第13章 抽象类和接口

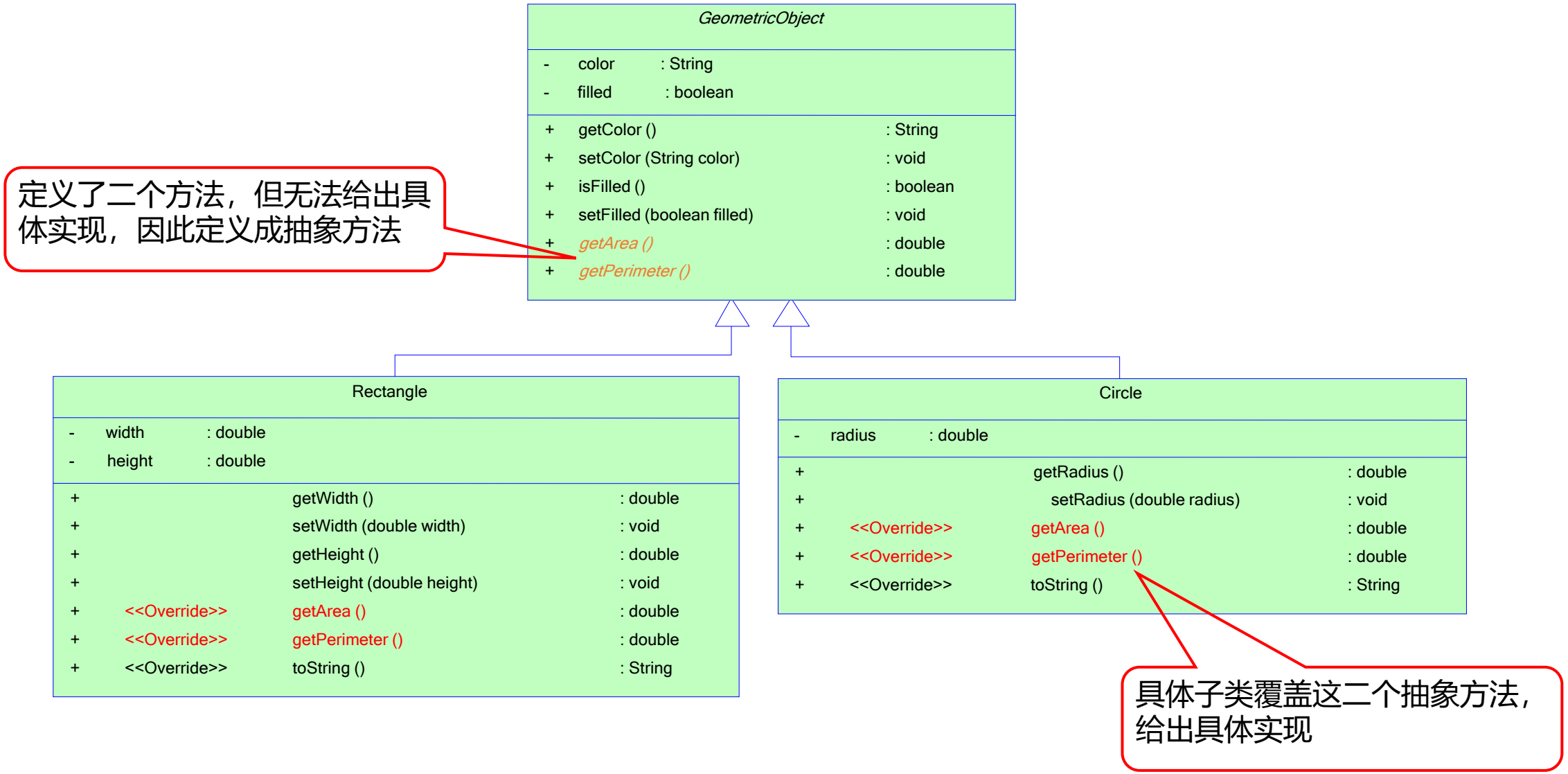
目 录

contents

13.1 抽象类

- 子类继承父类后，通常会添加新的属性和方法。因此沿着继承链越往下继承的子类其属性和方法越来越具体。相反，越上层的祖先类其实现越抽象，甚至无法给出具体实现。一个长方形图形有面积，但其祖先类GeometricObject的getArea()方法可能没法给出具体实现，这时可以定义成抽象方法。
- Java可定义不含方法体的方法，其方法体由子类根据具体情况实现，这样的方法称为抽象方法(abstract method)，包含抽象方法的类必须是抽象类(abstract class)。
- 抽象类和抽象方法的声明必须加上**abstract**关键字。
- 抽象方法的意义：加给子类的一个约束。例如Circle类和Rectangle类计算面积必须使用父类规定的函数签名。这样可以充分利用多态特性使得代码变得更通用

13.1 抽象类


GeometricObject
- color : String
- filled : boolean
+ getColor () : String
+ setColor (String color) : void
+ isFilled () : boolean
+ setFilled (boolean filled) : void
+ getArea () : double
+ getPerimeter () : double
定义了二个方法，但无法给出具体实现，因此定义成抽象方法
Rectangle
- width : double
- height : double
+ getWidth () : double
+ setWidth (double width) : void
+ getHeight () : double
+ setHeight (double height) : void
+ <<Override>> getArea () : double
+ <<Override>> getPerimeter () : double
+ <<Override>> toString () : String
Circle
- radius : double
+ getRadius () : double
+ setRadius (double radius) : void
+ <<Override>> getArea () : double
+ <<Override>> getPerimeter () : double
+ <<Override>> toString () : String
具体子类覆盖这二个抽象方法，给出具体实现

13.1 抽象类

```
abstract class GeometricObject{
    //属性和方法定义

    public abstract double getArea();
    public abstract double getPerimeter();
}
```

- 包含抽象方法的类必须是抽象类
- 抽象类和抽象方法必须用abstract关键字修饰
- 没有包含抽象方法的类也可以定义成抽象类

```
class Circle extends GeometricObject{
    //新的属性

    @Override
    public double getArea() {
        //给出具体实现
    }
    @Override
    public double getPerimeter() {
        //给出具体实现
    }
}
```

第13.2节程序清单13-2: GeometricObject.java,Circle.java, Rectangle.java, TestGeometricObject.java

13.1 抽象类

- 抽象方法：使用abstract定义的方法或者接口中定义的方法（接口中定义的方法自动是抽象的，可以省略abstract）。
- 一个类C如果满足下面的任一条件，则该类包含抽象方法且是抽象类：
  - 类C显式地包含一个抽象方法的声明；
  - 类C的父类中声明的抽象方法未在类C中实现；
  - 类C所实现的接口中有的方法在类C里没有实现
  - 只要类C有一个未实现的方法（自己定义的或继承的），就是抽象类
- 但是，一个不包含任何抽象方法的类，也可以定义成抽象类

13.1 抽象类

```
abstract class A {
    public abstract void m1();
    public abstract void m2();
}

abstract class B extends A{
     //B继承了二个抽象方法，但是只实现了m1,方法m2在B里还是抽象的，因此B必须是抽象类
     @Override
     public void m1() { }
}
class C extends B{
     //C继承B，又实现了方法m2，因此m1,m2二个方法在C里都有了具体实现
     //因此C就是可以是具体类
     @Override
     public void m2() { }

     //当然，C还可以继续覆盖B的m1，给出C的m1实现
}
```

13.1 抽象类

```
interface I {
    void m3();        //接口里方法编译器自动加上public abstract来修饰
    void m4();
}

abstract class D implements I{
    //class D声明实现了接口I，但只实现了一个接口方法m3
    //接口方法m4在D里还是抽象的，因此D只能是抽象类
    @Override
    public void m3() { }
}

//类E继承D，并实现了另一个接口方法m4
//因此类E是具体类
class E extends D implements I{ //注意既然E继承了D，所以这里的implements I可以不写
    @Override
    public void m4() { }

    //当然，E还可以继续覆盖D的m3，给出自己的的m3实现
}
```

13.1 抽象类

- 只有实例方法可以声明为抽象方法（Java里所有实例方法自动是虚函数，因此Java里没有virtual关键字）。
- 抽象类不能被实例化，即不能用new关键字创建对象（即new 右边的类型不能是抽象类）。
  - 但是抽象类可以作为变量声明类型、方法参数类型、方法返回类型
  - 为什么？因为一个抽象类型引用变量可以指向具体子类的对象
- 抽象类可以定义构造函数，并可以被子类调用。
- 抽象类可以定义变量、非抽象方法并被子类使用
- 抽象类的父类可以是具体类：自己引入了抽象方法。例如，具体类Object是所有类的祖先父类。

13.2 接口

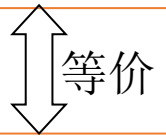
- 接口是公共静态常量和公共
范、协议的反映。
- 接口不是类：(1)不能定义
类可implements多个接口。
- 语法：[modifier] interface
constant_dec
abstract_meth
}
- 接口中的所有数据字段隐含
- 接口体中的所有方法隐含为public abstract

JDK版本	接口特性变化	核心描述
JDK 8	默认方法（default方法）、静态方法	允许接口包含具体实现的方法，解决兼容性问题；静态方法支持工具类功能。
JDK 9	私有方法	接口内部可定义私有方法，提升代码复用性和封装性。
JDK 17	密封接口（sealed和permits关键字）	限制接口的实现类范围，增强安全性和设计约束。
JDK 24	模块导入声明（import module）、模式匹配对接口的支持	简化模块化项目中接口的依赖管理；instanceof和switch直接匹配接口类型，提升代码简洁性和安全性。

13.2 接口-实例

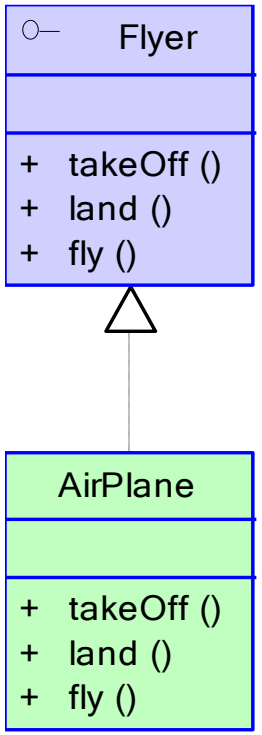
Java 接口不允许包含任何形式的初始化模块

```
public interface I1{
    public static final int k = 1; //可省略public static final
    public abstract void m();       //可省略public abstract
}
```




```
public interface I1{
    int k = 1;    //=1不可省略，因为它是final的，必须初始化
    void m( );    //不可定义函数体，它是abstract
}
```

13.2 接口-实例



注意：类和接口之间的实现关系的表示为三角箭头带虚线

```
public interface Flyer { //程序文件1
    void takeOff();
    void land();
    void fly();
}
public class Airplane implements Flyer {
    public void takeOff() {//程序文件2
        // 加速直到离地升空
        // 收起起落架
    }
    public void land() {
        // 放下起落架
        // 减速并降低副翼直到降落
        // 刹车
    }
    public void fly() {
        // 保持引擎运转
    }
}
```

接口描述了一种能力。Flyer接口描述了一种飞行的能力，飞行能力包括三个行为：takeOff、land、fly。但接口需要类来实现，因为接口描述的能力需要具体类的对象来体现。

一个类实现一个接口，表示这个类具有接口规定的能力。Airplane实现接口Flyer，表示Airplane具有飞行的能力，因此Airplane必须给出飞行能力的三个行为takeOff、land、fly的具体实现

类和接口之间是实现关系。这种实现关系是CANDO关系。例如Airplane实现接口Flayer，意味着Airplane的实例 CANDO Flyer

- 接口中的方法通过“接口类型的引用变量.方法名”调用，但接口类型的引用变量必须指向实现了该接口的类的实例对象
- 接口中的常量名通过“接口名.常量名”访问。

接口方法的访问

```
Flyer f = new AirPlane();

f.takeOff( ); //f传给this
f.fly( );
f.land( );     //f传给this
```

这时接口f的行为是飞机的飞行行为：多态

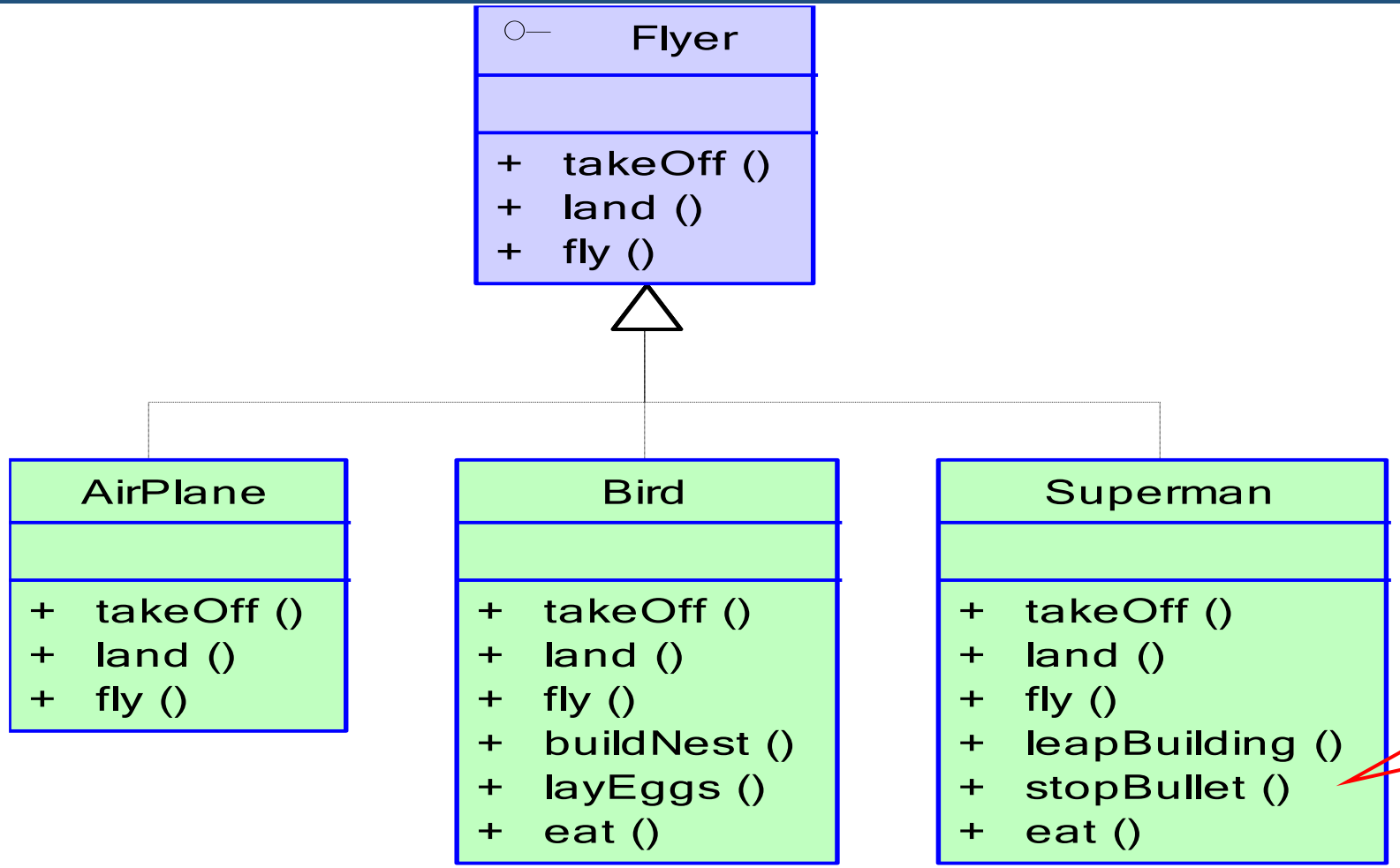
13.2 接口-用法

- 可以在能够使用任何其他数据类型的地方使用接口。
- 接口类型属于引用类型，接口类型的变量可以是：
  - 空引用(null)
  - 引用实现了该接口的类的实例
- 接口需要具体的类去实现。类实现接口的语法

```
[modifier] class className [extends superclass][implements interfaceNameList ] {
        member_declaration*
}
```

- 除非类为abstract，所有接口的成员方法必须被实现
- 一个类只能继承一个父类，但可以实现多个接口，多个接口以“，”分开。

13.2 接口-实例



接口方法的访问

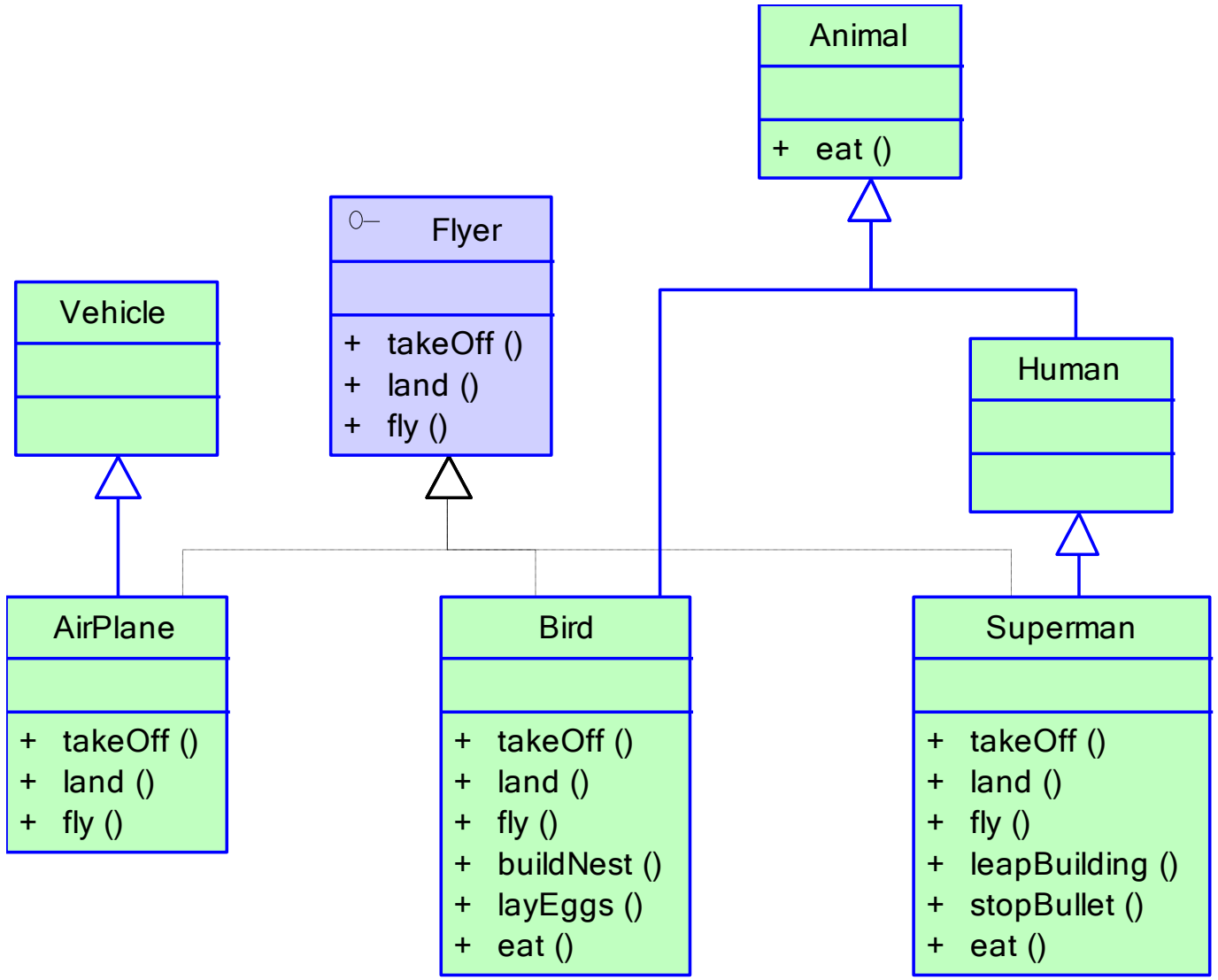
```
Flyer f = new Bird();

f.takeOff();
f.fly();
f.land();
```

这时接口f的行为是鸟的飞行行为：多态

注意一个具体类实现接口时，除了必须实现接口方法外，可以定义这个类其他的方法

这一个UML模型描述了AirPlane、Bird、Superman都实现了Flayer接口，因此这三个类型都具有飞行的能力(CANDO Fly)。但是他们各自的飞行行为可以不一样（每个类对takeOff、land、fly的具体实现都不一样），当用接口类型引用变量引用了三个类的实例时，通过接口引用变量调用接口方法就呈现出多态性。

13.2 接口-实例

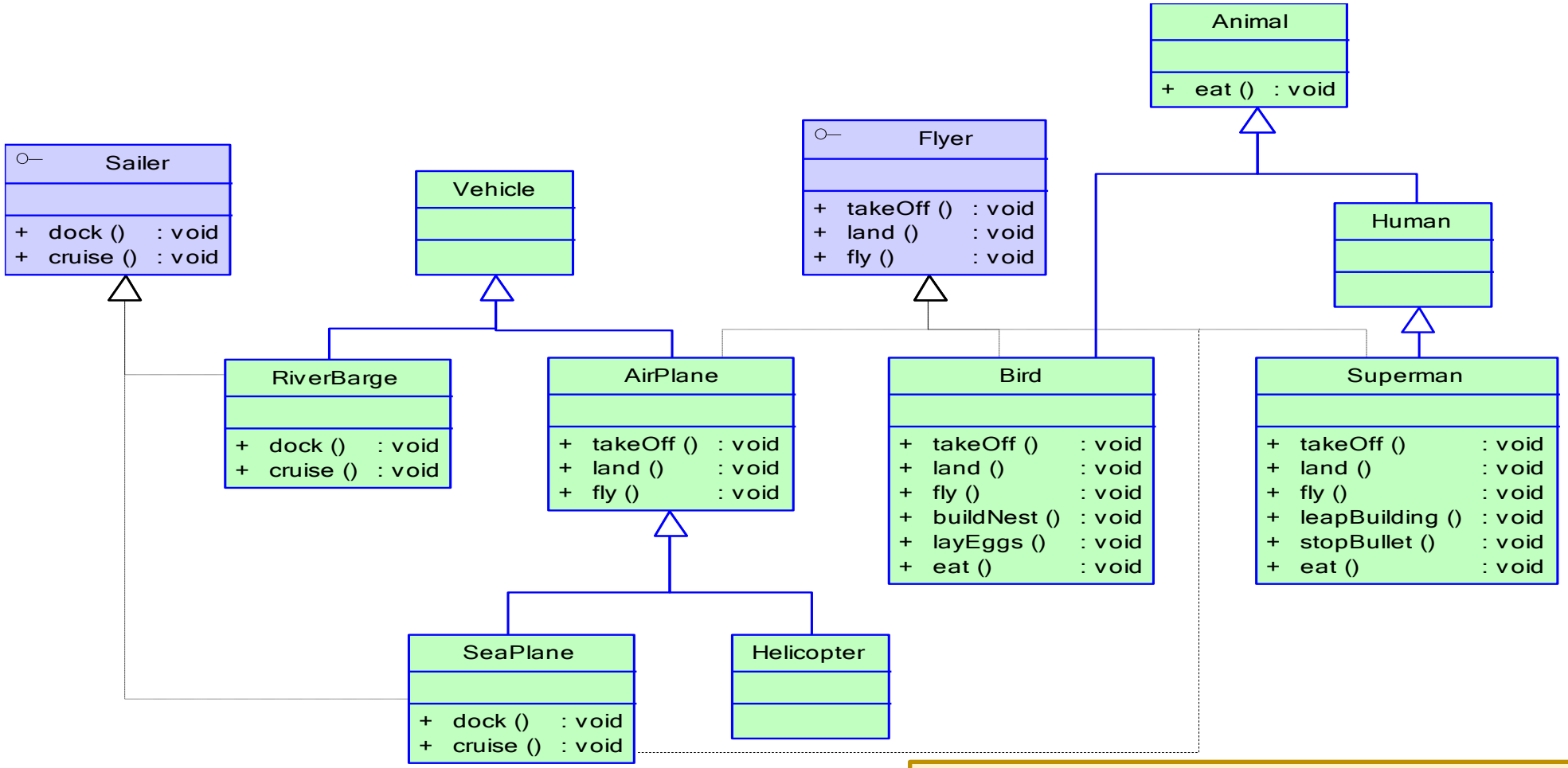


接口和继承的区别

接口描述了能力，继承描述了类之间的血缘关系。来自不同继承链（树）的类可以具有相同的能力。

例如：`Airplane`和`Superman`来自不同的继承链（即来自不同的家族，没有血缘关系），但他们都实现了接口`Flyer`，都具有飞行的能力。

13.2 接口



这个图描述了二个继承树（二个家族），二个接口，其中：
Airplane、Bird、Superman实现了接口Flyer
RiverBarge实现了接口Sailer
SeaPlane同时实现了接口Flyer和Sailer，这意味着Seaplane同时具有飞行能力和海上巡航的能力。这个例子说明了一个类可以实现多个接口（具备多种能力）

13.2 接口-继承

- 接口不是类（Java支持单继承类），一个接口可以继承多个接口。
- 语法

```
[modifier] interface interfaceName [extends interfaceNameList] {
    declaration*
}
```

- 如果接口声明中提供了extends子句，那么该接口就继承了父接口的方法和常量。被继承的接口称为声明接口的直接父接口。
- 任何实现该接口的类，必须实现该接口继承的其他接口。

13.2 接口-继承实例

```
public interface I1{
        public void m1();
}
```
I1.java

```
public interface I2 extends I1{
        public void m2();
}
```
I2.java

```
public interface I3 {
        public void m3();
}
```
I3.java

```
public class A implements I2, I3 {
        public void m1() { // implements}
        public void m2() { // implements}
        public void m3() { // implements}

}
```
A.java

当一个类实现多个接口时，
这个类的实例可以是多种类型
如下列表达式都返回true

```
A a = new A();

a instanceof I1          (true)
a instanceof I2          (true)
a instanceof I3          (true)
a instanceof Object      (true)

I1 i1 = new A();
I2 i2 = new A();
I3 i3 = new A();
```

13.2 接口-JDK的Comparable接口

- 有时需要比较二个对象，但不同类型对象的比较具有不同的含义，因此Java定义了Comparable接口。
- 因此，任何需要比较对象的类，都要实现该接口。
- Cloneable、Runnable、Comparable等接口均在包java.lang中：

```
package java.lang;
public interface Comparable{
    public int compareTo(Object o);
}
```

- CompareTo判断this对象相对于给定对象o的顺序，当this对象小于、等于或大于给定对象o时，分别返回负数、0或正数

13.2 接口- JDK的Comparable接口

- 有了Comparable接口，我们可以实现很通用的类来比较对象，例如实现一个从两个对象中找出最大者的方法。

```
public class Max{
    public static Comparable findMax (Comparable o1, Comparable o2){
        if(o1.CompareTo(o2) > 0 )
            return o1;
        else
            return o2;
    }
}
```

- 注意findMax方法的参数类型和返回类型都是Comparable（只要是实现了Comparable接口的对象都可以传进来。 Comparable接口描述了可以比较大小的能力，一个类实现了这个接口，意味着这个类的对象直接可以比较大小）
- Max.findMax与Comparable接口的具体实现子类无关。只要是实现了Comparable接口的具体类的二个对象（注意是同一个具体类的二个对象）传进来， Max.findMax都能工作。这就是接口的好处。（**程序存在的问题：如果是2个实现了Comparable接口的不同具体类对象传进来怎么办？最好通过泛型解决**）
- 另外要注意的是：o1.CompareTo(o2)调用是动态绑定（多态） （调用具体子类对象的CompareTo方法）

13.2 接口- JDK的Comparable接口

```
public class ComparableRectangle extends Rectangle implements Comparable {
    /** Construct a ComparableRectangle with specified properties */
    public ComparableRectangle(double width, double height) {
            super(width, height);
    }
    /** Implement the compareTo method defined in Comparable */
    public int compareTo(Object o) {
        if (this.getArea( ) >((ComparableRectangle)o).getArea()) return 1;
        else if (this.getArea( ) <((ComparableRectangle)o).getArea()) return -1;
        else return 0;
    }
}
```

注意由于篇幅所限没有用instanceOf检查o的类型。但如果o不是ComparableRectangle类型怎么办？这时返回什么样的整数都不合适，最好的这个问题最好的解决办法是用泛型。

- 对于ComparableRectangle的两个对象r1和r2，直接调用Max.findMax(r1,r2)找出最大的对象
- 对于实现了Comparable接口任何类的二个对象（同一个类）(不管其具体实现是什么)a1和a2，都可以调用Max.findMax(a1,a2)找出最大的对象。这就是接口和多态的威力。

13.2 接口-继承Cloneable接口

- Java定义了Cloneable接口，任何想克隆的类必须实现该接口，同时覆盖从Object类继承的clone方法，并将访问属性改为public
- Cloneable接口为空接口(未定义任何函数)，其定义为

```
package java.lang;
public interface Cloneable {     }
```

- 空接口称为标记接口(markup interface)
- 空接口有什么作用？唯一目的允许你用instanceof检查对象的类型：

```
if(obj instanceof Cloneable)…
```

例子见教材程序清单13-11House.java以及第11章PPT例子

13.3 接口与抽象类-比较

	接口	抽象类
多重继承	一个接口可以继承多个接口	一个类只能继承（extends)一个抽象类
方法	接口不能提供任何代码	抽象类的非抽象函数可以提供完整代码
数据字段	只包含public static final常量，常量必须在声明时初始化。	可以包含实例变量和静态变量以及实例和静态常量。
含义	接口通常用于描述一个类的外围能力，而不是核心特征。类与接口之间的是-able或者can do的关系，有instanceof关系（实现了接口的具体类对象也是接口类型的实例）。	抽象类定义了它的后代的核心特征。例如Person类包含了Student类的核心特征。子类与抽象类之间是is-a的关系，也有instanceof关系（子类对象也是父类实例）。
简洁性	接口中的常量都被假定为public static final，可以省略。不能调用任何方法修改这些常量的初始值。接口中的方法被假定为public abstract。	可以在抽象类中放置共享代码。可以使用方法来修改实例和静态变量的初始值，但不能修改实例和静态常量的初始值。必须用abstract显式声明方法为抽象方法。
添加功能	如果为接口添加一个新的方法，则必须查找所有实现该接口的类，并为他们逐一提供该方法的实现，即使新方法没有被调用。	如果为抽象类提供一个新方法，可以选择提供一个缺省的实现，那么所有已存在的代码不需要修改就可以继续工作，因为新方法没有被调用。

13.2 接口-新特性

- Java8接口-新特性
  - JavaSE8 允许在接口中增加静态方法
    - 可以不需要定义实现该接口的子类，直接用接口名.方法名访问该静态方法；
  - 可以为接口方法提供一个默认实现，必须使用 default 修饰符标记这个方法。
    - 为旧接口增加新方法，不影响已实现该接口的类
    - 相当于间接实现多继承，会导致二义性问题
      - 类实现的多个接口之间出现相同签名的缺省方法
      - 父接口和子接口之间出现相同签名的缺省方法
      - 父类和实现的接口之间出现相同签名的缺省方法
- Java9 新增私有方法（接口中使用，必须实现）

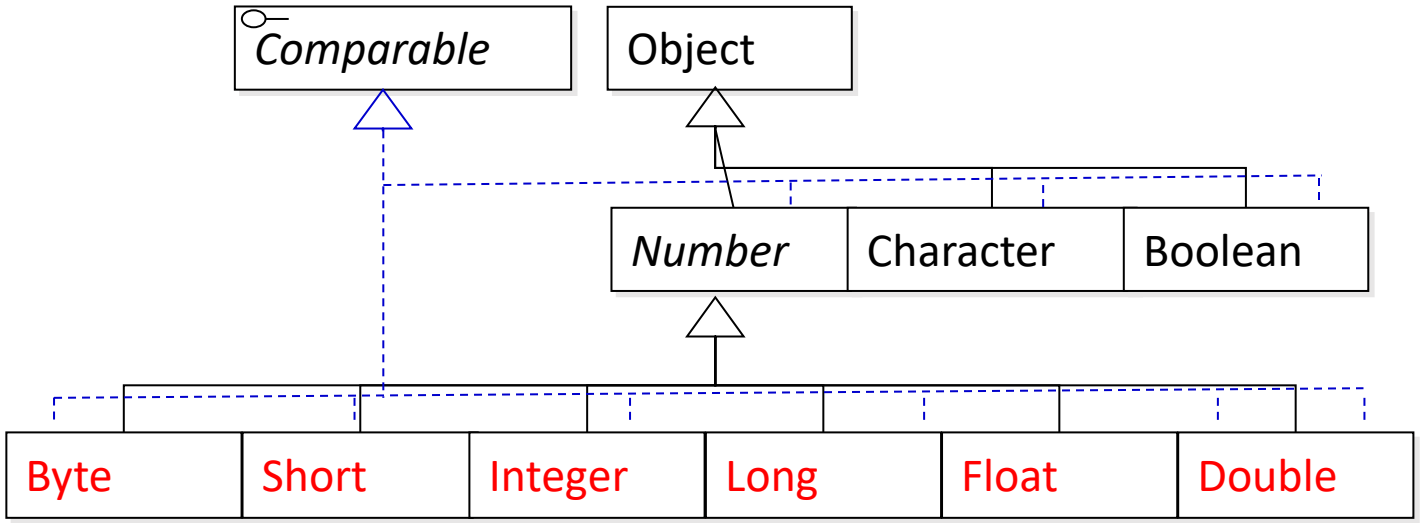
13.2 接口-新特性

```
interface  F1{default void f(){System.out.println("F1----f");}}
interface  F2{default void f(){System.out.println("F2----f");}}
interface  F3 extends F1{default void f(){System.out.println("F3----f");}}
class Base1 implements  F1{}
class Base2 implements  F2{public void f(){System.out.println("Base2----f");}}
class Base3 implements  F1,F2{public void f(){System.out.println("Base3----f");}}
class Base4 implements  F3{}
class Derive extends Base2 implements  F3{}
public class TestDefaultMethod {
    public static void main(String[] args){
        new Base1().f();
        new Base2().f();
        new Base3().f();
        new Base4().f();
        new Derive().f();
    }
}
```

```
F1----f
Base2----f
Base3----f
F3----f
Base2----f
```

13.4 包装类提供的接口和方法

- 基本数据类型包装类的作用
  - 为基本数据类型提供有用的方法和常量值
  - 用于只能处理对象引用的地方（比如Java所有的集合类里只能放对象）
- 包装类对象的值不变（内部valule是final的），只能读取。



13.4 包装类提供的接口和方法

- 构造函数
  - 以一个对应的基本数据类型为参数
  - 以字符串为参数(除了Character)
- 如Double类的构造函数如下:

```
public Double (double value);
public Double (String value);
```

- 例如

```
Double doubleObject = new Double(5.0);
Double doubleObject = new Double("5.0");
```

- 包装类没有无参构造方法

13.4 包装类提供的接口和方法

- 每一个数值包装类都有相应类型常量MAX_VALUE和MIN_VALUE。
  - MAX_VALUE对应本数据类型的最大值。
  - 对Byte ,Short ,Integer和Long, MIN_VALUE对应最小值
  - 对Float和Double, MIN_VALUE对应最小正值
- 上述常量用于排序算法时很有用。
- 直接用包装类名访问其常量值：

```
System.out.println("The maximum integer is"+
    Integer.MAX_VALUE); //MAX_VALUE是int类型
System.out.println("The minimum positive float
    is"+ Float.MIN_VALUE); //MIN_VALUE是float类型
```

13.4包装类提供的接口和方法-包装类->基本数据类型

- Number是基本数值类型包装类的抽象父类，里面有如下方法返回包装类对象对应的基本数据类型值：
  - public abstract int intValue()
  - public abstract long longValue()
  - public abstract float floatValue()
  - public abstract double doubleValue()
  - public byte byteValue()
  - public short shortValue()
- 如 int i = new Integer(10).intValue();
- 另外每个类的toString ()方法将数值转换成字符串

13.4包装类提供的接口和方法-包装类->基本数据类型

- 字符串转数值
- 转换为Byte, Short, Integer, Long, Float, Double

```
public static type parseType(String s)
public static type parseType(String s, int radix)
```

- 如

```
int i = Integer.parseInt(“11”,2); //3
int i = Integer.parseInt(“12”,8); //10
int i = Integer.parseInt(“1A”,16); //26
double d = Double.parseDouble(“3.14”);//3.14
```

13.4 包装类提供的接口和方法

- 方法valueOf创建一个新的包装对象，并将它初始化为指定字符串的值

- 例如：

```
Double doubleObject = Double.valueOf(“12.4”);
Integer integerObject = Integer.valueOf(“12”);
```

13.4包装类提供的接口和方法-基本数据类型->包装类

- JDK1.5开始允许基本类型和包装类之间的自动转换。
  - 将基本类型的值转换为包装类对象，称为装箱(boxing)
  - 将包装类对象转换为基本类型的值，称为开箱(unboxing)

```
Integer intObject = 2; //装箱
```
等价于
```
Integer intObject = new Integer(2);

Integer intObject1 = 2, intObject2 = 3 ;
System.out.println(intObject1 + intObject2 );//开箱
int j = intObject ; //开箱
```