Assignment 3 – 21325456 – Cathal Lawlor

Description

We learn about inheritance in this assignment.

In this case, the bird and fish extend off of animal,

where canary and ostrich extend off the former and trout and shark off the latter.

This stops a lot of duplication of code.

We have a toString method in each class to print out all the data it stores.

There's also an equals method to compare two classes to see if they are the same.

The test class then just creates two objects which do the following;

Test1 - creates an array of the animal objects and then prints off their corresponding info.

Test2 - creates an array of various animals, some duplicates.

It then uses the equals method in each class to compare them to other classes to find duplicates.

It prints the location, name and class type to the screen then.

AnimalTest

```
public class AnimalTest
{
   // instance variables - replace the example below with your own
     /**
    * Constructor for objects of class AnimalTest
    */
   public static void main(String[] args)
   {
     AnimalTest test = new AnimalTest();
     test.test1(); // calls the method holding our first test scenario in animalTest object
     test.test2(); // calls the method holding our second test scenario in animalTest object

   }

   public void test1()
   {
      System.out.println("\n---------- \n");
      System.out.println("First test case\n");
      System.out.println("Just an array showing our various leaf animals and their attriubtes they've inherited.");
      System.out.println("\n---------- \n");

      Animal[] animals = new Animal[4];

      animals[0] = new Ostrich("Graham");
      animals[1] = new Canary("Damon");
      animals[2] = new Shark("Alex");
      animals[3] = new Trout("David");



      for (int i = 0; i < animals.length; i++) {
```

```
        System.out.print(animals[i]); //printing out the animals toString methods
    }
}

public void test2() {
    System.out.println("\n\n---------- \n");
    System.out.println("Second test case\n");
    System.out.println("Using arrays to compare all of our animal objects");
    System.out.println("\nEmploying the equals method to compare them.");
    System.out.println("\n---------- \n");

    // New animal objects
    Animal[] animals = new Animal[11]; //an array holding lots of the animals with three pairs / matches
    animals[0] = new Trout("Evan");
    animals[1] = new Canary("Roisin");
    animals[2] = new Canary("Ciara");
    animals[3] = new Trout("Rob");
    animals[4] = new Shark("Bob");
    animals[5] = new Shark("Bob");
    animals[6] = new Ostrich("Finn");
    animals[7] = new Ostrich("Bill");
    animals[8] = new Canary("Jeb");
    animals[9] = new Canary("Jeb");
    animals[10] = new Ostrich("Bill");



    for (int i = 0; i < animals.length; i++) { // Goes through all animal objects comparing them
        System.out.printf("\nComparing using %s the %s ", animals[i].getName(), animals[i].getInstanceName() );
        int y = i; // y = i, as it there's no point backtracking on yourself checking already checked animals
        for (y = 0; y < animals.length; y++) {
            if (animals[i].equals(animals[y]) && i != y && !(y < i)) { // Comparing the animal[i] to all of the other animals between positions i + 1 and 9 - the rest of the conditions is to stop double counting
                // Printing out a match
```

```
            System.out.printf("\n%s the %s with the position %d in animal array matched with %s the %s with the position %d \n", animals[i].getName(), animals[i].getInstanceName(), i, animals[y].getName(), animals[y].getInstanceName(), y);

        } /*

          ** For if the two objects don't match - not using it as it fills up the screen with spam

          else {

          System.out.printf("%s the %s in position %d doesn't match %s the %s in position %d\n", animals[i].getName(), animals[i].getInstanceName(), i, animals[y].getName(), animals[y].getInstanceName(), y);

        }

          */


      }

    }

  }

}
```

Test1 output

```
-----------------

First test case

Just an array showing our various leaf animals and their attriubtes they've inherited.

-----------------

Canary;
Name: Damon; colour: yellow
Do I breathe?: true
Do I have skin?: true
Do I eat?: true
Do I have feathers?: true
Do I have wings?: true
Listen to me sing: tweet tweet tweet
Do I fly?: true
I fly 40 metres

Ostrich;
name: Graham; colour: yellow
Do I breathe?: true
Do I have skin?: true
Do I eat?: true
Do I have feathers?: true
Do I have wings?: true
Listen to me sing: gawk gawk gawk
Tall?: true
Leg type?: long thin legs!
Do I fly?: false
I can walk: 10 metres

Shark;
name: Alex; colour: black
Do I breathe?: true
Do I have skin?: true
Do I eat?: true
Do I have gills?: true
Do I have gills?: true
Dangerous: true
Do I bite you?:true
Look at me, I swam 500 metres!

Trout;
name: David; colour: brown
Do I breathe?: true
Do I have skin?: true
Do I eat?: true
Do I have gills?: true
Do I have gills?: true
Am I edible?: false
Do I have spikes?: true
How are new trout created?: I swim upriver to lay eggs.
Look at me, I swam 200 metres!
```

Test 2 Output

```
----------------

Second test case

Using arrays to compare all of our animal objects

Employing the equals method to compare them.

-----------------


Comparing using Evan the Trout
Comparing using Roisin the Canary
Comparing using Ciara the Canary
Comparing using Rob the Trout
Comparing using Bob the Shark
Bob the Shark with the position 4 in animal array matched with Bob the Shark with the position 6

Comparing using Finn the Ostrich
Comparing using Bob the Shark
Comparing using Bill the Ostrich
Bill the Ostrich with the position 7 in animal array matched with Bill the Ostrich with the position 10

Comparing using Jeb the Canary
Jeb the Canary with the position 8 in animal array matched with Jeb the Canary with the position 9

Comparing using Jeb the Canary
Comparing using Bill the Ostrich
```

Canary class

```
public class Canary extends Bird
{

   /**
    * Constructor for objects of class Canary
    */
   public Canary(String name)
   {
      super(); // call the constructor of the superclass Bird
      //Name & Colour inherited from animal class - Values assigned now override the inherited value
      this.name = name;
      colour = "yellow"; // overrides the value assigned in bird that's been inherited
      canSing = true;
      hasFeathers = true;



   }



   /**
    * Sing method overrides the sing method
    * inherited from superclass Bird
    */
   @Override // good programming practice to use @Override to denote overridden methods
   public String sing(){
      String temp = "tweet tweet tweet"; //canary singing – again overriding bird
      return temp;

   }

   @Override //this is for returning the instance type
   public String getInstanceName(){
```

```
    String classTemp = "Canary";
    return classTemp;
}



/**
 * toString method returns a String representation of the bird
 * What superclass has Canary inherited this method from?
 *
 */
@Override
public String toString(){
    String strng ="";
    strng+= "Canary;\n";
    strng+= "Name: ";
    strng+= name;
    strng+= "; ";
    strng+= "colour: ";
    strng+= colour;
    strng+= "\n";
    // TOD0 Your job is to include the fields and attributes inherited
    //from Bird and Animal in the String representation


    strng += "Do I breathe?: ";
    strng += breathes();
    strng += "\nDo I have skin?: ";
    strng += hasSkin();
    strng += "\nDo I eat?: ";
    strng += eats();
    strng += "\nDo I have feathers?: ";
    strng += hasFeathers();
    strng += "\nDo I have wings?: ";
    strng += hasWings();
```

```
    strng += "\nListen to me sing: ";
    strng += sing();
    strng += move(40);
    strng += "\n\n";



    return strng;
}



/**
 * equals method defines how equality is defined between
 * the instances of the Canary class
 * param Object
 * return true or false depending on whether the input object is
 * equal to this Canary object
 */

@Override
public boolean equals(java.lang.Object object){

    // Checking did we get given any object.
    if (object == null) {
        System.out.print("NULL object given\n");
        return false;
    }

    // instanceof checking if the given object is the same type, otherwise the object cannot be casted
    if (object instanceof Canary) {
        // Casting given object & running checks if all the details are the same as our existing object
        Canary canary = (Canary) object;
```

```
        //in this if statement I'm not checking everything as it would look messy, e.g. Lets assume that both birds breathe (hopefully)

        if (this.getName() == canary.getName() && this.getColour() == canary.getColour() && this.hasFeathers() == canary.hasFeathers()) {

            return true;

        }

    }


    return false;

  }

}
```

Ostrich Class

```
public class Ostrich extends Bird
{

   //String name; // the name of this Ostrich
   String legType;
   boolean isTall;

   /**
    * Constructor for objects of class Ostrich
    */
   public Ostrich(String name)
   {
      super(); // call the constructor of the superclass Bird
      //Name & Colour inherited from animal class - Values assigned now override the inherited value
      this.name = name;
      canSing = true;
      colour = "yellow"; // this overrides the value inherited from Bird
      //Fliesinherited from animal class
      legType = "long thin legs!";
      flies = false;
      isTall = true;
      distance = 10;
   }

   /**
    * Sing method overrides the sing method
    * inherited from superclass Bird
    */
   @Override // good programming practice to use @Override to denote overridden methods
   public String sing(){
      String temp = "gawk gawk gawk";
      return temp;
   }
```

```
@Override
public String getInstanceName(){ //this is for returning the instance type
   String classTemp = "Ostrich";
   return classTemp;
}

public boolean isTall(){
   return isTall;
}

public String legType() {
   return legType;
}



/**
 * toString method returns a String representation of the bird
 * What superclass has Ostrich inherited this method from?
 *
 */
@Override
public String toString(){
   String strng ="";
   strng+= "Ostrich;\n";
   strng+= "name: ";
   strng+= name;
   strng+= "; ";
   strng+= "colour: ";
   strng+= colour;
   strng+= "\n";



   strng += "Do I breathe?: ";
```

```
    strng += breathes();
    strng += "\nDo I have skin?: ";
    strng += hasSkin();
    strng += "\nDo I eat?: ";
    strng += eats();
    strng += "\nDo I have feathers?: ";
    strng += hasFeathers();
    strng += "\nDo I have wings?: ";
    strng += hasWings();

    strng += "\nListen to me sing: ";
    strng += sing(); //hehe

    strng +="\nTall?: ";
    strng += isTall;

    strng += "\nLeg type?: ";
    strng += legType();

    strng+= move(distance);
    strng += "\n\n";



    // TOD0 Your job is to include the fields and attributes inherited
    //from Bird and Animal in the String representation
    return strng;
}



/**
 * equals method defines how equality is defined between
 * the instances of the Ostrich class
 * param Object
 * return true or false depending on whether the input object is
```

```
     * equal to this Ostrich object
     */

    @Override
    public boolean equals(java.lang.Object object){

        // Checking did we get given any object.
        if (object == null) {
            System.out.print("NULL object given\n");
            return false;
        }

        // instanceof checking if the given object is the same type, otherwise the object cannot be casted
        if (object instanceof Ostrich) {
            // Casting given object & running checks if all the details are the same as our existing object
            Ostrich ostrich = (Ostrich) object;

            //in this if statement I'm not checking everything as it would look messy, e.g. Lets assume that both birds breathe (hopefully)
            if (this.getName() == ostrich.getName() && this.getColour() == ostrich.getColour() && this.isTall() == ostrich.isTall()) {
                return true;
            }
        }

        return false;
    }
}
```

Fish Class

```
public abstract class Fish extends Animal
{
   //instance variables (fields) - inherited by fish subclasses
   boolean hasFins;
   boolean hasGills;



   /**
    * Constructor for objects of class Fish
    */
   public Fish()
   { //all the subclasses of Fish inherit these properties and values
      super(); //calls the constructor of its superclass  - Animal
      colour = "black"; //overrides the value of colour inherited from Animal
      hasFins = true;
      hasGills = true;
      swims = true;
   }

   /**
    * move method overrides the move method
    * inherited from superclass Animal
    */
   @Override // good programming practice to use @Override to denote overridden methods
   public String move(int distance){  //move method overwritten for swimming
      distanceTxt = "Look at me, I swam ";
      distanceTxt += distance;
      distanceTxt += " metres!";


      return  distanceTxt;
   }
```

```
    /**
     * 'getter' method for the hasGills field
     */
    public boolean hasGills(){
        return hasGills;
    }

    /**
     * 'getter' method for the hasFins field
     */
    public boolean hasFins(){
        return hasFins;
    }
}
```

Shark Class

```
public class Shark extends Fish
{



   boolean isDangerous; //sharks bite!
   boolean bite;

   /**
    * Constructor for objects of class Shark
    */
   public Shark(String name)
   {
      super(); // call the constructor of the superclass Fish
      //Name & Colour inherited from animal class - Values assigned now override the inherited value
      this.name = name;
      isDangerous = true;
      bite = true;
   }



   public Boolean isDangerous(){
      return isDangerous;
   }

   /**
    * Sing method overrides the sing method
    * inherited from superclass Bird

   @Override // good programming practice to use @Override to denote overridden methods
   public void sing(){
      System.out.println("tweet tweet tweet");
   }*/
```

```
@Override
public String getInstanceName(){ //this is for returning the instance type
   String classTemp = "Shark";
   return classTemp;
}

/**
 * toString method returns a String representation of the fish
 * What superclass has Shark inherited this method from?
 *
 */
@Override
public String toString(){
   String strng ="";
   strng+= "Shark;\n";
   strng+= "name: ";
   strng+= name;
   strng+= "; ";
   strng+= "colour: ";
   strng+= colour;
   strng+= "\n";


   strng += "Do I breathe?: ";
   strng += breathes();
   strng += "\nDo I have skin?: ";
   strng += hasSkin();
   strng += "\nDo I eat?: ";
   strng += eats();
   strng += "\nDo I have gills?: ";
   strng += hasGills();
   strng += "\nDo I have gills?: ";
   strng += hasFins();
```

```
    strng+= "\nDangerous: ";
    strng+= isDangerous;

    strng += "\nDo I bite you?:";
    strng += bite;
    strng += "\n";

    strng += move(500);
    strng += "\n";

    // TOD0 Your job is to include the fields and attributes inherited
    //from Fish and Animal in the String representation
    return strng;
}



/**
 * equals method defines how equality is defined between
 * the instances of the Shark class
 * param Object
 * return true or false depending on whether the input object is
 * equal to this Shark object
 */

 @Override
public boolean equals(java.lang.Object object){

    // Checking did we get given any object.
    if (object == null) {
        System.out.print("Object given is NULL\n");
        return false;
    }
```

```
    // instanceof checking if the given object is the same type, otherwise the object cannot be casted
    if (object instanceof Shark) {
      // Casting given object & running checks if all the details are the same as our existing object
      Shark shark = (Shark) object;

      //in this if statement I'm not checking everything as it would look messy, e.g. Lets assume that both birds breathe (hopefully)
      if (this.getName() == shark.getName() && this.getColour() == shark.getColour() && this.isDangerous() == shark.isDangerous()) {
        return true;
      }
    }

    return false;
  }
}
```

Trout Class

```
public class Trout extends Fish
{

   boolean spikes;
   boolean isEdible; //smoked trout is a lovely dinner
   String spawnMethod;



   /**
    * Constructor for objects of class Trout
    */

   public Trout(String name)
   {
      super(); // call the constructor of the superclass Fish
      //Name & Colour inherited from animal class - Values assigned now override the inherited value
      this.name = name;
      colour = "brown";
      spawnMethod = "I swim upriver to lay eggs.";
      spikes = true;



   }



   public boolean spikes(){
      return spikes;
   }
   public boolean isEdible() {
      return isEdible;
   }
   public String spawnMethod() {
      return spawnMethod;
```

```
}
public boolean hasSpikes() {
   return spikes;
}



/**
 * Sing method overrides the sing method
 * inherited from superclass Bird

@Override // good programming practice to use @Override to denote overridden methods
public void sing(){
   System.out.println("tweet tweet tweet");
}*/

@Override
public String getInstanceName(){ //this is for returning the instance type
   String classTemp = "Trout";
   return classTemp;
}



/**
 * toString method returns a String representation of the fish
 * What superclass has Trout inherited this method from?
 *
 */
@Override
public String toString(){
   String strng ="";
   strng+= "\nTrout;\n";
   strng+= "name: ";
   strng+= name;
   strng+= "; ";
```

```
    strng+= "colour: ";
    strng+= colour;
    strng+= "\n";

    strng += "Do I breathe?: ";
    strng += breathes();
    strng += "\nDo I have skin?: ";
    strng += hasSkin();
    strng += "\nDo I eat?: ";
    strng += eats();
    strng += "\nDo I have gills?: ";
    strng += hasGills();
    strng += "\nDo I have gills?: ";
    strng += hasFins();

    strng += "\nAm I edible?: ";
    strng += isEdible();

    strng += "\nDo I have spikes?: ";
    strng += spikes();

    strng += "\nHow are new trout created?: ";
    strng += spawnMethod();

    strng += "\n";
    strng += move(200);

    // TOD0 Your job is to include the fields and attributes inherited
    //from Fish and Animal in the String representation
    return strng;
}


/**
```

```
    * equals method defines how equality is defined between
    * the instances of the Trout class
    * param Object
    * return true or false depending on whether the input object is
    * equal to this Trout object
    */

    @Override
   public boolean equals(java.lang.Object object){

      // Checking did we get given any object.
      if (object == null) {
         System.out.print("Object given is NULL\n");
         return false;
      }

      // instanceof checking if the given object is the same type, otherwise the object cannot be casted.
      if (object instanceof Trout) {
         // Casting given object & running checks if all the details are the same as our existing object
         Trout trout = (Trout) object;

         //in this if statement I'm not checking everything as it would look messy, e.g. Lets assume that both birds breathe (hopefully)
         if (this.getName() == trout.getName() && this.getColour() == trout.getColour() && this.hasSpikes() == trout.hasSpikes()) {
            return true;
         }
      }

      return false;
   }
}
```

This whole assignment was to get familiar with inheratince.

In this case, the bird and fish class extend from animal.

Both the bird and fish each have their own two leafs that extend off them; trout, shark, canary, ostrich.

All of these subclasses inherit methods and fields from the abstract classes above them.

For example - the animal class has a method for getName, where any of the animals can call it as they inherit it.

So even when it's called in ostrich, it still works even if the name is assigned and the method is in animal.

This is the same process for all the various fields / methods.