

1.00 / 1.001

Tutorial 5

Week of October 10, 2005

Inheritance, Abstract Classes,
Interfaces

Topics

- Inheritance
- Method Overriding
- Abstract Classes
- Interfaces
- Problem Set 4
- Method Overloading

Inheritance Basic Questions

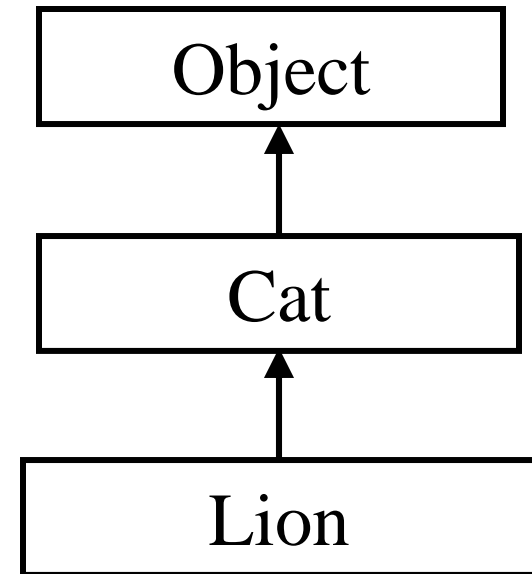
Use the following example to help you answer these questions:

- From which Java class do all other classes inherit implicitly?
- Which Java keyword is used to explicitly cause inheritance from another class?

Inheritance Example

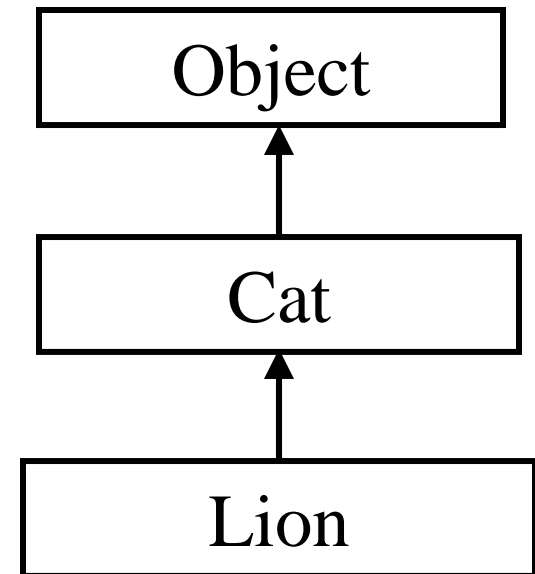
```
public class Cat {
    private int age;
    public Cat(int age){
        this.age = age;
    }
    public void pounce(){
        //implementation hidden
    }
}

public class Lion extends Cat{
    private String pride;
    public Lion(int age, String pride){
        //constructor (incomplete)
    }
}
```



Question 1

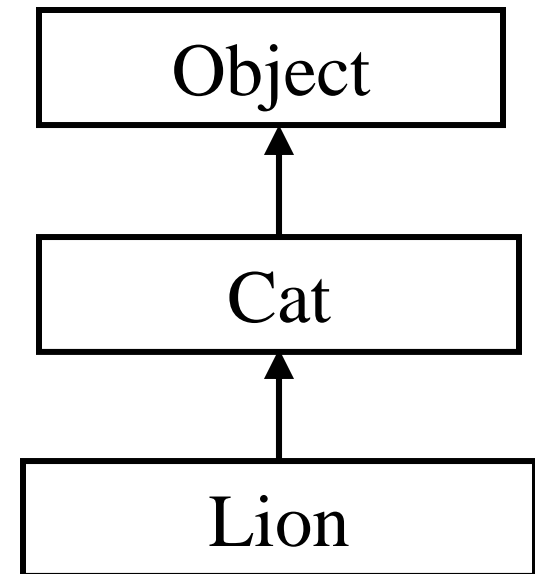
- What are the types of instances of
 - class `Cat`?
 - class `Lion`?
- What fields can each of the above classes access directly?



Question 1: Answers

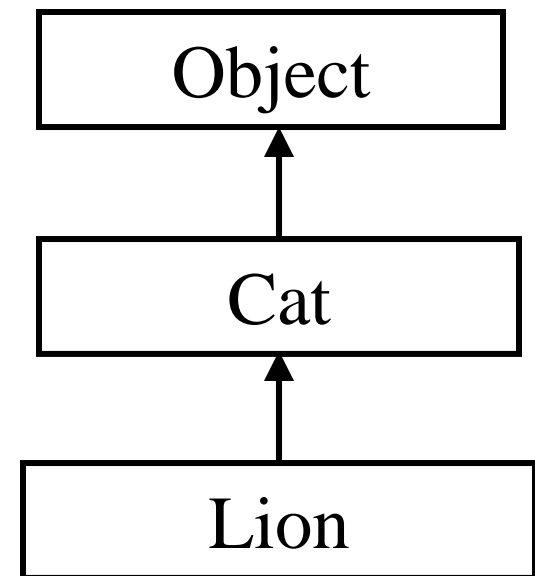
- Types

- Cat: Object, Cat
- Lion: Object, Cat, Lion



Question 1: Answers (cont'd)

- What fields can each of the above classes access directly?
 - Cat can access age
 - Lion can access pride
 - Lion has a data member age but it cannot access it directly, because it is inherited from Cat and is private.
 - Note: there are no fields in Object



Question 2

- Complete the constructor for the Lion class
 - How would you set the `age` data member for `Lion` ?
- Which of the following declarations is NOT allowed and why?
 - `Cat a1 = new Cat(79.6);`
 - `Cat a2 = new Lion(100.0);`
 - `Lion a3 = new Lion(100.0);`
 - `Lion a4 = new Cat(100.0);`

Question 2: Answers

- Complete the constructor for the Lion class

```
public class Lion extends Cat{
    private String pride;
    public Lion(int age, String pride){
        super(age);
        this.pride = pride;
    }
}
```

Question 2 – Answers (cont'd)

- Which of the following declarations is NOT allowed and why?

- `Cat a1 = new Cat(2);`
- `Cat a2 = new Lion(10, "SP5");`
- `Lion a3 = new Lion(10, "SP10");`
- `Lion a4 = new Cat(10);`

The object reference should be the same type as the object it refers to, or it should be a supertype.

Imagine what would happen if the assignment were legal and you wrote the code: `a4.getPride()`

[assuming that there is an accessible `getPride()` method in `Lion`]. Would the `Cat` object be able to execute that method?

Inheritance Basic Questions - Answers

- The subclass (e.g. `Lion`) inherits all methods and data members from its *superclasses* (e.g. `Cat`).
 - If the `Cat` class defines a `pounce()` method, the `Lion` class has one too.
 - The `Lion` class can override a method in `Cat` provided the method is not declared as `final`. Note that overriding a method is not the same as overloading a method.
- The keyword `extends` is used to express the hierarchical relationship.

The Big Question

- Why Inheritance?

The Answer

- Java models the hierarchical nature of categories with *inheritance*.
 - It helps you reuse components (code) *consistently*
 - It helps you to manage complexity (writing code where it makes sense and reusing it elsewhere in subclasses by using the superclass's methods)

Method Overriding & Inheritance

```
public class Lion extends Cat {  
    private String pride;  
    //constructor hidden  
  
    public void pounce() {  
        System.out.println("pounce() method of Lion");  
    }  
}
```

```
public class Cougar extends Cat {  
    private String family;  
    //constructor hidden  
  
    public void pounce() {  
        super.pounce();  
        System.out.println("pounce() method of Cougar");  
    }  
}
```

Method Overriding cont'd

```
public class Cat {
    private int age;
    //constructor hidden

    public void pounce() {
        System.out.print("pounce() method of Cat" + "\t");
    }

    public static void main(String [] args){
        Cat [] a = {new Lion(10,"SP5"), new Cougar(4,"AJ12")};
        a[0].breath();
        a[1].breath();
    }
}
```

What is the output of the main () method ?

(Note that overloading a method and overriding a method are different processes.)

Method Overriding Answer

Console output:

```
pounce() method of Lion  
pounce() method of Cat      pounce() method of Cougar
```

Abstract Classes

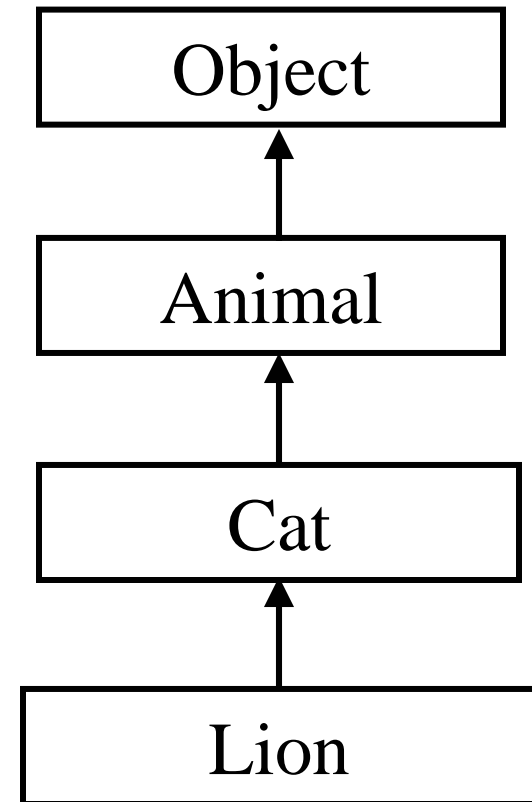
Abstract classes

- cannot be instantiated
- may have static and instance data fields and concrete methods
- may also contain abstract methods
 - Any subclass must implement all of the abstract methods OR it must be abstract itself.

Abstract Class Example

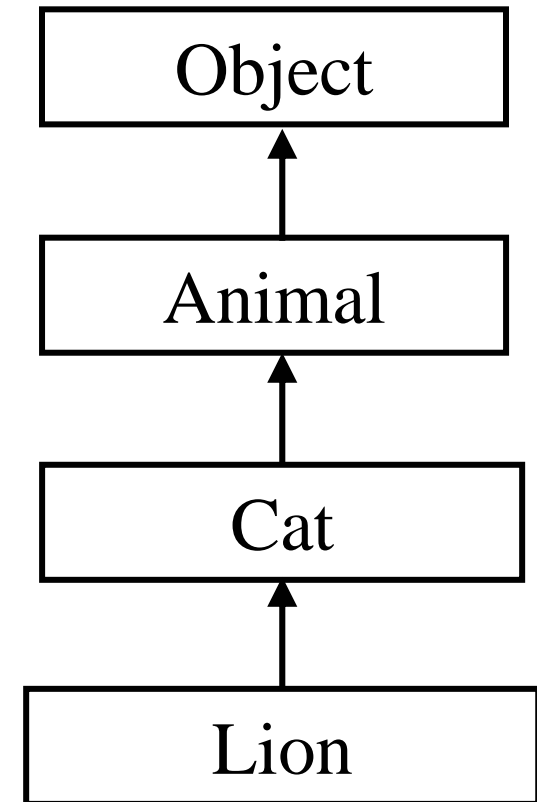
Here, we've modified the previous example. Now **Cat** extends an **abstract class** called **Animal**

```
public abstract class Animal {  
    private String habitat;  
    public Animal(String habitat){  
        this.habitat=habitat;}  
  
    public Animal(){ this("earth"); }  
}  
  
public class Cat extends Animal {  
    //implementation is same as before  
}  
  
public class Lion extends Cat{  
    //implementation is same as before  
}
```



Abstract Class Questions

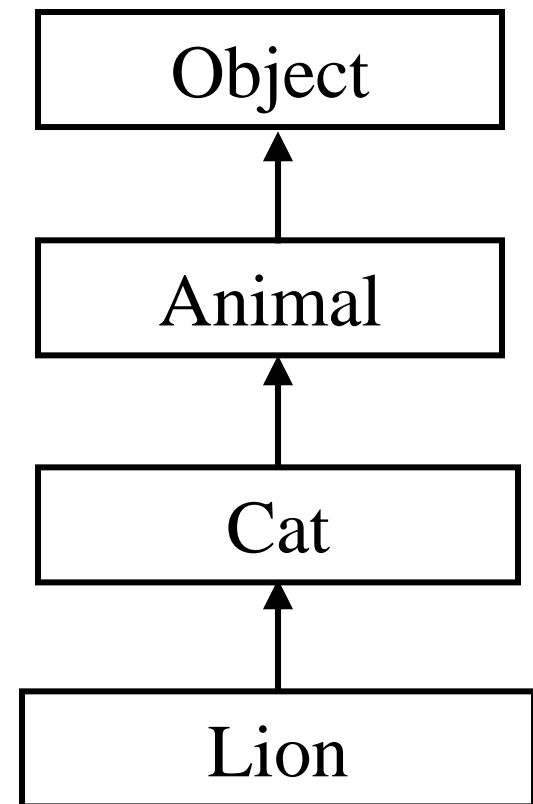
- Can you create an object using the `new` keyword and the constructor of `Animal`? Why/why not?
- Now what are the types of
 - `class Cat`
 - `class Lion`
- What fields can each of the above (`Cat`, `Lion`) classes access directly?



Abstract Class Answers

- Can you create an object using the `new` keyword and the constructor of `Animal`? Why/why not? What makes sense?

The statement `Animal a = new Animal();` is not legal because `Animal` is abstract. It doesn't make sense having an `Animal` that can't do anything – `move()`, `eat()`, etc. Even if we were to have such methods in `Animal`, how would we implement them so that all future subclasses of `Animal` would be able to reuse the code sensibly? Such methods should be abstract within `Animal` and implemented in its concrete subclasses.

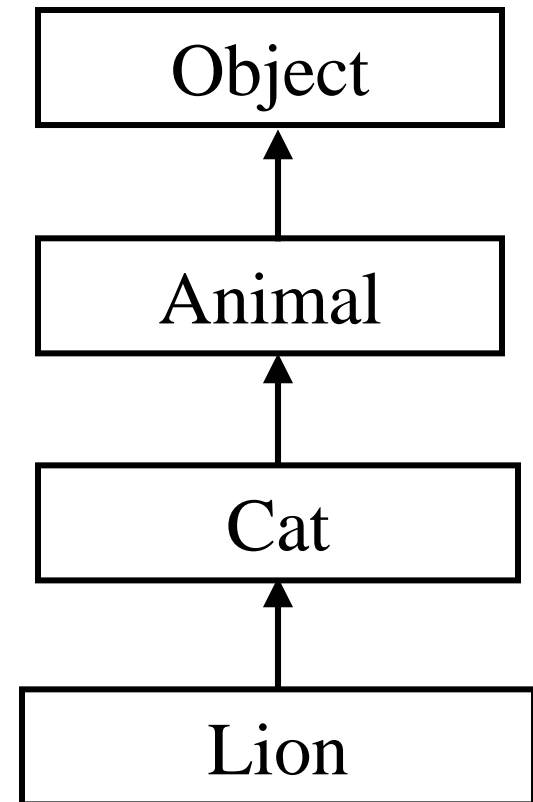


Abstract Class Answers cont'd

- Types

- Cat: Object,
Animal, Cat

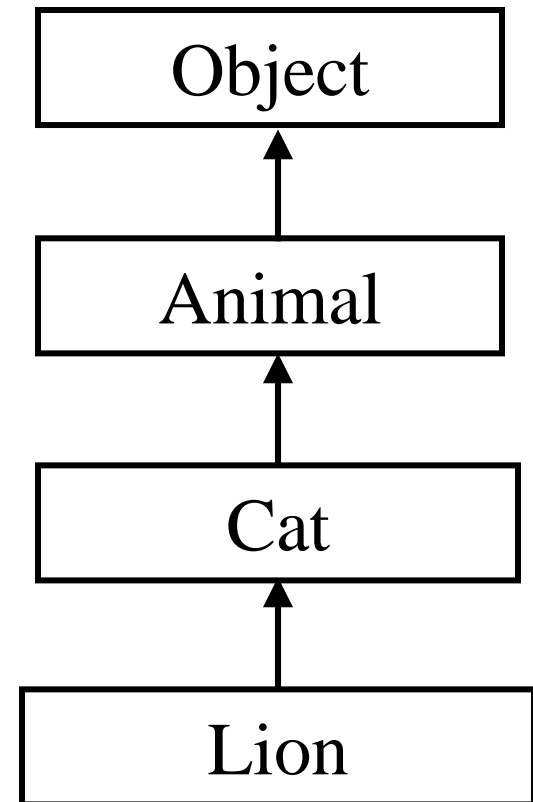
- Lion: Object,
Animal, Cat, Lion



Abstract Class Answers

cont'd

- What fields can each of the classes access directly?
 - Cat can access age
 - Lion can access pride
 - Both Cat and Lion have habitat data members but can only access them using getters/setters.
 - Lion also has an age data member but must use getter/setter methods to access it.



Interfaces Summary

- An interface comprises method declarations and optionally, special data members (public constants).
 - All methods are implicitly abstract - the `abstract` keyword is never actually used.
 - All methods are automatically `public`.
 - All data members are automatically `public static final`.
- An interface describes *what* its implementing classes should do
 - Ensure that some required piece of functionality is present in every implementing class.
 - Allow two totally different kinds of objects with no inheritance relationship to be handled using same code.
- Any class *implementing* a particular interface **must** define the *how*.
- Classes can implement one or more interface.
[Note: classes may *extend* one class only.]

Interface Exercise

- Write an interface called `Endangered`. It has two methods called `getPopulation()` and `isEndangered()` and one data member `threshold` with value `10,000`;
- Now modify the `Lion` class so that it implements the `Endangered` interface.
 - What additional methods are required in the `Lion` class ?
 - What additional data member should be added?

Interface Exercise - Answer

```
public interface Endangered{
    public static int threshold=10000;
    public int getPopulation();
}
```

```
public class Lion extends Cat implements Endangered{
    private int population;
    //constructor may also need to be changed
    //or overloaded.
    public int getPopulation(){
        return population;    }
    public boolean isEndangered(){
        return (population<=threshold);    }
}
```

COMPARISON OF ABSTRACT CLASSES AND INTERFACES

Abstract Class (A)	Interface (M)
Usually used as a base class at the top of a hierarchy (ex: Shape...)	No hierarchy implied. Can be used with disparate objects (ex: IAge, IColor...)
Other class inherit from A (keyword "extends")	Other classes implement M (keyword "implements")
A class can inherit from one abstract class only (multiple inheritance is not supported in Java)	A class can implement multiple interfaces
An abstract class can have instance variables and methods	An interface is usually a collection of method declarations only, but it also supports the declaration of constants (which are automatically final)
Methods can be private or public	All methods are automatically public
Methods can be concrete or abstract (with the keyword "abstract" used explicitly)	All methods are abstract (without actually being preceded by the abstract keyword), i.e. they have a name, return type and parameters but no implementation
Objects of A cannot be instantiated using the keyword "new" (Shape s = new Shape(); is not allowed)	Objects of M cannot be instantiated using the keyword "new" (IAge a = new IAge(); is not allowed)
A reference to an object of type A is allowed ("Shape s;" or "Shape s = new Square();" are allowed)	A reference to an object of type M is allowed (" IAge a;" or " IColor c = new Wall();" are allowed)
A concrete class inheriting from A must override the abstract methods of A	A concrete class implementing M must implement ALL methods of M

Problem Set 4

- Problem set 3 used 4 different classes with similar data members and methods
- Think about how the structure of the program can be modified to take advantage of this, using inheritance.

Reminder on Method Overloading

- Method overloading occurs when there are two or more methods with the same name defined within the same class.
- Each method in each class must have a unique combination of arguments. This means that the types and/or number and/or order of the arguments must be different for each method that shares one name.
- Remember, that having two methods with the same arguments but different return types is not legal.

Reminder on method overloading

- Remember that constructors are special methods. This fact allows constructors to be overloaded. The rules for constructors are the same as for other methods. Look at the constructors for the `Animal` class under the Abstract Classes topic of this tutorial for an example of constructor overloading.
- Examples of overloaded methods that you already know: `System.out.print()`, `System.out.println()`, `JOptionPane.showInputDialog()` [Check javadoc]