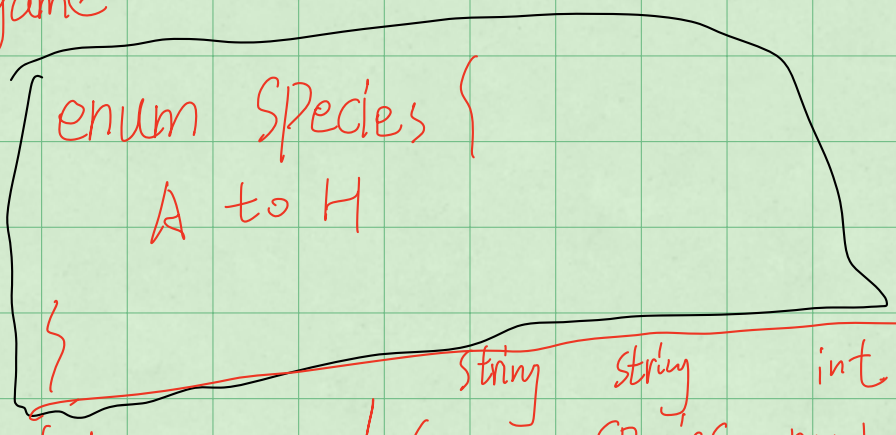
- src
  - comp1110.ass2
    - gui
      - Game
      - View
    - game
      - Species
      - Cards
      - Deck
      - DiscardPile
      - Position
      - Arboretum
      - Player
    - Event
      - Turn
      - RightToScore
      - getScore
  - Gameplay

- src
  - comp1110.ass2
    - gui
      - Viewer
      - Game
    - game



class : card(String name, String species, int number){

Constructor:{

this.name

this.species

this.number

}

Methods:

getName(){}

getSpecies(){}

getNumber(){}

}

```
class : deck (int speciesNum) {

    Constructor :{

        for species : speciesNums
            for int 1 to 8
                List<card>
                this.deck.add(New.card( ));

    }

    Methods

        getcard{
            random int from deck.length( )
            return card from.deck.pop(int)
        }

        isEmpty( )
    }
```

```
class : discardPile( ) {

    Constructor : New List<card>

    Method : getCards( )

}
```

class : position (x , y) {

Constructor : {

this.x

this.y

}

Methods

getX()

getY()

}

class : Arboretum (size)

Constructor : {

for int i size x int j size Position Card.

this.HashMap.put (position(i,j), null);

}

Methods :

addCard (Card card , position)

ifPosCanPlace(position) → Ture/False

showCurrentCards() → set of (postion, card)

showAvalidlePlace() → set of (postion)

CalculateScore(species) int score

resetMap() → after first card insert

Class: player() {

Constructor: (

this.name

this.arboretum = New Arboretum

this.discardPile = New DiscardPile

this.hand cards → list<card> = Null

}

Methods: {

getName();

getArboretum();

getDiscardPile();

getHands Card();

```
discard(card);
draw(location);
play(card, position);
}
}
```

V Event

```
class : Turn (Player, Deck){
    player.draw()
    player.draw()
    player.play()
    player.discard()
}
```

```
class : RightToScore ( players  List<player> )
    foreach player : players
        → which is higher

return { PlayerA : {S1, S2, S3...}   hashMap
         PlayerB : {S4, S5, S6}
         PlayerC : ---
         ----
       }
```

```
class: GetScore(Player, List<species>) {
        Socre = 0
        foreach specie : species {
            Socre += player.getArboretum. calculatScroe(sped)
        }
        return Socre
}
```

```
class: GamePlay(  ) {
    New Deck(G)
    New player(A)
    New player(B)
    while(!Deck.isEmpty) {
        New Turn(PlayerA, Deck);
        New Turn(PlayerB. Deck);
    }
    if (Deck.isEmpty) {
        map = RightToScore( List={PlayerA, PlayerB});
    }
    foreach i : map {
        getSore(i[0], i[1]);
    }
```

→ compare to winner !