四、面向对象（上）

面向对象的程序设计思想

1. 三大特征：封装（类与对象）、继承（基类与派生类）、多态（抽象类与接口）
2. 优点：可重用、可扩展、可维护

类中定义的属性和方法

1. 类中的域变量可以被类中所有的方法所访问
2. 方法中的形参和定义发局部变量的作用域仅限于方法，局部变量在使用前必须进行赋值初始化。**如果局部变量和类中的属性变量重名，则在方法中对同名变量改变的是局部变量**

对象

1. 对象与对象引用
   - 创建对象：new FighterPlane() （分配在堆上）
   - 声明一个对象引用： FighterPlane fp; （分配在栈上）
   - **实质是将新创建的FighterPlane对象的地址赋给对象引用fp，从此fp与该对象关联，通过fp即可操纵该对象**
2. 对象作为参数的特点
   - 普通数据类型作为参数传递是**值传递**，而对象是**引用传递**
   - 引用传递，当对象作为参数传递时，传递的是对象的地址

一维数组

1. 声明数组就是要确定数组名、维数和元素的数据类型
2. 声明时不可在方括号内指定数组大小
3. 使用关键字new——可将数组看成一个特殊对象
4. 数组初始化

```
int[] arr = new int[10];     // 此时分配了数组空间，每个单元初始化为默认值0
String[] example = new String[10];  // 对象引用数组
int[] a = {3,4,5,6}      // 不能分开写
```

5. 一维数组的相关方法：
   - 某些类的方法返回数组对象引用： linkedList调用toArray()返回数组，此时数组中为对象的引用
   - Arrays.sort方法
   - System.arraycopy()

二维数组

1. 初始化方式：
    - 数组名 = new 类型说明符[数组长度][];
    - 数组名 = new 类型说明符[数组长度][数组长度];
2. 当只定义第一维维数时，另一维的维数可以不一样。

构造方法

- 创建对象时的初始化顺序：
    1. 系统会对数据成员进行默认初始化
    2. 执行数据成员定义处的初始化语句
    3. 调用构造方法为数据成员指定初值
- finalize方法与垃圾回收
    - 什么是垃圾对象？对象没有任何引用
    - 垃圾对象何时回收？虚拟机在系统资源不够的情况下才会回收垃圾对象
    - 垃圾对象回收时调用**finalize**方法垃圾回收不能保证一定会发生

类成员属性和方法的其他修饰符

1. static

- 用static修饰符修饰的数据成员是不属于任何一个类的具体对象，而是属于类的静态数据成员
- 它被保存在类的内存区的公共储存单元中，而不是不保存在某个对象的内存区中。因此，一个类的任何对象访问它时，存取道德都是相同的数值。
- 访问方式为通过类名加点操作符来访问，也可通过对象引用来访问
- 静态数据成员仅在类加载时进行初始化，且只执行一次初始化

2. static修饰的方法有如下特点：
    - static方法是类方法，但可以被所有对象所访问，引用这个方法时，可以使用对象名做前缀，也可以使用类名做前缀
    - static方法内部的代码，只能访问类中的static属性或方法，不能直接访问类中非static属性或方法（因为那是对象方法），但非static方法（对象方法）可以访问static数据成员
    - main方法是特殊的静态方法，是Application程序入口点，必须写成public static void main(String args[])的形式
3. final
    - final修饰属性，则属性为常量
    - 如果修饰方法，则方法称为最终方法，在子类当中不能被覆盖，可防止任何继承类修改此方法，保证了程序的安全性和正确性

包

- 包是对Java类进行组织与管理的一种机制

五、面向对象（中）

封装

1. 访问控制
   - **public（接口访问权限）**
   - protected（包访问权限、继承访问权限）
   - "默认"（包访问权限）
   - **private（无法直接访问）**

类属性成员与方法\类前修饰符	**public**	默认
public	所有类	包中的类
protected	包中的类（对象引用）	包中的类
默认	包中的类	包中的类
private	本类	本类

2. protected属性具有包访问权限，可以被同一包中的所有类访问

3. 子类的类定义中可以访问父类的protected属性和方法，不论是否在同一包

```
// public类的private属性的单例设计模式
public class FighterPlane {
    private String name;
    private int missileNum;
    private static FighterPlane fp;
    private FighterPlane(String _name, int _missileNum) {
        name = _name;
        missileNum = _missileNum;
    }
    public static FighterPlane getInstance(String _name, int _missileNum) {
        if(fp==null) {
            fp = new FighterPlane(_name,_missileNum);
        }
        return fp;
    }
}
public class RunPlane {
    public static void main(String args[]) {
        FighterPlane fp;
        fp = FighterPlane.getInstance("苏35",6);
    }
}
```

4. 访问权限首先看类前的修饰符，再看方法前的修饰符。

5. 面向对象程序 = 对象 + 消息

    - 消息的实质就是引用向对象发出服务请求，是对数据成员和成员方法的调用，例如fp.name和fp.fire()就是发送消息

6. 能否发送消息取决于：

    - 引用必须引用了特定的对象，否则会在运行时抛出NullPointerException
    - 对象必须定义了相应的属性或方法，否则编译不会通过
    - 被访问的属性或方法必须具有

继承

- 通过**extends**关键字实现
- 子类继承了父类的所有属性和方法，但只有public、protected的属性和方法在子类是可见的
- 子类在继承父类的时候，首先应该满足父类可被访问，例如当子类和父类不在同一个包当中时，
- 子类在继承父类的时候，首先应该满足父类可被访问，例如当子类和父类不在同一个包当中时，父类修饰符必为public

1. 子类可以直接访问父类的protected属性和方法

2. 子类不能直接访问父类的private属性和方法，可以调用父类的公共方法来间接访问私有属性

3. Object类是所有类的**共同祖先**，即使定义类时没有写extends Object

多态

1. Java中提供两种多态的机制：重载与覆盖

2. 覆盖：子类对父类的同名方法（方法名称相同，参数相同，返回类型相同）重新进行定义。

3. 覆盖——注意：

    - 子类的访问修饰符权限**应等于或大于**父类
    - 同名的static方法和非static方法不同相互覆盖
    - 方法前有final修饰符，此方法不能在子类方法中进行覆盖
    - 抽象类中如果存在抽象方法，则具体子类必须对抽象方法进行覆盖

六、面向对象（下）

this与super

1. this
    - 表示当前对象引用
    - 表示当前对象
    - 调用当前类的构造函数

```
// 1
public class FighterPlane {
    String name = "苏35";
    void init(String name) {
        this.name = name;
    }
```

```
}
// 2
public A(FighterPlane fpp) {
    this.fp = fpp;
    fpp.setA(this); //将当前对象传给FighterPlane
}
// 3
class AddClass {
    public int x=0,y=0,z=0;
    AddClass(int x) {
        this.x = x;
    }
    AddClass(int x,int y) {
        this(x);
        this.y = y;
    }
}
```

2. super

    - 访问当前类的直接父类
    - 子类的数据成员或成员方法与父类的数据成员或成员方法名字相同时，当要调用父类的同名方法或同名数据成员时则可用super来指明。
    - super(参数)，表示调用父类构造方法

抽象类

1. 抽象类的概念：要你管abstract修饰的类称为**抽象类**，用abstract修饰的成员方法称为抽象方法
    - 抽象类中可以有零个或多个抽象方法，也可以包含非抽象方法
    - 抽象类不能创建对象，创建对象由具体子类来实现，但可以有**声明**，声明能引用所有具体子类的对象
    - 对于抽象方法，在抽象类中只指定方法名及类型，而不写实现代码。抽象类必定要派生子类，若派生的子类是具体类，则具体子类中必须实现抽象类中定义的**所有抽象方法**（覆盖）
    - 若子类还是抽象类，如果父类中已有同名abstract方法，则子类中就不能再有同名的抽象方法了
    - abstract不能和与final并列修饰同一个类；abstract不能与private，staitc（因为static修饰的方法比如被直接调用），final或native并列修饰同一个方法

接口

1. 两种含义：

    - 可以被引用调用的方法
    - 同"类"概念地位相当的专有概念interface，interface是方法说明的集合

2. 定义接口要注意：

    - 接口定义用关键字interface，而不是class，interface前的修饰符要么为public，要么为缺省
    - 接口定义的数据全是final static。即使没有修饰符，其效果也等效，访问级别要么为public，要么为缺省

- 接口中没有构造方法；**所有成员方法都是抽象方法**。即使没有修饰符，其效果完全等效，访问级别要么为public，要么为缺省
- 接口具有**继承性**，可通过extends关键字声明接口的父接口

```
// 定义接口
interface Washer {
    abstract void startUp();
    abstract void letWashIn();
}
// 实现接口
class RoaseBrand implements Washer {
    public void startUp() {
        System.out.println("startup");
    }
    public boif letWasherIn() {
        System.out.println("letWaterIn");`
    }
}
// 使用接口
public class Consumer {
    public static void main(String args[]) {
        Washer w = new RoaderBrand();
        w.startUp();
        w.letWaterIn();
        //w.dehydrate(); 当接口调用接口未定义的功能方法，编译会报错
    }
}
```

3. 接口实现到注意几点
   - 一个类可以实现**多个**接口
   - 接口的实现者可以继承接口中定义的常量
   - 如果实现某接口的类不是abstract的抽象类，则在类的定义部分必须实现接口的所有抽象方法
   - 如果实现接口的类是abstract类，则它可以不实现该接口的所有方法。

抽象类与接口

name	抽象类	接口
共同点	二者都可具有抽象方法，都不能实例化，但都可以有自己的声明，并能引用子类或实现类对象	
属性变量	可以有变量	不能有，只能是静态变量
成员方法	可以有具体方法（而且具体方法可以调用抽象方法）	如果有方法，则全部是抽象方法
实现策略	必须有子类继承	必须有实现类实现

name	抽象类	接口
扩展性	弱	强

引用

1. equals方法：equals方法是Object类的方法，比较本引用和参数指明的某个引用啊是否相等，即是否指向同一对象。返回true或false
2. 使用==比较：如果两边是引用则比较的是它们的引用是否相同；如果两边是数值，则比较的是它们的值
3. instanceof的用法：判断对象是否为某个类可引用的实例

类的其他内容

1. 内部类
2. 匿名内部类
3. 匿名对象