Ashvan Wal, Student ID: A01023474, Set A
Phuong T Ho, Student ID: A01050900, Set A

ASSIGNMENT 1
OBJECT ORIENTED PROGRAMMING

1. How does your design implement the four pillars of OOP (abstraction, encapsulation, inheritance and composition, and polymorphism).

ABSTRACTION

AbstractTalent entity maintains only relevant details (attribute and methods) of a talent.

- Attributes: id, first name, last name, talent number, date debut
- Methods: get_id, set_id, get_first_name, get_last_name, get_full_name, get_talent_num, get_date_debut, get_year_debut, get_years_debut

Abstract methods (*get_details* and *get_type*) are declared without implementation.

```
@abstractmethod
def get_details(self) -> str:
    '''Return details of talent'''
    raise NotImplementedError("Must be implemented in subclass")

@abstractmethod
def get_type(self) -> str:
    '''Return type of talent'''
    raise NotImplementedError("Must be implemented in subclass")
```

The abstract methods merely define a contract that derived classes (Actor and Model) must implement.

Actor implementation:

```
def get_details(self) ->str:
    '''Returns the actor's details in a printable format'''
    if self.get_num_award() < 1:
        return '{} id {} talent number {} year debuted {} has no award'.format(self.get_full_name(),
                                                                                self.get_id(),
                                                                                self.get_talent_num(),
                                                                                self.get_year_debut())
    else:
        return '{} id {} talent number {} year debuted {} has {} awards'.format(self.get_full_name(),
                                                                                 self.get_id(),
                                                                                 self.get_talent_num(),
                                                                                     self.get_year_debut(),
                                                                                     self.get_num_award())

def get_type(self) ->str:
    '''Return actor for any object in this class'''
    return Actor.ACTOR_TYPE
```

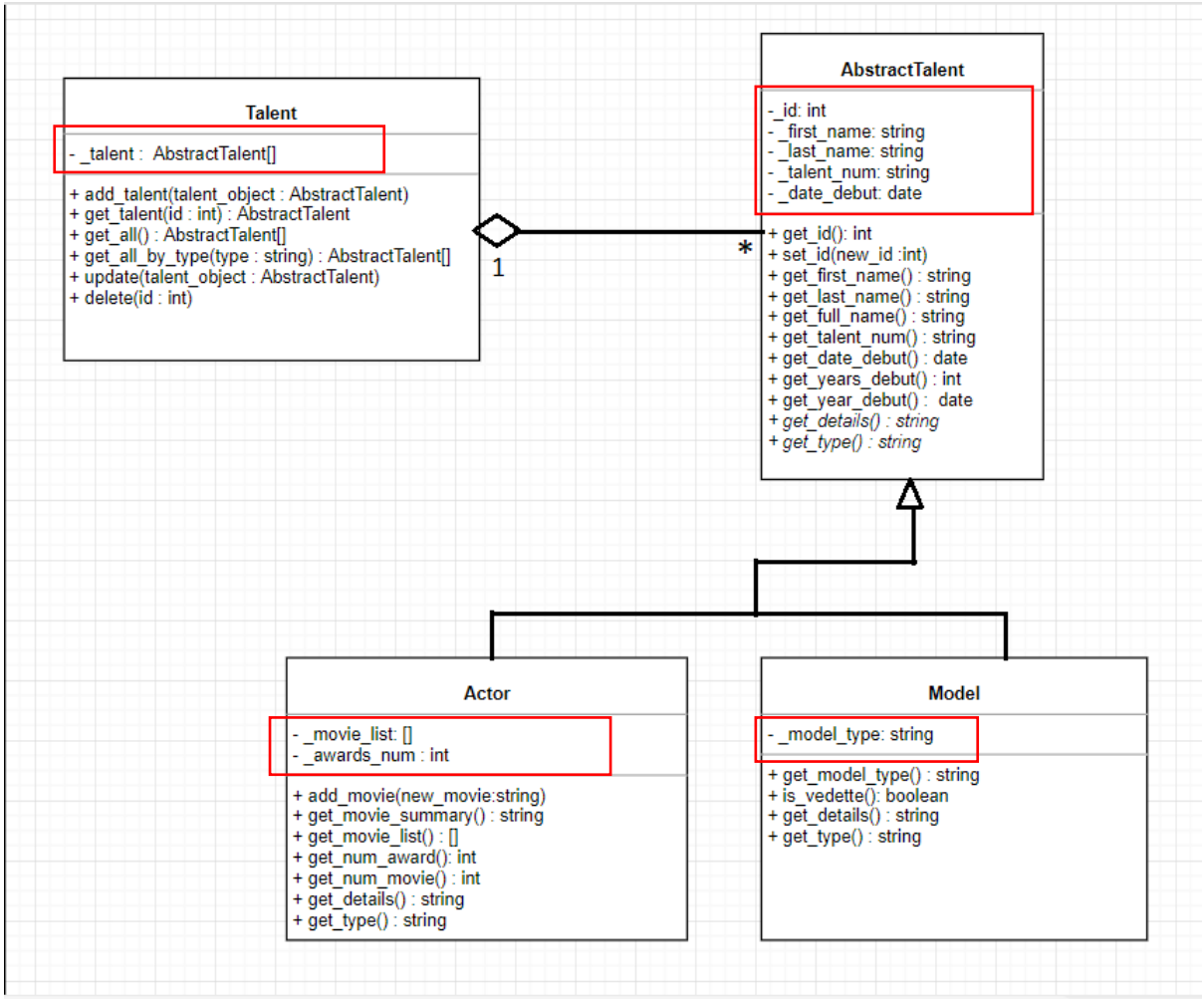
Model implementation:

```
def get_details(self) ->str:
    '''Returns the model's details in a printable format'''

    return '{} with id: {} , talent number: {}, Year debut: {} is a {} model'.format(self.get_full_name(),self.get_id(),
                                                                                     self.get_talent_num(),self.get_year_debut(),
                                                                                     self.get_model_type())

def get_type(self) ->str:
    '''Return model for any object in this class'''
    return Model.MODEL_TYPE
```

ENCAPSULATION

Encapsulation is the process of hiding implementation, functional details through public and private attributes and methods.

Attributes declared in the constructors were private as shown below. Static methods used to validate input are also private methods. By doing this, you can restrict access to methods and variables. This can prevent the data from being modified by accident.



INHERITANCE

Actor and Model are child class that inherit attributes and methods from base class AbstractTalent.

- Attributes: id, first name, last name, talent number, date debut
- Methods: get_id, set_id, get_first_name, get_last_name, get_full_name, get_talent_num, get_date_debut, get_year_debut, get_years_debut

Talent Entity maintains 'is a' relationship with Actor and Model class. A talent can be a model or an actor.

COMPOSITION

Talent Entity maintains 'has a' relationship with AbstractTalent. Talent is a class which composes instances of other classes (Actor and Model).

Example: Talent list is a list of actors and models that are instances of the Actor and Model Class.

2. Why are your entity classes good abstractions (i.e., models) of the real-world entities?

We model a talent management agency. So we think that our entity classes are good abstraction because talent management agency might demand those information for their operation and management on model and actors.

First, those classes maintain most relevant information for a talent such as names, id, employment history, etc. For each type of talent (model/actor), specific methods and attributes are designed to give talent agency the flexibility to hold specific types of information.

Second, TalentAgency class maintains general methods allows them add, remove, update, delete and get information about their talents.