Inheritance

Method Replacement and Refinement

COMP2603
Object Oriented Programming 1

Week 3, Lecture 2

Outline

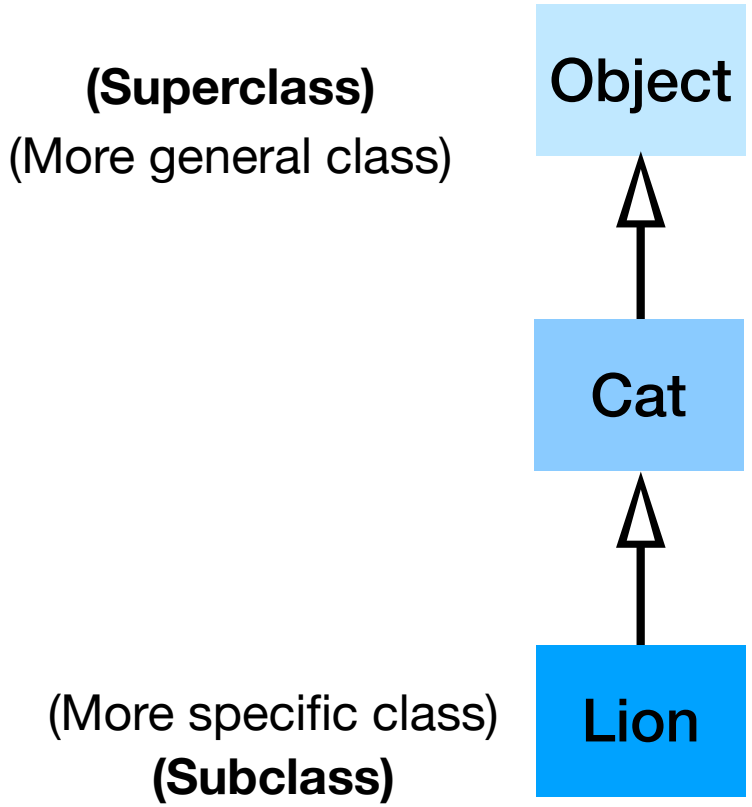
- Inheritance
  - Creating and Manipulating Subclass Instances
  - Constructors
  - Method Refinement
  - Method Replacement
  - Access modifiers and inheritance
  - Preventing Inheritance and Overriding

Generalisation vs Specialisation

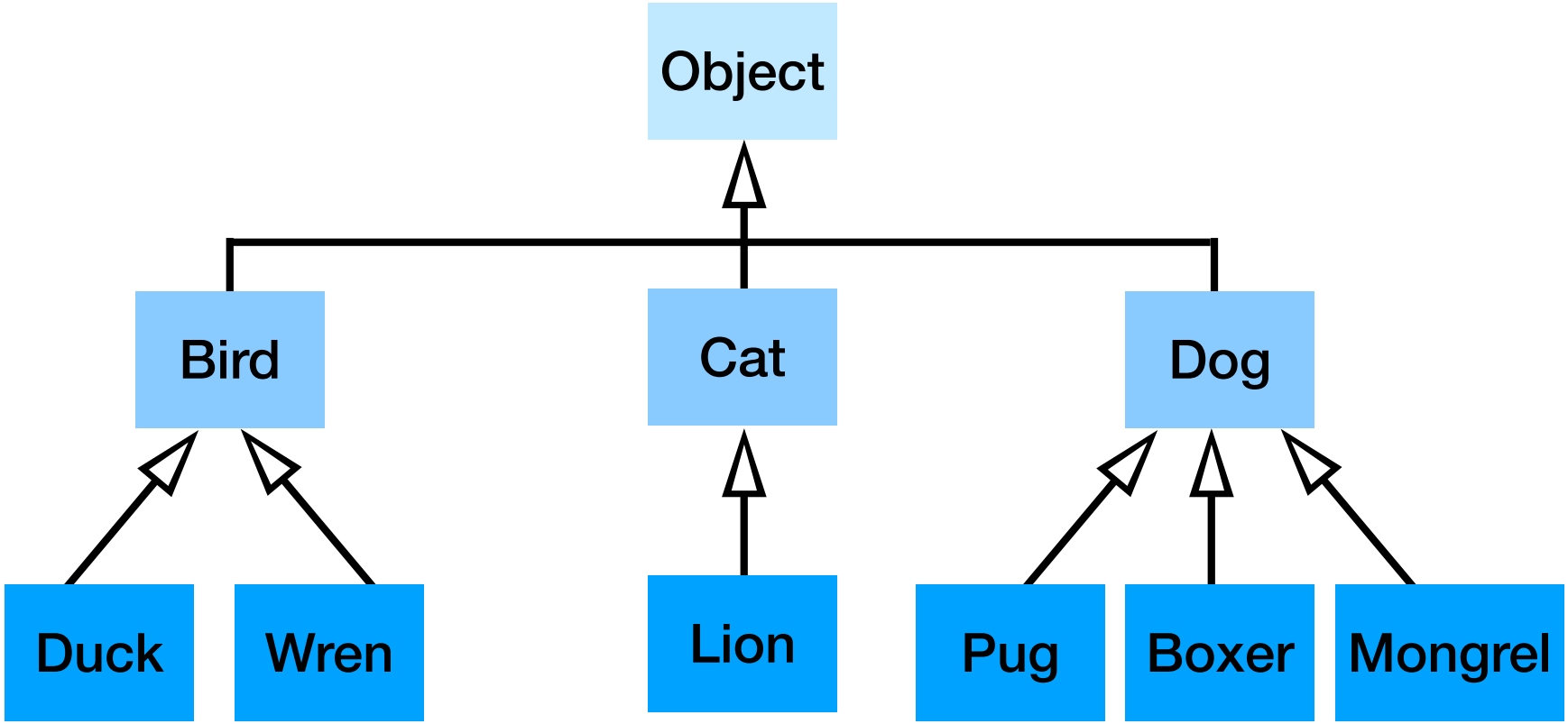
Generalisation is used to model a relationship between classes in which one class represents a more general concept and another class represents a more specialised concept.

Example - Inheritance

The Cat class is a specialisation of Object (superclass) whereas it is a generalisation of Lion (subclass).



Example - Inheritance



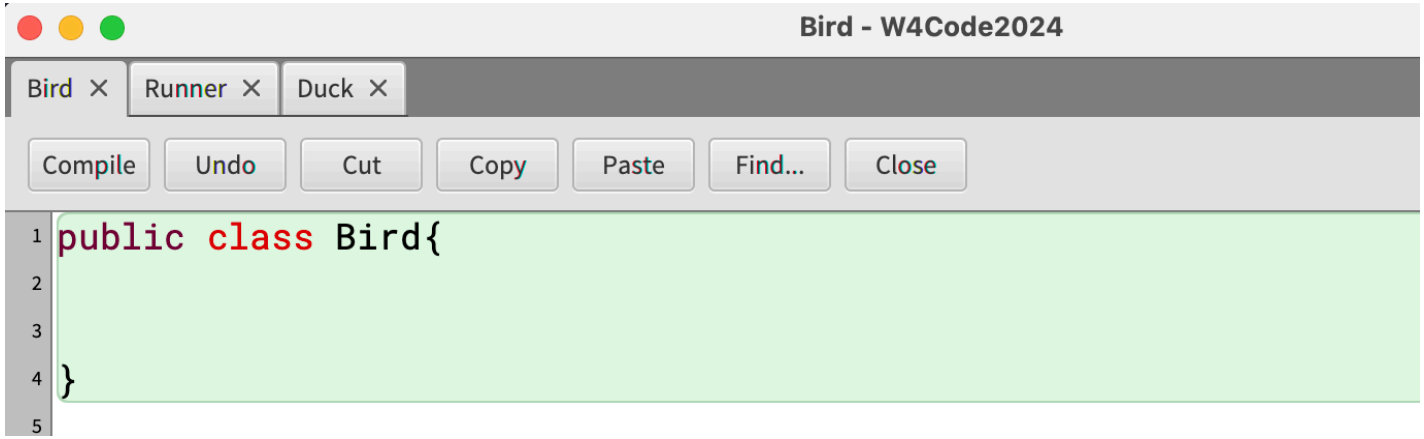
Constructors

The default constructor is a simple, no-argument constructor. Subclasses therefore can use the no-argument constructors (provided by default).

When arguments are required by a superclass constructor, the subclasses must supply these parameters and explicitly invoke the parent constructor.

Example - Inheritance (Slide 5)

Suppose we fill out the Bird class like this:



Example - Inheritance (Slide 5)

In the Runner class, the toString( ) method compiles without error because the parent class (Object) has a toString( ) method. That method is invoked on line 4 and the output originates from the way the toString( ) from the Object class works.

Runner - W4Code2024

Bird × | Runner × | Duck ×

Compile | Undo | Cut | Copy | Paste | Find... | Close

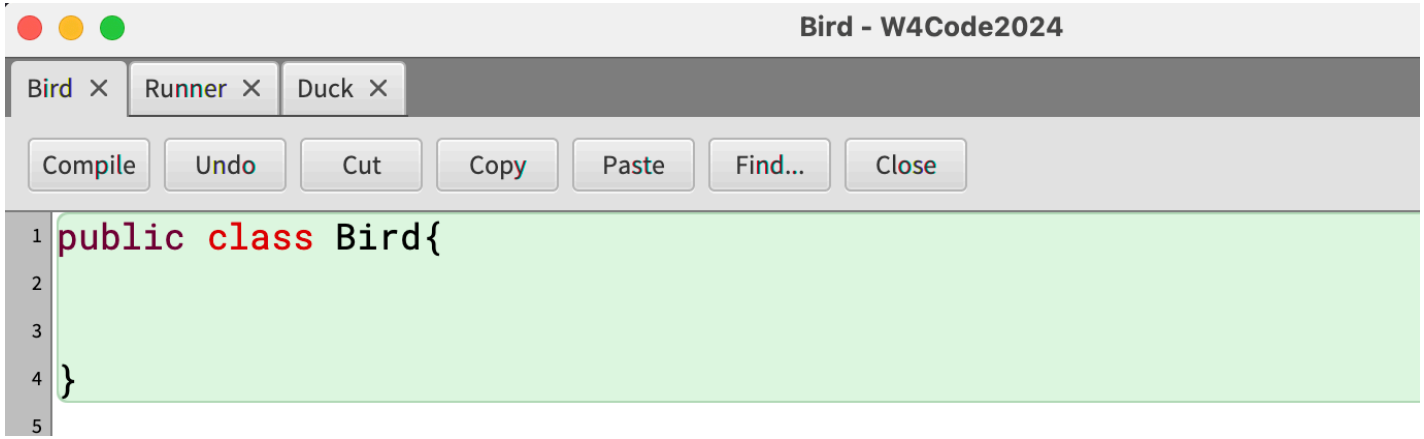
```
1 public class Runner{
2     public static void main(String[] args){
3         Bird b1 = new Bird();
4         System.out.println(b1.toString());
5
6
7     }
8 }
```

BlueJ: Terminal Window - W4Code2024

```
Bird@7f7fe722
```

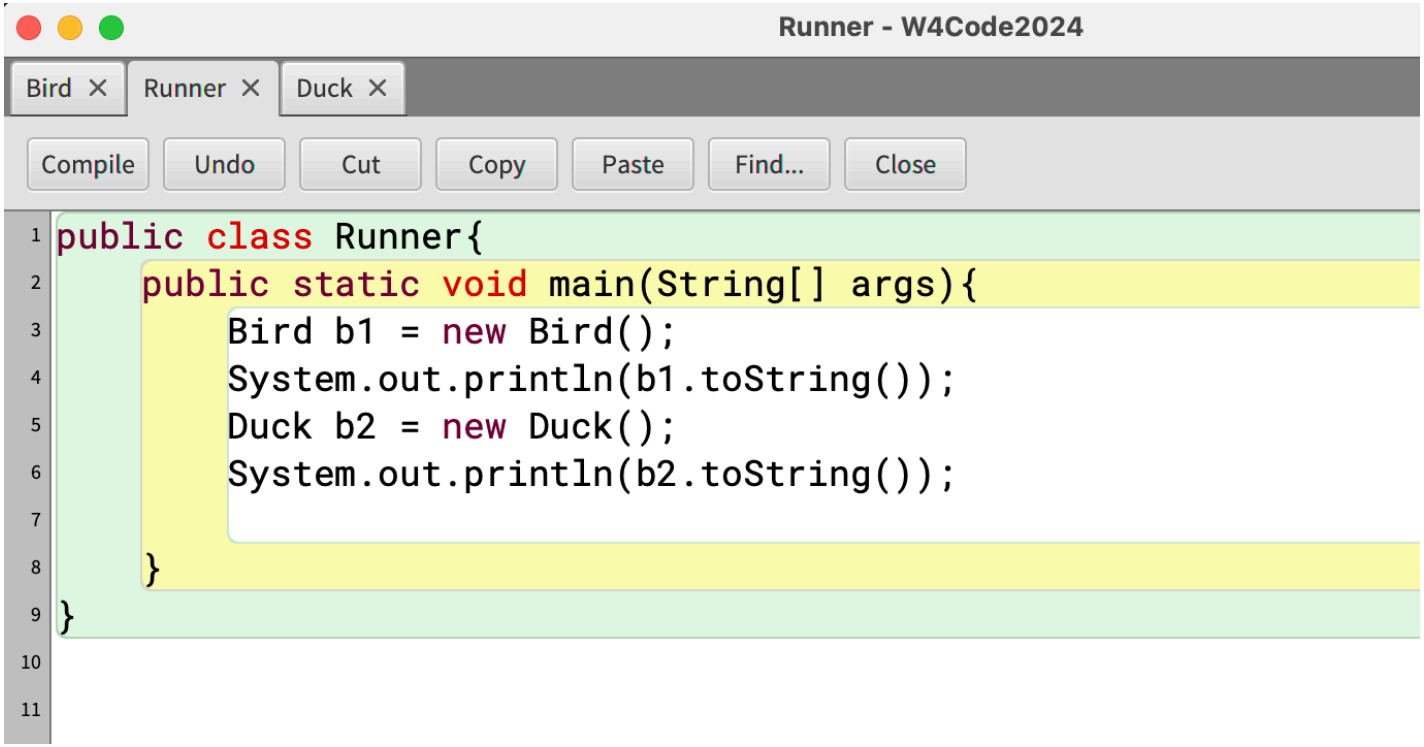
Example - Inheritance (Slide 5)

Suppose we fill out the Bird and Duck classes as follows.



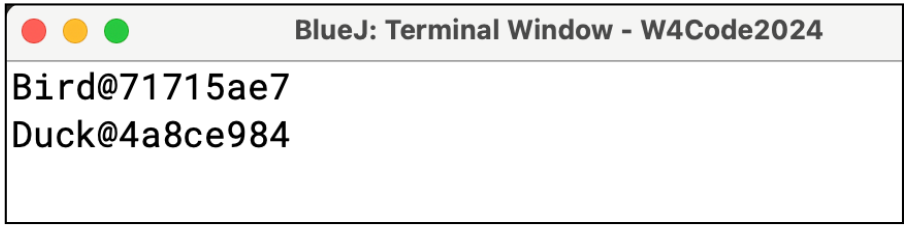
Example - Inheritance (Slide 5)

The Runner class looks like this:



Example - Inheritance (Slide 5)

The output is produced below:

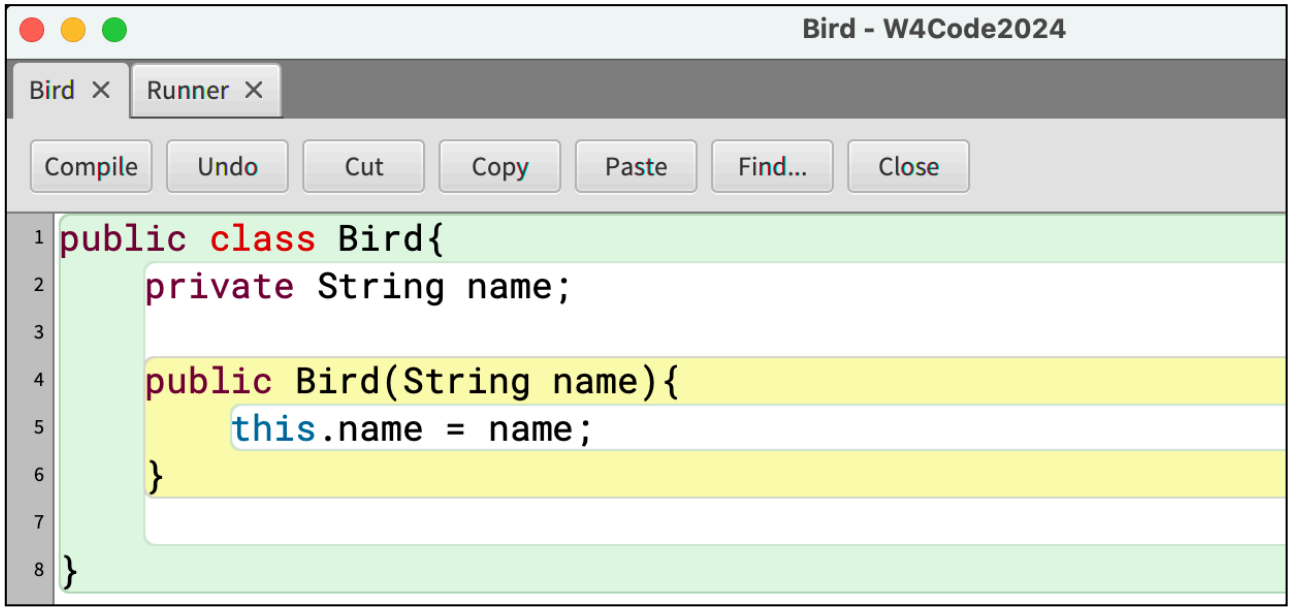


Bird and Duck do not have any custom constructors, so the default Java constructors are used. No arguments are needed which means that there is no problem with connecting Duck as a subclass of Bird.

Bird and Duck are subclasses of Object and they do not have a toString( ) of their own. Consequently, the parent's toString( ) is used.

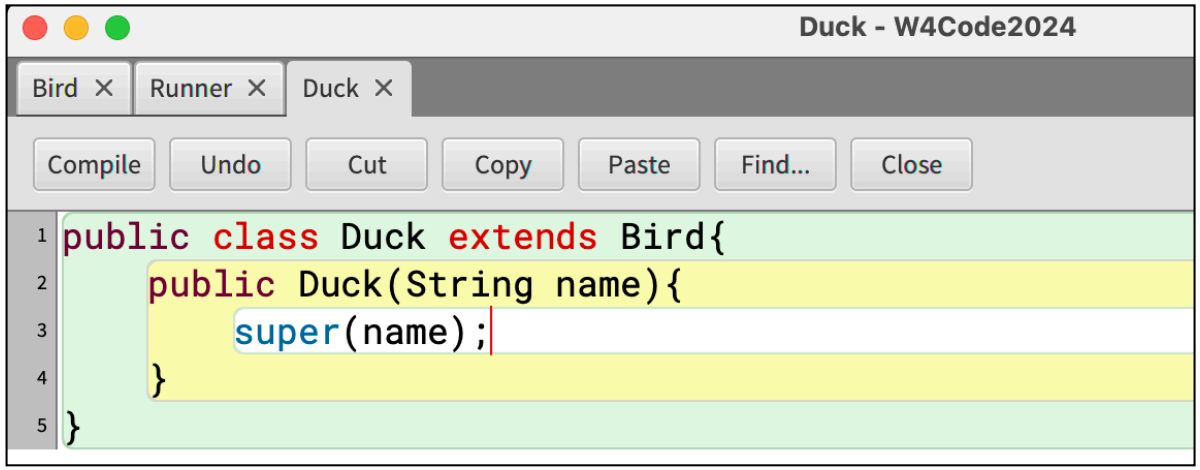
Example - Inheritance (Slide 5)

Suppose we fill out the Bird class with a custom constructor as follows.



Example - Inheritance (Slide 5)

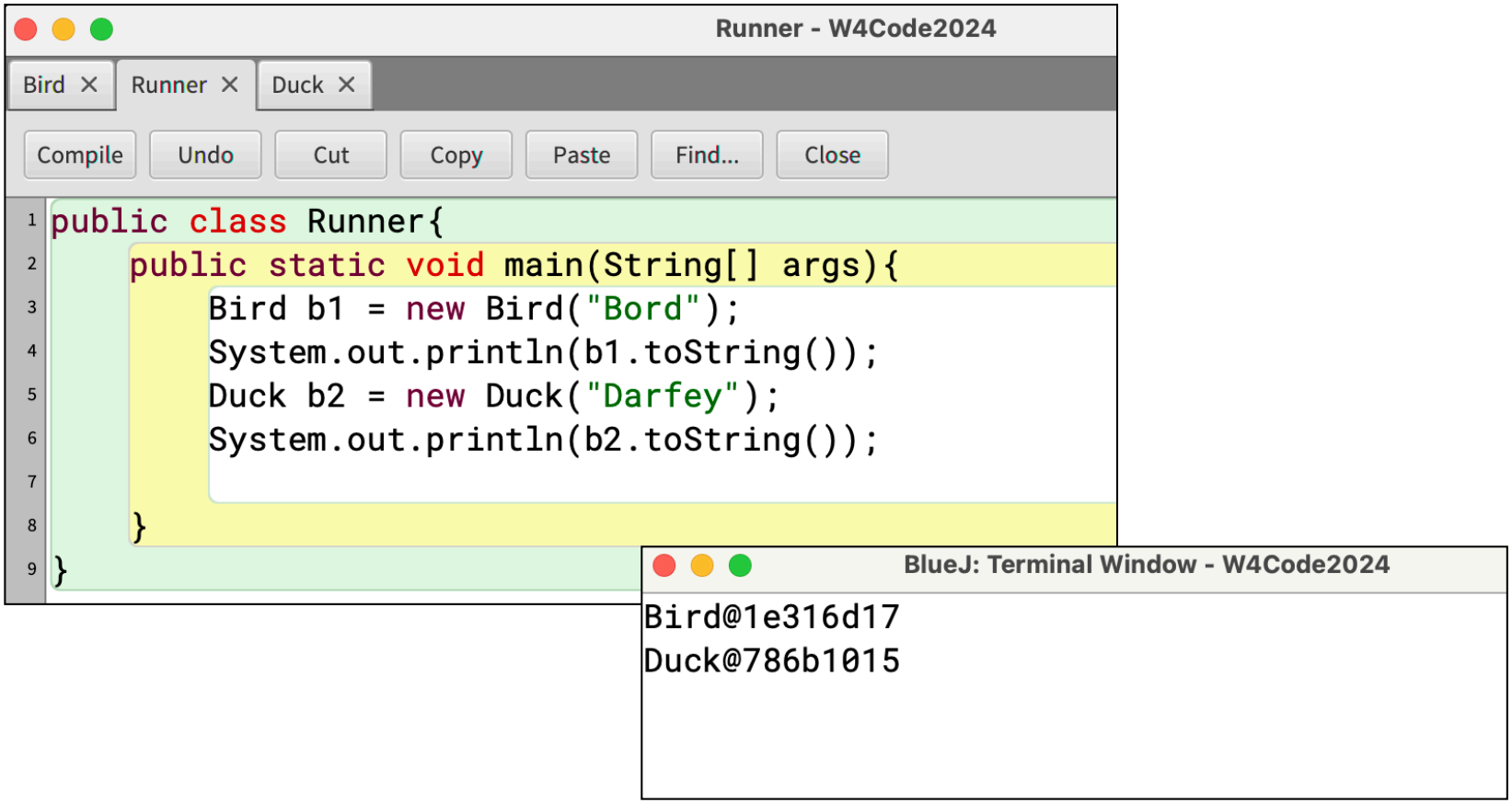
The Duck class must be modified as follows:



Recall that the superclass (Bird) constructor now requires a String and this is why a constructor must be provided in the subclass (Duck) that explicitly passes up a String to the parent constructor via super(..)

Example - Inheritance (Slide 5)

The Runner class will need to be modified to use the new constructors, but the output remains similar (i.e. from the Object toString() method).



Object Class: toString( )

toString

```
public String toString()
```

Returns a string representation of the object. In general, the `toString` method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read. It is recommended that all subclasses override this method.

The `toString` method for class `Object` returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@', and the unsigned hexadecimal representation of the hash code of the object. In other words, this method returns a string equal to the value of:

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

Returns:

a string representation of the object.

This means that every object has a toString( ) method that is inherited from the Object class. We are encouraged to override this method with one that provides a meaningful representation of the object.

String API: https://docs.oracle.com/en/java/javase/17/docs/api/

Method Replacement and Refinement

A subclass can change the methods inherited from its superclass. The method signature has to be maintained for this to happen.

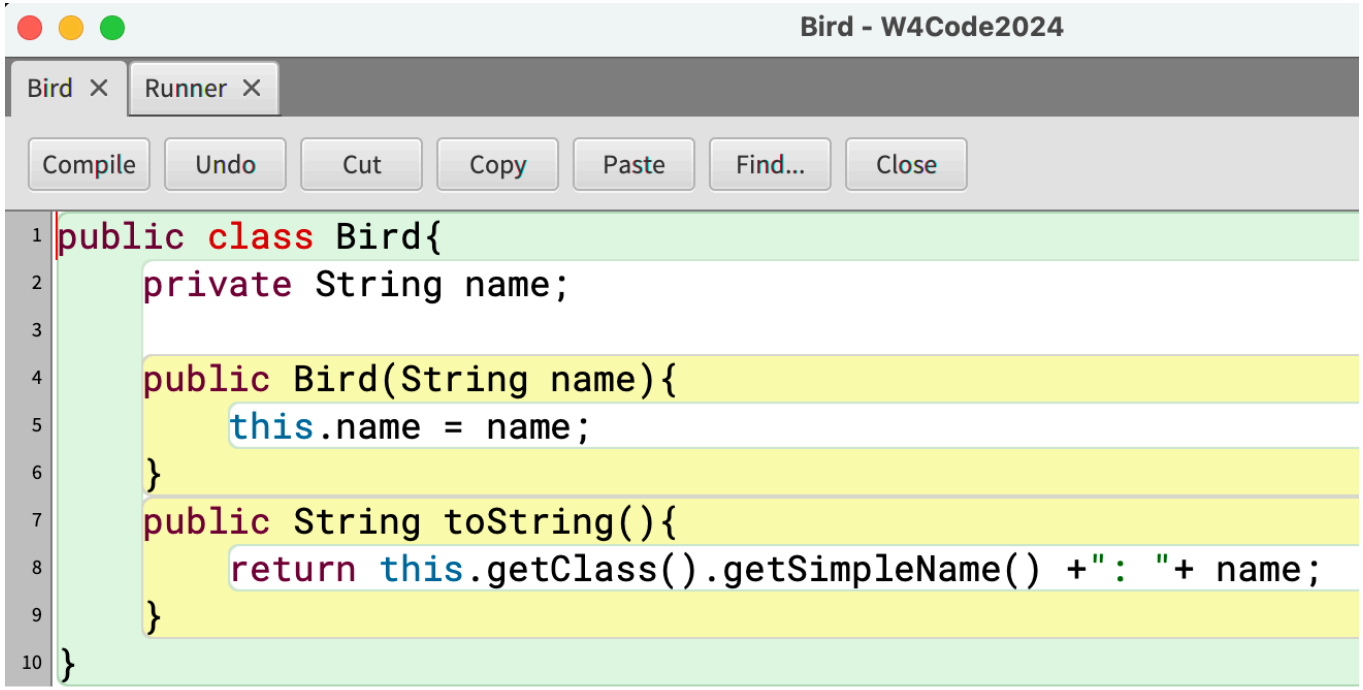
- Method Replacement
- Method Refinement

Method Replacement

- Method Replacement: the method in the subclass replaces the inherited behaviour.
- The superclass code is never executed, even though it still exists.
- New behaviour is introduced which becomes the default.
- Method replacement is also called method overriding.

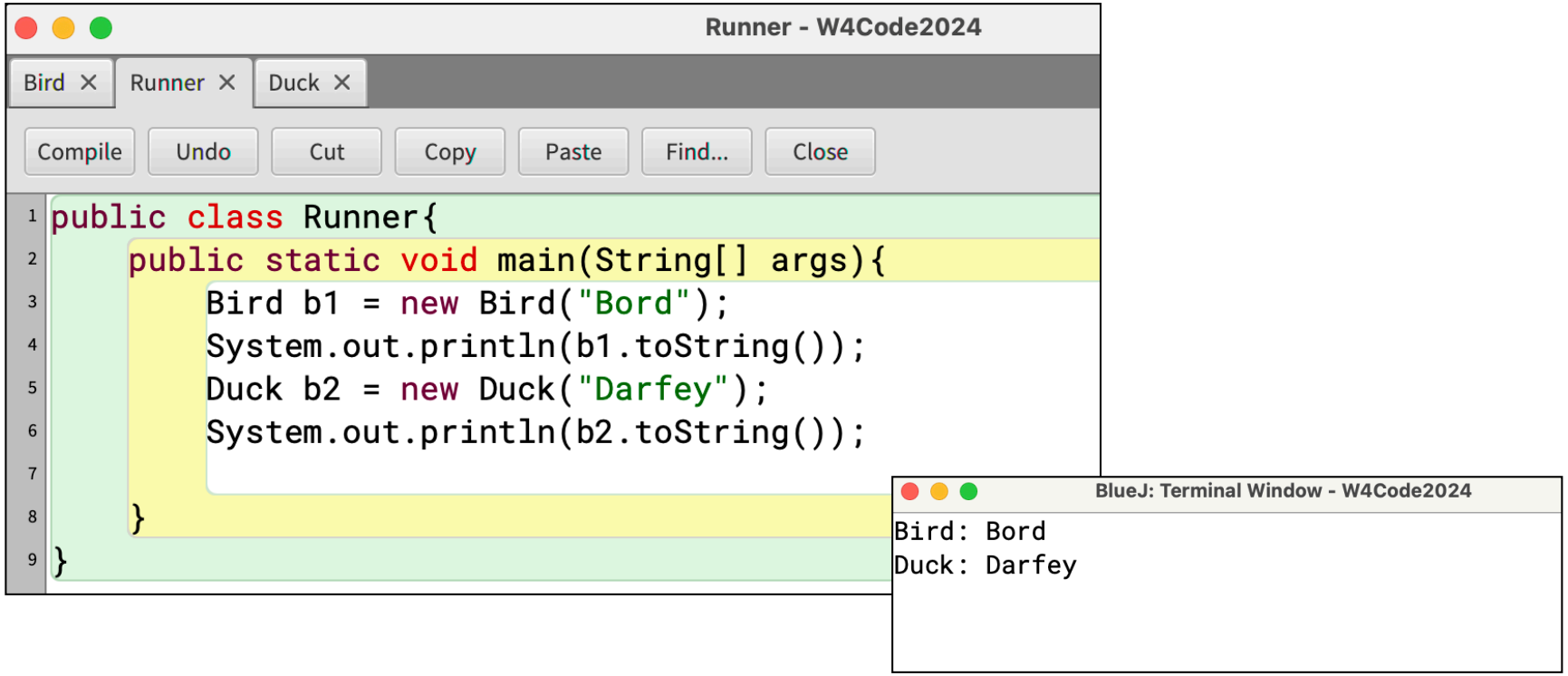
Example - Method Replacement

If the toString( ) is modified with custom logic as shown below, the output is different. This happens because the parent toString( ) is not called. Instead, the custom toString( ) intercepts the invocation and it is called instead. The behaviour of the parent's toString( ) is never called.



Example - Method Replacement

The Runner class looks like this:



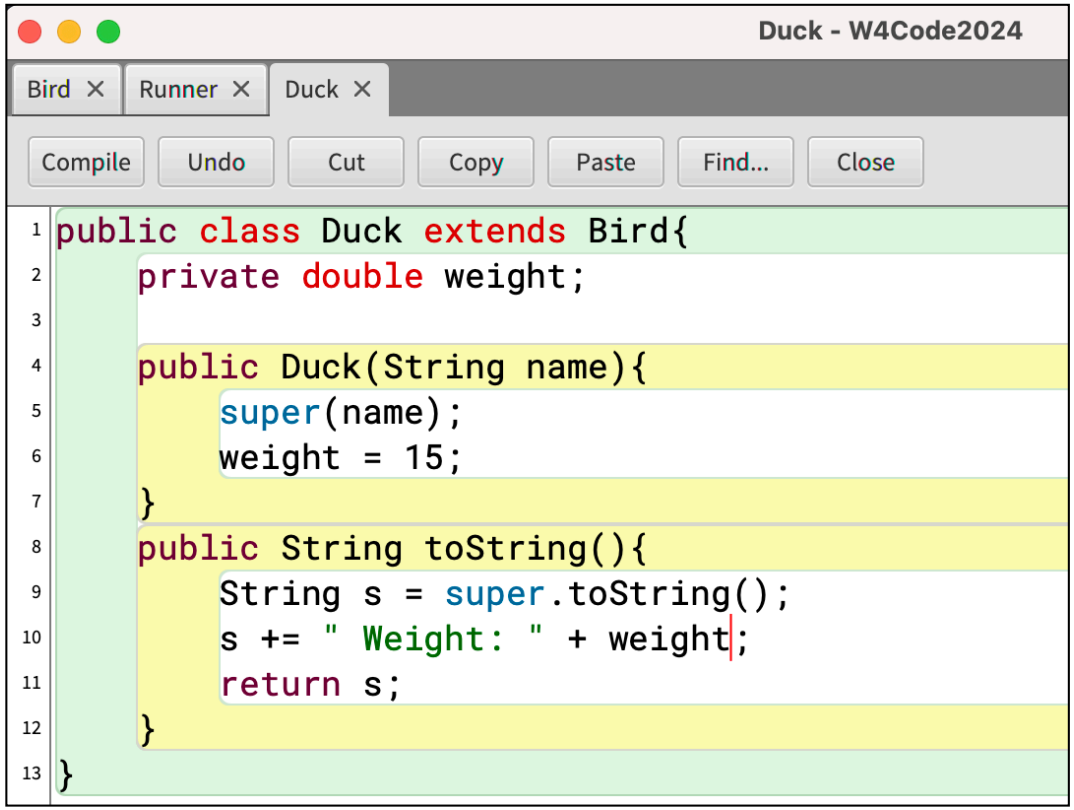
The Bird object's name is printed along with the object's type (Bird). This also occurs in the Duck class because the Duck class inherits the custom toString( ) behaviour of the Bird class.

Method Refinement

- Method Refinement: the method in the subclass adds some extra behaviour of its own while maintaining the original behaviour that was inherited. The superclass code is executed with some extra code in the subclass.
  - The keyword `super` is used to invoke the superclass' method within the refined method in the subclass.

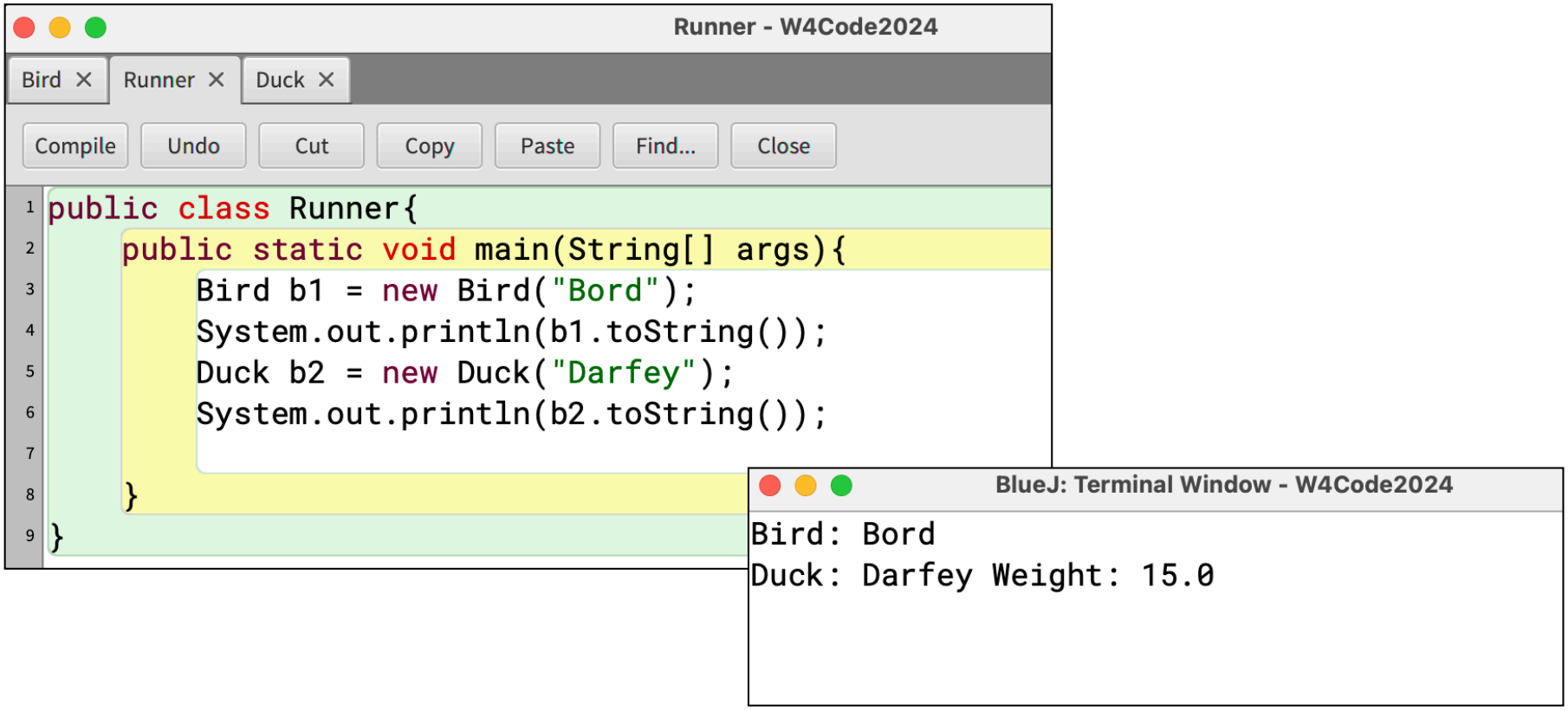
Example - Method Refinement

Suppose the Duck class reuses some the parent toString( ) behaviour but adds on specific data of its own.



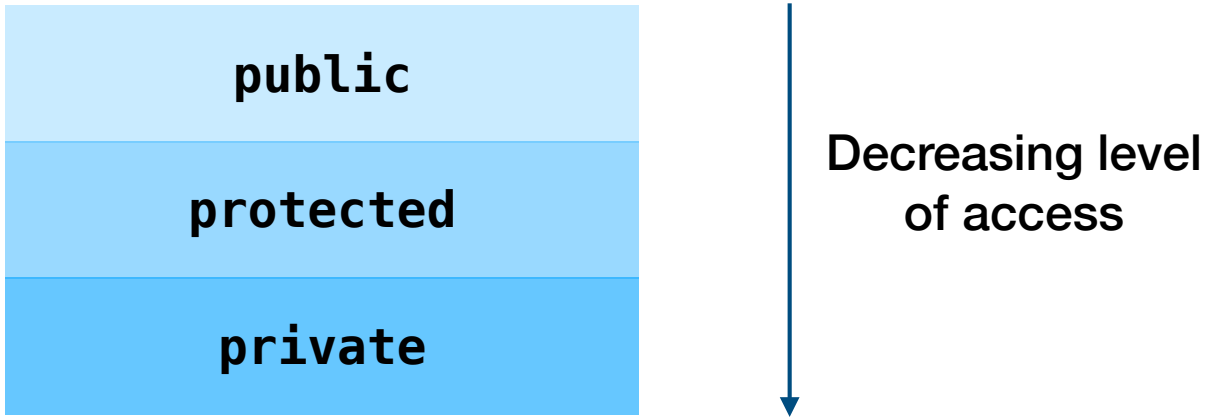
The parent's method is called via super.toString( ) within the refined Duck toString( ) method. If super is omitted, this degrades incorrectly to a recursive call, so be careful!

Example - Method Refinement



The Duck object's name is printed along with the object's type (Duck), but this time the inherited custom toString( ) behaviour of the Bird class is refined in the Duck class with extra data (about weight).

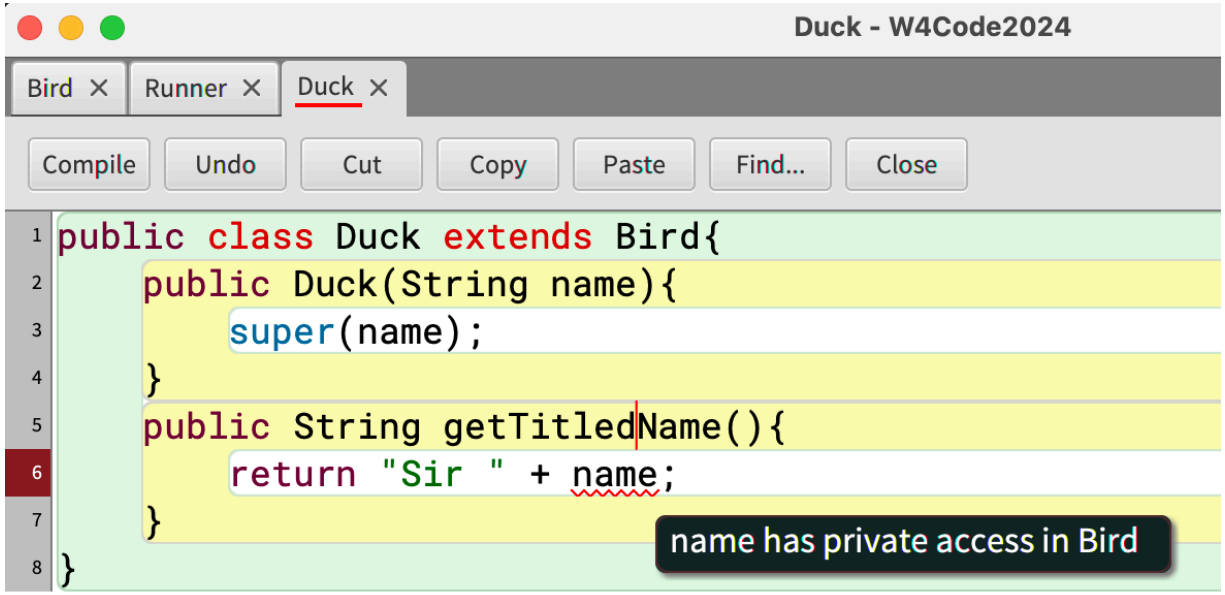
Access Modifiers



- Public: A declaration that is accessible to all clients
- **Protected**: A declaration that is accessible only to the class itself, its **subclasses**, and its friends
- Private: A declaration that is accessible only to the class itself and its friends

Example - Inheritance (Slide 5)

Suppose the Duck class is modified as follows:



A compilation error occurs because the name variable, though inherited, is not accessible in the subclass. The parent class can be modified in two ways to fix this: (1) use the protected access modifier for name or (2) provide an accessor for name.

Inherited Attributes and Methods

The access modifiers (protected, public) allow subclasses to use the inherited methods and the attributes from the superclass.

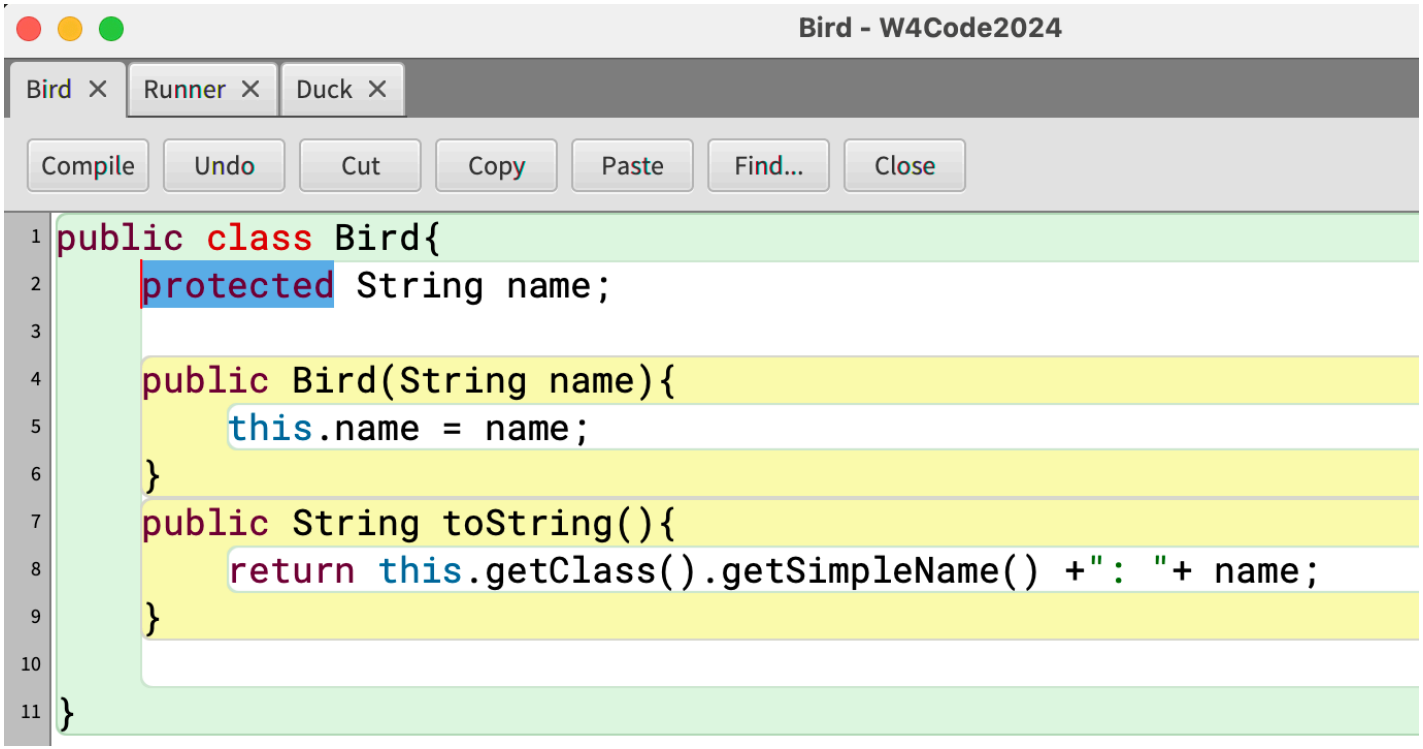
```
public class Bird{
  protected String name;
  //… rest of class
}
```

Solution #1: change the access modifier in the superclass to protected.

```
public String getTitledName(){
  return "Sir " + getName( );
}
```

Solution #2: use the accessors and mutators

Inheritance - Access Modifiers



The Duck class no longer has an error since the access control has been loosened to allow subclasses to direct use the name variable. This applies to methods as well.

Preventing Overriding

Child classes can be prevented from overriding a parent's inherited method.

This means that the method implementation in the parent class will always be used whenever it is called on any child instance.

Careful consideration must be done since all child classes would have that particular behaviour.

Preventing Overriding

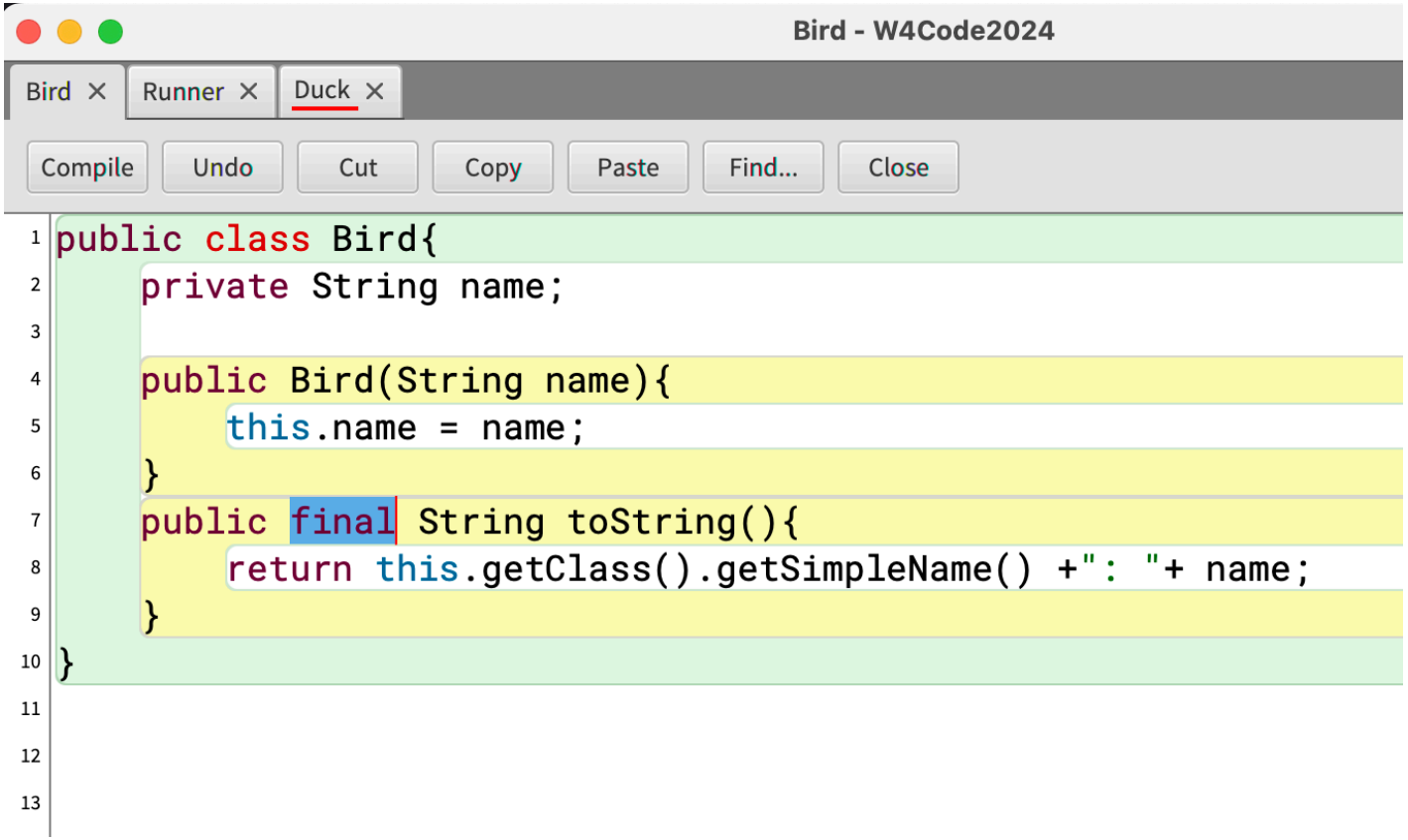
A final method takes the following form:

```
access-modifier final return-type method-
Name( parameter-type parameter-name);
```

Example: In the Bird class

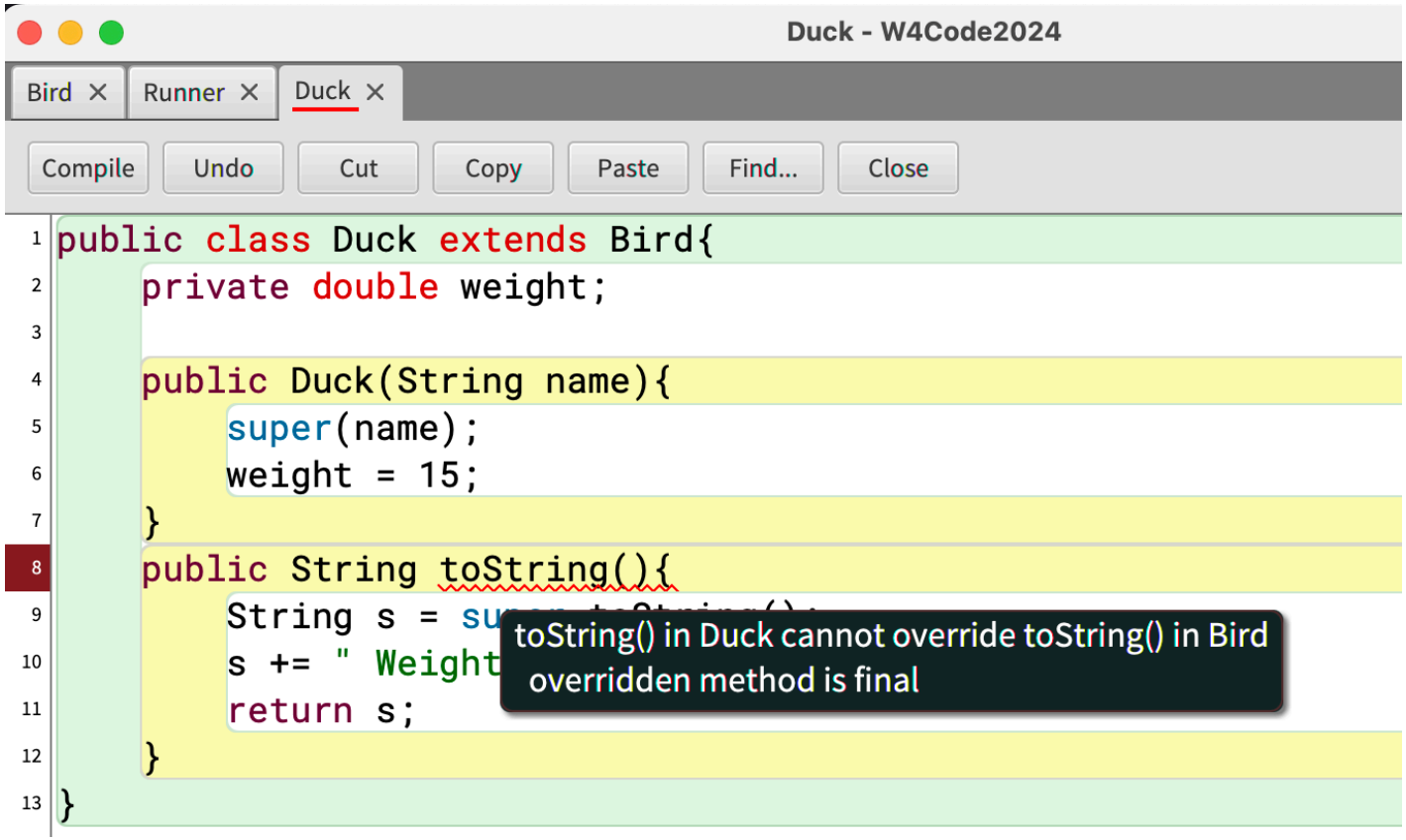
```
public final String toString( ){
  // code
}
```

Example - Preventing Overriding



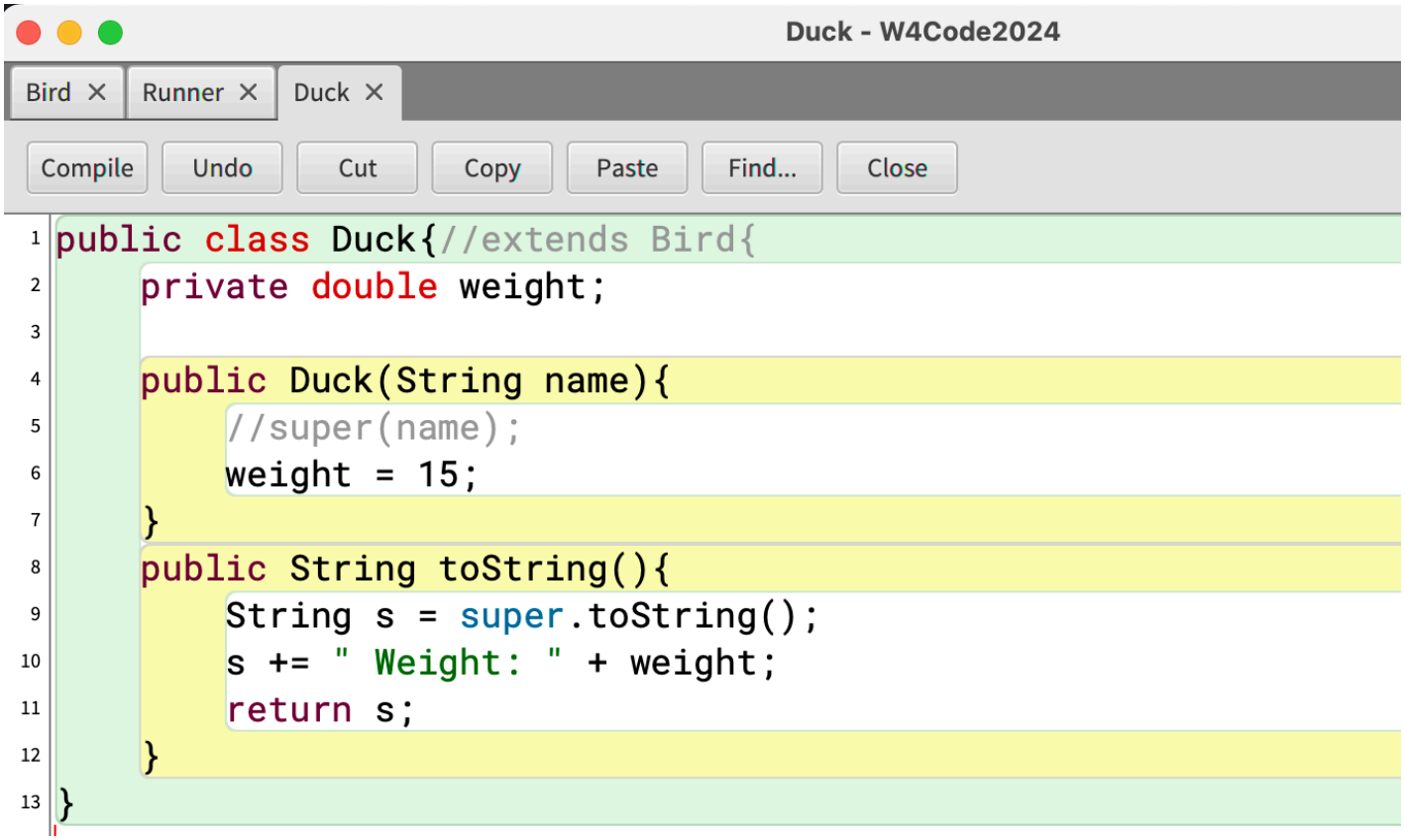
If final is applied to toString( ) method's signature, it means that the method cannot be overridden in any subclass. The Duck class throws an error as a result.

Example - Preventing Overriding



The error occurs because the Duck class is a subclass of the Bird class.

Example - Preventing Overriding



Suppose the Duck is disconnected as a subclass (Line 1 and 5 comments).

Example - Preventing Overriding

Runner - W4Code2024

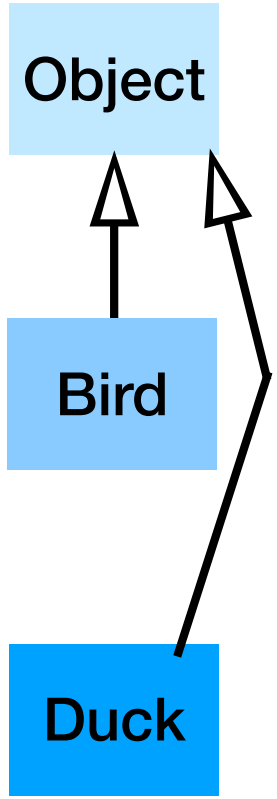
Bird × Runner × Duck ×

Compile Undo Cut Copy Paste Find... Close

```
1  public class Runner{
2      public static void main(String[] args){
3          Bird b1 = new Bird("Bord");
4          System.out.println(b1.toString());
5          Duck b2 = new Duck("Darfey");
6          System.out.println(b2.toString());
7
8      }
9  }
```

BlueJ: Terminal Window - W4Code2024

```
Bird: Bord
Duck@4a70a4a2 Weight: 15.0
```



The output for the Duck object changes. Since the Bird is no longer a parent, the Duck now refines the Object parent's toString( ).

Preventing Inheritance

Inheritance of an entire class can be prevented by using the **final** keyword as well.

This means that the class will not be subclassed at all.

Preventing Inheritance

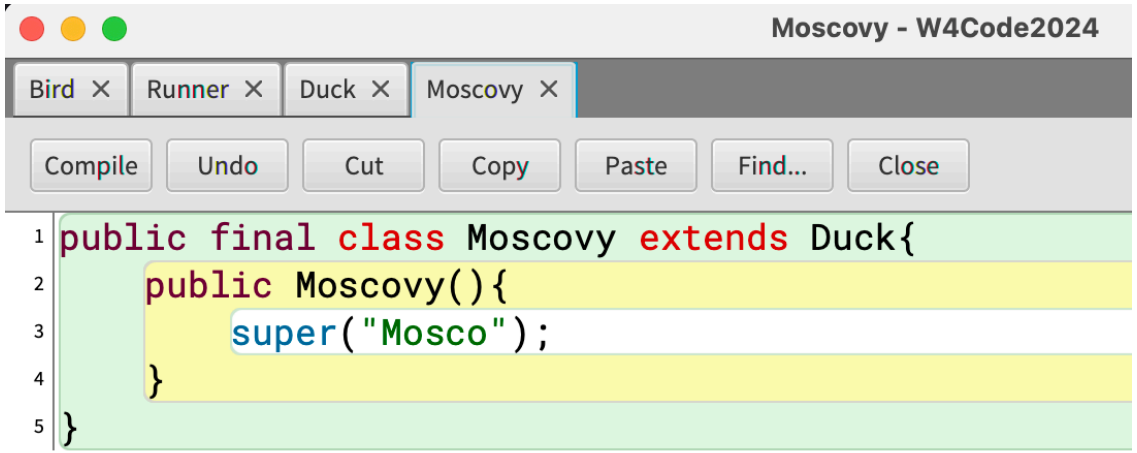
A final class takes the following form:

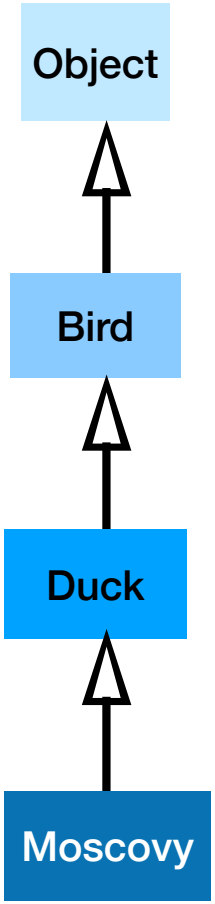
```
access-modifier final class className
```

Example:

```
public final class Moscovy extends Duck{
 // Moscovy class body
}
```

Example: Preventing Inheritance



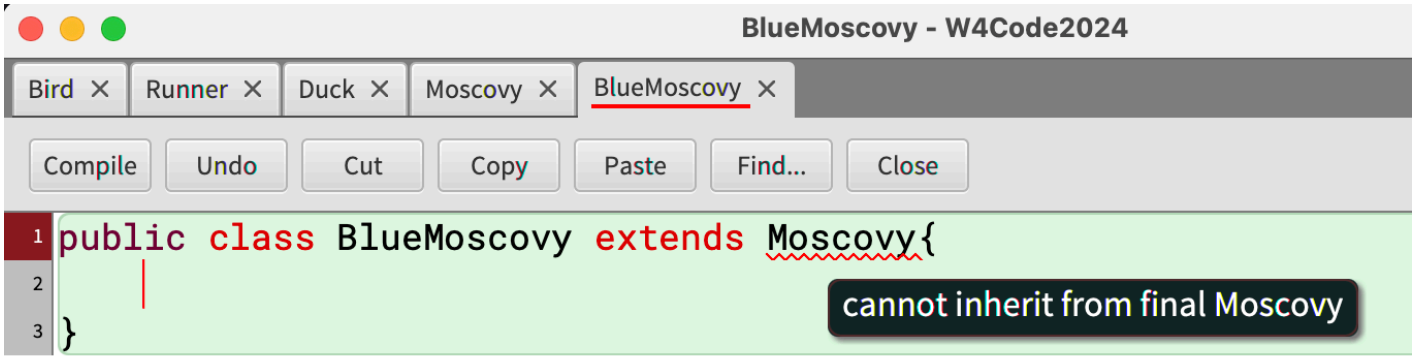


Suppose the Duck and Bird classes are restored to their code prior to Slide 28. Suppose a Moscovy class is introduced as a subclass of Duck but it is made **final**.

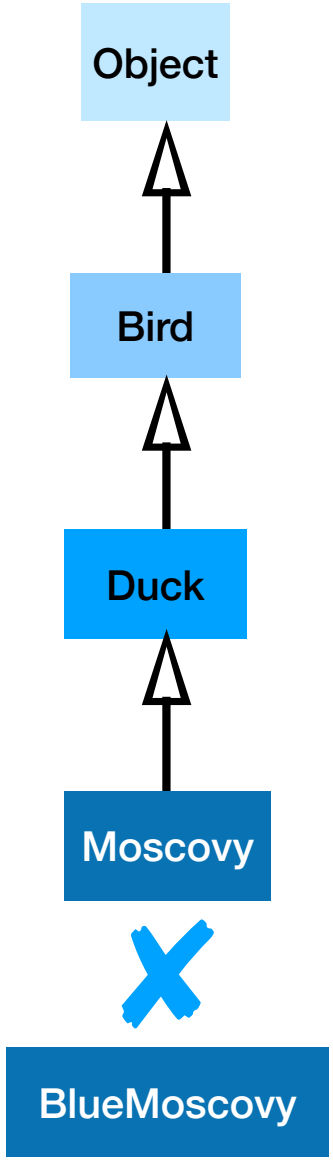
At this point, no subclasses of Moscovy can be created.

https://breeds.okstate.edu/poultry/ducks/muscovy-ducks.html

Example: Preventing Inheritance



If we try to connect a new subclass BlueMoscovy to the Moscovy class, this causes a syntax error.



https://breeds.okstate.edu/poultry/ducks/muscovy-ducks.html

Questions - Inheritance (Slide 5)

1. Explain why the following is TRUE or FALSE

   (a) Wren is a subclass of Bird

   (b) Cat is a superclass of Lion

   (c) Pug and Boxer are superclasses

   (d) Object is not a subclass

2. Fill in the blanks

   (a) Duck has _____ superclass(es).

   (b) Cat is a superclass of ____ subclasses

   (c) Mongrel has _____ sibling class(es)

   (d) Object has ____ subclass(es)

Questions - Inheritance (Slide 5)

3. Match the keyword with its purpose

(a) protected	I. Used to refer to a parent class from child class
(b) super	II. Provides access to an attribute or method to a child class
(c) super( )	III. Creates a subclass relationship in a class signature
(d) extends	IV. Prevents subclassing and method overriding
(e) final	V. Used to invoke a direct parent constructor from a child class

4. Construct the class signatures for the following classes

(a) Dog: _____

(b) Boxer: _____

(c) Mongrel (but it cannot be subclassed):_____

Summary

Today you learned about:

- Inheritance
  - Creating and Manipulating Subclass Instances
  - Constructors
  - Method Refinement
  - Method Replacement
  - Preventing subclassing and overriding

References

- Booch, G. (2007) Object-Oriented Analysis and Design. Chapter 2 - the Object Model
- Chapter 2 Objects: Using, Creating, and Defining: https://runestone.academy/ns/books/published/javajavajava/chapter-objects.html
- Chapter 3 Methods: Communicating With Objects: https://runestone.academy/ns/books/published/javajavajava/chapter-methods.html