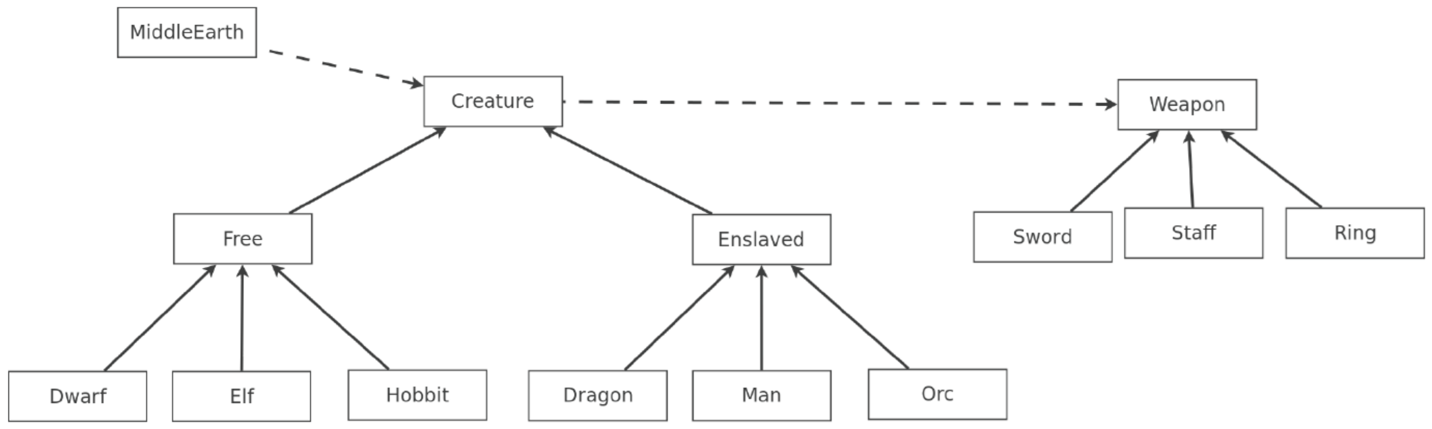
1. Fill out the following terms in your own words as they relate to the object-oriented paradigm:

Term	**Definition**	**Purpose**
Class	A blueprint for a thing; the definition of state and behavior for an entire class of things	To describe the idea of something. To describe all the things an object can be and can do
An object	An instance of a class, with specific values assigned to instance variables	To be a "physical" representation of a class. A class contains variables whereas an object contains values for those variables
Class variable	A value that is shared among all the instances of a class	When you want a value to be held by all instances of a class
Instance variable	A variable defined within a class. Each object can have a different value for their copy of the instance variable.	When you want an object to hold on to a specific value
Constructor	A special method in a class that is automatically invoked when a new instance of the class is instantiated; usually performs initialization tasks (e.g., assigning default or specified values to the instance variables).	To create and initialize instance variables
Magic Function	Functions in Python that have special meaning. They are normally called automatically by Python in the right situations (sometimes we call them explicitly).	Purpose depends on the magic function used
The "Object" class	The base class for all user-defined objects; the top-most superclass	To serve as a base super class so all classes have some baseline level of functionality (i.e. a basic set of methods that all classes inherit)
Inheritance	A relationship among classes that permits a class to inherit the state and behavior of another class; see **is-a**.	Helps with the concept of "DRY" or Don't Repeat Yourself. Inheritance allows you to specify state and behavior in a super class and reuse it in all subclasses
Accessors	A special method in a class that wraps an instance variable for the purpose of providing read access	To control access to an instance variable

Term	Definition	Purpose
Mutators	A special method in a class that wraps an instance variable for the purpose of providing write access; usually implements input validation	To control modification to an instance variable for the purpose of input validation
Super class	A class that another class inherits state and behavior from	To serve as a base class for other classes to grab functionality from (i.e. to grab state and behavior from)
Sub class	A class that inherits state and behavior from another class	Same purpose as the purpose of inheritance. This is just a class that inherits from another class so the purpose is similar to the purpose being having inheritance
Class diagram	A diagram that models the classes of a system or application, their relationships, and their members.	To visualize an entire system (or a large part of a system) by "zooming out" and viewing just the classes and relationships.

2. Write a description of the scenario whose components and relationships are captured by the class diagram below.



There are many correct answers. Here is one:

I live in middle earth. In middle earth there are creatures. A creature is either free or enslaved. Dwarfs, elves and hobbits are free whereas dragons, men and orcs are enslaved. Every creature has a weapon. A weapon is either a sword, staff or a ring.

3. List all rules you should follow when naming a class.

All class names should

- ↘ use singular nouns,
- ↘ start with an uppercase letter (and follow camel case)
- ↘ not contain spaces or other invalid characters as defined by identifier rules

4. Consider the following python program

```
1.   class Dog(object):
2.       def __init__(self, breed, owner = "pound"):
3.           self.owner = owner
4.           self.breed = breed

5.       @property
6.       def owner(self):
7.           return self._owner

8.       @owner.setter
9.       def owner(self, value):
10.          self._owner = value

11.      @property
12.      def breed(self):
13.          return self._breed

14.      @breed.setter
15.      def breed(self, value):
16.          self._breed = value

17.      def __str__(self):
18.          return "Breed = {},\tOwner = {}".format(
19.                 self.breed, self.owner)

20.  d1 = Dog("Beagle")
21.  d2 = Dog("Rottweiler", "James")
22.  d3 = Dog("Poodle")
23.  print("d1: {}".format(d1))
24.  print("d2: {}".format(d2))
25.  print("d3: {}".format(d3))
```

a) Using the line numbers specified, list the order in which the statements are executed.

```
20, 1, 2, 3, 9, 10, 4, 15, 16,
21, 1, 2, 3, 9, 10, 4, 15, 16,
22, 1, 2, 3, 9, 10, 4, 15, 16,
23, 17, 18, 12, 13, 6, 7,
24, 17, 18, 12, 13, 6, 7,
25, 17, 18, 12, 13, 6, 7
```

b) What is the output generated by the program?

```
d1: Breed = Beagle,     Owner = pound
d2: Breed = Rottweiler, Owner = James
d3: Breed = Poodle,     Owner = pound
```

For extra practice:

5. Create a class for the main player of a video game. You decide the instance variables, class variables, properties and methods the class will have. This largely depends on the type of game the player is going to be in, which is completely up to you. Feel free to think about games you have played before and try to model the player class based on those games. After creating your class, create a player object and utilize some of the methods you created.

Answer not given. Multiple answers are possible.

6. Create a class for a basic geometric shape. This shape class is going to be a base class for more specific types of shapes. Think of what all shapes can have in common and add this to the base shape class. Now create 3 derived classes (i.e. classes that inherit from the shape class). What types of simple geometric shapes are there? (e.g. triangle, hexagon, square, parallelogram, rhombus, etc.). Does any of your specific shapes have features that are not common to all shapes? If so, make sure to add those to that derived class and not to the base shape class. Finally, create an object for each one of your specific shape classes and play around with its methods and variables.

Answer not given. Multiple answers are possible.

7. Create a class for an employee of a company. Each employee of this company has an employee id number which is exactly 5 digits long; a birthday for the employee which includes a valid month, day, and year; a position title which is either 'engineer', 'lead engineer' or 'supervisor'; and a salary which is automatically calculated based on their position title - which is $50,000 for 'engineer', $65,000 for 'lead engineer', and $80,000 for 'supervisor'. Make sure to use properties for correct validation of all instance variables. Also, implement the __str__ method to display all information about an employee in a nice, easy-to-read format. Create several employee objects, some with all valid information and some with invalid information. Feel free to handle invalid information however you see fit.

Answer not given. Multiple answers are possible.