CT2106
Object Oriented Programming

Dr. Frank Glavin
Room 404, IT Building
Frank.Glavin@University*of*Galway.ie
School of Computer Science

University *of* Galway.ie

Lecture Topic

Polymorphism

For examples, see: https://www.javatpoint.com/runtime-polymorphism-in-java

Ollscoil na Gaillimhe
University of Galway

Animal Code

```
public abstract class Animal
{

    public abstract void move(int y);

}
```

```
public class Bird extends Animal
{
    @Override
    public void move(int y)
    {
        System.out.printf("I fly %d metres", y);
    }
}
```

```
public class Frog extends Animal
{
    @Override
    public void move(int y)
    {
        System.out.printf("I hop %d metres", y);
    }
}
```

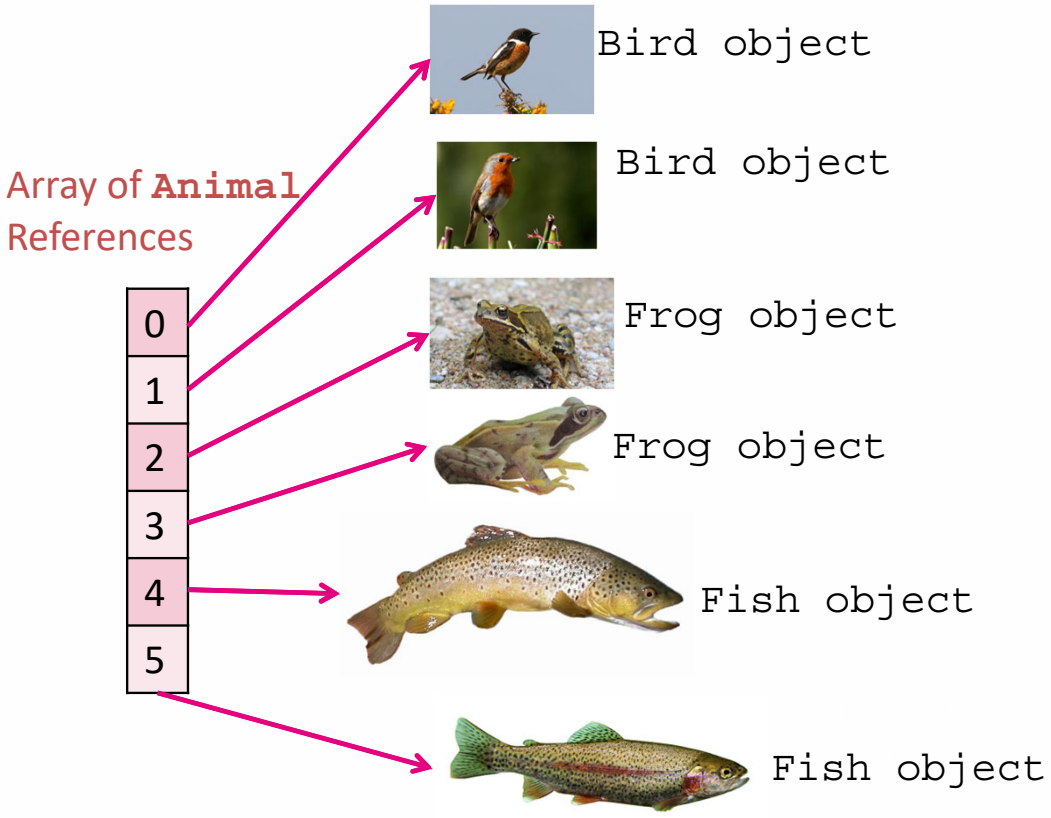
```
public class Fish extends Animal
{
    @Override
    public void move(int y)
    {
        System.out.printf("I swim %d metres", y);
    }
}
```

Ollscoil na Gaillimhe
University of Galway

Write the code

Write the code to add a reference to a different animal in each array location

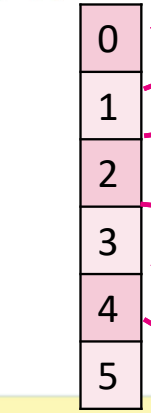
E.g. a bird in the first location
a bird in the second location
A Frog in the third location
And so on



```
public class AnimalTest
{

    public static void main(String[] args)
    {
        Animal[] animals = new Animal[6];
        animals[0] = new Bird();
        animals[1] = new Bird();
        animals[2] = new Frog();
        animals[3] = new Frog();
        animals[4] = new Frog();
        animals[4] = new Fish();
        animals[5] = new Fish();


    }
}
```




Bird object


Bird object


Frog object


Frog object


Fish object

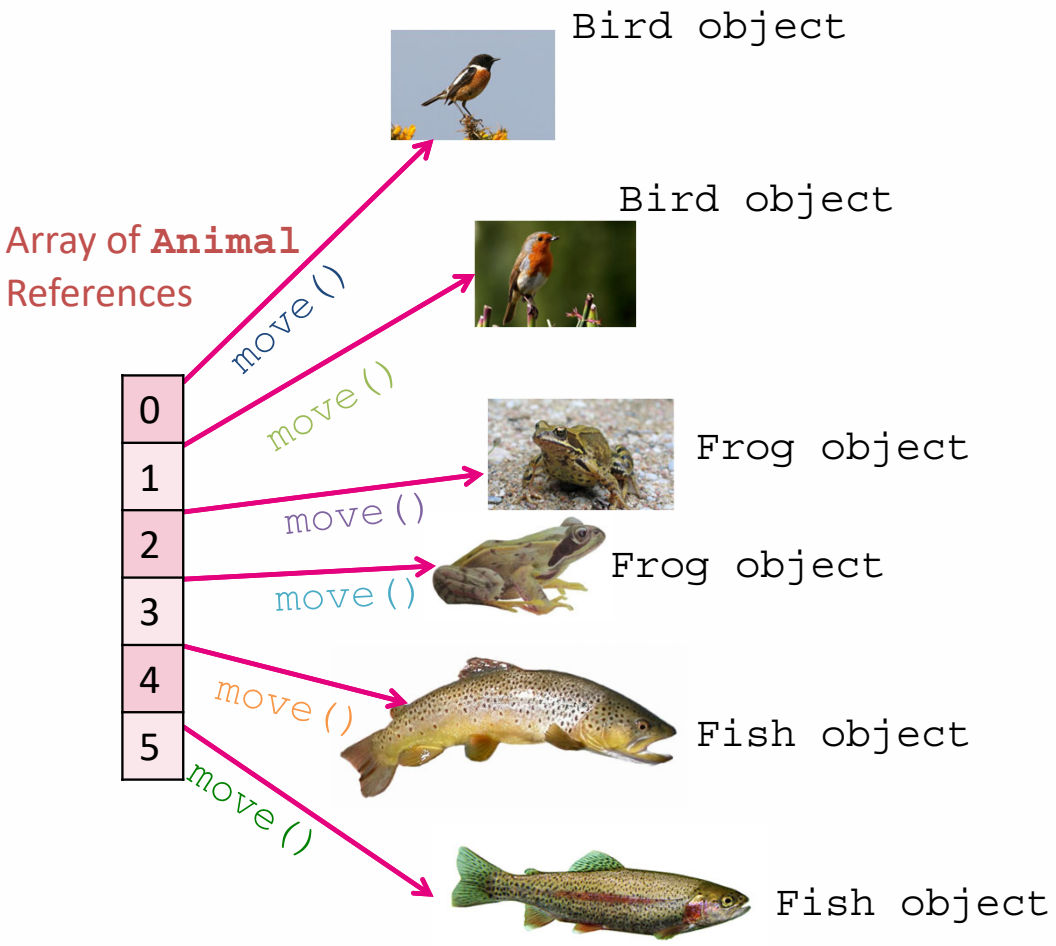

Fish object

Example

Now write the code
to call the move() method
from each
reference in the array
Use a *for* loop

```
for(Animal animal: animals){
    animal.move(5);
}
```

Run the code from the main
method



Example

- Note how you haven't explicitly called the move methods of Bird, Frog or Fish
- Just the move method of Animal (which is abstract)

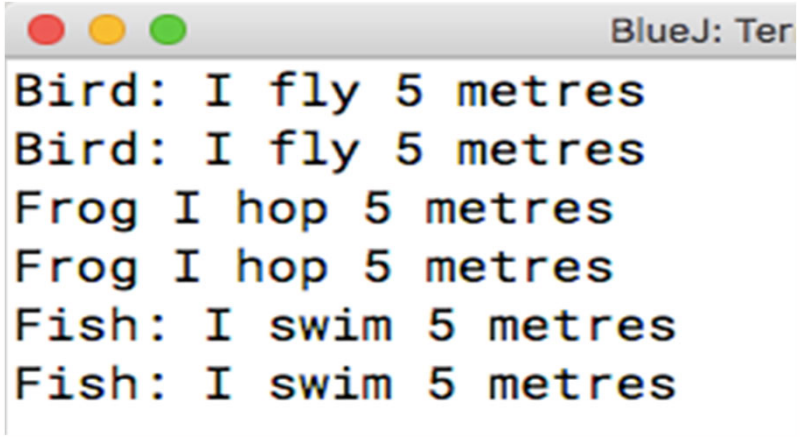
```
for(Animal animal: animals){
    animal.move(5);
}
```

Ollscoil na Gaillimhe
University of Galway

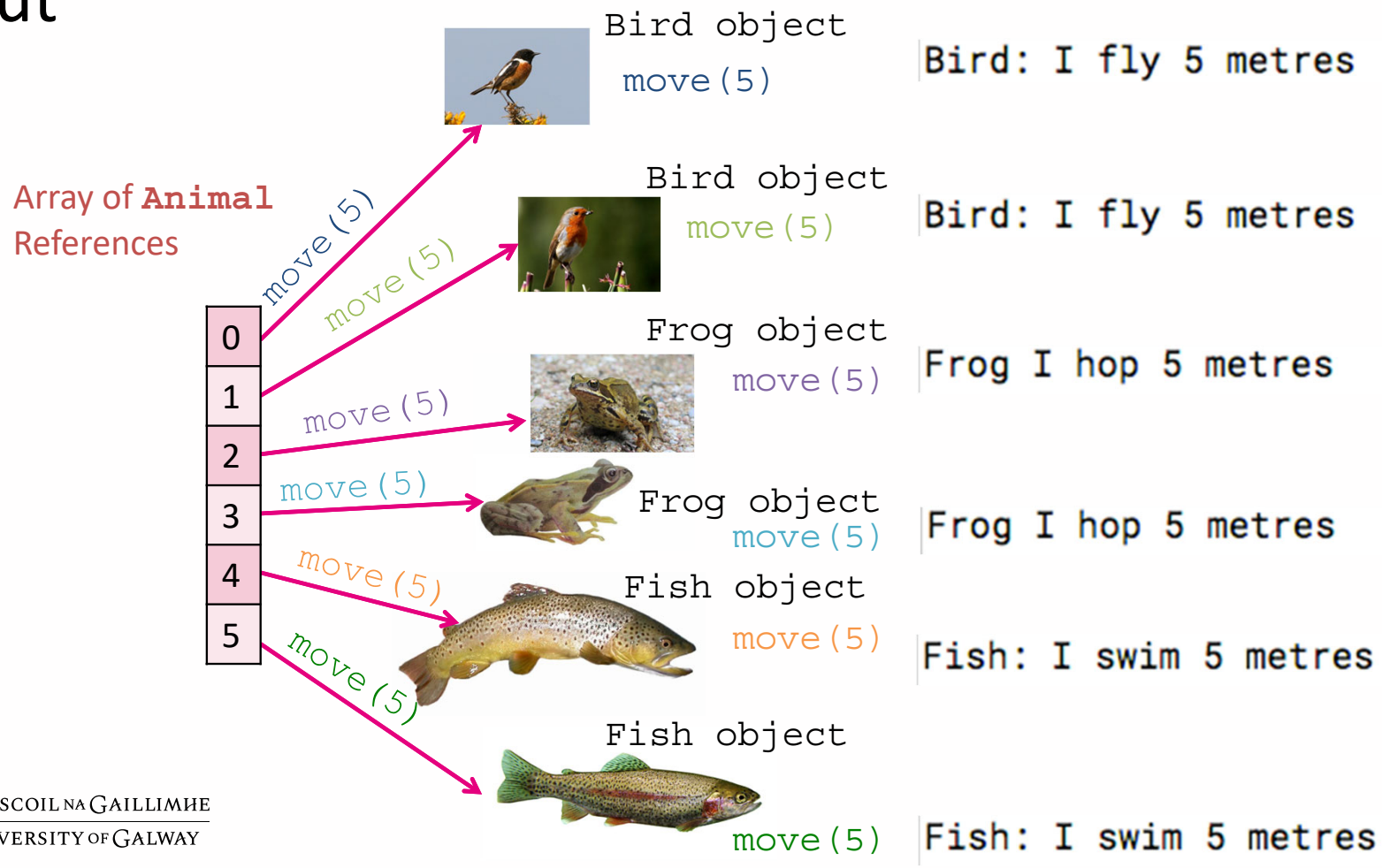
Output

Examine the output produced in the terminal

The specific *move* method of each of the referenced animal objects(Bird, Frog, Fish) has been called

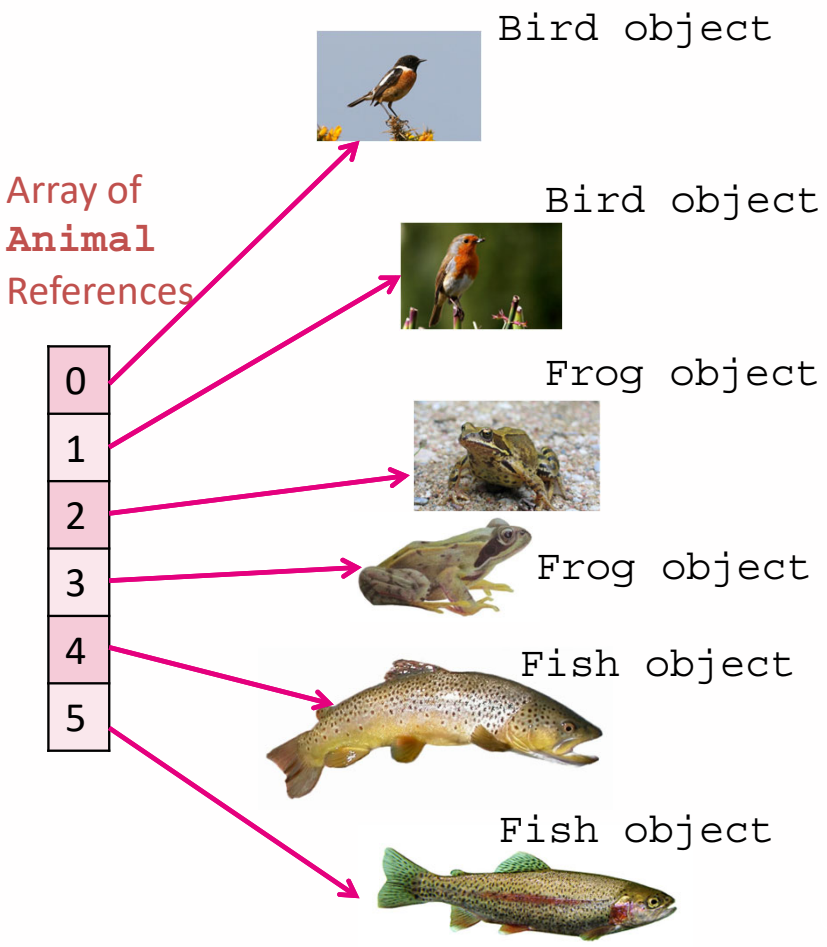


Output


Array of Animal References
0
1
2
3
4
5
move(5)
move(5)
move(5)
move(5)
move(5)
move(5)
Bird object
move(5)
Bird: I fly 5 metres
Bird object
move(5)
Bird: I fly 5 metres
Frog object
move(5)
Frog I hop 5 metres
Frog object
move(5)
Frog I hop 5 metres
Fish object
move(5)
Fish: I swim 5 metres
Fish object
move(5)
Fish: I swim 5 metres

Explanation

- Each element in the array contains a reference variable of type `Animal`
- Each reference points to a `Bird`, `Frog` or `Fish` object
- So when the `move()` method is called from the `Animal` references in the array it is the `move()` method of the respective `Bird`, `Frog`, `Fish` objects that is invoked



Dynamic Dispatch/Late binding

- This an example of what is called **dynamic dispatch** or **late binding**
- The decision as to which method to invoke is decided at program runtime, not compilation time
- If at run time, `animals[0]` points to a `Bird` object, then `animals[0].move()` invokes the `move()` method of the `Bird` object
- If `animals[0]` points to a `Fish` object, then `animals[0].move()` invokes the `move()` method of the `Fish` object

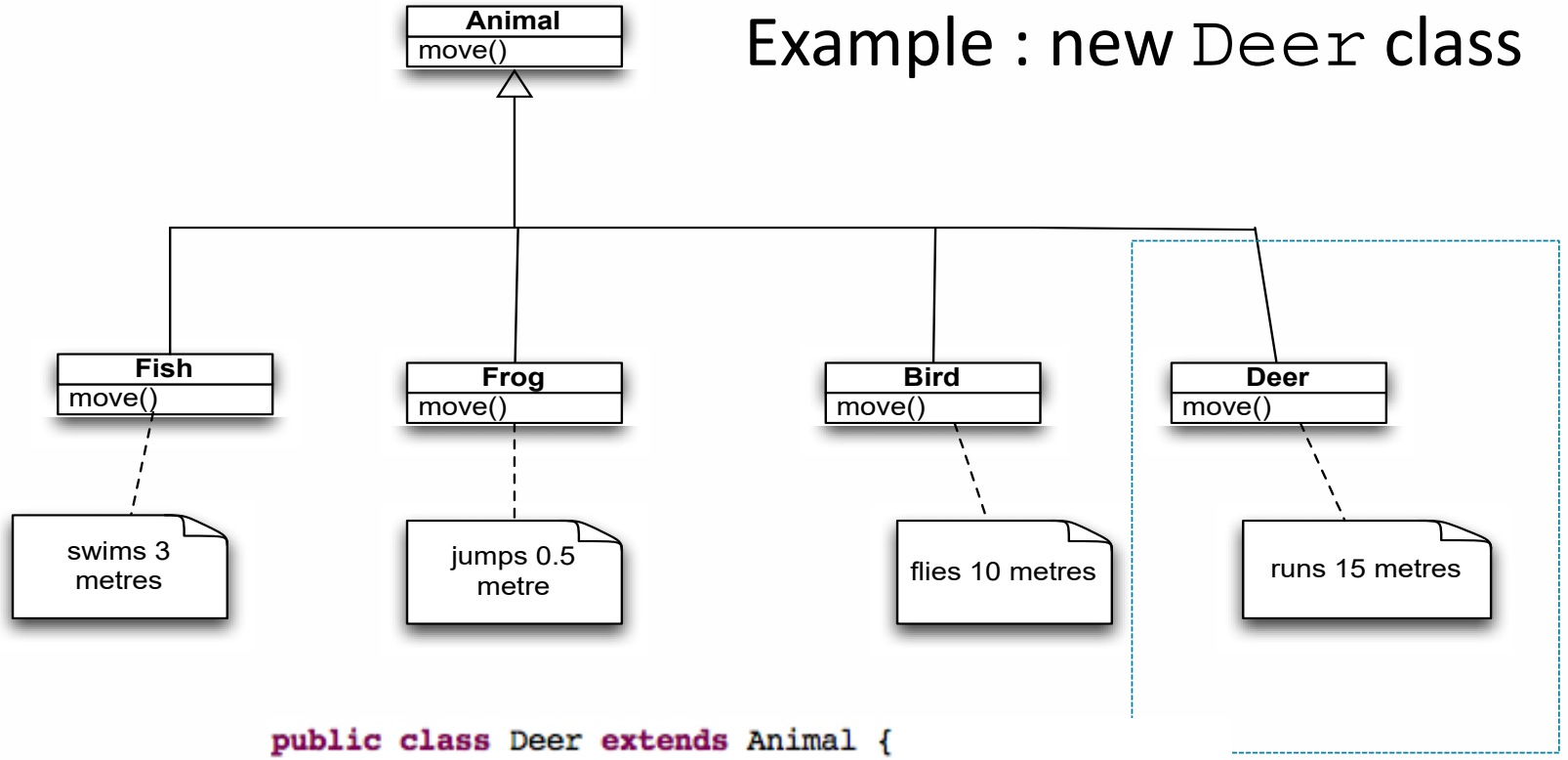
Ollscoil na Gaillimhe
University of Galway

Polymorphism

- We can add new `Animal` types with new `move()` behaviours to the array of `Animal` references
- As long as these are subclasses of `Animal`, their `move()` method will always be called

Ollscoil na Gaillimhe
University of Galway

Example : new `Deer` class




```java
public class Deer extends Animal {

    @Override
    public void move(){
        // TODO  code for flying 10 metres
        System.out.println("Deer: I run 15 metres");
    }

}
```

Ollscoil na Gaillimhe
University of Galway

Create a deer object

- Place a reference to a Deer object in the array and run the program again.

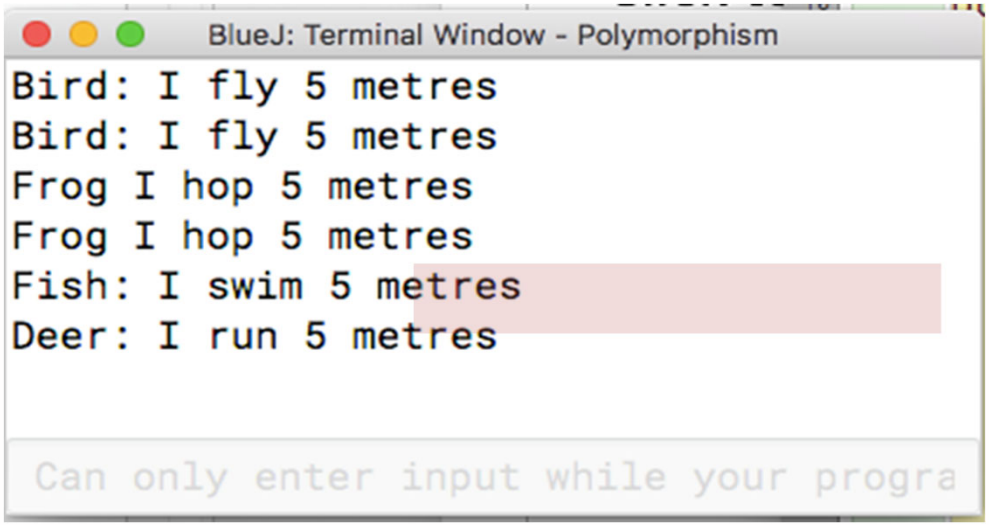
```
Animal[] animals = new Animal[6];
animals[0] = new Bird();
animals[1] = new Bird();
animals[2] = new Frog();
animals[3] = new Frog();
animals[4] = new Frog();
animals[4] = new Fish();
animals[5] = new Fish();
animals[5] = new Deer(); // this replaces the previous value

for(Animal animal: animals){
    animal.move(5);
}
```

Ollscoil na Gaillimhe
University of Galway

Output

- Key message we can change the behaviour of a program **without changing its code**
- E.g. this piece of code remains the same




```
for(Animal animal: animals){
    animal.move(5);
}
```

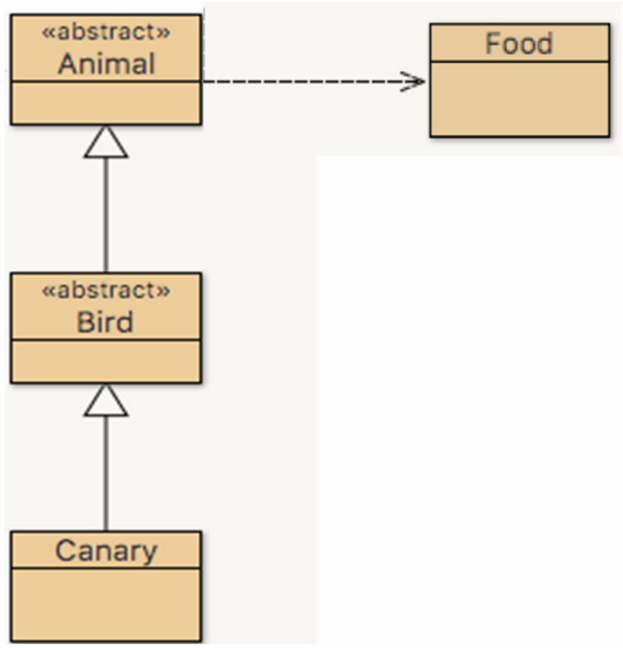
Implications

- With polymorphism, we can design and implement systems that are easily *extensible*
- New classes with new behaviours can be added with little or no modification to the general portions of the program

Ollscoil na Gaillimhe
University of Galway

Let’s look at applying these ideas

Open the code we first looked at yesterday



Ollscoil na Gaillimhe
University of Galway

Instructions

Food:

Make Food an abstract class

Give it two abstract methods *getCalories* and *getFat* with a return type *int*

Animal: make *eat* method abstract

- Create an abstract subclass of Food called **Vegetable**
- Create a concrete subclass of Vegetable called **Seed**
- Seed has two fields *calories* and *fat*
- Canary must implement a concrete version of the *eat* method
- Canary's eat method checks if Food object is an *instanceof* Seed; if it is, the Canary calls Food's getCalories method and moves the distance returns. She also calls the sing method.

Ollscoil na Gaillimhe
University of Galway

Lecture wrap up

- We looked at polymorphism – the facility by which an object can be referenced by a variable of its Superclass
- This allows us to create code that is easily extensible
- We saw that we can create variables of abstract types (classes)

Ollscoil na Gaillimhe
University of Galway