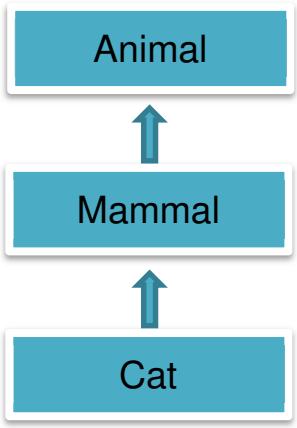
Apu Animal Park – Version 1

1 General

Before you start programming, take a good amount of time to think out a good design for your solution. Model your solution using a class diagram, or a sketch on a piece of paper making notes of required classes and relations between them (aggregation or inheritance). You can also use the Class Diagram template in Visual Studio (Project – Add – New Item and then select Class Diagram).

Begin writing the class **Animal**. In this class, define instance variables (e.g. name, age) and methods that are common to all animals. Define some methods that all animal objects should use, properties connected to instance variables, constructors, and so on.



Create a **Mammal** class that inherits all data and operations from the **Animal** class and extends it by defining more data and more features that are particularly related to the **Mammal** category. The Mammal class should only have fields and members that are specific to mammals and that are not already existing in the Animal class.

Likewise, you can create Bird, Insect and Reptile classes all derived from the Animal class.

Back to mammals, write a class **Dog** which inherits the **Mammal** class but then it handles all about dogs, a special type of mammal. You can proceed to define classes for other species like Cat,, Lion, as sub-classes to the Mammal class. A **Cat** object "is a " mammal which in turn "is an" Animal object.

Continue this pattern for the other categories too. When you have the basic structure there, then think about the GUI and how you can exchange data back and forth between the GUI (**MainForm**) and your objects, i.e. saving input to your objects and retrieving data to display on the GUI (showing result or updating the GUI).

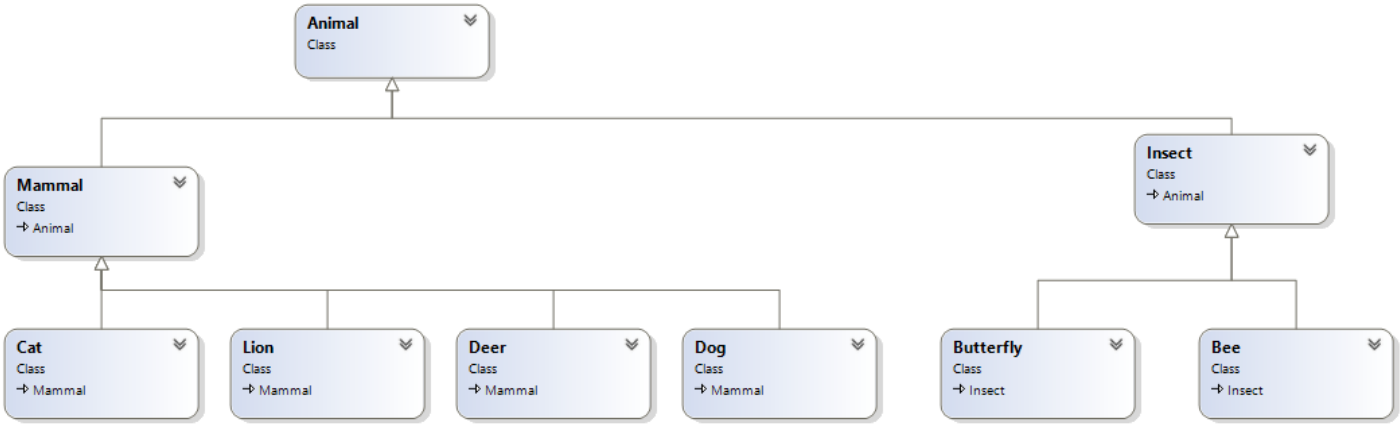
More specifically, if the user wants to register an object of a Cat (say the cat Tom is to be hosted in the Park), you would need to do the following:

- Create an object of Cat.
- Read values that are general, name, age, gender from controls on the GUI and save them in the variables you have reserved for them in the object's Animal part (variables defined in the Animal class).
- Read values from GUI pertaining to the mammals (data defined in the Mammal class).

- Read values from GUI that are specifically related to the Cat object (variables defined in the Cat class).

Tom has its special data but it also has data that come all the way from the **Animal** part. The object **tom** is registered in the system with all its data saved in the object. In much the same way, you can save another object of Cat or sveral other objects of Cat.

The following class diagram may serve as a guide to structuring your classes.



The enumerations **CategoryType** and **GenderType** may also be useful and make life easier.

Category.cs

```
public enum CategoryType
{
    Bird,
    Insect,
    Mammal,
    Marine,
    Reptile
}
```

Gender.cs

```
public enum GenderType
{
    Female,
    Male,
    Unknown
}
```

2 Dynamic binding.

You can always create an object of any type using the type directly, as in the code below:

```
Cat tom = new Cat(…Constructor parameters, if any);
```

The above code binds the object created by the statement at the right of the equal sign to the reference variable **tom** (of the type Cat) at the left of the equal sign. This binding is static as the

compiler can perform the binding at compile time. Tom is a **Cat**, a Cat is a **Mammal**, a Mammal is an **Animal** and this way we have an inheritance hierarchy. The object **tom** can access all the public (and protected) members of both the **Mammal** and the **Animal** classes.

```
tom.Name = "Tommy";      //Name defined in the Animal class
tom.NumOfTeeth = 38;     //NumOfTeeth defined in the Mammal class
tom.Color = Color.White; //Color defined in the Cat class
```

If we now create another object, like Dog, Butterfly or Dove, we have to repeat the same code again for new objects.

To minimize the amount of code by not repeating same or similar statements, we can use polymorphism and dynamic binding. Declare a variable of the type Animal, handle data for the special type of animal (e.g Cat, Dog); then use same code for data for all mammals and finally for all animals.

- Declare a reference variable of the type Animal
  `Animal animal = null;`

- Create the selected concrete object (Cat, Dog, Falcon, etc.)
  `animal = new Dog(…);`

- Save the object-specific data (saving in the object, instance variables in the Dog class)
  `((Dog) animal).Breed =` `data from GUI`

- Save data for the selected category, (saving in instance variables in the Mammal class)
  `animal.TailLength = data from GUI`

- Save data that are common to all animals(saving in instance variables in the Animal class)
  `animal.Name = …data from GUI`
  `animal.Age = data from GUI`

Through dynamic binding, you can write code to create objects dynamically at run time based on the user's choice. In the code snippet below, **animalObj** is declared as an Animal type (compile-time), but based on the user's option, it is going to refer to an object of a Dog or a Butterfly, etc. at run-time.

CreateMammal är a static method in the class MammalFactory that returns an object of mammals, e.g. Dog, Cat

```
private void AddAnimal()
{
    Animal animalObj;

    CategoryType animalCategory = GetSelectedCategory();

    switch (animalCategory)
    {
        case CategoryType.Mammal:
        {
            MammalSpecies animalSpecie = (MammalSpecies)Enum.Parse(typeof(MammalSpecies), lstAnimals.Text);
            animalObj = MammalFactory.CreateMammal(animalSpecie);   //Late binding - animalObj is now dog,cat, etc.
            break;
        }
```

3 Type-casting to access members of sub-classes

The reference variable **animal** being declared as an object of the class Animal, cannot access (or see) members in the sub-classes. We need to direct the compiler to such a member by an explicit type-casting:

```
int teeth = 0;
if (int.TryParse(txtInput1.Text, out teeth))
{
    ((Mammal)animalObj).Teeth = teeth;
```

If you want to know which object you are working with at a certain time, a simple way is to use **typeOf** as shown below, although there are other (better) ways to accomplish the same thing (**animalObj** is declared as a type of **Animal** and **Dog** is the class name):

```
if (animalObj  is Dog)
{
    animalObj.ExtraAnimalInfo = ((Dog)animalObj).GetAnimalSpecificData();
}
```

Well, discuss other questions in the forum and share your tips and hints with your classmates. Have durability and maintainability in mind and don't forget our motto of this course:

We work Strongly Object-oriented and Well-documented!

Good Luck!

Farid Naisan,

Course Responsible and Instructor