Object Oriented Programming In Python

Activity Sheet

Week 4 The 20Q Example

1 Introduction

The 20Q example illustrates the use of inheritance with overriding of method. This section introduces the 20Q game.

1.1 The 20Q Game

The 20 questions game is a classic illustration of computational thinking. One player chooses a secret object – say a 'toaster' or 'panda'. The other player now has 20 questions to guess the object. The questions usually have only yes / no answers (though the game starts with a special question – “animal, vegetable or mineral?”).

How is this possible? The best algorithm is to make sure that the question divides the set of possible answer approximately in half. So you start with general questions such as "do you find one at home" rather than “do you find it in the kitchen”. With this algorithm, 20 questions can distinguish up to 1 million different objects.

In this version of the game, the computer has:

- a fixed set of objects (many fewer than a million);
- a fixed set of questions with an answer for each object

The game is played as follows:

- the computer chooses on of the objects – a secret
- the computer displays the list of possible objects and the questions
- the player chooses one of the questions and the computer gives the answer
- after a small number of questions, the player must guess the secret object. If the guess is wrong, the correct answer is displayed

1.2 Choosing a Topic

This simple version of 20Q works best when the objects from a specified general topic or area. The example code is for animals. The organisation of the code does not depend very much on the area chosen.

2 Task 1: Review the Provided Code

Run the provided program and read the code carefully. Answer the questions below:

2.1 Classes and Class Diagram

The provided program has three classes. What are these? Draw the class diagram.

2.2 Overriding and Inheritance

Complete the following table that shows the methods and classes:

Method	Classes		
	Animal	Mammal	
hasFur	Implemented	Overridden	
canSwim	Implemented		
getSize	Implemented	Inherited	
getLegs	Implemented		

2.3 Comparing Legs and Swimming

The different Animals must all differ. There are two ways that are used to do this in the code:

- The animals belong to different subclasses of Animal.
- The class constructors have parameters used to initialise the attributes of the class.

Explain the way that:

i. The function canSwim() returns a different value for a Tiger and a Dolphin
ii. The function getSize() returns a different value for a Tiger and Badger.

3 Task 2: getDescription Function

When the game ends, it would be good to print out a full description of the secret object. For example, it might print:

```
The Tiger has the following characteristics:
  It cannot swim; it does not have fur
  It has 4 legs; it is Large; it eats meat
```

Implement this feature in the following steps:

3.1 Step 1: Which Class

Decide which class or classes need to be modified to add this functionality, given that it behaves differently for different animals.

3.2 Step 2: Implementation

Implement and test this function.

4 Task 3: Adding More Animals

The game would be more fun with more animals and more questions. Following the steps to add more of both; each step (except the first) incudes some coding – remember to test as you go. You may also add the animals a few at a time.

4.1 Step 1: List the new Animals

Write a list of possible new animals and then suggest additional subclasses.

Bee – insect, ….

Make a substantial list at this stage; we will add them little by little. Redraw the class diagram.

4.2 Step 2: More Functions like hasFur

Suggest other functions that could be used to distinguish between the now larger set of animals.

Implementation:

i. Add the new functions to the Animal class
ii. Override the new functions in the existing subclasses as necessary
iii. Add the new subclasses, overriding the new and existing functions as necessary.

Testing: the program should still run. At the command line, you can call the new functions on animals from the list allAnimals.

4.3 Step 3: More Attributes

Do you have enough information to distinguish between the animals you plan to add, once they are allocated to the different classes? If not, you may wish to add more attributes that are initialised in the constructor.

Make a list of these attributes here:

Implementation:

i. Add the new attributes to the constructors of the existing and new classes: if the value is the same of all members of a subclass, then no parameter is needed in the subclass for this attribute.
ii. Modify the existing constructors calls (for Horse, Badger, Tiger etc) to include the appropriate values of the new attributes.
iii. Add 'get' functions to the top class (Animal) for the attributes.

Testing: the program should still run. At the command line, you can call the new 'get' functions on animals from the list allAnimals.

4.4 Step 4: More Questions

More questions are needed to distinguish the large set of animals. It must be possible to answer each questions for all the animals using:

- The methods implemented for each class (step 2)
- The attributes 'get' functions (step 3)
- A combinations of these (not done at present)

Implementation:

i. Add the text of the question to the list 'questions'.
ii. Add code to answer the question in the function 'answerQuestion'.
iii. Now that the list of question is longer, the range checking in the use interface code is in correct: fix this but also improve the code by removing the literal '9'. Can you suggest how to avoid having to make this change every time the list of questions grows?
iv. You may wish to increase the number of questions the player can get answered before having to make the guess. Again, avoid literals as far as possible.

Testing: the program should run and the new questions should be offered for the existing animals.

4.5 Final Step: The New Animals

Add the animals to the function 'animals'.

Testing: test the completed program thoroughly.

5 Task 4 (Advanced): More Objects

The 20Q program uses a hierarchy of classes for the animals, but other parts of the program make less use of OOP. The aims of introducing more classes are a) maximise the reuse of code by generalising it b) managing complexity by separating the code into separate classes. A version of the program is available that uses OOP throughout:

- The main program is much shorter – most of the code is in one of the classes.
- New classes have been introduced for Topic and Question.
- A subclass of Topic is a specific topic: for example, the AnimalTopic is about animals.

Run the program (open 20Q.py).

Complete the following table showing the classes (each is in a separate file).

Class/File	Description	Imports
Animal.py		
Mammal.py		
MarineMammal.py		
20Q.py	The main program. The program is run from here.	

Draw the class diagram.

Notes: Design is subjective: do you like this design?