Lecture 31

- Covers
  - Inheritance
- Reading: Savitch 7.1,7.2

Photos in this lecture courtesy of Dr Geoff Witten

Inheritance

- Define a general class
- Make specialised versions
- Specialised versions “inherit” the properties from the base class
- Superclass / subclass
- Base class / derived class

Advantages of inheritance

- Code reuse
- Real-world correlation
- Collectively referring to different types of objects

Example

- The La Trobe Lizard Museum has commissioned you to write a program to help them catalogue their collection of lizards. Each lizard specimen is given a specimen number and this must be recorded along with the date it was collected and the name of the zoologist who collected it.
- Every lizard has its snout-vent length recorded (the distance from the lizard's nose to the base of its tail) as well as its tail length (the distance from the base of its tail to the end of its tail). Both of these measurements are recorded in millimetres.

Lizard class

Lizard
specimenNumber collectorName dateCollected snoutVentLength tailLength

Lizard( )
display( )
getSnoutVentLength( )
getTailLength( )

Lizard class

```
Lizard
----------------------------------
int specimenNumber
String collectorName
String dateCollected
double snoutVentLength
double tailLength
----------------------------------
Lizard(int, String, String,
          double, double)
void display( )
double getSnoutVentLength( )
double getTailLength( )
```

Defining the Lizard class

```
public class Lizard
{
   private int specimenNumber;

   private String lizardType;

   private String dateCollected;
   private String collectorName;

   private double snoutVentLength;
   private double tailLength;
}
```

$\longleftarrow$ *will use later when we have different types of lizards*

Defining the Lizard constructor

```
public Lizard(int sn, String collector, String date,
                    double svLength, double tLength)
{
    specimenNumber = sn;
    collectorName = collector;
    dateCollected = date;
    snoutVentLength = svLength;
    tailLength = tLength;
    lizardType = "lizard";
}
```

Defining Lizard methods

```
public void display( )
{
    System.out.println("Specimen Number:    " + specimenNumber);
    System.out.println("Type of Lizard:     " + lizardType);
    System.out.println("Date collected:     " + dateCollected);
    System.out.println("Collected by:       " + collectorName);
    System.out.println("Snout-vent length:  " + snoutVentLength);
    System.out.println("Tail length:        " + tailLength);
}
```

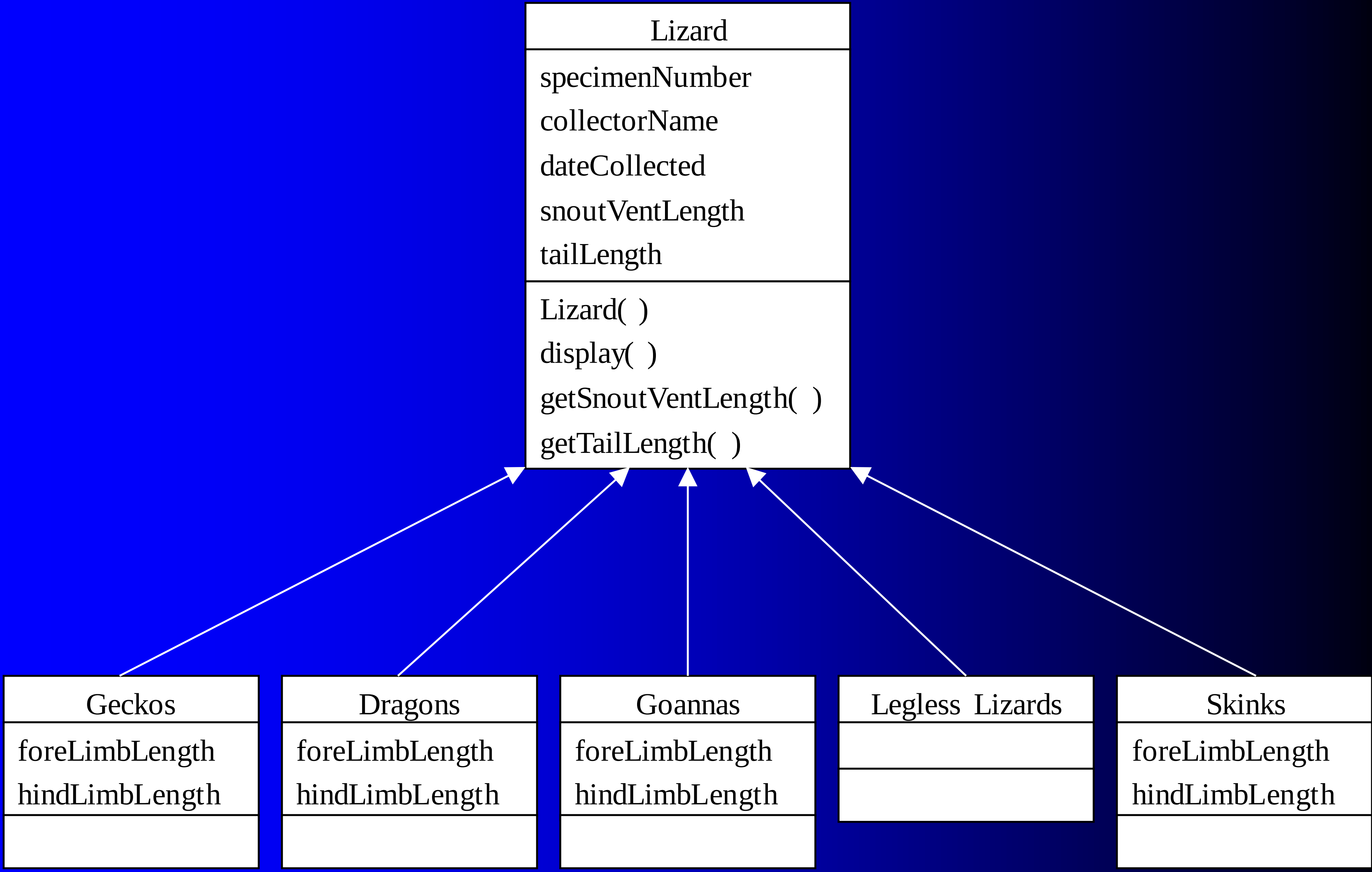
Defining Lizard methods

```
public double getSnoutVentLength( )
{
   return snoutVentLength;
}
public double getTailLength( )
{
   return tailLength;
}
public void setLizardType(String lt)
{
   lizardType = lt;
}
```

will use later when we have different types of lizards

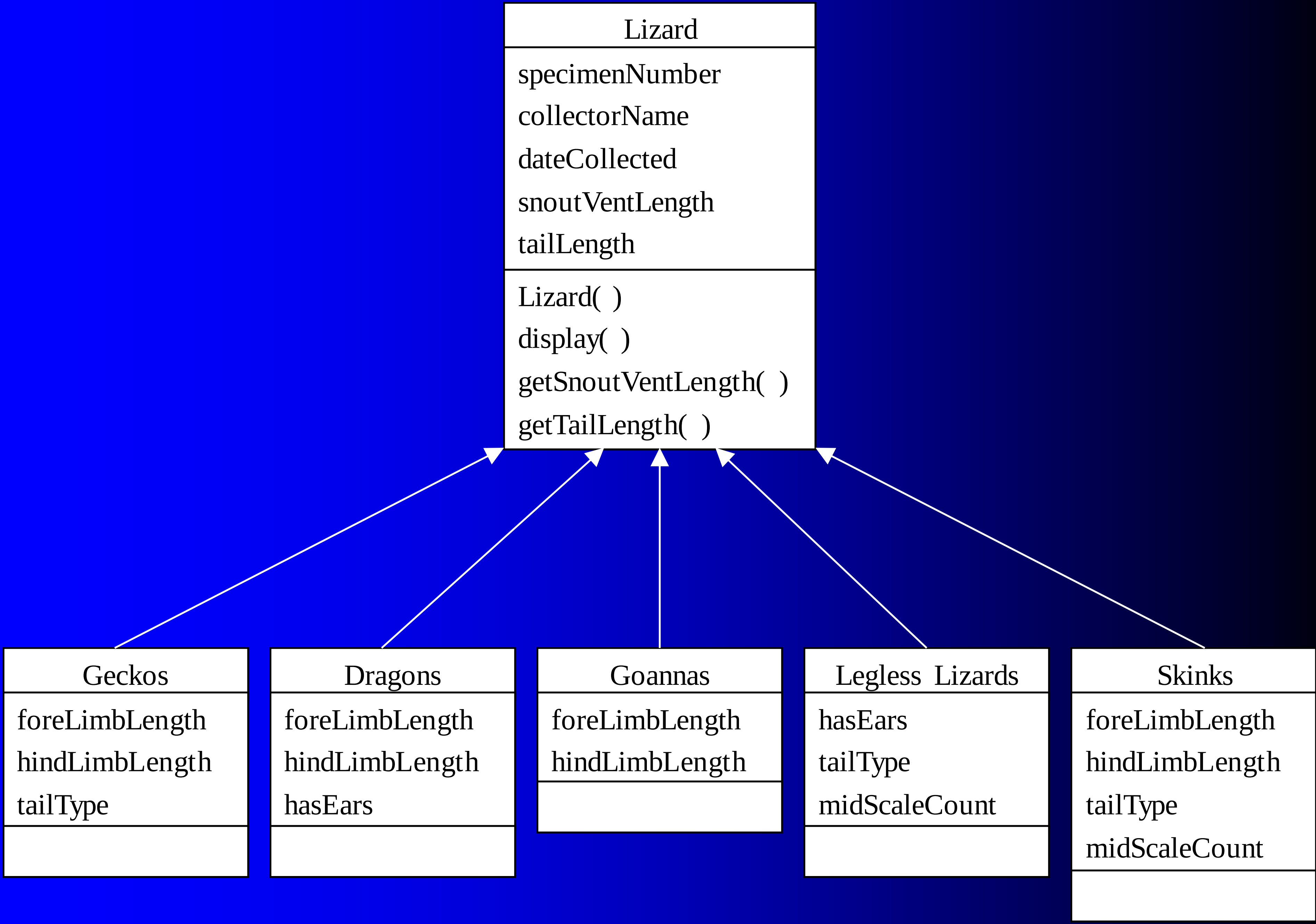
Example

- Lizards found in Australia can be divided into the following groups: Geckos, Dragons, Goannas, Legless Lizards and Skinks
- The system needs to record different attributes of the lizard specimens depending on which type of lizard they are
- All types of lizards have their forelimb (front leg) length and hindlimb (back leg) length recorded (again in millimetres), except for Legless Lizards who have no forelimbs and may not have hindlimbs


Lizard
specimenNumber
collectorName
dateCollected
snoutVentLength
tailLength
Lizard( )
display( )
getSnoutVentLength( )
getTailLength( )
Geckos
foreLimbLength
hindLimbLength
Dragons
foreLimbLength
hindLimbLength
Goannas
foreLimbLength
hindLimbLength
Legless Lizards
Skinks
foreLimbLength
hindLimbLength

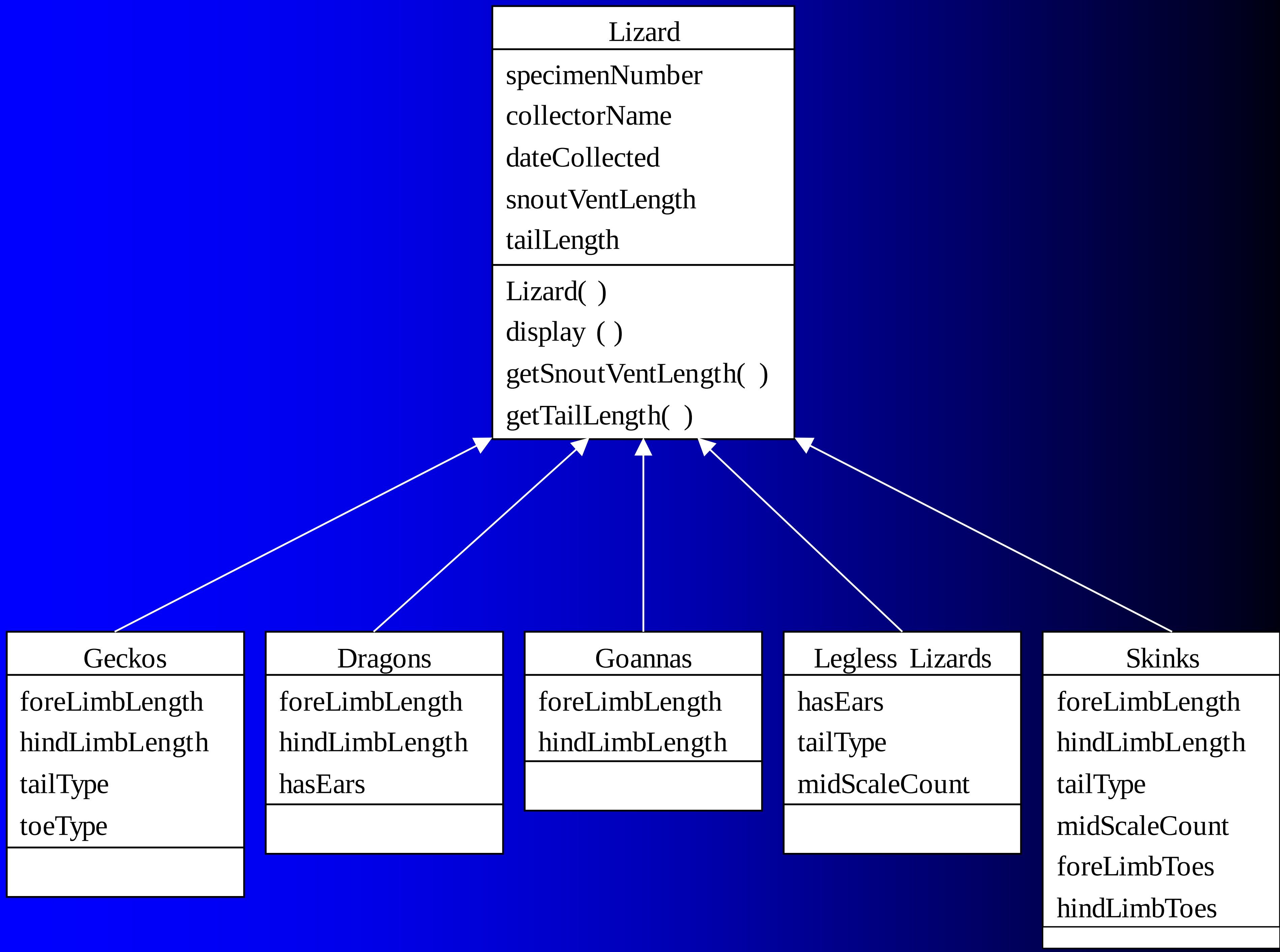
Example

- The presence or absence of ears must be recorded for Dragons and Legless Lizards only while a mid-body scale count must be recorded for Legless Lizards and Skinks only
- Geckos, Legless Lizards and Skinks may all lose their tails to save themselves if they are caught by a predator. Therefore, when recording tail length for these types of lizards, whether or not they had a missing, damaged or regenerated tail is also recorded


Lizard
specimenNumber
collectorName
dateCollected
snoutVentLength
tailLength
Lizard( )
display( )
getSnoutVentLength( )
getTailLength( )
Geckos
foreLimbLength
hindLimbLength
tailType
Dragons
foreLimbLength
hindLimbLength
hasEars
Goannas
foreLimbLength
hindLimbLength
Legless Lizards
hasEars
tailType
midScaleCount
Skinks
foreLimbLength
hindLimbLength
tailType
midScaleCount

Example

- Geckos have their type of toes recorded. Their toes are classified as either claws or pads
- Skinks, on the other hand, have variable numbers of toes, so each specimen of Skink has a forelimb toe count and a hindlimb toe count recorded


Lizard
specimenNumber
collectorName
dateCollected
snoutVentLength
tailLength
Lizard( )
display ( )
getSnoutVentLength( )
getTailLength( )
Geckos
foreLimbLength
hindLimbLength
tailType
toeType
Dragons
foreLimbLength
hindLimbLength
hasEars
Goannas
foreLimbLength
hindLimbLength
Legless Lizards
hasEars
tailType
midScaleCount
Skinks
foreLimbLength
hindLimbLength
tailType
midScaleCount
foreLimbToes
hindLimbToes

Extending the Lizard class

- All lizards share some common properties
- Each of the different subclasses of the Lizard class add to, or extend, the properties defined in the Lizard class
- For example, Geckos have all the properties of Lizards and some additional properties: foreLimbLength, hindLimbLength, tailType, toeType
- In Java, we can use a class as a base class and create new derived classes that inherit from it
- Use the keyword extends

Defining the Gecko class

```
public class Gecko extends Lizard
{
   private double foreLimbLength;
   private double hindLimbLength;
   private String tailType;
   private String toeType;
}
```

Defining the Gecko constructor

```
public Gecko(int sn, String collector, String date,
             double svLength, double tLength, double fll, double hll,
             String tail, String toe)
 {
     // set the attributes of the Gecko object to the values
     // of the parameters passed in
 }
```

Q: How do we set the inherited properties when they are declared to be private in the Lizard class?

Defining the Gecko constructor

```
public Gecko(int sn, String collector, String date,
            double svLength, double tLength, double fll, double hll,
            String tail, String toe)
{
    super(sn, collector, date, svLength, tLength);
    foreLimbLength = fll;
    hindLimbLength = hll;
    tailType = tail;
    toeType = toe;
    setLizardType("gecko");
}
```

We can use the base class constructor to set the inherited attributes' values

We must use the public method setLizardType to set the lizardType attribute as it was private in Lizard

Defining the Gecko methods

```
public double getForeLimbLength( )
{
   return foreLimbLength;
}
public double getHindLimbLength( )
{
   return hindLimbLength;
}
```

Defining the Gecko methods

```
public void display( )
{
    // display all the attributes inherited by Gecko from
    // the Lizard class and the ones defined in the
    // Gecko class itself
}
```

This method overrides the display method inherited from the Lizard class

Q: How to display the inherited attributes that are declared as private in the Lizard class?

Defining the Gecko methods

```
public void display( )
{
    super.display( );
    System.out.println("Forelimb length:   " + foreLimbLength);
    System.out.println("Hindlimb length:   " + hindLimbLength);
    System.out.println("Tail type:         " + tailType);
    System.out.println("Toe type:          " + toeType);
}
```

Using the Lizard and Gecko classes

```
public class LizardTester
{
  public static void main(String[ ] args)
  {
    Lizard liz1 = new Lizard(1, "Julie", "April 1", 30, 50);
    liz1.display( );

    System.out.println("liz1 tail: " + liz1.getTailLength( ) + "\n");

    Gecko g1 = new Gecko(2, "Kinh", "May 2", 40, 55, 15, 25, "original", "pads");
    g1.display( );

    System.out.println("g1 tail: " + g1.getTailLength( ) + "\n");

    Lizard liz2 = new Gecko(3, "Mary", "June 3", 50, 50, 20, 20, "missing", "claws");
    liz2.display( );
    System.out.println("liz2 tail: " + liz2.getTailLength( ) + "\n");
  }
}
```

Using the Lizard and Gecko classes

```
Specimen Number:   1
Type of Lizard:    lizard
Date collected:    April 1
Collected by:      Julie
Snout-vent length:  30.0
Tail length:        50.0
liz1 tail: 50.0
```

```
Specimen Number:   2
Type of Lizard:    gecko
Date collected:    May 2
Collected by:      Kinh
Snout-vent length:  40.0
Tail length:        55.0
Forelimb length:   15.0
Hindlimb length:   25.0
Tail type:          original
Toe type:           pads
g1 tail: 55.0
```

```
Specimen Number:   3
Type of Lizard:    gecko
Date collected:    June 3
Collected by:      Mary
Snout-vent length:  50.0
Tail length:       50.0
Forelimb length:   20.0
Hindlimb length:   20.0
Tail type:          missing
Toe type:           claws
liz2 tail: 50.0
```

References to the superclass

- If the type of a variable x is class C, then x can refer to an object of class C or an object of any of its derived classes
- We can have a variable that is a reference to a Lizard, and it can refer to a Lizard object or an object of a derived class of Lizard such as the Gecko class
- Therefore a Gecko object can be considered as having multiple types: It is a Gecko, but it is also a Lizard

Overriding

- A method declared in the base class can be given a new definition in the derived class
- The definition in the derived class "overrides" the definition inherited from the base class
- Calling an overridden method on an object of the derived class results in the method defined in the derived class being executed (e.g. the display( ) method)

Overriding

- If a method is not overriden in the derived class, then calling that method on a derived class object will result in the inherited method being executed (e.g. the getTailLength( ) method)

Class exercise

- Define the class header and attributes of the Dragon class

Class exercise

- Define the constructor for the Dragon class

Class exercise

- Define the display( ) method for the Dragon class

Referring to objects collectively

```
public class LizardTester
{
    public static void main(String[ ] args)
    {
        Lizard[ ] liz = new Lizard[4];
        liz[0] = new Lizard(1, "Julie", "April 1", 30, 50);
        liz[1] = new Gecko(2, "Kinh", "May 2", 40, 55, 15, 25, "original", "pads");
        liz[2] = new Gecko(3, "Mary", "June 3", 50, 50, 20, 20, "missing", "claws");
        liz[3] = new Dragon(4, "Hongen", "June 3", 5, 15, 8, 8, true);

        for (int i = 0; i < liz.length; ++i)
        {
            liz[i].display( );
            System.out.println( );
        }
    }
}
```

Covered only if time permits

Example

- There are two calculations that zoologists use when examining and classifying lizards: the relative tail length and the relative limb length
- The relative limb length is calculated as the forelimb length divided by the hindlimb length
- The only variation in calculating this ratio is for skinks where one or both of the limb lengths may be 0. In the case where both limb lengths are 0, the value -1 is used to denote the relative limb length. In the case where only the forelimb length is 0, 0 is used to denote the relative limb length
- For Legless Lizards (no recorded limb lengths) -2 is used to denote the relative limb length

Abstract methods

- Each derived class of the Lizard class has a way of calculating the relative limb length, but the base class Lizard does not
- If we want to have a collection of Lizards and calculate the correct relative limb length depending on the type of lizard, we need to define the method in the base class

Abstract methods

```
public static void main(String[ ] args)
{
   Lizard[ ] liz = new Lizard[4];
   liz[0] = new Dragon(1, "Julie", "April 1", 30, 50, 20, 25, false);
   liz[1] = new Gecko(2, "Kinh", "May 2", 40, 55, 15, 25, "original", "pads");
   liz[2] = new Gecko(3, "Mary", "June 3", 50, 50, 20, 20, "missing", "claws");
   liz[3] = new Dragon(4, "Hongen", "June 3", 5, 15, 8, 8, true);

   for (int i = 0; i < liz.length; ++i)
   {
      liz[i].display( );
      System.out.println("Relative Limb Length: " + liz[i].relativeLimbLength( ));
   }
}
```

Abstract methods

- We can define the overriding methods in the derived classes easily as we know how to calculate the relative limb lengths, e.g.

```
public double relativeLimbLength( )
{
      return foreLimbLength / hindLimbLength;
}
```

- But to be able to call the method on a generic Lizard, we need to declare the method in the base class
- What code should it have?

Abstract methods

- There is no sensible code to calculate the relative limb length for a generic Lizard object
- We can declare the method abstract in the base class with the keyword abstract
- No method body is then defined in that class

```
public abstract double relativeLimbLength( );
```

Abstract methods

- We can no longer create instances of the Lizard class as one of the methods is abstract

```
Lizard liz = new Lizard(…);
liz.relativeLimbLength( );
```

✗

No code to call here!

Abstract classes

- When we cannot create instances of the class, we call the class an abstract class
- It must be labelled as such

```
public abstract class Lizard
{

}
```

- If a class has an abstract method, it must be declared to be an abstract class

Homework exercise

- Relative tail length is calculated as the tail length divided by the snout-vent length
- For the three types of lizards that may lose their tails, a specimen whose tail has been damaged, lost or regenerated will not have an accurate relative tail length with this calculation
- In these cases, 0 is instead used to denote the relative tail length

Next lecture

- Introduction to applets
- HTML basics
- The paint( ) method