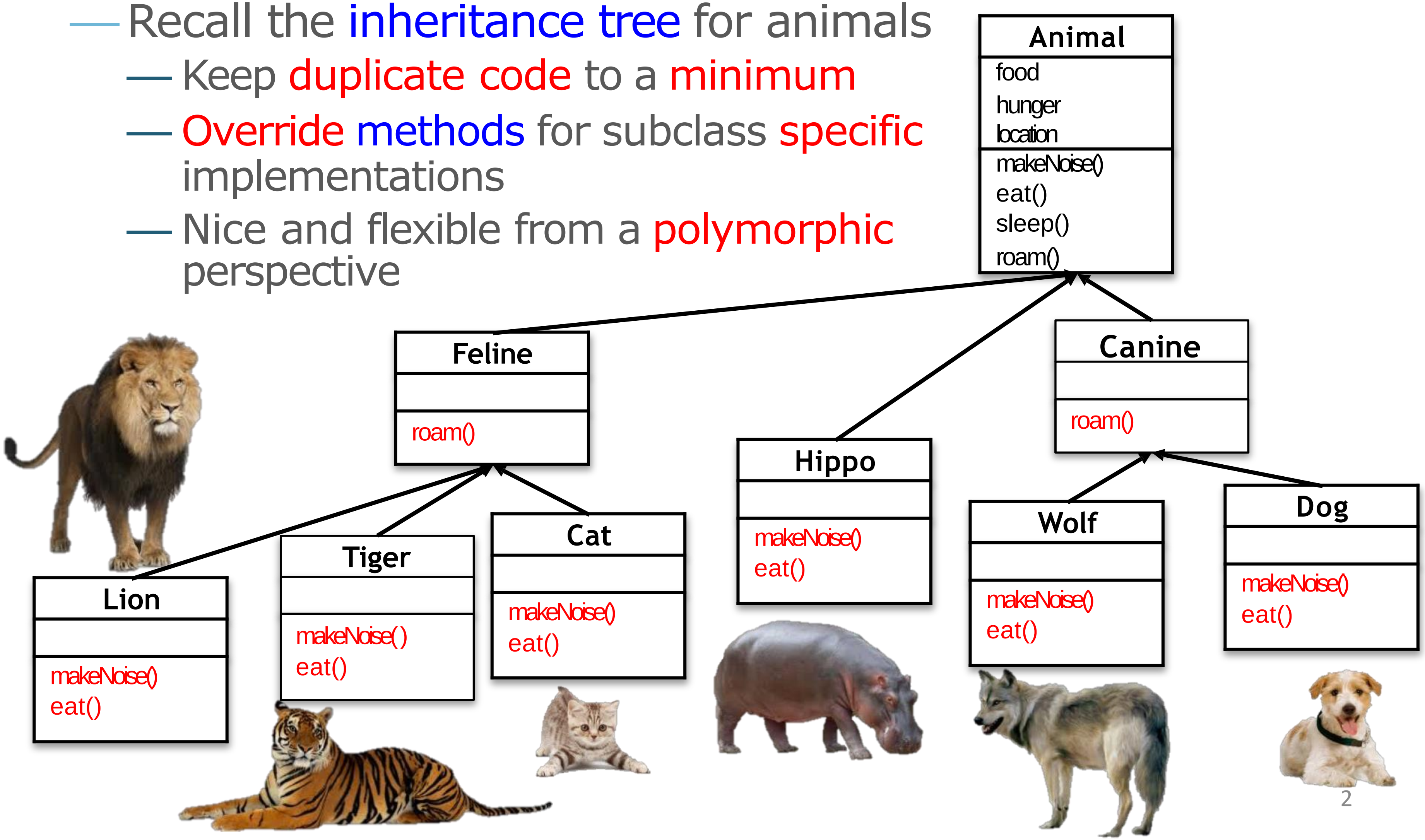
Chapter 7.

Abstract Classes and Interfaces

2020-2021
COMP2396 Object-Oriented Programming and Java
Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)
Department of Computer Science, The University of Hong Kong

The Animal Inheritance Tree

- Recall the inheritance tree for animals
  - Keep duplicate code to a minimum
  - Override methods for subclass specific implementations
  - Nice and flexible from a polymorphic perspective



Animal
food
hunger
location
makeNoise()
eat()
sleep()
roam()

Feline
roam()

Canine
roam()

Hippo
makeNoise()
eat()

Lion
makeNoise()
eat()

Tiger
makeNoise()
eat()

Cat
makeNoise()
eat()

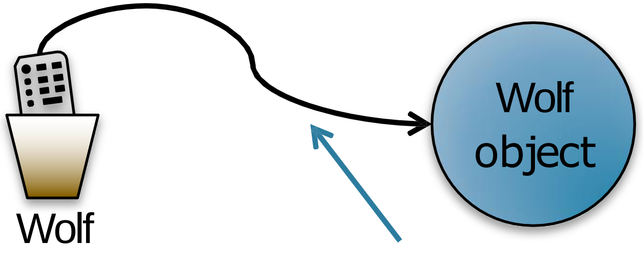
Wolf
makeNoise()
eat()

Dog
makeNoise()
eat()

The Animal Inheritance Tree

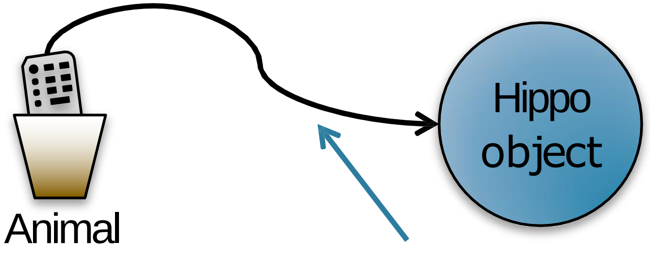
- Examples of instantiating a class from the tree

```
Wolf aWolf = new Wolf();
```



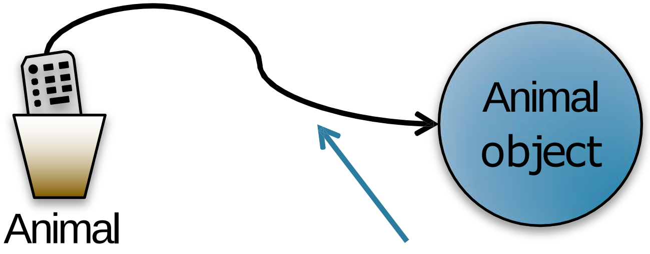
A Wolf reference to a Wolf object

```
Animal aHippo = new Hippo();
```



An Animal reference to a Hippo object

```
Animal anim = new Animal();
```



What does an Animal object look like?

An Animal reference to an Animal object

Abstract Class

- It makes sense to create a Wolf object, a Hippo object or a Tiger object
- What exactly is an Animal object? How does it look like? What is its shape, color and size? How many legs…
- The Animal class is needed for inheritance and polymorphism
- Programmers are, however, supposed to instantiate only the less abstract subclasses of the Animal class, but not the Animal class
- A simple way to prevent a class from ever being instantiated is by marking it as abstract

Abstract Class

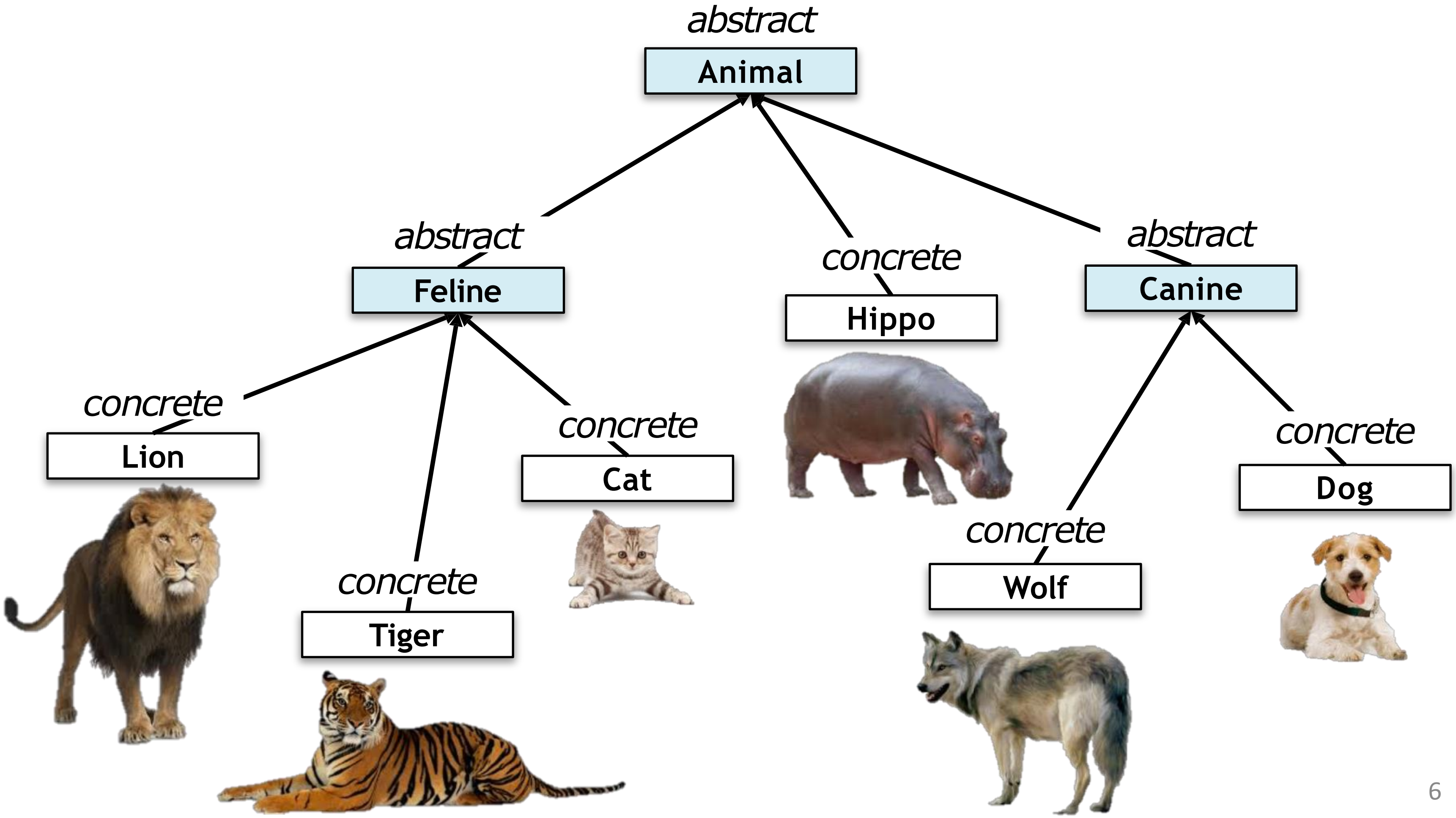
- The keyword abstract is used to define an abstract class

```
abstract class Canine extends Animal
   {  public void roam() { ... }
}
```

- The compiler will stop anyone from instantiating an abstract class using the new operator
- An abstract class can be used as a reference type (e.g., using it as a polymorphic argument or return type, or to make a polymorphic array)
- When designing an inheritance tree, one must decide which classes are abstract and which are concrete (concrete classes are those that are specific enough to be instantiated)

Abstract Class

- Example: Which classes are abstract / concrete?



abstract
Animal

abstract
Feline

concrete
Hippo

abstract
Canine

concrete
Lion

concrete
Tiger

concrete
Cat

concrete
Wolf

concrete
Dog

Abstract Method

- Besides classes, methods can also be marked as abstract
- An abstract class means the class must be extended, whereas an abstract method means the method must be overridden
- An abstract method has no method body, just ends with a semicolon

```
public abstract void eat();
```

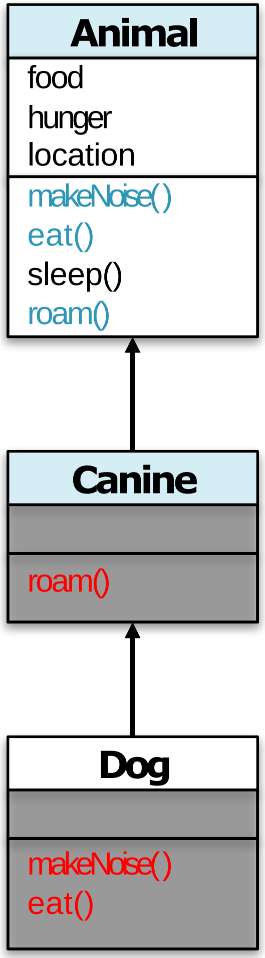
- A class must be marked as abstract if it has at least one abstract method. It is illegal to have an abstract method in a non-abstract class!
- An abstract class, on the other hand, can have either or both abstract and non-abstract methods

Abstract Method

- Mark a method as abstract when you cannot come up with any generic code that subclasses would find useful (e.g., makeNoise() and eat() in Animal class)
- Abstract methods are important in the sense that they define part of the protocol for a group of subtypes (subclasses) used in polymorphism
- A concrete class in the inheritance tree must implement all the abstract methods from its superclass
- An abstract class, on the other hand, may implement none or some of the abstract methods from its superclass, leaving the rest to its subclasses to complete the implementation
- Implementing an abstract method in a subclass is just like overriding a method

Abstract Method

- Example
  - Suppose makeNoise(), eat() and roam() are abstract methods in the abstract class Animal
  - Canine extends Animal
  - Since Canine is also an abstract class, it can choose to implement either none, some or all of the abstract methods from Animal
  - Canine only implements the roam() method from Animal
  - Dog extends Canine
  - Since Dog is a concrete class, it must therefore implement all the abstract methods from Canine, including those Canine inherited from Animal (i.e., makeNoise() and eat())



Pet Shop Program

- Ocean was asked to design a pet shop program
- He would like to modify the Animal inheritance tree he designed previously and introduce some pet behaviors such as `beFriendly()` and `play()`
- He came up with a few options
  1. Put the pet methods in the Animal class
  2. Put the pet methods in the Animal class and make them abstract
  3. Put the pet methods only in the classes where they belong (e.g., in Dog class and Cat class)

Pet Shop Program: Option 1

- Pros:
  - All animal subclasses will instantly inherit the pet behaviors
  - No need to touch the existing animal subclasses
  - Any future animal subclasses will also get to take advantage of inheriting these methods
  - The Animal class can be used as a polymorphic type in any program that wants to treat animals as pets
- Cons:
  - Not all animals can be kept as pets (Could be dangerous to give non-pet animals the pet methods!)
  - Almost certainly will have to touch the pet classes like Dog and Cat because they tend to implement pet behaviors very differently

Pet Shop Program: Option 2

- Pros:
  - Give us all the benefits of Option 1, but without the drawback of having non-pet animals running around with pet behaviors
  - All subclasses must implement the pet methods, but the non-pet classes can make these methods do nothing
- Cons:
  - A waste of time to implement all the pet methods in the non-pet classes
  - Every non-pet class would still announce to the world that it, too, has the pet methods, even though these methods wouldn’t actually do anything when being called

Pet Shop Program: Option 3

- Pros:
  - The pet methods are where they belong, and ONLY where they belong
- Cons:
  - There is no contract for the pet classes
  - The compiler has no way to check if you have implemented the methods correctly
  - Cannot exploit polymorphism for the pet methods

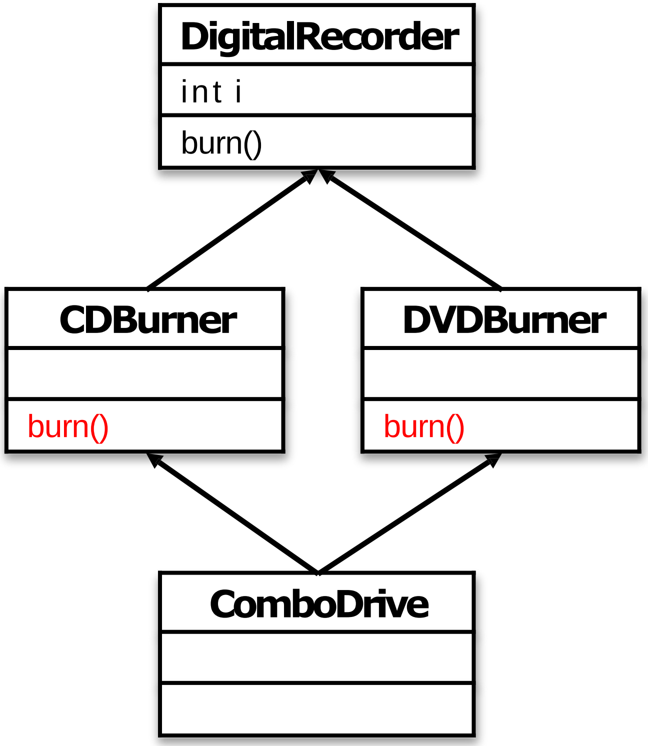
Pet Shop Program

- What we really need is
  - A way to have pet behaviors in just the pet classes
  - A way to guarantee that all pet classes have all of the same pet methods defined
  - A way to take advantage of polymorphism so that all pets can have their pet methods called, without having to use specific arguments, return types, and arrays for each and every pet class

How about making a new abstract superclass called Pet, giving it all the pet methods, and making Dog and Cat also a subclass of Pet?

Deadly Diamond of Death

- The “two superclasses” approach is called multiple inheritance
- It has a problem known as the Deadly Diamond of Death (DDD)
- Example
  - CDBurner and DVDBurner both extend DigitalRecorder
  - Imagine that the instance variable i is used by both CDBurner and DVDBurner, with different values
  - What happens if ComboDrive need to use both values of i?
  - Which burn() method runs when you call burn() on the ComboDrive?



Interface

- Java does not allow multiple inheritance
- Java solves the problem of multiple inheritance through the use of an interface
- A Java interface is like a 100% pure abstract class, with all methods being abstract
- A Java interface is defined using the keyword interface

```
public interface Pet {
   public abstract void beFriendly();
   public abstract void play();
}
```

Interface methods are implicitly public and abstract, so typing in ‘public’ and ‘abstract’ is optional

Interface

- With the interface methods being abstract, a class ‘implementing’ the interface must implement all the interface methods
- At runtime, the JVM will therefore not be confused about which of the 2 inherited versions it is supposed to call
- A class implements an interface using the keyword implements

```
public class Dog extends Canine implements Pet {
   public void beFriendly() { … }
   public void play() { … }

   public void makeNoise() { … }
   public void eat() { … }
}
```

implement Pet methods

normal overriding methods

Interface

- An interface can be used as a polymorphic type
- Classes that implement an interface can come from anywhere in the inheritance tree, or even from completely different inheritance trees
- An object can be treated by the role it plays, rather than by the class type from which it was instantiated
- Example: Any object that needs to be able to save its state to a file should implement the Serializable interface
- Better still, a class can implement multiple interfaces

```
public class Dog extends Canine implements Pet, Serializable { … }
```

General Design Rules

- Make a class that does not extend anything when your new class does not pass the IS-A test for any other type
- Make a subclass only when you need to make a more specific version of a class and need to override or add new behaviors
- Use an abstract class when you want to define a template for a group of subclasses, and you have at least some implementation code that all subclasses could use
- Make a class abstract when you want to guarantee nobody can make objects of that type
- Use an interface when you want to define a role that other classes can play, regardless of where these classes are in the inheritance trees

Chapter 7.

End

2020-2021
COMP2396 Object-Oriented Programming and Java
Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)
Department of Computer Science, The University of Hong Kong