Goals

Prior to building the system, our aim is to identify the design goals to clarify what the system will focus on. Below are our design goals, many of them are included in the analysis stage of Roomie, in non-functional requirements.

End-User

Reliability

The reliability of the system is dependent to the reliability of Android. Exception handling will be carried out during the execution of the Roomie using both Android and Java's native Exception handling methods.

Usability

We aim to plan how the users of Roomie will engage with the app, looking at Roomie from their perspective, to see how they will understand the app and use it correctly. This will lead Roomie to create a positive user experience. Another approach we are going to take to create a positive user experience is to make Roomie require as few steps as possible. We will

implement Roomie in a way that the user will go through fewer steps, pages and buttons. We plan to design the app in a way that all options make sense to the user, spending time to figure out the best structure and see how all features of Roomie will fit together in an organized way.

Extensibility

Since the game is designed to be object-oriented, it can easily be modified and extended. The design architecture reduces the malfunctioning possibility and makes it possible to extend and change particular features without having to modify other classes. Its design makes Roomie highly extensible. Roomie is also suitable to be reused and extended for further versions and other works.

Adaptability

We are using Android to fulfill adaptability feature, Roomie will be available on Android devices with Android version 6.0 and above.

Ease of learning

Our design will be simple and effective so that the user can easily understand and perform actions. We plan to make the user interface easy to use, and use icons to guide the users and use iconography to lead the users, this makes Roomie easier to learn and play, even for user without prior knowledge of the game.

Portability

Roomie is highly portable since it will run on any Android device with Android version 6.0 and above.

Availability

Since Android API 23 (Marshmallow) is going to be used in the project development, Roomie will be available to those with Android version 6.0 and above.

Developer

Minimum # of errors

The design of Roomie aims to have minimum number of errors.For this purpose, we will use the libraries that Android provides. Complicated and long codes will be avoided which will result in easier code to debug. Many classes will be utilized to have a more stable structure in our implementation. We will employ unit tests to validate that every class works fine. To develop a more reliable system, we will make sure that necessary exception handlings are done.

Modifiability

This aspect is not a must for Roomie. However, it is intended that Roomie will be highly modifiable. To fulfill this aspect, our aim is to reduce the number of modules that are directly affected by change. To achieve this, the coupling of the subsystems will be minimized so that a change in a system component does not have a huge effect in the system.

Reusability

It is aimed to make Roomie suitable to be reused and extended for further versions and other works, but it is not a priority.

Tradeoffs

Efficiency *- Reusability*

Efficiency is an aspect that matters for Roomie. Roomie is implemented in an efficient way by using graph data structure. The events are kept in a graph design since Roomie is an event-based story game. Arrays are not utilized in the fundamental design to keep the implementation efficient. Also, when the players save a game on Roomie, the whole game is not saved. Instead, only the key points that are changed during the game will be saved, and the game will be loaded with those changed state savings. The game is implemented by adopting from what Android has to offer which makes the implementation more efficient. Roomie's code could be reused in the case of extending the game and building on the design. The storyline might be enlarged by modifying the graph structure and other features might be added.

Functionality - ***Usability***

Roomie's prior concern is to have a simple usage rather than having a complex functionality. The game will not be too complex to play, since this would decrease the interest in Roomie. The game will have a simple user interface and easy to learn actions that the user can perform. The user will not have to spend time understanding what can be performed, the game will proceed easily without confusing the player. With its minimal features and functionality, Roomie will be as user-friendly as possible.

Space - ***Speed***

Roomie does not occupy a large space since the game contains of only source code files and multimedia files that are used in the user-interface. Also the game will save and load into the Internal Storage Device, again not taking up storage space. Roomie tends to be a dynamically working, high performance game and provide its users a fast and vital game environment.

1.3. References

[1] “Storage Options.” Android Developers, 8 Nov. 2016, developer.android.com/guide/topics/data/data-storage.html#pref.

[2] Rogers, Rick. Learning Android Game Programming: A Hands-on Guide to Building Your First Android Game. Addison-Wesley, 2012.

[3] “Design Patterns MVC Pattern.” Www.tutorialspoint.com, Tutorials Point, 15 Aug. 2017, www.tutorialspoint.com/design_pattern/mvc_pattern.htm.

1.4. Overview

In the Introduction section of this Design Report, the purpose of Roomie, which is to simulate the life of an university student is explained in depth. The design goals of Roomie are detailed. Reliability, usability, extensibility, adaptability, ease of learning, portability and availability compose our prior concerns regarding Roomies design. Among efficiency and reusability Roomie favors efficiency, since our prior concern is to develop an efficient game rather than writing reusable code. Our user-friendly design and not complex functionalities of Roomie leads Roomie to choose usability over functionality. Regarding speed and space, Roomie opts for speed, it is designed in a way that lessens the storage use and is a fast game environment.

Software Architecture

2.1. Overview

In this chapter, the architecture of our design will be revealed in more detail. The aim is to define an abstract framework for the components of Roomie and to set a firm structural organization. Along with structural design, more details about user access, data management and boundary conditions will be disclosed.

2.2. Subsystem Decomposition

In this section, the aim is to decompose our system into smaller subsystems to have a more stable and understandable structure. The whole system of our project is divided into three main sub-categories. This structural design meets our non-functional requirements such as modifiability and extendibility. Their connection with each other, and their independent roles makes the whole implementation easier and logical. This makes any possible further modifications easier to implement.

The first subsytem is called User Interface, and it holds viewpoints. The ActivityManager in this system responds the requests that are coming from the lower subsystem which is called Game Manager. Game Manager is responsible for receiving user input and evaluating these inputs, Game Manager sends appropriate requests to User Interface to update the screen view. Along with these two subsystems, Game subsystem provides the main logic of the game. All of the game events, player objects, and other components that are related to the internal logic of the game are hold in this subsystem. See the figure that is below for more detailed

view of these three subsystems' decomposition

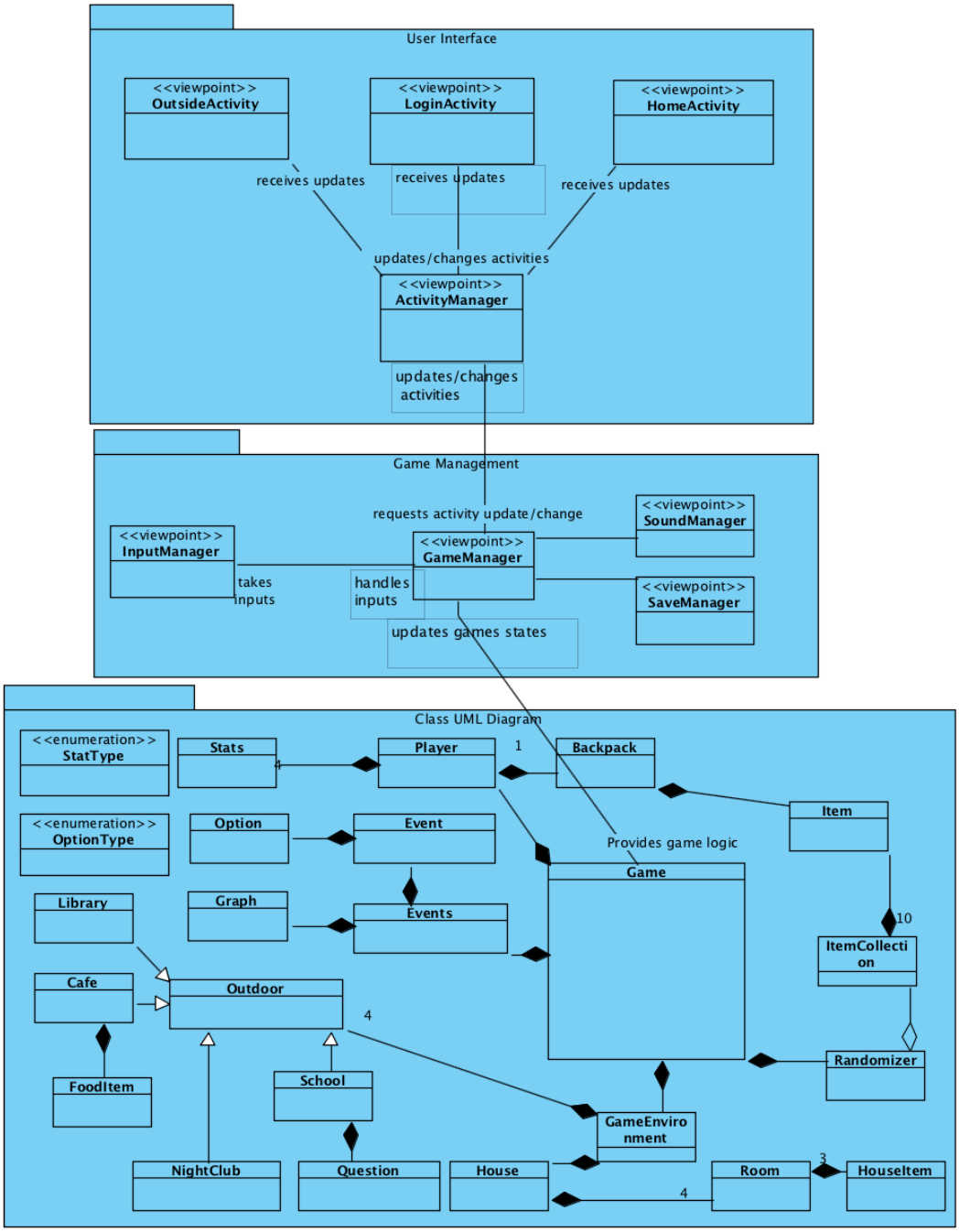


Figure 1: Basic Subsystem Decomposition

2.3 Architectural Design

Roomie follows Model-View-Controller Architectural Pattern in its design. Using this model, the separation between the UI/Interaction layer and the whole logic layer becomes apparent. There is also another layer that arranges the relationship between these two other layers, the controller layer. The view layer will be responsible for the visualization of the model

layer [3]. The model layer contains all the objects that contain data about the game. The control layer is to update the view layer whenever there is change in data of the model layer. The corresponding components in our design are listed below:

1. View Layer - User Interface: This layer is about the UI that is displayed to user while playing the game.
2. Controller Layer - GameManager Class: This class is to provide connection between the view layer and the model layer. It can be thought as an action listener, its duty is to listen user inputs and direct those inputs to view layer for displaying appropriate screens.
3. Model Layer - Game Class: The class that implements the whole internal logic and flow of the game. All of the game related objects (e.g Player, Stats, School) are implemented in this layer of the architecture.

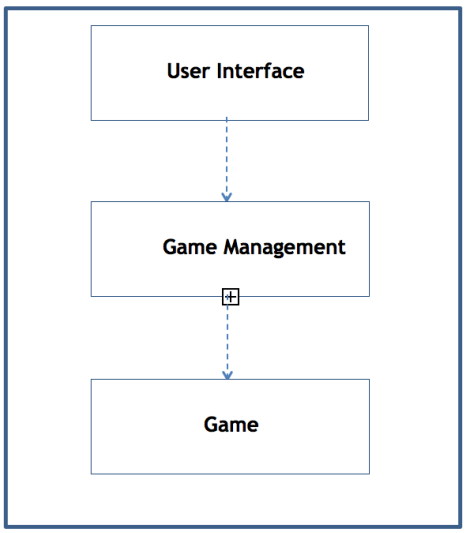


Figure 3: Three layers of MVC Design Pattern

2.4 Hardware / Software Mapping

Roomie will be implemented in Java programming language for Android. JDK 1.8 (Java Development Kit) will be used. As hardware setup, a working Android phone is enough to run Roomie. The touchscreen of the phone, its speakers and microphone are the components

where the game will take input from. Roomie does not require Internet/ Network connection to operate, and therefore it is an offline game.

2.5 Persistent Data Management

To manage data, Android's SharedPreferences class will be utilized. This class lets applications to "save and retrieve persistent key-value pairs of primitive data types." [1] Using this class, the objects of the game will be saved as strings. Also, the point that the player leaves the game will be saved along with the player's status points. Since Roomie is implemented using graph data structure, the design will be able to observe and understand the event that user leaves the game. Using this class, all of this data information will be directly saved on the device's internal storage. These files that are stored are private and the player cannot access them. If the player uninstalls Roomie, all of these files will be removed.

2.6 Access Control and Security

In Roomie, there will not be any access control. Anyone who installs the game could play the game. Since there is no network connection, user registration is also not possible. However, the player will be able to create a character for himself before starting the game. There are not security concerns about the game, considering that the content of the game and its limitations are not matters of security.

2.7 Boundary Conditions

Initialization

Since Roomie is an Android game, it will require an installment from Google Play. After installing the game, touching on the Roomie icon is enough to initiate the game.

Termination

Roomie can be terminated / closed by clicking "Home" button of the phones, the main menu of the game will also let the players close the game. If player wants to quit during game play, system provides an automatic save and the player is able to continue the game from the same point.

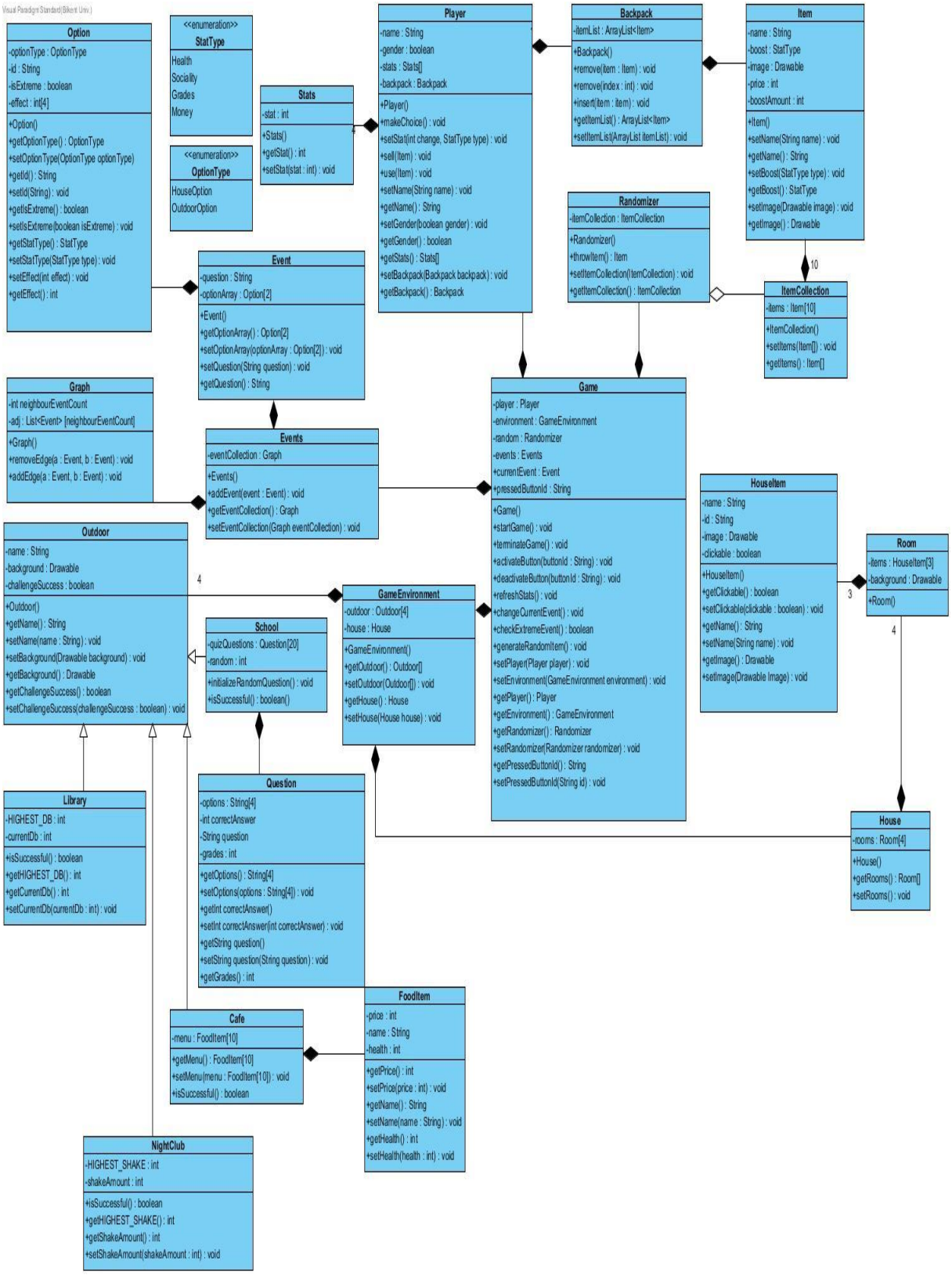
When the user completes a game cycle, main menu screen will be directed to the player.

Error

If program does not respond because of a performance/compatibility issue, the player will need to restart the game and lose all of the current data.

Subsystem Services

3.1 Game(Model)



3.1.1. Game Class

Attributes:

- **private Player player:** This attribute holds an instance of a Player class which holds the player related information and actions of the Roomie.
- **private GameEnvironment environment:** This attribute is containing the game environments in which implemented as GameEnvironment Class.
- **private Randomizer random:** This element holds the randomizer reference for randomizing the items which dropped to the player's Backpack.
- **private Events events:** This attribute holds the evets for the game.
- **public Event currentEvent:** This attribute holds the initial event of the Roomie
- **public String pressedButtonId :** This attribute stores the pressed button's ID.

Constructors:

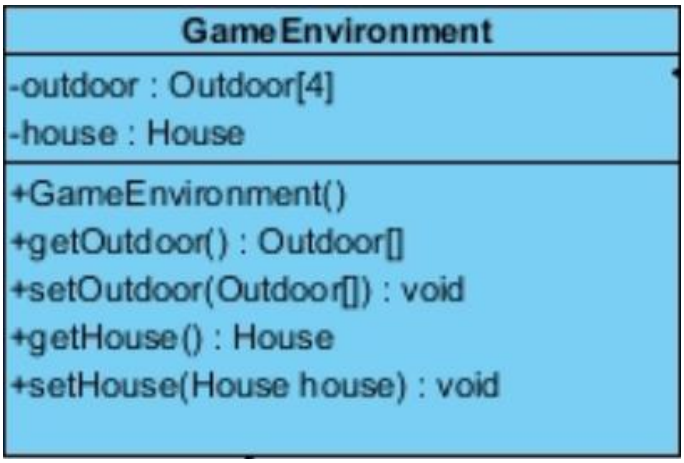
- **public Game():** Initialize the game instance for the Roomie which holds the information of the player, environment, events, current event, pressed button and random class.

Methods:

- **public void startGame():** This method starts the game with an environment and event also uses the player's backpack information.

- **public void activateButton(String buttonId):** This method activates a button with the give button ID.
- **public void deactivateButton(String buttonId):** This method deactivates a button with the give button ID.
- **public void refreshStats():** This method refreshes the player stats.
- **public void changeCurrentEvent():** This method changes the current event from the event list.
- **public boolean checkExtremeEvent():** This method decides whether or not an extreme case is going to occur in the current game session.
- **public void generateRandomItem():** This method generate a random item for the item collection of the player called backpack
- **public boolean startChallenge():** This method initiates the challenge of the Roomie events.
- **public void setPlayer(Player player):** This method sets the player
- **public void setEnvironment(GameEnvironment environment):** This method sets the Game environment.
- **public void setRandomizer(Randomizer random):** This method mutates the Randomizer.
- **public void setPressedButtonId(String Id):** This method sets the PressedButton.
- **public Player getPlayer():** This method returns the Player.
- **public GameEnvironment getEnvironment():**This method returns the GameEnvironment.
- **public Randomizer getRandomizer():** This method returns the Randomizer.
- **public String getPressedButtonId():** This method returns the pressed button's ID.

3.1.2. Game Environment Class



Attributes:

- **private Outdoor[] outdoor:** This attribute holds the outdoor Game Environment.
- **private House house:** This attribute holds the House object.

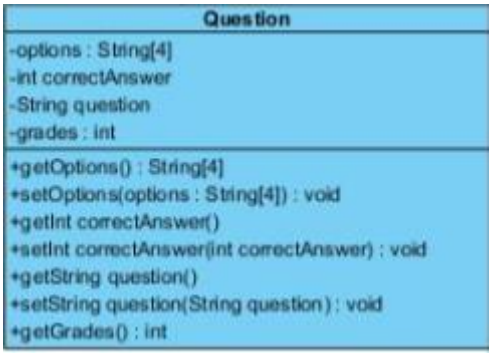
Constructors:

- **public GameEmvironment():** Constructs the game environments.

Methods:

- **public Outdoor[] getOutdoor():** Returns the out door.
- **public void setOutdoor(Outdoor[] outdoor):** This method set the outdoor by the outdoor parameter.
- **public House getHouse():** This method returns the house object.
- **public void setHouse(House house):** This method sets the house by the parameter.

3.1.3 Question Class



Attributes:

- **private String[4] option:** Holds the answers fort he question.
- **private int correctAnswer:** Holds the correct answer's index in the array.
- **private String question:** This variable stores the question as string
- **private int grades:** holds the players grade.

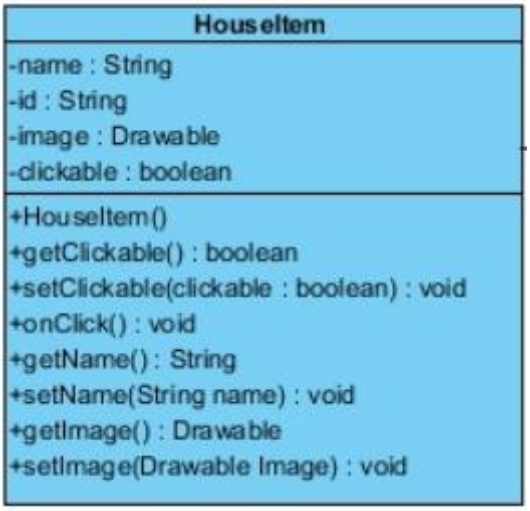
Constructors

- **public Question():** Initialize and constructs the question itself.

Methods:

- **public String[] getOptions():** Returns the Options.
- **public void setOption(String[] options):** Sets the options.
- **public int getCorrectAnswer():** returns the Correct Answer.
- **public void setCorrectAnswer(int correctAnswer):** Sets the answer to the parameter.
- **public String getQuestion():** returns the question
- **public void setQuestion(String question):** Sets the question to the parameter.
- **public void setGrades(int grade):** Sets the grades.
- **public int getGrades():** Returns the grades.

3.1.4. HouseItem Class



Attributes:

- **private String name:** This variable stores the name of the item.
- **private String id:** This attribute holds the id of the house item.
- **private Drawable image:** This drawable variable holds the image of the house item.
- **private Boolean clickable:** This Boolean attribute stores the condition of the clickable case of the item.

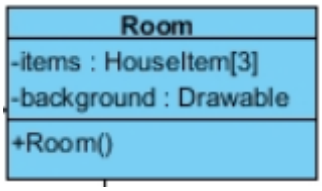
Constructors:

- **public HouseItem():** Construtor for the House Item Class.

Methods:

- **public Boolean getClickable():** Return true if item is clickable, false if otherwise.
- **public void setClickable (boolean clickable):** This function overwrites the boolean attribute of the clickable.
- **public void onClick():** Extended from the ClickListener, implemented for the click action.
- **public String getName():** Returns the name of the house item
- **public void setName(String name):** Set the name of the house item.
- **public Drawable getImage():** Returns the Drawable object of the House item.
- **public void setImage(Drawable Image):** Mutate the Drawable object of the house item.

3.1.5. Room Class



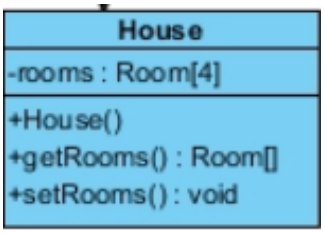
Attributes:

- **private HouseItem[3] items:** Stores the house item in an HouseItems array.
- **private Drawable background:** Holds the background of the room.

Constructors:

- **Public Room():** Construct the room with the items and background.

3.1.6. House Class



Attributes:

- **private Room[4] rooms:** Holds the Room in the house.

Constructors:

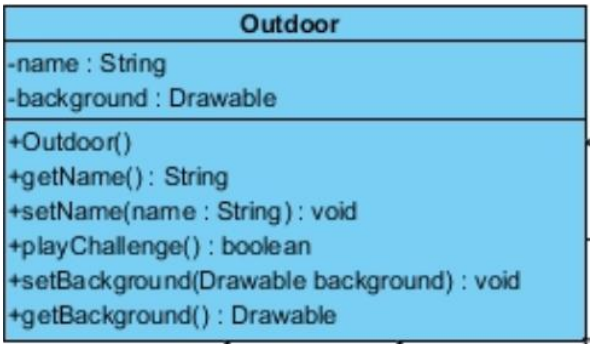
- **public House():** The constructer of the House class Initiates the rooms.

Methods:

- **public Room[] getRooms():** This function returns the rooms of the house as Room Array.

public void setRooms(): This functions sets the rooms of the house.

3.1.7. Outdoor Class



Attributes:

- **private String name:** This attribute holds the name of the outdoor.
- **private Drawable background:** This attribute holds the background of the Outdoor.

Constructors:

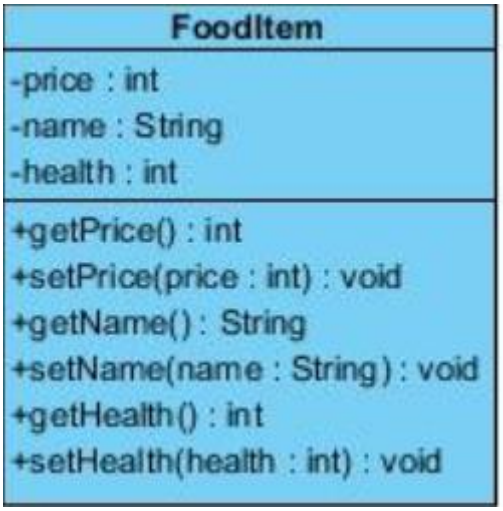
- **public Outdoor():** This constructor Initiates the ourdoor object.

Methods:

- **public String getName():** Return the name of the outdoor object as String
- **public void setName(String Name):** Sets the name of the string

- **public boolean playChallange():** This function plays the challenge and returns te result as boolean variable
    - If the playChallange() return true, then the challenge successfully completed:
    - If the playChallange() function returns false, then the challenge is failed.
    -
- **public void setBackground(Drawable background):** The function sets the Background of the outdoor form the given parameter.
- **public Drawable getBackground():** This accessor returns the Image of the background as Drawable object.

3.1.8. FoodItem Class



Attributes:

- **private int price:** Holds the price of the food
- **private String name:** Holds the name of the food.
- **private int health:** Holds the health boost amount.

Methods:

- **public int getPrice():** Returns the price of the food.
- **public void setPrice(int price):** Sets the price of the food.
- **public String getName():** Returns the name of the food.
- **public void setName(String name):** Set the name.
- **public int getHealth():** Returns the health of the food.
- **public void setHealth(int health):** Set the health.

3.1.9. Cafe Class

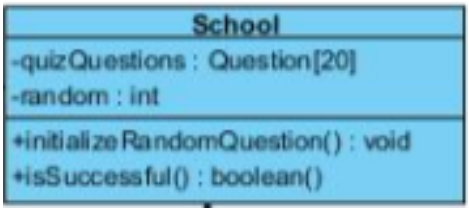
Attributes:

- **private FoodItem[] menu:** Holds all the food in the roomie.

Methods:

- **public FoodItem[] getMenu():** Returns the menu.
- **public void setMenu(FoodItem[] menu):** Sets the new menu.
- **public boolean isSuccesful():** Returns true if the café challenge completed successfully, false in the case of failing.

3.1.10. School Class



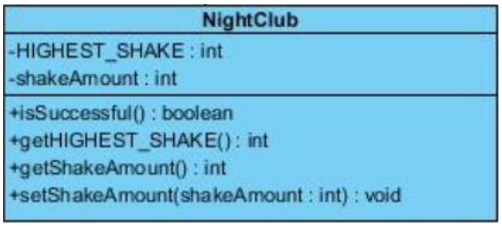
Attributes:

- **private Question[] quizQuestions:** Holds the quiz questions.
- **private int random:** Random Question index.

Methods:

- **public void initializeRandomQuestion():** This method returns a random integer between 0 and 19 to determine a unique ordered sequence.
- **public boolean isSuccesful():** Returns true if the School challenge completed successfully, false in the case of failing.

3.1.11. NightClub Class



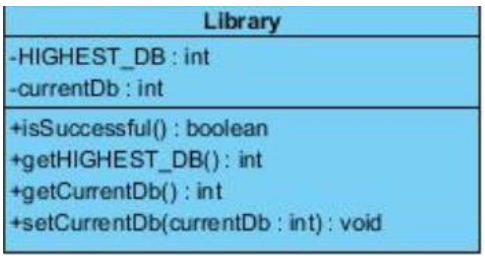
Attributes:

- **public int HIGHEST_SHAKE**: stores the highest amount of shake as constant.
- **public int shakeAmount():** holds the players shake amount.

Methods:

- **public boolean isSuccesful():** Returns true if the NightClub challenge completed successfully, false in the case of failing.
- **public int getHIGHEST_SHAKE():** Returns the highest shake constant.
- **public int getShakeAmount():** Returns the shake amount of the player.
- **Public void setShakeAmount(int shakeAmount):** Sets the shakeAmount.

3.1.12. Library Class



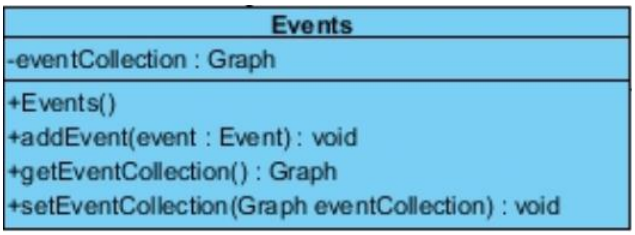
Attributes:

- **private int HIGHEST_DB:** Stores the highest Decibel
- **private int currentDb:** Stores the current Db level.

Methods:hi

- **public boolean isSuccesful():** Returns true if the Library challenge completed successfully, false in the case of failing.
- **public int getHIGHEST_DB():** Returns the maximum Db unit.
- **Public int getCurrentDb():** retrieves the current Db level as integer.
- **Public void setCurrentDb:** This function is going to be mutate the current Db level.

3.1.13. Events Class



Attributes:

- **privates Graph eventCollection:** Holds the graph Data Structure for storing the events.

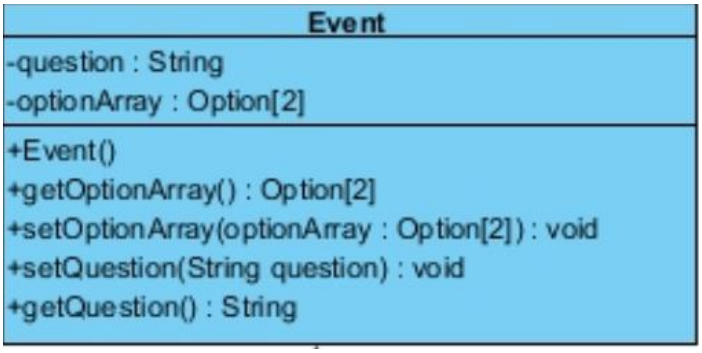
Constructors:

- **public Events():** Constructs the Events Class by the events graph

Methods:

- **public void addEvent(event: Event):** This function add the event which is taken from the parameter to the eventCollection graph.
- **public Graph getEventCollection():** Returns the eventCollection graph.
- **public void setEventCollection(Graph eventCollection):** Set the eventCollecction of the Events class to the parameter eventCollection.

3.1.14. Event Class



Attributes:

- **private String question:** This String stores the question.
- **private Option[2] optionArray:** This Option array stores the possible answer to the question.

Constructors:

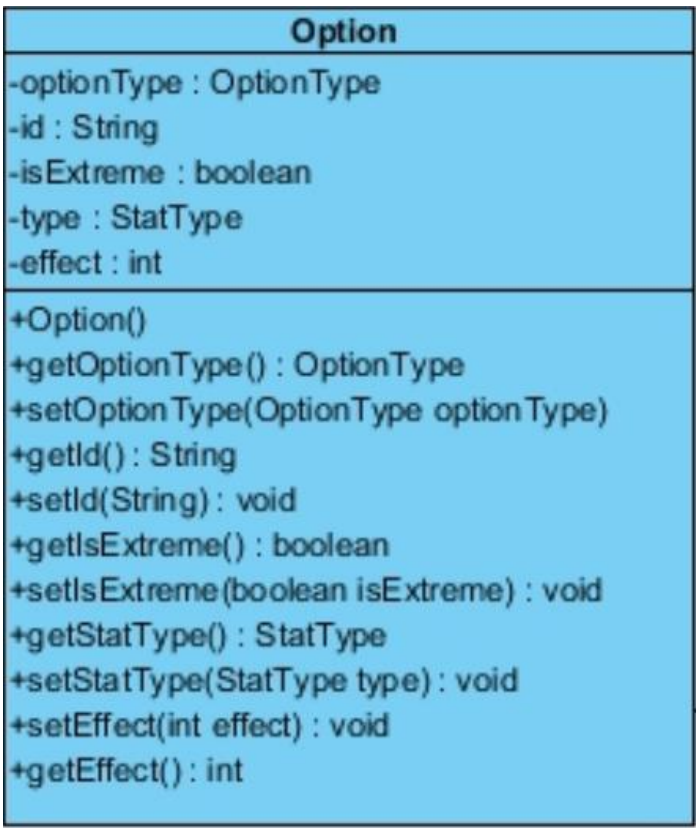
- **public Event():** Constructs the question and the options

Methods:

- **public Option[] getOptionArray:** This function returns the optionArray.
- **public void setOptionArray(Option[] optionArray):** This function sets the optionArray parameter to the array itself.

- **public void setQuestion(String question):** This function loads the question parameter to the Class' question.
- **public String getQuestion():** This function returns the question as String

3.1.15. Options Class



Attributes:

- **private OptionType optiontype:** This attribute holds the optionType.
- **private String id:** This variable holds the id of the Option.
- **private boolean isExtreme:** This variable is the flag variable for the extreme case since the options are going to treated differently for extreme cases.
- **private StatType type:** This variable holds the StatType attribute of the option
- private int effect: This integer holds the effect amount in case of use this option.

Constructors:

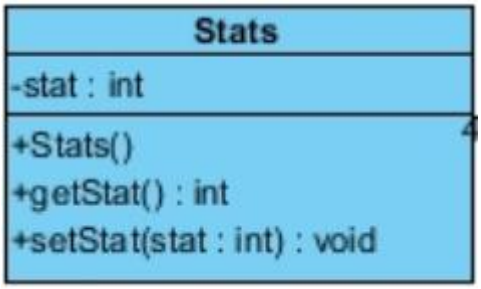
- **public Option():** This constructor initialize the Option object.

Methods:

- **public OptiontType getOptionType():** This function returns the OptionİType.
- **public void setOptionType(OptionType optionType):** Mutates the OptionType as the parameter.
- **public String getId():** Returns the Id as String.

- **public void setId(String id):** This function sets the newly given id string to the old one.
- **public boolean getIsExtreme():** This function returns the extreme case.
- **public void setIsExtreme(Boolean isExtreme):** Sets the option extreme case to the given boolean parameter.
- **public StatType getStatType():** This function returns the StatType of the Option
- **public void setStatType(StatType type):** This function overwrites the StatType with the given parameter.
- **public void setEffect(int effect):** This function sets the effect amount by the given parameter.
- **public int getEffect():** This function returns the affect integer.

3.1.16. Stats Class



Attributes:

- **private int stat:** This parameter stores the value of the stat as integer

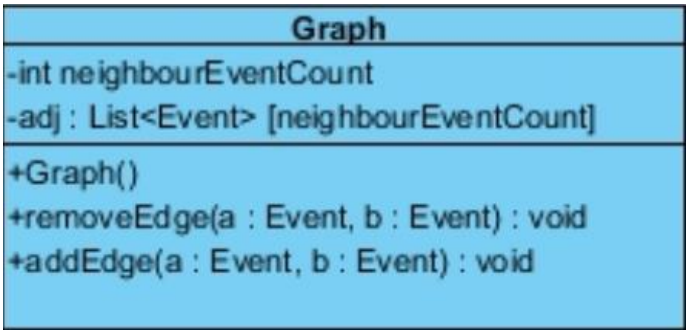
Constructors:

- **public Stats():** Constructs the Stats Object.

Methods:

- **public int getStat():** Returns the stat integer
- **public void setStat(stat : int):** Sets the stat integer by the given parameter.

3.1.17. Graph Class



Attributes:

- **private int neighbourEventCount:** This integer stores the adjacent edges on the graph. These edges are manifested as the neighbourEventCount.
- **private List<Event> [neighbourEventCount] adj:** Hold the linked edges to the one event as in the Graph implementation.

Constructors:

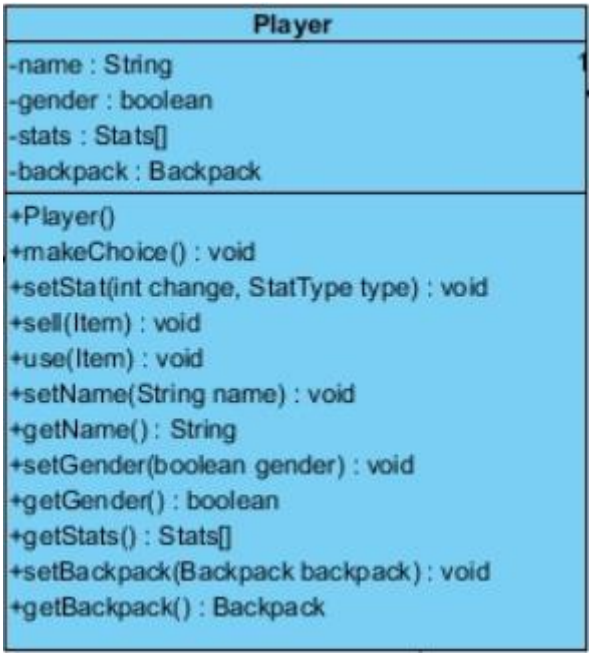
- **public Graph():** Constructs the Graph by the Attributes.

Methods:

public void removeEdge(a: Event, b:Event): Removes vertexes between event a and event b(events are treated as vertex)

public void addEdge(a: Event, b: Event): Adds vertexes between event a and event b(events are treated as vertex)

3.1.18. Player Class



Attributes:

- **private String name:** This variable holds the Name of the Player.
- **private boolean gender:** This variable holds the gender of the player.

- **private Stats[] stats:** This array stores the stats of the player.
- **private Backpack backpack:** This attribute denoted the itemCollection, known as BackPack (Inventory)

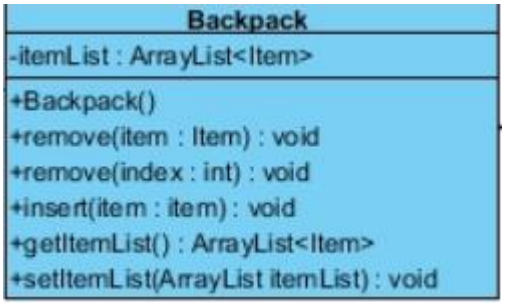
Constructors:

- **public Player():** Constructs the Player.

Methods:

- **public void makeChoice():** This function enable the player to make a choice for the event.
- **public void setStat(int change, StatType type):** This function overwrites the Stats as change amount and the type.
- **public void sell(Item item):** Sell an item from the BackPack which is going to worth for some money.
- **public void use(Item item):** Use an item from the Backpack to increase the stats.
- **public void setName(String name):** Sets the name of the player.
- **public String getName():** Returns the name of the player.
- **public void setGender(boolean gender):** sets the Gender of the player.
- **public boolean getGender():** Returns the gender of the player.
- **public Stats[] getStats():** Returns the Stats of the Player.
- **public void setBackpack(Backpack backpack):** Mutate the backpack by the parameter.
- **public Backpack getBackpack():** returns the Backpack of the player.

3.1.19. Backpack Class



Attributes:

- **Private ArrayList<Item> itemList:** This arraylist stores the items in the Backpack

Constructors:

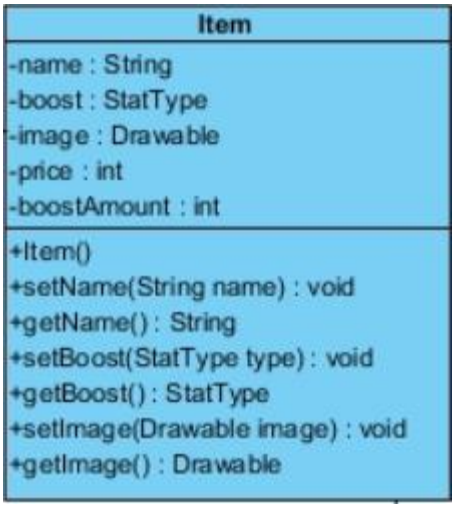
- **public Backpack():** Creates the Backpack

Methods:

- **public void remove (Item item):** Removes an item from the backpack
- **public void remove(int index):** Remove an item from the backpack with the given index.
- **public void insert(Item item):** Inserts the item to the Backpack.

- **public ArrayList<Item> getItemList():** Returns the backpack
- **public void setItemList(Arraylist itemList):** Sets the backpack with the given parameter.

3.1.20. Item Class



Attributes:

- **Private String name:** This attribute holds the name of the item.
- **private StatType boost:** This variable stands for the boost type of the item when it is used by the player.
- **private Drawable image:** Stores the image of the Item.
- **private int price:** Holds the price of the item.
- **private int boostAmount:** This variable stands for the boost amount of the item when it is used by the player.

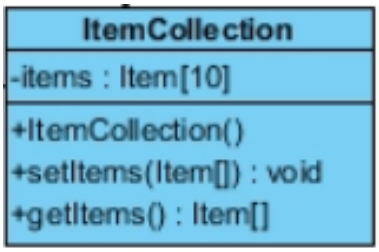
Constructors:

- **public Item():** Create and initialize the Item

Methods:

- **public void setName(String name):** Set the name of the Item
- **public String getName():** Returns the name of the Item as String Variable.
- **public void setBoost(StatType Type):** Sets the boosttype of the Item.
- **public StatType getBoost():** Returns the boost type of the object.
- **public void setImage(Drawable image):** Sets the image of the item.
- **public Drawable getImage():** Returns the image of the Item.

3.1.21. Item Collection Class



Attributes:

- **Private Item[10] items:** This array stores all items in the Roomie

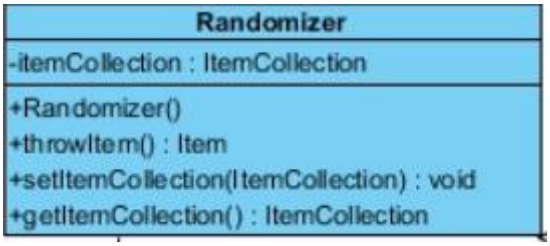
Constructors:

- **public ItemCollection():** creates the items and itemCollection.

Methods:

- **public void setItems(Item[]):** Sets the items form the parameter.
- **public Item[] getItems():** Returns the items in an array.

3.1.22. Randomizer Class



Attributes:

- **private ItemCollection itemCollection:** Holds the Items which are possibly to drop.(Added to the backpack)

Constructors:

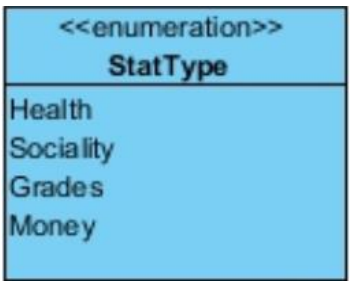
- **public Randomizer():** Initialize the Randomizer.

Methods:

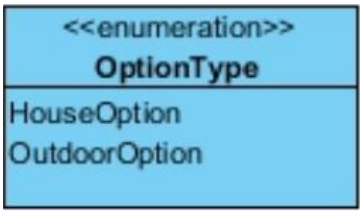
- **public Item throwItem():** Throws a random item to the backpack of the player form ItemCollection
- **public void setItemCollection(ItemCollection):** Sets predefined itemCollection which is taken from the parameter.
- **public ItemCollection getItemCollection():** Returns the itemCollection.

3.1.23. Enumaration Classes:

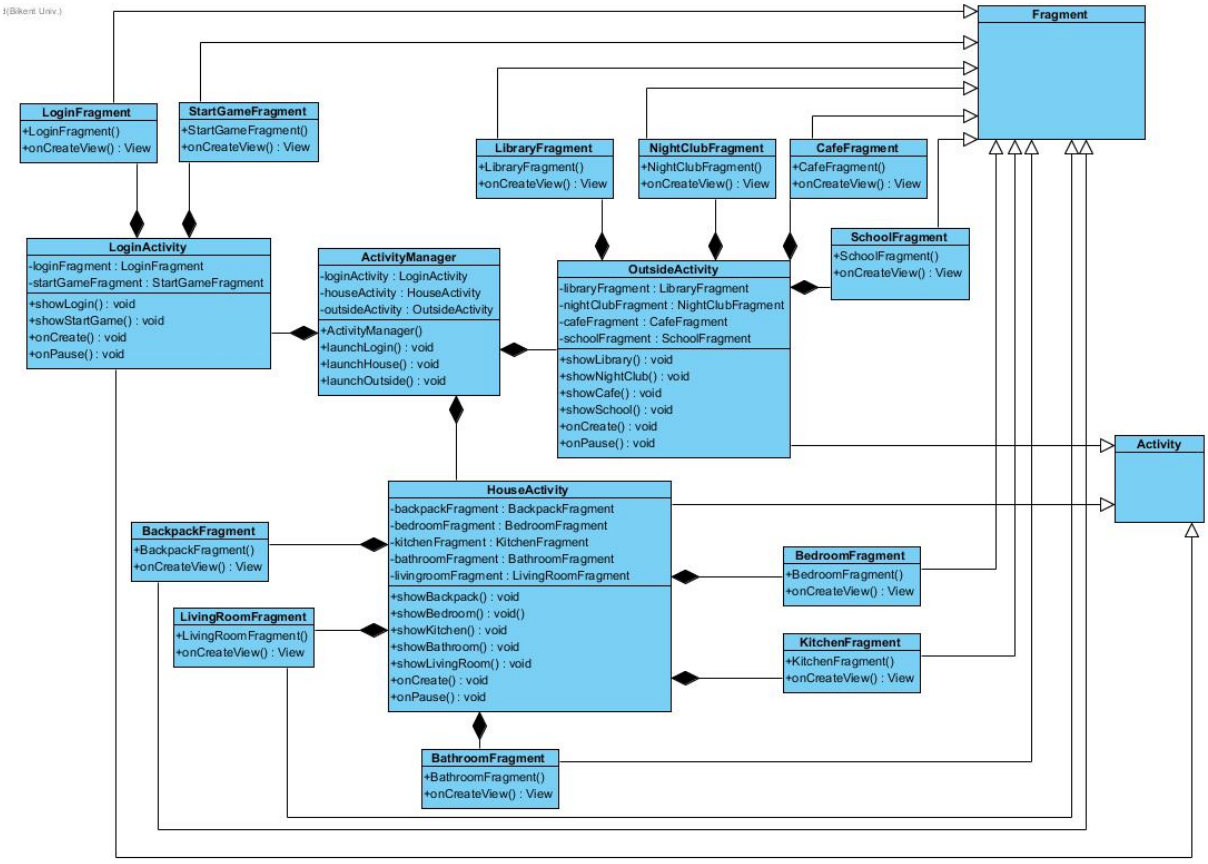
StatType



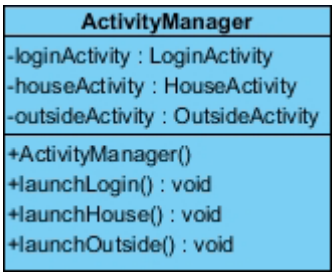
OptionType



3.2 User Interface(View)



3.2.1 ActivityManager Class



Attributes

- **private LoginActivity loginActivity:** This is the activity which Roomie starts with.
- **private HouseActivity houseActivity:** This is the activity which shows house and backpack of the player.
- **private OutsideActivity outsideActivity:** This is the activity when an outside event is choosen.

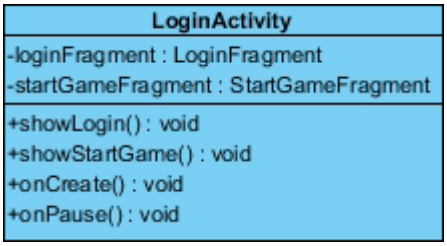
Constructors

- **public ActivityManager:** Initializes loginActivity, houseActivity and outsideActivity.

Methods

- **public void launchLogin():** This method lauches the login activity.
- **public void launchHouse():** This method lauches the house activity.
- **public void launchOutside():** This method lauches the outside activity.

3.2.2 LoginActivity Class



Attributes

- **private LoginFragment loginFragment:** This is the fragment when Roomie is first launched. It is constructed by some ui components such as buttons, labels and textareas.

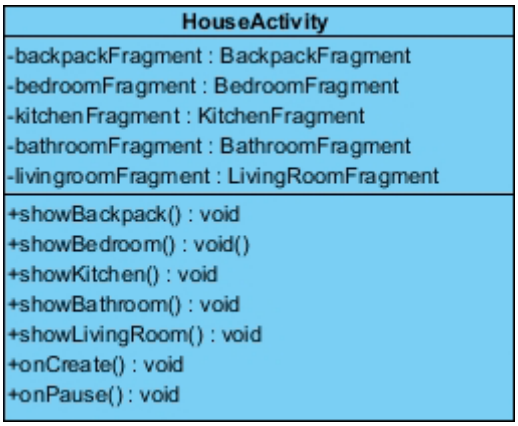
- **private StartGameFragment startGameFragment:** This is the fragment when Roomie is launched. It has Roomie's logo and a button to start the game.

Methods

- **public void showLogin():** This method sends loginFragment to the front.

- **public void showStartGame():** This method sends startGameFragment to the front.
- **public void onCreate():** This is an overridden method from extended Activity class. It basically behaves like a constructor.
- **public void onPause():** This is an overridden method from extended Activity class. It pauses the activity when other activities is launched.

3.2.3 HouseActivity Class



Attributes

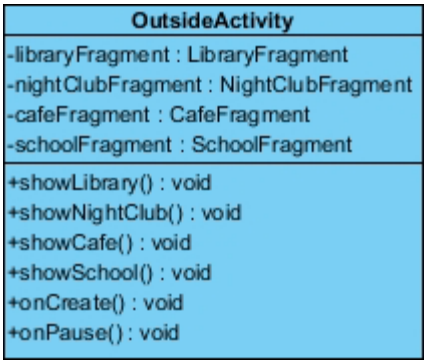
- **private BackpackFragment backpackFragment:** This fragment shows player's backpack. The backback has some items in it. It is constructed by some buttons and a listview.
- **private BedroomFragment bedroomFragment:** This fragment shows house's bedroom. It has some buttons with images to show items of the room.
- **private KitchenFragment kitchenFragment:** This fragment shows house's kitchen. It has some buttons with images to show items of the room.
- **private BathroomFragment bathroomFragment:** This fragment shows house's bathroom. It has some buttons with images to show items of the room.
- **private LivingRoomFragment livingroomFragment:** This fragment shows house's living room. It has some buttons with images to show items of the room.

Methods

- **public void showBackpack():** This method sends backpackFragment to the front.
- **public void showBedroom():** This method sends bedroomFragment to the front.
- **public void showKitchen():** This method sends kitchenFragment to the front.
- **public void showBathroom():** This method sends bathroomFragment to the front.
- **public void showLivingRoom():** This method sends livingroomFragment to the front.
- **public void onCreate():** This is an overridden method from extended Activity class. It basically behaves like a constructor.

- **public void onPause():** This is an overridden method from extended Activity class. It pauses the activity when other activities are launched.

3.2.4 OutsideActivity Class



Attributes

- **private LibraryFragment libraryFragment:** This fragment shows library and it's event. Library event requires player to stay quite in a some amount of time. It does not have additional UI components.
- **private NightClubFragment nightClubFragment:** This fragment shows night club and it's event. Night club event requires player to shake the device become successful. It does not have additional UI components.
- **private CafeFragment cafeFragment:** This fragment shows cafe and it's event. Cafe event requires player to choose a food item from the menu. The menu is constructed by using some labels, buttons and a listview.
- **private SchoolFragment schoolFragment:** This frahment shows school and it's event. School event requires player to fill out a quiz. The quiz is constructed by using some labels and buttons.

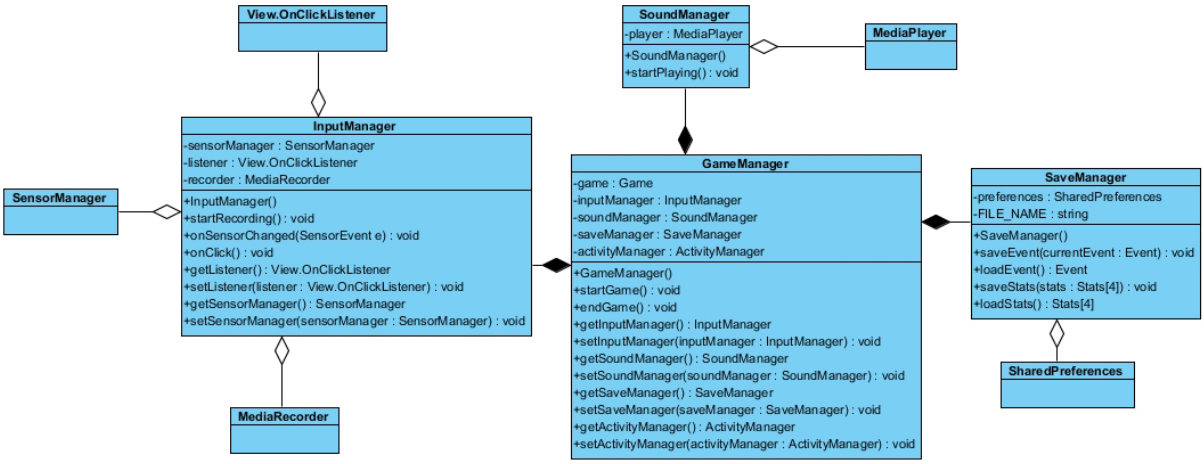
Methods

- **public void showLibrary():** This method sends libraryFragment to the front.
- **public void showNightClub():** This method sends nightClubFragment to the front.
- **public void showCafe():** This method sends cafeFragment to the front.
- **public void showSchool():** This method sends schoolFragment to the front.
- **public void onCreate():** This is an overridden method from extended Activity class. It basically behaves like a constructor.
- **public void onPause():** This is an overridden method from extended Activity class. It pauses the activity when other activities are launched.

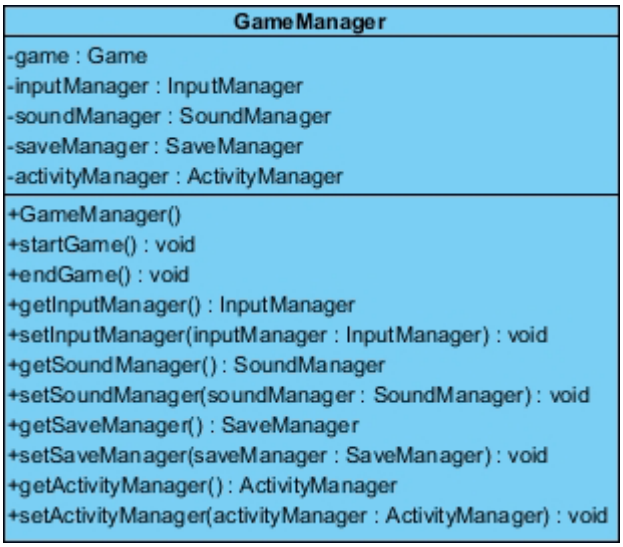
3.2.5 Other Classes

The other 11 classes are fragment classes. They have same functional properties. They are the subclass of the Android's fragment class. They all have an empty constructor which is neccessary. They also have onCreateView method to initialize properties and create UI components.

3.3 Game Management(Controller)



3.3.1 GameManager Class



Attributes

- **private Game game**: An instance of the game.

- **private InputManager inputManager:** An instance of the inputmanager.
- **private SoundManager soundManager**: An instance of the soundmanager.
- **private SaveManager saveManager**: An instance of the savemanager.
- **private ActivityManager activityManager:** An instance of the activitymanager.

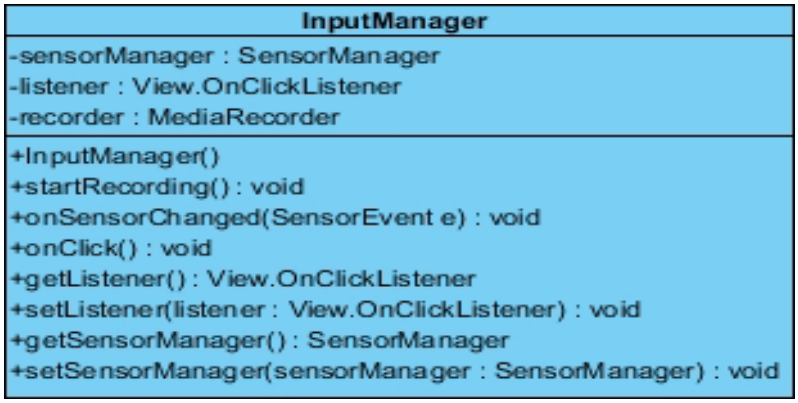
Constructors

- **public GameManager():** A constructor for initializing class properties.

Methods

- **public void startGame():** This method starts the game.
- **public void endGame():** This method ends the game.
- **The other methods are the getters and setters.**

3.3.2 InputManager Class



Attributes

- **private SensorManager sensorManager:** An instance of the sensormanager. This sensor provides gyroscope values to determine player's shake.
- **private View.OnclickListener listener:** An instance of the onclicklistener. This listener handles button clicks.
- **private MediaRecorder recorder:** An instance of the mediarecorder. This recorder provides microphone input to calculate decibel level.

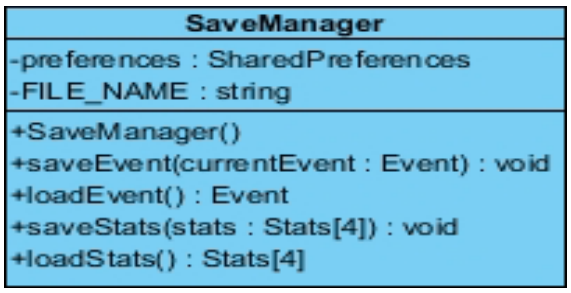
  Constructors
- **public InputManager():** A constructor for initializing the properties.

Methods

- **public void startRecording():** This methods starts the microphone for recording.

- **public void onSensorChanged(SensorEvent e):** This method is called when gyroscope sensor is changed.
- **public void onClick():** This method handles the button clicks.
- **The other methods are the getters and setters.**

3.3.3 SaveManager Class



Attributes

- **private SharedPreferences preferences:** This is an Android class that helps saving and loading key values.
- **private String FILE_NAME:** This is a constant file name to save and load the values.

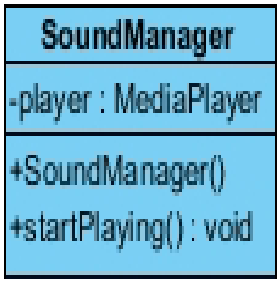
Constructors

- **public SaveManager():** A constructor for initializing the properties.

Methods

- **public void saveEvent(Event currentEvent):** This method saves the current event of the game. This method is called when the application is stopped.
- **public Event loadEvent():** This method loads the current event of the game. This method is called when the application is launched.
- **public void saveStats(Stats [] stats):** This method saves the stats of the player. This method is called when the application is stopped.
- **public Stats [] loadStats():**This method loads the stats of the player . This method is called when the application is launched.

3.3.4 SoundManager Class



Attributes

- **private MediaPlayer player:** This is an Android class that plays sounds.

Constructors

- **public SoundManager():** A constructor for initializing the properties.

Methods

- **public void startPlayin():** This methods starts mediaplayer and the device starts to generate sounds.

3.4 Detaited System Design

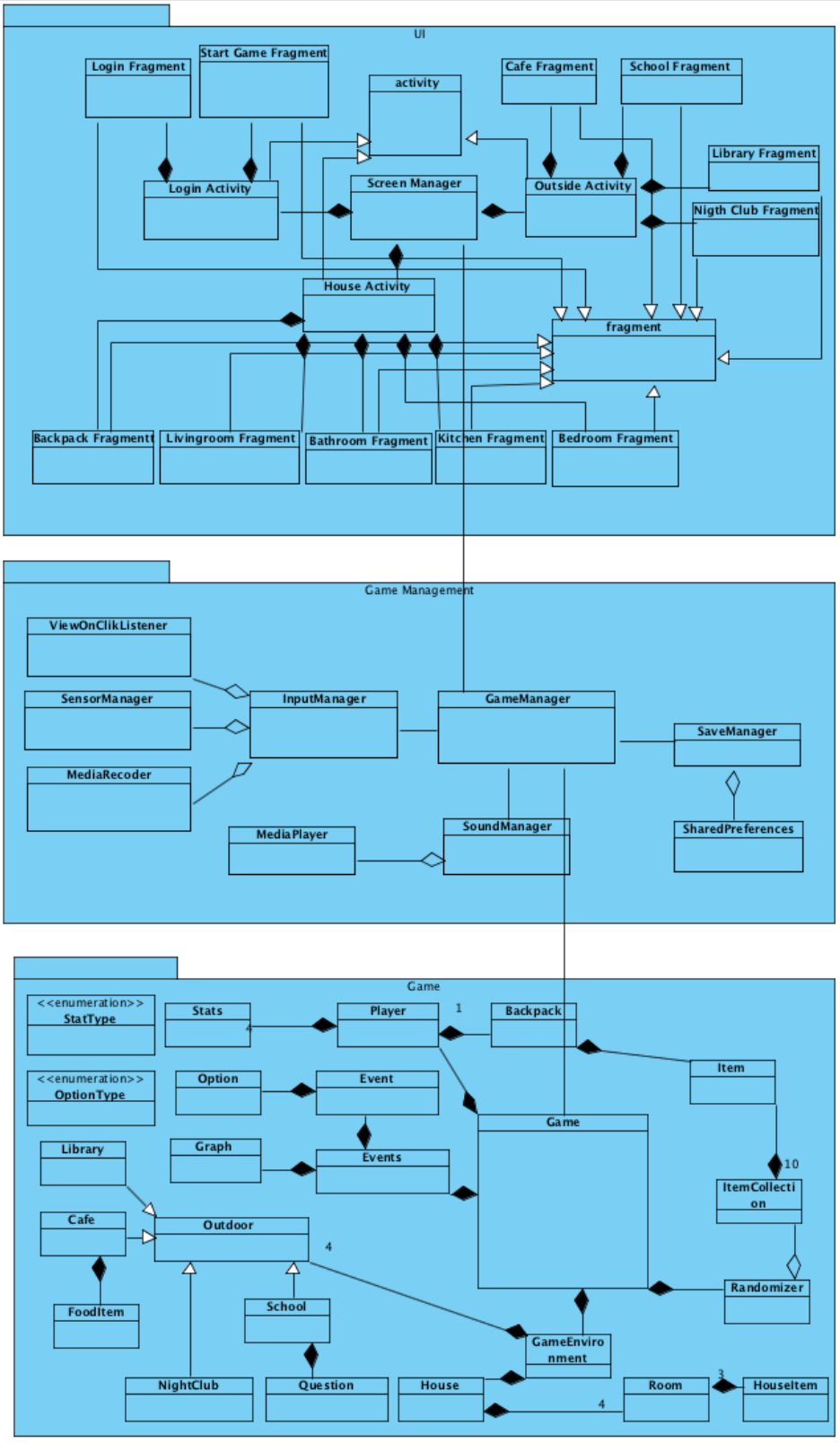


Figure 2: Detailed Subsystem Decomposition