Java语言与系统设计

第7讲 抽象类和接口

- static 关键字
- main 方法
- final 关键字
- 抽象类和抽象方法
- 接口
- 内部类

1. static关键字

- 类描述了其对象的属性和行为，并没有产生实质上的对象，只有通过new关键字才会产生出对象，系统才会分配内存空间给对象，其方法才可以供外部调用。
- 有时候希望无论是否产生了对象或无论产生了多少对象，某些特定的数据在内存空间里只有一份。例如，班上所有学生都共享学校的名称，不需要在每一个学生的实例对象中都单独分配一个用于代表学校名称的变量。

实例：

```
class Customer{
 String name;
 String bankName; // static
}
 public class test1 {
  public static void main(String[] args) {
  Customer zhangsan = new Customer();
  zhangsan.name = “zhangsan";
  zhangsan.bankName = “China Bank";
  Customer lisi = new Customer();
  lisi.name = “lisi";
 System.out.println("lisi.bankName="+lisi.bankName);
 } }
```

1. static

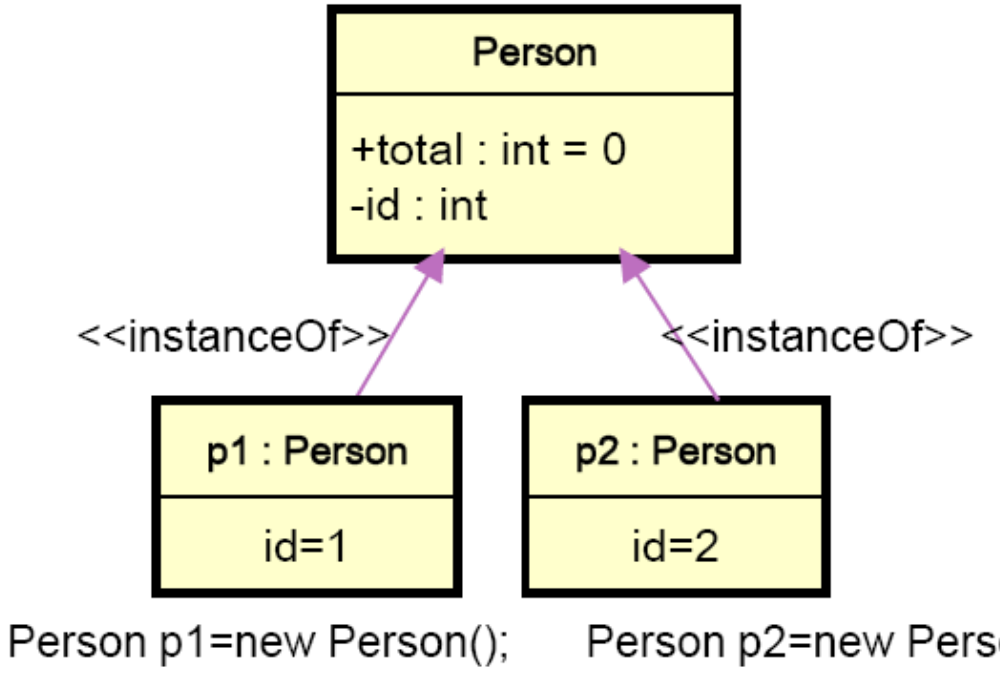
- 类属性作为该类各个对象之间共享的变量。在设计类时，分析哪些属性不因对象的不同而改变，将这些属性设置为类属性。相应的方法设置为类方法。
- 如果方法与调用者无关，则这样的方法通常被声明为类方法，由于不需要创建对象就可以调用类方法，从而简化了方法的调用。

1. static

- Java类中可用static修饰属性、方法、代码块、内部类
  - 被修饰后的成员具备以下特点:
  - 随着类的加载而加载
  - 优先于对象存在
  - 修饰的成员，被所有对象所共享
  - 访问权限允许时，可不创建对象，直接被类调用

1. static

- 类（静态）变量：类变量（类属性）由该类的所有实例共享

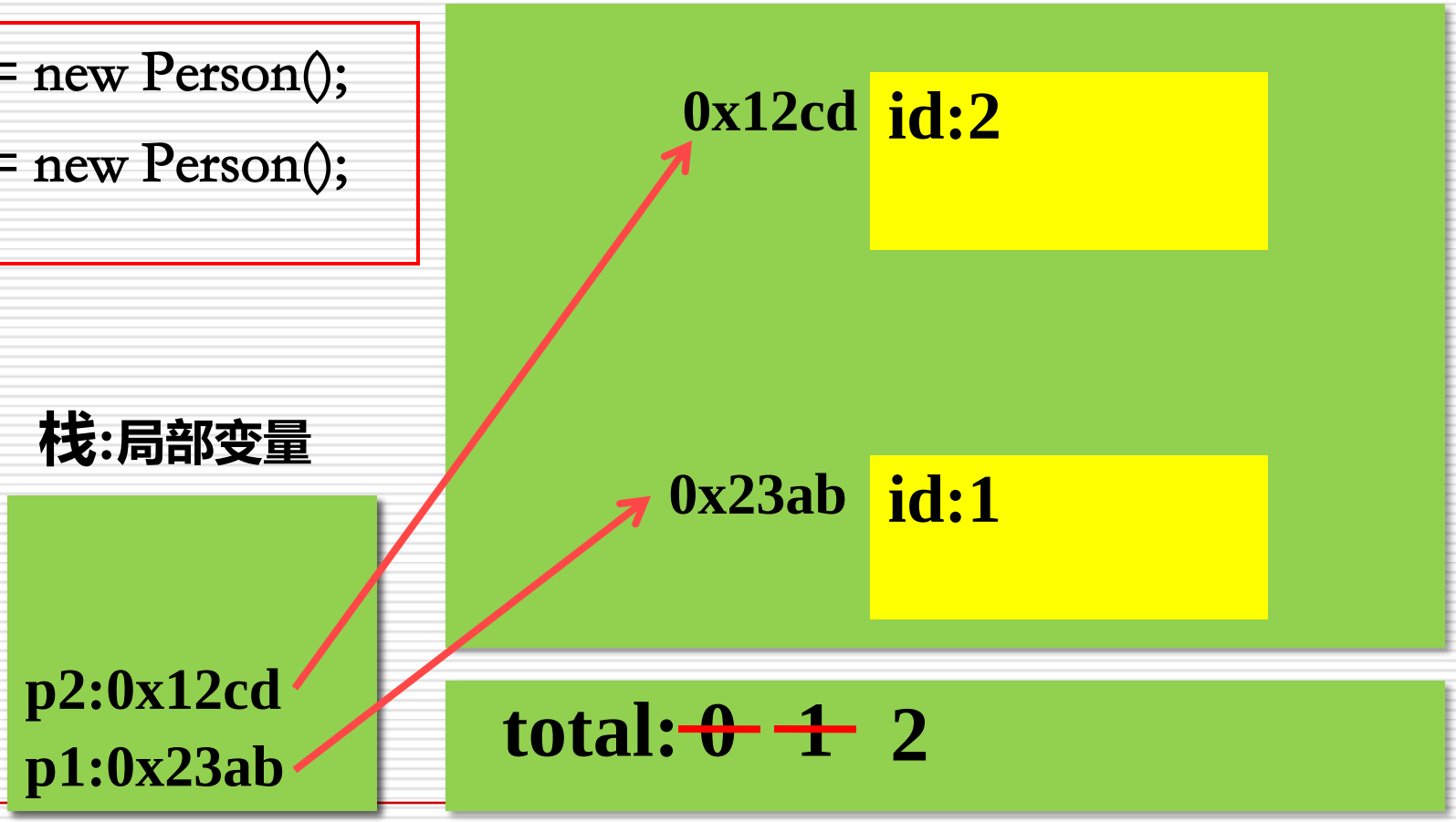



```
public class Person {
    private int id;
    public static int total = 0;
    public Person() {
        total++;
        id = total;
    }
}
```

静态变量的内存存储实现

堆: new出来的结构：对象、数组

```
Person p1= new Person();
Person p2= new Person();
```



方法区：类的加载信息、静态域、常量池

1. static

- 类方法：类变量（类属性）由该类的所有实例共享
  - 没有对象的实例时，可以用类名.方法名()的形式访问由static修饰的类方法。
  - 在static方法内部只能访问类的static修饰的属性或方法，不能访问类的非static的结构。
  - 因为不需要实例就可以访问static方法，因此static方法内部不能有this和super。
  - static修饰的方法也不能被重写。

```
class Person {
      private int id;
      private static int total = 0;
      public static int getTotalPerson() {
              //id++;       //非法
              return total;}
      public Person() {
              total++;
              id = total;
      }}
public class PersonTest {
       public static void main(String[] args) {
              System.out.println("Number of total is " + Person.getTotalPerson());
              //没有创建对象也可以访问静态方法
              Person p1 = new Person();
              System.out.println( "Number of total is "+ Person.getTotalPerson());
       }}
```

The output is:
Number of total is 0
Number of total is 1

2. main方法

- 由于Java虚拟机需要调用类的main()方法，所以该方法的访问权限必须是public。又因为Java虚拟机在执行main()方法时不必创建对象，所以该方法必须是static的，该方法接收一个String类型的数组参数，该数组中保存执行Java命令时传递给所运行的类的参数。
- 因为main() 方法是静态的，我们不能直接访问该类中的非静态成员，必须创建该类的一个实例对象后，才能通过这个对象去访问类中的非静态成员。

实例：

```
class mainTest {
   int a=10; //static
   public static void main(String[] args) {
     //mainTest test=new mainTest();
     //System.out.println(test.a);
     System.out.println(a);
     }
    }
```

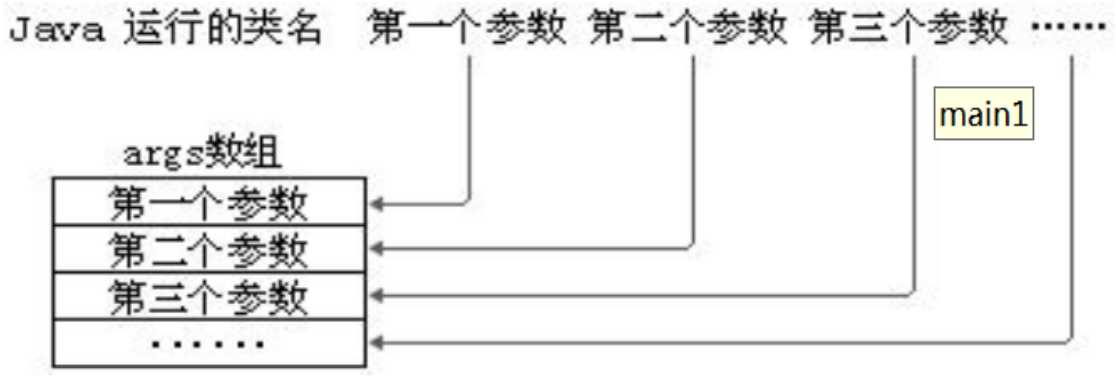
2. main方法

- 命令行参数

```
public class CommandPara {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println("args[" + i + "] = " + args[i]);
        }
    }
}
```

//运行程序CommandPara.java

java CommandPara “Tom" “Jerry" “Shkstart"



输出结果：

args[0] = Tom
args[1] = Jerry
args[2] = Shkstart

static method不能直接call non-static methods。可改成"System.out.println("s.doSomething() returns " + s.doSomething());"。同理，static method不能访问non-static instant variable。

- 查错

```
public class Something {
      public static void main(String[] args) {
          Something s = new Something();
          System.out.println("s.doSomething()
    returns " + doSomething());
      }
      public String doSomething() {
          return "Do something ...";
      }
}
```

练习

```
public class Demo {
    private static int j = 0;
    private static boolean methodB(int k) {
        j += k;
        return true;
    }
    public static void methodA(int i) {
        boolean b;
        b = i < 10 | methodB(4);
        b = i < 10 || methodB(8);
    }
    public static void main(String args[]) {
        methodA(0);
        System.out.println(j);
    }
}
```

3. 代码块

- 代码块(或初始化块)的作用：对Java类或对象进行初始化。
- 一个类中代码块若有修饰符，则只能被static修饰，称为静态代码块(static block)，没有使用static修饰的，为非静态代码块。
- static代码块通常用于初始化static的属性

```
class Person {
public static int total;
static {
total = 100;//为total赋初值
}
…… //其它属性或方法声明
}
```

3. 代码块

```
class Person {
public static int total;//静态属性
public int age; //非静态属性
public static void func(){//静态方法
System.out.println("in static method!"); }
static{ total=100;
System.out.println(“in static block!");}
//静态代码块
{age=50;System.out.println("in block!"); }
//非静态代码块//能否删掉括号？
}
```

3. 代码块

```
public class PersonTest{
public static void main(String[] args) {
//System.out.println("total = "+ Person.total);
//System.out.println("total = "+ Person.total);
//Person p1=new Person();
//Person p2=new Person();
//p1.func();
//System.out.println(p1.age);
}
}
```

3. 代码块

静态代码块：用static 修饰的代码块

1. 可以有输出语句。
2. 可以初始化类的属性。
3. 不可以对非静态的属性初始化。即：不可以调用非静态的属性和方法。
4. 若有多个静态的代码块，那么按照从上到下的顺序依次执行。
5. 静态代码块的执行要先于非静态代码块。
6. 静态代码块随着类的加载而加载，且只执行一次。

非静态代码块：没有static修饰的代码块

1.可以有输出语句。
2.可以初始化对象的属性。
3.除了调用非静态的结构外，还可以调用静态的变量或方法。
4.若有多个非静态的代码块，那么按照从上到下的顺序依次执行。
5.每次创建对象的时候，都会执行一次。且先于构造函数执行。

3. 代码块

程序中成员变量赋值的执行顺序

练习

```
public class Test {
    static int x, y, z;
    static {int x = 5;x--;}
    static {x--;}
    public static void main(String[] args) {
        System.out.println("x=" + x);
        z--;
        method();
        System.out.println("result:" + (z + y + ++z));
    }
    public static void method() {y = z++ + ++z;}
}
```

x=-1

result:3

//y=0,z=3

4. final 关键字

- 在Java中声明类、变量和方法时，可使用关键字final来修饰，表示“最终的”。

(1) final标记的类不能被继承，提高安全性，提高程序的可读性。

  - String类、System类、StringBuffer类

(2) final标记的方法不能被子类重写。

  - 比如：Object类中的getClass()

4. final 关键字

(3) final标记的变量(成员变量或局部变量)即称为常量。名称大写，且只能被赋值一次。

- final标记的成员变量必须在声明时或在每个构造函数中或代码块中显式赋值，然后才能使用。

  例如：Final double MY_PI=3.14;

  - 如果每个对象的变量值相同，初始化赋值；
  - 如果每个对象的变量值不同，用构造函数；
  - 如果变量值需要通过代码执行才能赋值，用代码块。

4. final 关键字

- final修饰局部变量，则该变量变成了常量
- final修饰方法的形式参数，则该形参数在方法中不能修改，是常量

注意：

- static final修饰属性：全局常量

练习

- 查错

```
public class Something {
    public int addOne(final int x) {
        return ++x;

    }
}
```

练习

```
public class Something {
    public static void main(String[] args) {
        Other o = new Other();
        new Something().addOne(o);
    }
    public void addOne(final Other o) {

        o.i++;    //没有问题
    }
}
class Other {
    public int i;
}
```

练习

- 查错

```
class Something {
    int i;
    public void doSomething() {
     System.out.println("i = " + i);
    }
}
```

//i=0

练习

- 查错

```
class Something {
    final int i;
    public void doSomething() {
     System.out.println("i = " + i);
    }
}
```

final的instant variable没有default value，必须在constructor (构造器)结束之前被赋予一个明确的值。可以修改为"final int i = 0;"

5. 抽象类和抽象方法

- 随着继承层次中一个个新子类的定义，类变得越来越具体，而父类则更一般，更通用。类的设计应该保证父类和子类能够共享特征。有时将一个父类设计得非常抽象，以至于它没有具体的实例，这样的类叫做抽象类。

5. 抽象类和抽象方法

- 用abstract关键字来修饰一个类，这个类叫抽象类。
- 用abstract来修饰一个方法，该方法叫抽象方法。
- 抽象方法：只有方法的声明，没有方法的实现。以分号结束：

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

- 含有抽象方法的类必须被声明为抽象类。

实例：

```
abstract class BBB {//平面几何图形的抽象类
    double x,y;                          //平面的长和宽
    BBB() {x=0; y=0;}                    //无参构造函数
    BBB(double xx, double yy) {x=xx; y=yy;}
     //带参构造函数
    abstract double area();   //求面积的抽象方法
    abstract double girth();   //求周长的抽象方法
}
```

```
class CCC extends BBB {        //矩形类
    CCC() {super();}
    CCC(double xx, double yy){super(xx,yy);}
    double area() {return x*y;} //长*宽
    double girth() {return 2*(x+y);}
    //2*(长+宽)
}
```

```
class DDD extends BBB {//直角三角形类
    DDD() {super();}
    DDD(double xx, double yy){super(xx,yy);}
    double area() {return x*y/2;} //长*宽/2
    double girth()
    {return x+y+Math.sqrt(x*x+y*y);}
}
```

假定利用下面的主类程序来使用上面定义的3个类。

```
public class abstractTest
{ public static void main(String[] args)
 {
 BBB b;      //定义抽象类的引用对象
 b=new CCC(3,4);  //b指向矩形类对象
 System.out.println(b.area()+", "+b.girth());
 b=new DDD(3,4);  //b指向直角三角形类对象
 System.out.println(b.area()+", "+b.girth());
 }
}
```

5. 抽象类和抽象方法

注意：

- 抽象类不能被实例化。抽象类是用来被继承的。
- 抽象类的子类必须重写父类的抽象方法，并提供方法体。若没有重写全部的抽象方法，仍为抽象类。
- 不能用abstract修饰变量、代码块、构造函数；
- 不能用abstract修饰私有方法、静态方法、final的方法、final的类。

5. 抽象类和抽象方法

问题1：为什么抽象类不可以使用final关键字声明?

问题2：一个抽象类中可以定义构造函数吗?

问题3：是否可以这样理解：抽象类就是比普通类多定义了抽象方法，除了不能直接进行类的实例化操作之外，并没有任何的不同?

判断：包含抽象方法的类，一定是抽象类；反之，抽象类中可以没有抽象方法。

填空：

定义抽象类和抽象方法的关键字是（）。抽象类中（）（可以/不可以）有抽象方法，（）（可以/不可以）有普通方法（）（可以/不可以）有属性；一个类中定义了抽象方法，那这个类（）（必须/不必须）用abstract修饰，即抽象类。

练习

- 查错

```
abstract class Name {
private String name;
public abstract boolean isName(String name) {}
}
```

//抽象方法定义没有{}，以；号结束

练习

- 查错

```
abstract class Something {
    private abstract String doSomething ();
}
```

//抽象和私有是冲突的，抽象和final也是

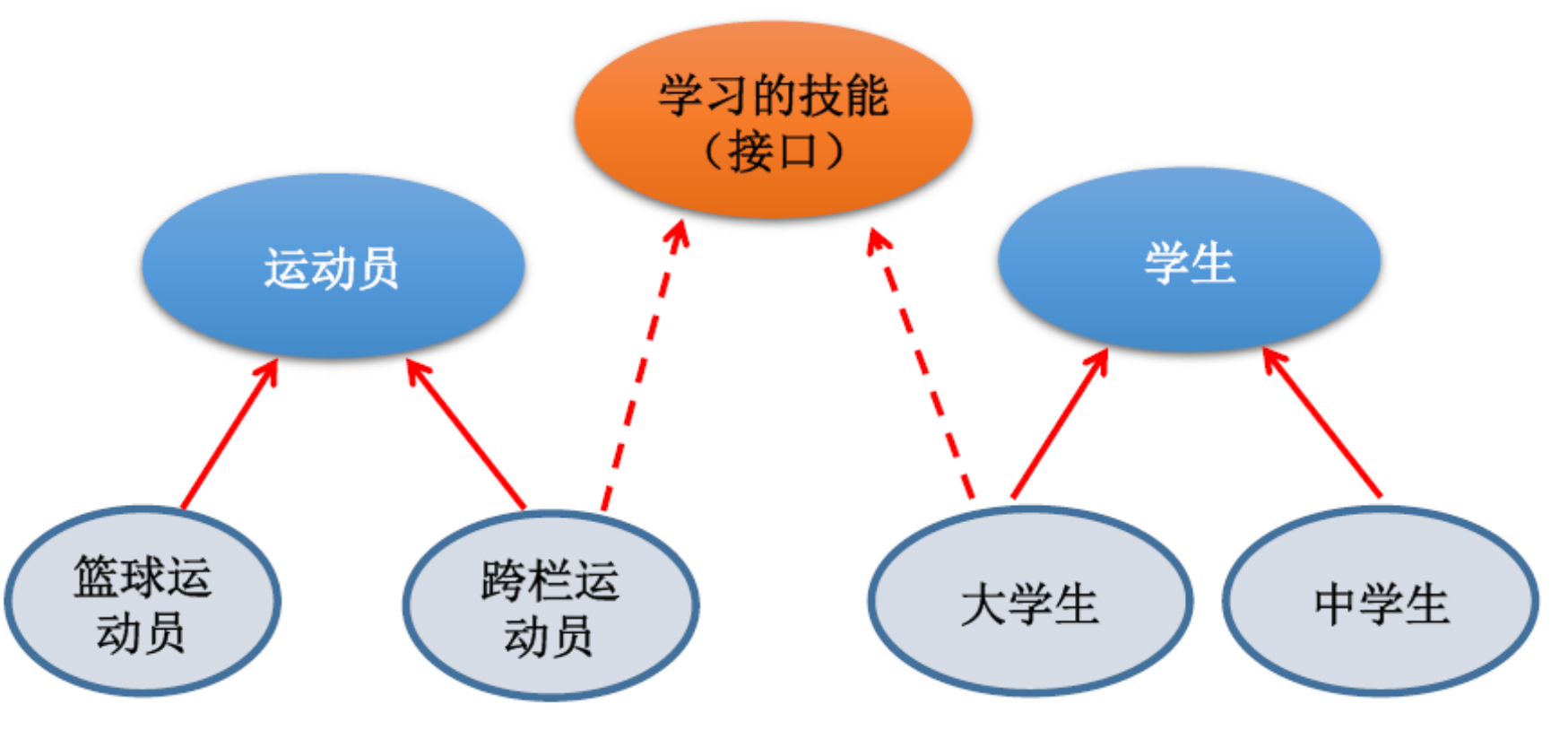
6. 接口

- 有的时候希望能从几个类中派生出一个子类，继承它们所有的属性和方法。但是，Java不支持多重继承。接口就可以得到多重继承的效果。
- 从几个类中抽取出一些共同的行为特征，但是它们之间并没有is-a的关系，仅仅是具有相同的行为特征而已。
- 例如：QQ、微信都支持视频聊天。

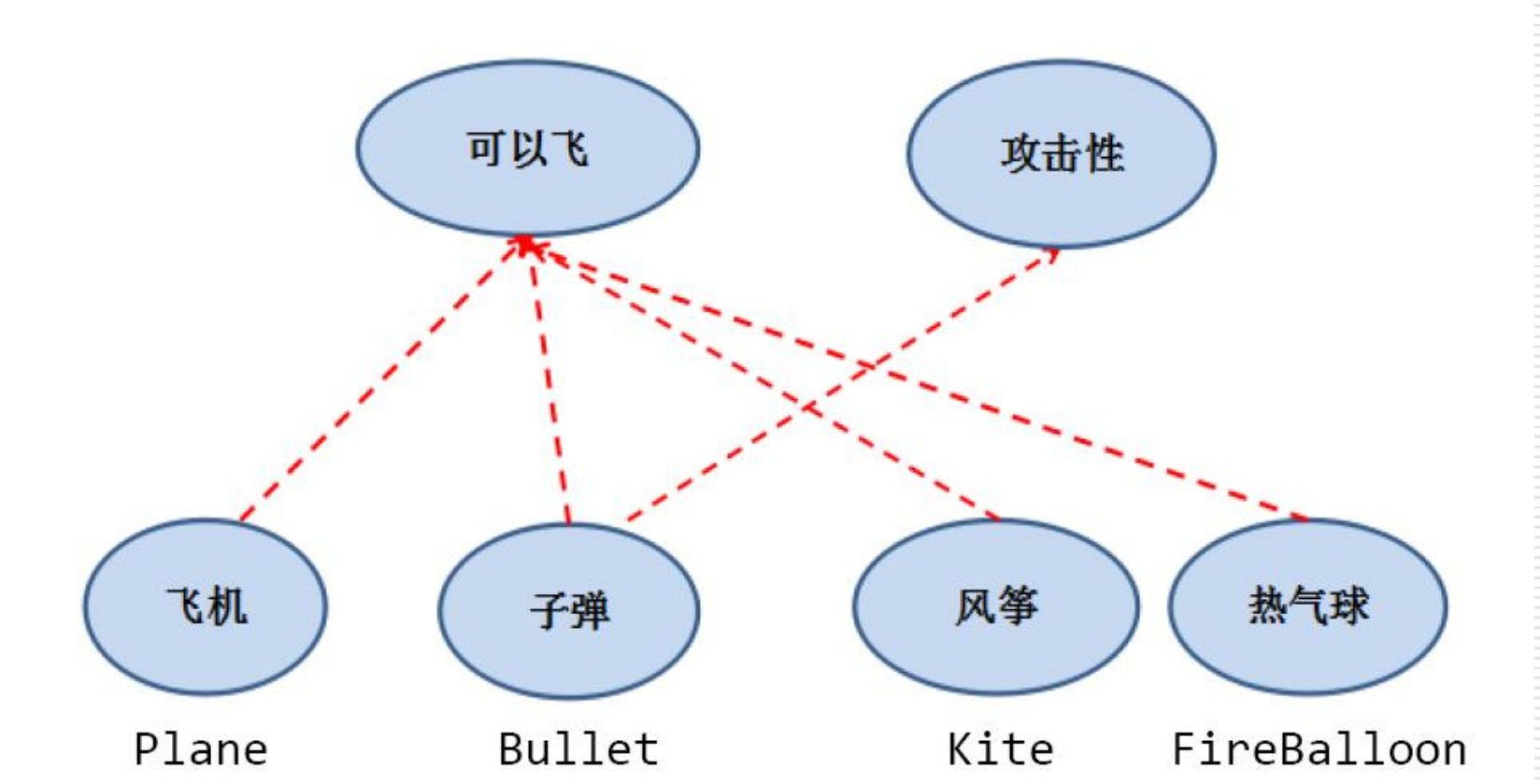
6. 接口

- 接口就是规范，定义的是一组规则，体现了现实世界中“如果你是/要...则必须能...”的思想。
- 继承是一个“是不是”的关系，而接口实现则是“能不能”的关系。
- 接口的本质是契约，标准，规范，制定好后大家都要遵守，实现接口的功能。

6. 接口



6. 接口



6. 接口

- 接口(interface)是抽象方法和常量值定义的集合。特点如下：
  - 用interface来定义。
  - 所有成员变量都默认是public static final修饰（全局常量）
  - 所有抽象方法都默认是由public abstract修饰
  - 没有构造函数
  - 采用多继承机制

```
public interface Runner {
    int ID = 1;
    void start();
    public void run();
    void stop();
}
```

```
public interface Runner {
    public static final int ID = 1;
    public abstract void start();
    public abstract void run();
    public abstract void stop();
}
```

6. 接口

- 定义Java类的语法格式：先写extends，后写implements

```
Class SubClass extends SuperClass implements InterfaceA{}
```

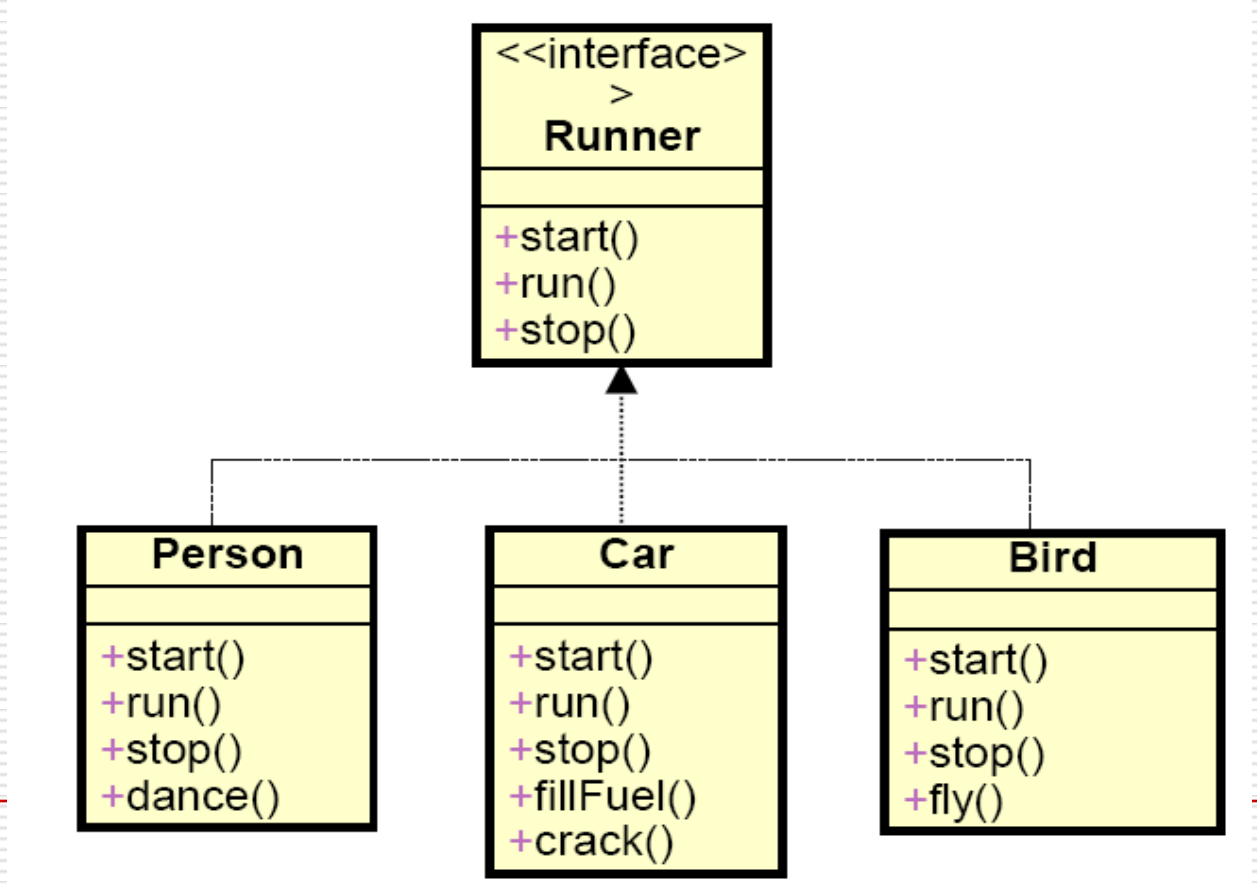
- 一个类可以实现多个接口，接口也可以继承其它接口。
- 实现接口的类中必须提供接口中所有方法的具体实现内容，方可实例化。否则，仍为抽象类。

6. 接口

- 接口的主要用途就是被实现类实现
- 接口之间可以多继承
- 与继承关系类似，接口与实现类之间存在多态性
- 接口和类是并列关系，或者可以理解为一种特殊的类。从本质上讲，接口是一种特殊的抽象类，这种抽象类中只包含常量和方法的定义(JDK7.0及之前)，而没有变量和方法的实现。

6. 接口

- 接口的主要用途就是被实现类实现:



6. 接口

- 举例：

```
interface Runner {
    public void start();
    public void run();
    public void stop();
}
class Person implements Runner {
    public void start() {
        // 准备工作：弯腰、蹬腿、咬牙、瞪眼
        // 开跑
    }
    public void run() {
        // 摆动手臂
        // 维持直线方向
    }
    public void stop() {
        // 减速直至停止、喝水。
    }
}
```

6. 接口

- 一个类可以实现多个无关的接口

```
interface Runner { public void run();}
interface Swimmer {public double swim();}
class Creator{public int eat(){…}}
class Man extends Creator implements Runner ,Swimmer{
        public void run() {……}
        public double swim()  {……}
        public int eat() {……}
}
```

6. 接口

- 一个接口可以继承多个接口

```
interface AA{
void method1();
}
interface BB{
void method2();
}
interface CC extends AA, BB{    }
```

6. 接口

- 实现类SubAdapter必须给出接口SubInterface以及父接口MyInterface中所有方法的实现。否则，SubAdapter仍需声明为abstract的。

```
interface MyInterface{
        String s="MyInterface";
        public void absM1();
}
interface SubInterface extends MyInterface{
        public void absM2();
}
public class SubAdapter implements SubInterface{
        public void absM1(){System.out.println("absM1");}
        public void absM2(){System.out.println("absM2");}
}
```

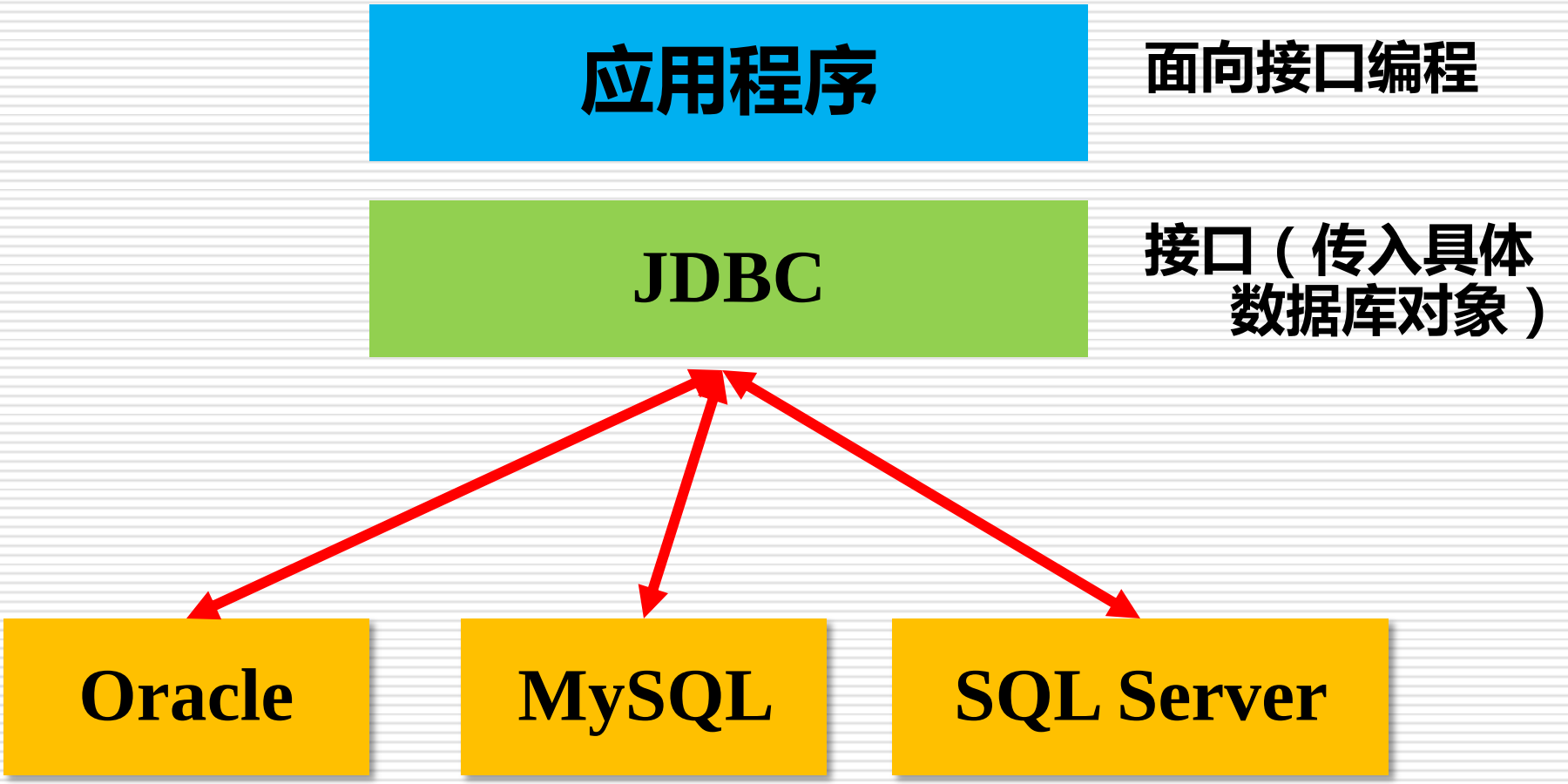
6. 接口

- 与继承关系类似，接口与实现类之间存在多态性

```
public class usbTest{
public static void main(String[] args){
  Computer com =new Computer();
      Flash flash =new Flash();
      com.transferData(flash);}
}
class Computer{
  void transferData(USB usb){// USB usb= new Flash()
  usb.start();
  usb.stop();}
}
```

6. 接口

```
interface USB{
public void start();
public void stop();
}
class Flash implements USB{
 public void start(){System.out.println("U盘启动");}
public void stop(){System.out.println("U盘停止");}
}
class Printer implements USB{
 public void start(){System.out.println("打印机启动");}
public void stop(){System.out.println("打印机停止");}
}
```


应用程序
面向接口编程
JDBC
接口（传入具体数据库对象）
Oracle
MySQL
SQL Server
每个类重写接口的操作方法

6. 接口

区别点	抽象类	接口
定义	包含抽象方法的类	主要是抽象方法和全局常量的集合
组成	构造方法、抽象方法、普通方法、常量、变量	常量、抽象方法、(jdk8.0:默认方法、静态方法)
使用	子类继承抽象类(extends)	子类实现接口(implements)
关系	抽象类可以实现多个接口	接口不能继承抽象类，但允许继承多个接口
常见设计模式	模板方法	简单工厂、工厂方法、代理模式
对象	都通过对象的多态性产生实例化对象	
局限	抽象类有单继承的局限	接口没有此局限
实际	作为一个模板	是作为一个标准或是表示一种能力
选择	如果抽象类和接口都可以使用的话，优先使用接口，因为避免单继承的局限	

提示：在开发中，常看到一个类不是去继承一个已经实现好的类，而是要么继承抽象类，要么实现接口。

练习

父类和接口是并列关系，不清楚x是属于谁的。
System.out.println（super.x）；System.out.println（A.x）；
或者变量定义成不同名字
再问一下：如果A是B父类，B是C的父类，C输出x值，结果如何?

- 查错

```
interface A {
    int x = 0;
}
class B {
    int x = 1;
}
class C extends B implements A {
    public void pX() {
        System.out.println(x);
    }
    public static void main(String[] args) {
        new C().pX();
    }
}
```

练习

play 重写2个借口的play()方法
ball 是常量， 不能重新赋值了。

```
interface Playable {
    void play();
}

interface Bounceable {
    void play();
}

interface Rollable extends Playable,
Bounceable {
    Ball ball = new Ball("PingPang");

}
```

```
class Ball implements Rollable {
    private String name;

    public String getName() {
        return name;
    }

    public Ball(String name) {
        this.name = name;
    }

    public void play() {
        ball = new Ball("Football");
        System.out.println(ball.getName());
    }
}
```

编译失败：因为A接口中并未定义func方法。

练习

```
interface A{}
class B implements A{
public String func(){
return "func";}
}
class Demo{
public static void main(String[] args){
A a=new B();
System.out.println(a.func());}
}
```

练习

1)下面关于接口的说法中不正确的是（）。

A．接口中所有的方法都是抽象的

B．接口中所有的方法都是public访问权限

C．子接口继承父接口所用的关键字是implements

D．接口是Java中的特殊类，包含常量和抽象方法

2)Java语言接口间的继承关系是（）。

A．单继承　B．多重继承 C．不能继承 D．不一定

3)一个类实现接口的情况是（）。

A．一次可以实现多个接口　B．一次只能实现一个接口

C．不能实现接口　　D．不一定

练习

接口是否可继承接口？抽象类是否可实现(implements)接口？抽象类是否可继承实体类(concrete class)？

答案：

接口可以继承接口。

抽象类可以实现(implements)接口，

抽象类可继承实体类。

7. 内部类

- 当一个事物的内部，还有一个部分需要一个完整的结构进行描述，而这个内部的完整的结构又只为外部事物提供服务，那么整个内部的完整结构最好使用内部类。
- 在Java中，允许一个类的定义位于另一个类的内部，前者称为内部类，后者称为外部类。

7. 内部类

- Inner class一般用在定义它的类或语句块之内，在外部引用它时必须给出完整的名称。Inner class的名字不能与包含它的外部类类名相同。
- 成员内部类（static成员内部类和非static成员内部类）
- 局部内部类

7. 内部类

- 成员内部类作为类的成员的角色:
  - 和外部类不同，Innerclass还可以声明为private或protected;
  - 可以调用外部类的成员
  - Innerclass可以声明为static的，但此时就不能再使用外层类的非static的成员变量;

7. 内部类

- 成员内部类作为类的角色:
  - 可以在内部定义4种属性、方法、构造器等结构
  - 可以声明为abstract类，因此可以被其它的内部类继承
  - 可以声明为final的
  - 编译后生成OuterClass$InnerClass.class字节码文件
  - 非static的成员内部类中的成员不能声明为static的，只有在外部类或static的成员内部类中才可声明static成员。

7. 内部类

- 实例化成员内部类对象，注意静态和非静态的区别
- 访问成员内部类的成员，需要“内部类.成员”或“内部类对象.成员”的方式
- 成员内部类可以直接使用外部类的所有成员，包括私有的数据

7. 内部类

- 局部内部类：局部内部类可以用成员、代码块、构造函数形式定义
- 只能在声明它的方法或代码块中使用，而且是先声明后使用。除此之外的任何地方都不能使用该类

```
public class Outer {
    private int s = 111;
    public class Inner {
        private int s = 222;
        public void mb(int s) {
            System.out.println(s);
            System.out.println(this.s);
            System.out.println(Outer.this.s);
        }
    }
    public static void main(String args[]) {
        Outer a = new Outer();
        Outer.Inner b = a.new Inner();
        b.mb(333);
    }
}
```

```
333
222
111
```

```
public class Test{
 public Test(){
    Inner s1= new Inner();
    s1.a= 10;
    Inner s2= new Inner();
    s2.a= 20;
    Test.Inner s3= new Test.Inner();
    System.out.println(s3.a);
 }
 class Inner{public int a =5;}
 public static void main(String[] args){
    Test t= new Test();
    Inner r= t.new Inner();
    System.out.println(r.a);}}
```

```
5
5
```

一.静态内部类可以有静态成员，而非静态内部类则不能有静态成员。 故A、B 错

二.静态内部类的非静态成员可以访问外部类的静态变量，而不可访问外部类的非静态变量； 故D 错

```
public class OuterClass {
  private double d1 = 1.0;
    //insert code here }
A. class InnerOne{
     public static double methoda() {return d1;}}
B. public class InnerOne{
     static double methoda() {return d1;}}
C. private class InnerOne{
     double methoda() {return d1;}}
D. static class InnerOne{
     protected double methoda() {return d1;}}
```

编译失败，非静态内部类中不可以定义静态成员。
内部类中如果定义了静态成员，该内部类必须被静态修饰。

```
class TD {
    int y = 6;
    class Inner {
        static int y = 3;
        void show() {
            System.out.println(y);
        }
    }
}

class TC {
    public static void main(String[] args) {
        TD.Inner ti = new TD().new Inner();
        ti.show();
    }
}
```