Bunny class

Bird class

Frog class

Crocodile class

A Computer Game is based on Animals.

`A Bunny IS-A Animal.`

`A Frog IS-A Animal.`

`A Bird IS-A Animal`

`A Crocodile IS-A Animal`

Let's take a look at the `Bird` class:

```
public class Bird{
  //class properties/attributes
  private String name;
  private int score;
  //constructor method
  public Bird(String name){
    this.name = name;
    this.score = 0;
  }
  //accessor methods
  public String getName(){
    return this.name;
  }
  public int getScore(){
    return this.score;
  }
  //mutator methods
  public void setScore(int points){
    this.score = this.score+points;
  }
}
```

Let's also take a look at the `Frog` class.

```
public class Frog{
  //class properties/attributes
  private String name;
  private int score;
  //constructor method
  public Frog(String name){
    this.name = name;
    this.score = 0;
  }
  //accessor methods
  public String getName(){
    return this.name;
  }
  public int getScore(){
    return this.score;
  }
  //mutator methods
  public void setScore(int points){
    this.score = this.score+points;
  }
}
```

Compare the two classes.

What do you notice about the attributes and methods of each class?

Are there any differences between the classes?

Write your notes here:

CODING TASK 1

In your IDE, start a new project and create the Bird and Frog classes.

Construct an instance of[1] a `Bird` and a `Frog` in a main class
Write the 2 lines of code here.

-
-

Write more code to display the 2 Animals' names.

[1] instantiate

We can imagine that because a Crocodile **IS-A** Animal, then the design of the `Crocodile` class will be very similar to the design of other kinds of `Animals`.

CODING TASK 2

In your IDE, design the `Crocodile` class.

ISSUE

So, the game designers have decided that every `Animal` requires a new behaviour - `shout`!

This method takes a boolean parameter. If the parameter is true, the Animal shouts its name. If the parameter is false, the Animal shouts "I'm busy!"

CODING TASK 3

In your IDE, code the `shout` for the following classes:

- Frog
- Bunny
- Crocodile

In your Main file, make any of your animals shout!

ISSUE

So when the program designers need to make changes to one of the classes, they need to make the same change to all of the classes. This is:

- time-consuming
- introduces the risk of error
- Can you think of another disadvantage of this approach?

A SOLUTION - INHERITANCE

One solution to the above issue is to take advantage of **INHERITANCE**.
Using inheritance, a **superclass** can be created which defines all Animals.

We will identify all properties and methods which are **common** to all types of Animal and **encapsulate** them in a class called `Animal`.

```
public class Animal{

  //class properties/attributes

  private int name;

  private int score;

  //constructor method

  public Animal(String name){

    this.name = name;

    this.score = 0;

  }

  //accessor methods

  public String getName(){

    return this.name;

  }

  public int getScore(){

    return this.score;

  }

  //mutator methods

  public void setScore(int points){

    this.score = this.score+points;

  }

}
```

[2] A Frog or An Animal?!

CODING TASK 4

In your IDE, code the `Animal` superclass. Don't forget to include the shout method!

Now, we need to **redesign** the other classes as subclasses of this super class. This will help explain that:

```
A Bunny IS-A Animal.

A Frog IS-A Animal.

A Bird IS-A Animal

A Crocodile IS-A Animal
```

In Java, we use the keyword **extends** to implement this relationship:

```
public class Bird extends Animal{
  //class properties/attributes
  //constructor method
  public Bird(String name){
    //to be updated
  }
  //accessor methods
  //mutator methods
}
```

Note that all of the attributes are described in the superclass - we no longer need to list them in subclasses.

CODING TASK 5

In your IDE, redesign all of the subclasses. At the moment all we need is a constructor method. All **properties** and other **methods** will be handled by the **superclass**.

ISSUE

Note that when we want to create an instance of a Bird, we give it a String as a parameter eg:

```
Bird tweeter = new Bird("Tweeter");
```

But look at the redesigned `Bird` class. Find the **constructor** method above. Currently there is no way to process this parameter. We **can't** do this like before:

```
public Bird(String name){
      this.name = name;
}
```

because the `name` property no longer exists in the `Bird` class. It is handled by the superclass's **constructor** method

ie if you look at the `Animal` superclass, this is where the `name` property will be processed.

So, in the `Bird` constructor, we have to send this parameter to the super constructor, like this:

```
public class Bird extends Animal{
  //class properties/attributes
  //constructor method
  public Bird(String name){
    super(name);
  }
  //accessor methods
  //mutator methods
}
```

Task
Try it! Can you still construct a Bird, display its name and make it shout?

ISSUE

The superclass **encapsulates** data and methods related to Animals.

The properties are **private**.

Can **subclasses** directly access private properties of a **superclass**?

For example, can we do this in another class:

```
Bird bird = new Bird("Tweety");

System.out.println(bird.name);
```

Or do we still need to use **accessor/mutator methods**, even though the classes have an **IS-A** relationship?

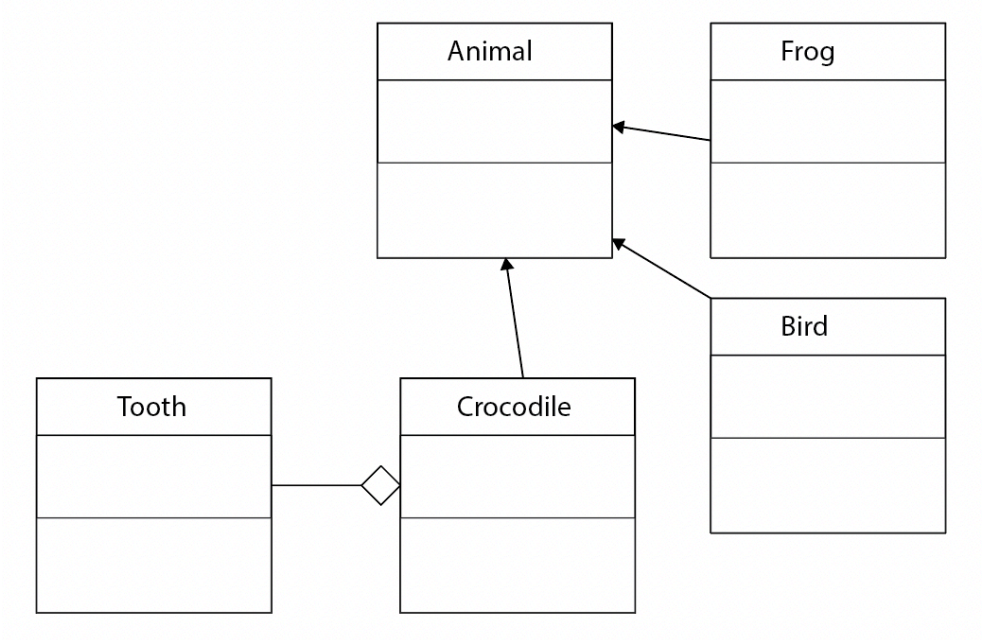
Write your findings here:

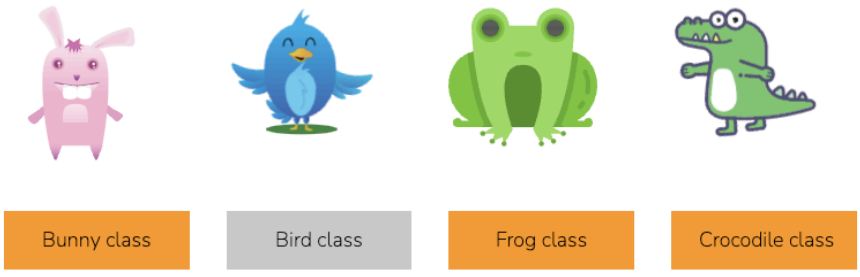
Some Advantages of Inheritance

- It promotes **code reuse** because the superclass holds common data and actions which speeds up development time;
- It **reduces maintenance time/cost** because you only have to update the superclass
- Can **speed up development time** because superclass will (hopefully) already have been tested
- We have the ability to extend a class without having to create completely new classes, saving development time

A note on UML Diagrams

Similar to aggregation (HAS-A), we can demonstrate inheritance (IS-A) relationships between classes in UML diagrams. Just use an arrow pointing from the subclass(es) to the super class.





What a bunch of Animals!!