# Isaac Computer Science Student Activity Booklet

**Object Oriented Programming**

**Activity 1:** Pet object Attributes

Produce a formal definition of the pet object in the table below. You should include the data type of each attribute, and an example of the data, so that you are clear about the nature of each attribute.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Example** |
| Name | String | “Tiggy” |
| Type | String | “Cat” |
| Colour | String | “Ginger” |
| Mood | String | “Playful” |

**Activity 2:** Class Diagrams

Use your formal definition of the Cat class from Activity 1 to complete the top and middle sections of this class diagram. We will consider the bottom section, the methods, later.

**Note:** In UML, classes are represented by boxes that contain three sections:

* The top section contains the name of the class
* The middle section contains the attributes of the class
* The bottom section contains the methods of the class (the operations that the class can execute.)

|  |
| --- |
| Class Name |
| + Name: string + Type: string + Colour: string  + Mood: string |
|  |

**Activity 3:** Getter and Setter Methods

Complete the bottom section of the Pet class diagram below. You should include a getter method for each attribute and a setter only if you can justify it’s inclusion for the children’s game.

|  |
| --- |
| Pet |
| + Name: string + Type: string + Colour: string + Mood: string |
| + get\_name(): string + get\_type(): string + get\_colour():string + get\_mood():string + set\_name(string) + set\_type(string) + set\_colour(string) |

**Activity 4: Trinket:** <https://trinket.io/python3/137fad7687>

**A screenshot of a computer program

Description automatically generated**

**Task 1:** Create a new pet object, a cat called 'Romeo'

my\_object = Pet('name', 'type', ‘colour’)

**Task 2:**  Use the setter method to change it’s name to 'Juliet'

my\_object.set\_name('new name')

**Task 3:** Use the getter method to check the change has been successful

print(my\_object.get\_name())

**Task 4:** Complete the ‘describe’ method, so that it returns the pet object’s name, type and colour as a string.

return "I am a " + str(self.colour) + " " + str(self.type)\

+ " called " + str(self.name)

**Task 5:** Call the new ‘describe’ method of your pet object

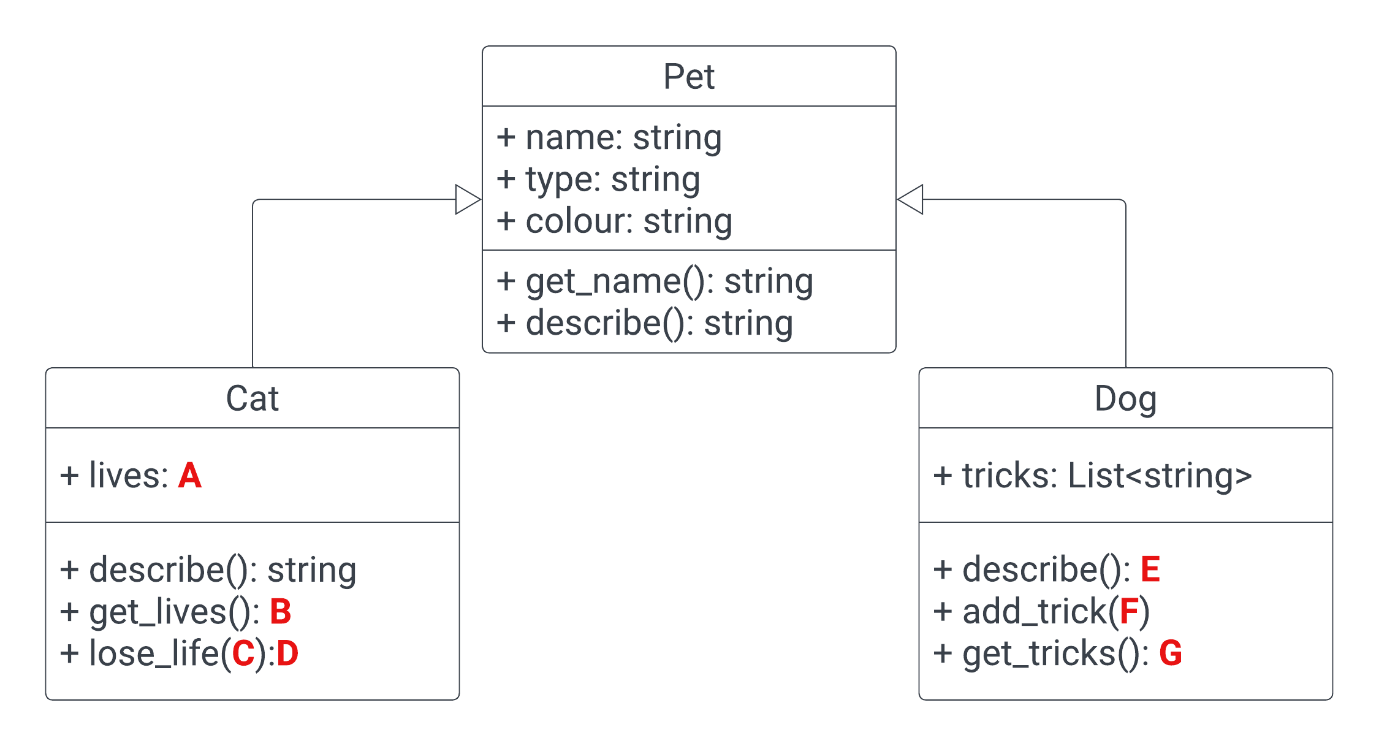
print(my\_object.describe())

**Solution:**

**A screenshot of a computer program

Description automatically generated**

**Activity 5:** Inheritance



* This UML diagram depicts the relationship between the superclass Pet and the subclasses Cat and Dog. Missing from the diagram are the data types for some of the attributes, method parameters and return values. Enter the missing values in the table below.

|  |  |
| --- | --- |
|  | **Data Type** |
| A | integer |
| B | integer |
| C | None |
| D | None |
| E | string |
| F | String |
| G | List<string> |

**Activity 6:** Super & Subclasses

A computer screen shot of text

Description automatically generated

1. Study the class definitions for the Pet class (line 1) and the Cat class (line 15). How can you determine, which is the superclass and which the subclass?  
   * The Cat class definition is on line 15: **class** **Cat(Pet):**
   * Here, Cat is defined with (Pet) next to its name. This means that the Cat class is inheriting from the Pet class.
   * The class mentioned inside the parentheses (Pet) is the superclass from which the Cat class inherits attributes and methods. In this case, Pet is the superclass.
   * The class that is doing the inheriting, which is Cat in this instance, is the subclass.
2. In the constructor of the Cat class, there is call to the constructor of the Pet class (line 18). Why is this necessary?

* Calling the Pet class's constructor from the Cat class's constructor ensures that the Cat object is properly initialised with all the attributes and methods it inherits from the Pet class, while also allowing for the addition of attributes and methods specific to the Cat class.

1. "If the describe() method exists in both the Pet class and the Cat subclass, it represents method overriding, a type of polymorphism. What would be the differing outputs when invoking the describe() method on instances of the Pet and Cat classes?"

* When the describe() method is called on a Pet object, the output would be in the format:

**I am a {colour} {type} called {name}**

* For instance, if we have a Pet object with attributes: colour="blue", type="bird", and name="Tweety", the output would be:

**I am a blue bird called Tweety**

* However, when the describe() method is called on a Cat object, there is an additional line of output at the beginning. The output would be in the format:

**Meow, I am a {mood} {colour} {type} called {name}**

* For instance, if we have a Cat object with attributes: mood="playful", colour="white", type="cat", and name="Snowy"

**Meow,I am a playful white cat called Snowy**

**Activity 7:**  Trinket: <https://trinket.io/python3/10c64b72c4>

Use the **Activity 7** trinket or **activity7.py** python file to complete the following tasks:

1. Scroll down to the bottom of the code (**line 38**)
2. Check if the pet\_type entered by the user is "cat" (case insensitive).
3. If it's a cat, create a new instance of the Cat class with the provided pet\_name and pet\_colour.
4. Then call the describe() method of the Cat object. This will invoke the describe() method of the Cat class and not the parent Pet class due to method overriding.
5. Next, call the get\_lives() method and print the current number of lives the cat has.
6. Call the lose\_life() method to decrement the cat's lives by one.
7. Finally, call the get\_lives() method again and display the updated number of lives.

**Solution:**

A computer code with text

Description automatically generated

A screenshot of a computer

Description automatically generated