Java

使用类库：继承或调用方法

**协议**：互相遵守的规则 **耦合**：层次的依赖性

Java中的对象默认继承Object

**C++内存管理：**

动态分配的内存空间，其地址pAddress, 可能有多个变量获得此地址。地址的特点：获得地址信息即可操作内存单元。而这多个地址操作的不同步，可能会报错或bug。于是，需要内存管理。而复制构造函数就要注意这问题，即深拷贝。

**垃圾内存及回收：**

动态分配的内存空间而没有被指向，回收没有被引用的内存，保证有引用的内存不被释放(null无指向)

**环境变量：**

可执行文件在当前目录找不到时，会在环境变量的路径中寻找

**基本数据类型：**

局部变量（函数内），必须初始化。当一个对象创建时，数据成员默认全为零。

**对象变量 实际为 引用类型（即地址）**

拓宽类型widening a type，可自动转换

缩窄类型narrowing a type，必须显式完成

装箱、拆箱：

基本数据类型 -> Object -> 基本数据类型，无疑性能下降

**参数传递：** Only值传递

**类的外部访问控制符： （对内无限制）**

**Private：**当前类可访问（不对外开放）

**//Default（不写）：**当前包

**Protected：**当前包可访问，可被其他包中的类继承使用（跨包子类内部使用）

**Public：**任何包任何类 不限制（跨包访问）

外部访问：1、类名.成员 2、对象.成员

（类只有public和default权限,一个文件中最多一个public class）

**Static：**

静态成员属于类的，只能访问静态成员，为类的所有对象共享。编译时绑定

对于静态变量在内存中只有一个拷贝（节省内存），JVM只为静态分配一次内存，在加载类的过程中完成静态变量的内存分配，可用类名直接访问

**构造函数：no destructor**

函数名为类名，却不写返回值类型. 成员函数名可以是类名。（C++不能）

对象创建时，自动调用.

**（**若自定义构造函数，则编译器不会产生默认的构造函数**）**

区别：C++不允许数据成员定义时，初始化。而Java允许，之后执行构造函数

**This：指向本身**

同一类的不同对象共用一个非静态方法，根据对象的this引用来操作.

public A(int age)

{

this.age = age;

}

**继承：子类是父类的拓展**

一个新类（派生类）从已有类（基类）获得属性和方法（除构造和析构外）

子类可以继承父类的所有非私有的成员。

（泛化）父类 -> 子类 【一般 –> 特殊】

Java支持单一继承，不允许多重继承（一个类可继承多个父类）.

不支持多继承原因: 多重继承的二义性

**Super : 指向父类**

调用基类的构造函数，super(实参表); （派生类构造函数中第一条语句）

**多态：One接口，Multi实现**

父类的引用类型可以指向父类或子类，根据当前时刻的指向而采取不同的方法

Java语言全面支持动态绑定，而C++语言只对虚函数使用动态绑定

@Override在子类中重写只是隐藏了父类的方法, 权限不能变低

一个接口，多种实现。便于扩充子类

（继承：特殊的不断具体化）

子类可以当成父类看待，但父类不是子类的一种

**Final：**

最终的，perfect，不可改变,前期绑定

-属性，只读

-方法，不可被重写（编译时绑定）

-类，不可被继承

**接口Interface：交互的通道** （如:USB接口，驱动程序就是implement）

服务器 与 客户端 之间的契约，可功能扩展

【规定一系列的（public static final）属性和（public abstract）**方法**的模型】

接口之间可以继承，甚至多继承

一个类实现接口的**部分**方法，则一定是抽象类，先extends后implements

Instanceof 判断是否包含接口的实现

由于使用某接口须实现所有方法，以通用实现类为拓展，重写特定方法，很有必要。

**抽象类：**

不可实例化，**Abstract，**可带有部分方法的实现

通常，作为一个类族的最顶层父类，表示一类事物的共性

**包Package：**

java…核心库 javax…拓展库

逆域名命名：com.sun

真正的类名，为包名和原类名的组合

给该文件中的所有类打包，新建文件夹

（若没显式声明，则在默认包中）

【Javac –d . res.java 可以自动建立包层】

一个源文件中，只有一个公共类，可以含多个非公共类

**异常Exception：**便于调试，将错误处理与主干代码分开

try + catch或finally（可以解决if-else无能为力的情况）

运行时的不希望发生的事件，就抛出特定类型的异常对象,若不捕获处理，则终止程序

必须处理 非RuntimeException的Exception类型异常

throws… 表明 潜在的异常交由调用者处理

建议：方法内部已处理异常，则不用throws

对throws的所有异常处理

finally{} 提供程序的统一出口，收尾工作

1. **子类抛出的异常不能是父类的异常之外的。（继承的多态）否则无法用一个函数，对父类及其子类统一处理**
2. **假定 A extends Exception; B extends A; C extends B**

**则捕获次序：C->B->A ∵既然父类都出错了，那么**基于父类的**子类必出错**

功能的强大，必然带来操作的繁多。不要依赖异常处理