* Player
* User
* AI
* Field
* Deck
* Active Cards
* Hand
* Card

**Player:**

The player class is the Super class of User and AI (Artificial Intelligence) classes. Both User and AI have common attributes and methods so, the parent class named as Player is created having inheritance (generalization) relationship with them. The Player class it created because there will be many player objects needs to be created in the game play. Each player will have its unique attributes and methods. The User class and AI class uses the Player class for reusability as the objects is created multiple times for different users object and AI object. So, inheritance of Player class with User and AI class is needed. The base classes (User and AI) use Player class for reusability.

**User:**

The User class is Sub class of the Player class. This class inherits the attribute and methods of its parent class named as Player class. This class is created because when a new player will start the game, its object will be created. Each User have different class object.

**AI:**

The AI (Artificial Intelligence) class is also the Sub class of the Player class. It inherits the attributes and methods from the Player class. The object of this class will be created when the user starts the game. An opponent Player class object will be created against the User class. This class is created because each user will play against the AI opponent in the game. There are certain levels in the game and different levels have different AI class objects to play against the User class object as an opponent. The difficulty levels of the AI class objects are different for different levels.

**Field:**

Field class is created to provide the platform for User and AI to play against each other. This class represents the interface where both User and AI play. The field sides of both User and AI are separate to each other. The user face different opponents of AI on different stages and each stage will have certain field where both AI and User will play against each other.

**Deck:**

The Deck class contains a collection of card class objects. The deck object is created at the start of the game. The cards are distributed one by one to the User object and AI object on each turn within the gameplay interface. The Deck object is dependent on the collection of the card objects so, the strong relation of composition is consisted between Card and Deck classes.

**Active Cards:**

The Active Cards class contains the current cards ‘on the field’ for either player, so User will have an Active Cards class or AI will have one as well. The class contains Card objects which are added and removed by the player ‘activating’ the Card object or by the object’s hp value being depleted.

**Hand:**

This class describes the User or AI class objects active cards on to attack the opponent. The Hand class contains the object of the Player to define its side. This class allows the Player to view Hand (collection of active cards) during the game.

**Card:**

This class is created as each card has its specific attribute to attack the opponent and defend itself. The object of Card class is created when a new Player object starts the game. The objects of cards are created and given to the Player to get Hand and attack the opponent. This class has a strong relation of composition with the Deck, Active Cards and Hand classes. The card object is only removed via ‘destroying it’ in the Active Cards object, as it loses its HP, by playing it from the Hand object, or by removing it from the Deck object in the Deck Manager.

**RELATIONSHIPS BETWEEN CLASSES**:

* **Inheritance (Generalization)** relationship is present between Player, User, AI and Field classes.
  + **Super Class:** Player
  + **Sub Classes:** User, AI and Field
* **Inheritance (Generalization)** relationship is present between Field, Deck and Hand classes.
  + **Super Class:** Field and Deck
  + **Sub Classes:** Hand
* **Inheritance (Generalization)** relationship is present between Field and Active Deck classes.
  + **Super Class:** Active Cards
  + **Sub Classes:** Field
* **Association** relationship is present between Hand and Active Cards classes.
* **Associate Composition** relationship is present between Deck, Active Cards, Hands and Card classes.
  + **Dependent classes:** Deck, Active Cards and Hand

**ATTRIBUTES AND METHODS:**

**Player Class:**

**Attributes:**

* **ID:** This attribute has the integer (int) datatype. It specifies the unique identifier of each player. Each player is identified by its unique ID provided to him.
* **Name:** This attribute has the string datatype. Each player object will have a name which will be displayed in the screen of the game.
* **Decks:** The Decks attribute contains a Map object of key, value type int, Deck. Map (int, Deck) where int is the unique identifier for the Deck, and it contains the selected Card objects for that unique Deck. This attribute is passed in from either the User or AI class.
* **HP:** HP is Health Points of the Player, used during gameplay
* **Mana:** Mana is Magic Points, they are used to play cards since cards have a cost in order to use them.

**Methods:**

* **newPlayer:** This method will be called by the Player class when its object is created. It passes the parameter of ID as integer.
* **selectedDeck:** selectedDeck() will actually return the Deck selected in the pre-game screen, that is, this method sends the chosen Deck object to the Field class.

**User Class:**

**Attributes:**

* **ID:** This attribute has the integer (int) datatype. It specifies the unique identifier of each user. Each user is identified by its unique ID provided to him.
* **Name:** This attribute has the string datatype. Each user object will have a name which will be displayed in the screen of the game.
* **Decks:**This attribute has the map int, Deck datatype. It is the object of Deck class which will be uses by the User Class.

**Methods:**

* **newUser:** This method will be called by the User class when its object is created. It passes the parameter of integer as ID.
* **viewDeck:** This method will be called when a player starts the game and view his selected deck. It passes the parameter of Deck object.

**AI Class:**

**Attributes:**

* **ID:** This attribute has the integer (int) datatype. It specifies the unique identifier of each AI. Each AI is identified by its unique ID provided for each level.
* **Name:** This attribute has the string datatype. Each AI object will have a name which will be displayed in the screen of the game.
* **Decks:**This attribute has the map int, Deck datatype. It is the object of Deck class which will be uses by the AI Class.

**Methods:**

* **newAI:** This method will be called by the AI class when its object is created. It passes the parameter of string.
* **viewDeck:** This method will be called when the AI object gets the turn and using this method he can select the deck. It passes the parameter of integer.

**Field Class:**

**Attributes:**

* **Sides:** This attribute has the list datatype of the players object. When the field object is created, the sides are distributed as Player side and AI side.
* **PlayerSide:** This attribute has the active cards datatype. The player side gets the active cards when the object of Field is created.
* **AISide:**This attribute has the active cards datatype. The AI side gets the active cards when the object of Field is created.

**Methods:**

* **DefeatPlayer:** This method will be called by the Field class when the player is defeated. It passes the parameter of Boolean which results either yes or no.
* **WinPlayer:** This method will be called by the Field class when the player is wins. It passes the parameter of Boolean which results either yes or no.

**ActiveDeck Class:**

It contains all of the stored cards of the selected Deck. This will be used on the Field class for sending Card objects to the Hand. It will also shuffle the contained Card objects.

**Attributes:**

* **ID:** This attribute has the integer (int) datatype. It specifies the unique identifier of each deck. Each deck is identified by its unique ID.
* **Name:** This attribute has the string datatype. Each deck object will have a name which will be displayed in the screen of the game.
* **Size:**This attribute has the integer datatype. It specifies the size of the deck.
* **Container:** The ActiveDeck class contains a map object map (int, card) in a key, value relationship with int being the unique identifier of the card, and Card being the card data.

**Methods:**

* **CreateDeck:** This method will be called by the Deck class when the player starts the game and deck containing cards will be present for the game.
* **EditDeck:** This method will be called by the Deck class when the level of the player is changed then the deck containing card will also be changed.
* **DeleteDeck:** This method will be called by the Deck class when the player ends the game and the deck is deleted.

**Active Cards Class:**

**Attributes:**

* **Active:** It contains the list of active cards in a map(int,Card). This may change to a list of type list(Card).
* **Side Player:** This attribute has the player datatype as an object of Player class. It specifies the side of the player is either User or AI.
* **Amount:** This attribute determined how many Cards are in the active slot. This tracks how many cards are contained within the ActiveCards Class, which is determined by calling map.size() or list.size().

**Methods:**

* **ViewActive:** This method will be called by the ActiveCards class when the player gets turn. It returns the Boolean values as the result of card activate or not in the turn of the player.
* **AddCard:** AddCard() is the method called when the player wants to add a card to the field, as in when they ‘play’ a Card.
* **RemoveCard:** This is the opposite of AddCard(), when a Card is ‘destroyed’ by running out of HP, then this is called.

**Hand Class:**

**Attributes:**

* **Side Player:** This attribute has the player datatype as an object of Player class. It specifies the side of the player is either User or AI.
* **Container:** The Hand class contains a map object map (int, card) in a key, value relationship with int being the unique identifier of the card, and Card being the card data.

**Methods:**

* **ViewHand:** This may or may not be used as a form of toggle for seeing the opposite Side’s Hand.
* **DrawCard:** This specifies when a Card is added to the Hand from the ActiveDeck.
* **PlayCard:** This will send the selected Card to the ActiveCards container.

**Card Class:**

**Attributes:**

* **ID:** This attribute has the integer (int) datatype. It specifies the unique identifier of each card. Each card is identified by its unique ID.
* **Name:** This attribute has the string datatype. Each card object will have a name which will be displayed in the screen of the game.
* **HP:** This attribute has the integer (int) datatype. It specifies the Health points of the card.
* **Attack:** This attribute has the integer (int) datatype. It specifies the attacking power of the card.
* **Defense:** This attribute has the integer (int) datatype. It specifies the defending power of the card.

**Methods:**

* **AttackTarget:** This method will be called by the card class to attack the opponent’s active card. It uses the parameters as hp, attack, defense, Active Cards and side player to target the values of the opponent’s active card.
* **AttackTarget:** This method will be called by the card class to attack the opponent directly. It uses the parameters as hp, attack, defense and player to target the values of the opponent.