### **Reading**

This material is covered in Chapter 13 of book.

### **Interfaces**

Specify common behavior for a set of classes. Eg. can specify the objects are comparable, cloneable, etc.

Use the keyword **interface** instead of class.

Define an interface as follows:

public **interface** *NameOfInterface* {

// Any number of final, static fields

// Any number of abstract method declarations

}

An example of an interface is:

public **interface** Mammalian {

// describe what an mammal does

public abstract void eat();

public abstract void travel();

}

By default methods in interfaces are public and abstract and data fields are public and static. So the above is the same as:

**interface** Mammalian {

// describe what an mammal does

void eat();

void travel();

}

It is treated as a special class. It is compiled into its own .class file.

Like abstract class:

* can not create an instance of interface using new operator
* can use interface as a data type

Can use Mammalian interface to specify if an object is a mammal.

Use the **implements** keyword to say a class implements this interface.

Eg. Elephant class and Human class implement the Mammalian interface to specify elephants and humans behave as mammals.

When a class implements an interface, it implements all methods defined in the interface. Else it will be an abstract class.

This is also know as **interface inheritance**, and is the same as class inheritance.

public class Elephant **implements** Mammalian {

public void eat() {

System.out.println("Elephant eats");

}

public void travel() {

System.out.println("Elephant travels");

}

public int numberOfLegs() {

return 4;

}

}

public class Human **implements** Mammalian {

public void eat() {

System.out.println("Human eats");

}

public void travel() {

System.out.println("Human travels");

}

public int numberOfLegs() {

return 2;

}

}

Note that the numberOfLegs() method is not part of the interface Mammal.

### **Interfaces and Abstract Classes**

Let's mix an abstract class into this example.

public abstract class Organism {

public abstract String sound();

}

public class Elephant **extends** Organism **implements** Mammalian {

...

public String sound() {

return "trumpet";

}

}

public class Human **extends** Organism **implements** Mammalian {

...

public String sound() {

return "words";

}

}

The Organism class defines a sound() method. It is an abstract method and needs to be implemented by a concrete class.

### **Multiple Interfaces**

A class can only extend (inherit from) one superclass. Multiple inheritance is not allowed.

However a class can implement multiple interfaces.

So, single inheritance for classes, but multiple inheritance for interfaces.

Question: What issue does multiple inheritance create?

public **interface** Matriarchal {

public String leader();

}

public class Elephant extends Organism **implements** **Mammalian, Matriarchal** {

...

public String leader() {

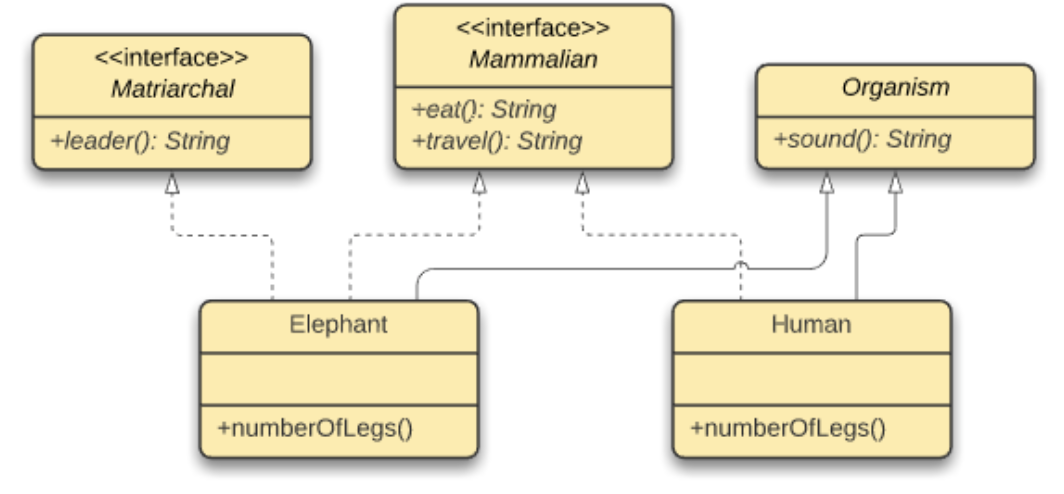
return "head cow";

}

}

### **UML Representation**

The above classes and interfaces are represented in UML as below.



Note the "<<interface>>" above the interface name. The interface name and the methods names are italicized.

As dashed line and hollow triangle points to the interface.

### **Operator instanceof**

The **instanceof** operator is used to test whether an object is an instance of the specified type (class or subclass or interface).

It returns either true or false.

For example

class TestProgram {

public static void main(String args[]){

Object obj = new Human();

System.out.println(obj instanceof Matriarchal); // false

System.out.println(obj instanceof Mammalian); // true

System.out.println(obj instanceof Organism); // true

System.out.println(obj instanceof Elephant); // false

System.out.println(obj instanceof Human); // true

}

}

It can be used check availability of method as follows:

if (obj instanceof Matriarchal)

System.out.println(((Elephant).leader()));

In-class Exercise: Are these examples of legal abstract classes?

(a)

interface A {

void print() { };

}

(b)

interface A {

void print();

}

(c)

abstract interface A {

void print();

}

(d)

abstract interface A extends Mammalian, Matriarchal {

abstract void print();

}

### **Abstract Classes vs. Interfaces**

|  | **Interfaces** | **Abstract Classes** |
| --- | --- | --- |
| **When to use** | "is-kind-of" relationship | Clear "is-a" relationship |
| **Name** | Typically Adjectives | Typically Nouns (like concrete classes) |
| **Variables** | must be "public static final" | No restrictions |
| **Constructors** | No constructors | Invoked by subclass via constructor chaining |
| **Methods** | must be"public abstract" | No restrictions |