Java-面向对象

- 面向对象
  - 程序=对象+消息
  - 三大特征
    - 封装：类与对象
    - 继承：基类和派生类
    - 多态：抽象类与接口
  - 优点
    - 可重用（通用代码）
    - 可扩展（插入新的功能）
    - 可维护（代码结构优良，接口清晰）

类与对象

- 对象是对客观世界的抽象
- 类是对对象的抽象，在语言中是一种抽象的数据类型
- 对象是类的实例，类是对象的模板

属性与方法

- 属性(域)：事物静态特征的抽象
  - 类中定义的属性可以被类中所有的方法访问
  - 方法中使用的形参及局部变量，作用域仅限于这一方法。同名的局部变量会覆盖类属性，使得方法仅访问局部变量
- 方法(操作)：事物动态特征的抽象

对象与对象引用

```
ClassA obj = new ClassA();
// 实质是将新创建的CLassA对象的地址赋给对象引用obj
```

- 创建对象：`new ClassA();` （分配在堆上）
- 声明一个对象引用，相当于别名：`ClassA obj;` （分配在栈上）
- 对象作为参数的特点：
  - 参数的传值机制：普通数据类型作为参数传递是值传递，而对象是引用传递
  - 也就是说传参时只会传递对象的地址，并不会为形参开辟新的内存空间。

对象的创建与回收

- 创建对象的初始化顺序
  - 系统会对数据成员进行默认初始化
  - 执行数据成员定义处的初始化语句，如`int x=1`中令x为1
  - 调用构造方法为数据成员制定初值。
- 对象的回收
  - Java中没有析构函数，对象在没有任何引用（垃圾对象）时由系统自动回收。
  - 系统在内存不足时会释放垃圾，也可以手动 System.gc() 释放垃圾。

封装

访问控制

- 访问控制符包括`public`，`protected`，`private`. 这些修饰符可以用于修饰类、类的属性成员和类的成员方法，用于标记不同主体对这一类/成员的访问权限。
  - 类前修饰符：可以写public或缺省(不写)
    - 声明为public的类，可以被所有类的对象访问到，否则只能够被相同包(package)中的类访问到。
    - 若将该类声明为公共类，则这一源文件的文件名须与类名保持相同。
    - 注：除了访问控制符外，类前只能用`final`，`abstract`进行修饰。
  - 类成员(属性&方法)前修饰符：三种访问控制符都可以使用
    - public: 所有类的对象都可以访问到该成员（在该类是public类的情况下）
    - protected: 同一个包下的所有类可以访问到该成员。同时该类的所有子类对象都可以访问到该成员，忽略包的限制。
    - 缺省：同一个包下的所有类可以访问到该成员。
    - private: 只有本类的对象可以访问到该成员。
  - 

- 注：“访问”的主体是调用该方法或对象的代码所属的主体。即这行代码在哪个类里面，访问主体就是谁。

继承与子类

- 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` 和 `protected` 的属性和方法在子类是可见的。（如果子类和父类在同一个包，缺省成员也可见，详见上文）
- Java中可以多重继承，但不存在多继承（一个类不能extends多个类）。

Object类

- Object类是所有类的共同祖先。所有类都隐式地 `extends Object`
- 在Object中定义了许多方法，它们都可以被所有子类所继承。常见的方法如下：
  - `clone()`：将当前对象克隆
  - `Boolean equals(Objects obj)`：判断两个引用是否指向同一对象
  - `Class getClass()`：获得当前对象的类对象
  - `void finalize()`：对象被释放时使用
  - `int hashCode()`：当前对象的哈希值
  - `String toString()`：代表这个对象的字符串

this与super关键字

- super关键字：通过 super 关键字来实现对父类成员的访问，用来引用当前对象的父类。
  - 在子类的构造函数中，我们可以通过 `super(父类构造函数的参数)` 来调用父类的构造方法。
  - 在子类方法中，可以通过 `super.方法名` 调用父类的方法。通常用于子类重写父类同名方法/覆盖同名数据成员的情况。
- this关键字：指向自己的引用。
  - 采用 this 关键字是为了解决实例变量和局部变量之间发生的同名的冲突。

构造器

- 子类是不继承父类的构造器(a.k.a 构造方法or构造函数）的，它只是调用（隐式或显式）。如果父类的构造器带有参数，则必须在子类的构造器中显式地通过 `super` 关键字调用父类的构造器并配以适当的参数列表。
- 如果父类构造器没有参数，则在子类的构造器中不需要使用 `super` 关键字调用父类构造器，系统会自动调用父类的无参构造器。
- 子类在进行构造时，先调用父类的构造方法，再执行当前子类的构造方法。

多态

- 多态的概念：一个程序中同名的不同方法共存的情况。

- Java中实现多态共有三种方法。

重写(Override)和重载(Overload)

- 重写(Overriding a.k.a 覆盖)是子类对父类的允许访问的方法的实现过程进行重新编写，返回值和形参都不改变。**即外壳不变，核心重写。**
  - 重写的好处在于子类可以根据需要，定义特定于自己的行为。 也就是说子类能够根据需要实现父类的方法。
  - 可以在方法前标记`@Override`，表示该方法重写了父类的方法。
  - 返回值和形参
    - 参数列表与被重写方法的参数列表必须完全相同。
    - 返回类型与被重写方法的返回类型也可以不相同，但是必须是父类返回值的派生类。
  - 访问权限问题
    - 访问权限不能比父类中被重写的方法的访问权限更低。例如：如果父类的一个方法被声明为 public，那么在子类中重写该方法就不能声明为 protected。
    - private 和 final 的方法不能被重写。
- 重载(overloading) 是在一个类里面，方法名字相同，而参数不同。返回类型可以相同也可以不同。每个重载的方法都必须有一个独一无二的参数类型列表。最常用的地方就是构造器的重载。

区别点	重载方法	重写方法
参数列表	必须修改	一定不能修改
返回类型	可以修改	一定不能修改
异常	可以修改	可以减少或删除，一定不能抛出新的或者更广的异常
访问	可以修改	一定不能做更严格的限制（可以降低限制）

抽象类

- 用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。
  - 抽象类除了不能实例化对象之外，类的其它功能依然存在，成员变量、成员方法和构造方法的访问方式和普通类一样。
  - 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。只要有一个抽象方法，类前就必须有abstract修饰。若没有抽象方法，类前也可有abstract修饰。
- 由于抽象类不能实例化对象，所以抽象类必须被继承，才能被使用。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）。
- 父类包含了子类集合的常见的方法，但是由于父类本身是抽象的，所以不能使用这些方法。

接口

- 接口（英文：Interface），在Java编程语言中是一个抽象类型，是抽象方法的集合，接口通常以interface来声明。一个类通过实现接口的方式，从而来实现接口的抽象方法。
  - 除非实现接口的类是抽象类，否则该类要定义接口中的所有方法。
  - 接口无法被实例化，但是可以被实现。一个实现接口的类，必须实现接口内所描述的所有方法，否则就必须声明为抽象类。另外，在 Java 中，接口类型可用来声明一个变量，他们可以成为一个空指针，或是被绑定在一个以此接口实现的对象。
  - 接口不是被类继承了，而是要被类实现。类使用`implements`关键字实现接口。在类声明中，Implements关键字放在class声明后面。

接口的实现

- 一个类可以同时实现多个接口。
- 一个类只能继承一个类，但是能实现多个接口。
- 一个接口能继承另一个接口，使用`extends`关键字。

接口特性

- 接口是隐式抽象的，当声明一个接口的时候，不必使用**abstract**关键字。
- 接口没有构造方法。
- 接口中每一个方法也是隐式抽象的，接口中的方法会被隐式的指定为 **public abstract**（只能是 public abstract，其他修饰符都会报错）。
- 接口中可以含有变量，但是接口中的变量会被隐式的指定为 **public static final** 变量（并且只能是 public，用 private 修饰会报编译错误）。
- 接口中的方法是不能在接口中实现的，只能由实现接口的类来实现接口中的方法。

抽象类和接口的区别

- 1. 抽象类中的方法可以有方法体，就是能实现方法的具体功能，但是接口中的方法不行。
- 2. 抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是 **public static final** 类型的。
- 3. 接口中不能含有静态代码块以及静态方法(用 static 修饰的方法)，而抽象类是可以有静态代码块和静态方法。
- 4. 一个类只能继承一个抽象类，而一个类却可以实现多个接口。

		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。	
不同点	属性变量	可以有变量	不能有，只能是静态常量。
	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
	扩展性	弱	强

修饰符

static

- static修饰属性或方法后，属性和方法不在属于某个特定的对象，而是属于类的静态数据成员。也可以说是static成员不依赖某个对象，在类加载时就被初始化（只被初始化一次）。
- 访问：static修饰的属性或方法，可以直接使用类名调用，而不用先实例化对象再调用。如Math类中的诸多方法都是静态方法。
- 静态方法不能访问非静态属性和方法，因为静态方法可能在对象之外被调用，此时非静态属性方法没有被初始化。
- 在类中也可以使用不包含在任何方法体中的静态代码块。当类被装载的时候执行，且只执行一次，通常用来初始化静态属性。

final

- final 可以用来修饰变量（包括类属性、对象属性、局部变量和形参）、方法（包括类方法和对象方法）和类。类比C++中的`const`.
- 使用 final 关键字声明类，就是把类定义定义为最终类，不能被继承。
- 或者用于修饰方法，该方法不能被子类重写。
- 用final修饰的变量必须初始化，且之后不能赋值。
- 注： final 定义的类，其中的属性、方法不是 final 的。