1. 多态
   1. 静态多态 --- 函数重载 、运算符重载
   2. 动态多态 --- 父子之间继承 + 虚函数
   3. 动态多态满足条件
      1. //1、父类中有虚函数
      2. //2、子类重写父类的虚函数
      3. //3、父类的指针或者引用 指向子类的对象
   4. //重写 子类重新实现父类中的虚函数 ，必须返回值、函数名、参数一致才称为重写
   5. //子类在做重写时候 ，可以不加关键字 virtual
2. 多态原理
   1. 当父类中存在虚函数后，内部发生结构变化
   2. 多了指针 vfptr 虚函数表指针 --- 指向 虚函数表 vftable
   3. 虚函数表内部 记录着虚函数的地址
   4. 当子类发生重写后，会修改子类中的虚函数表中的函数地址，但是并不会影响父类中的虚函数表
   5. 深度剖析内部工作原理
3. 多态案例 --- 计算器案例
   1. //多态的好处： 对扩展性提高，组织性强，可读性强
   2. //如果父类中有了虚函数，子类并没有重写父类的虚函数，那么这样的代码是毫无意义的
   3. //如果子类不重写父类虚函数，那么没有用到多态带来的好处 ，而且内部结构还变得更为复杂
   4. 开闭原则 对扩展进行开放 对修改进行关闭
4. 纯虚函数和抽象类
   1. 纯虚函数语法 virtual void func() = 0;
   2. 如果类中有了纯虚函数，那么这个类也称为 抽象类
   3. 抽象类 是无法实例化对象的
   4. 继承了抽象类的子类，必须要重写父类中的纯虚函数，否则的话，子类也是属于抽象类，无法实例化对象了
5. 虚析构和纯虚析构
   1. 如果子类中有属性创建在堆区，那么多态情况下，不会调用子类的析构代码，导致内存泄露
   2. 解决方案：利用虚析构或者纯虚析构
   3. 虚析构 在析构前加 关键字 virtual 即可
   4. 纯虚析构 virtual ~函数名() = 0
   5. 纯虚析构 类内声明、类外必须要实现
   6. 如果一个类中有了纯虚析构函数，那么这个类也属于抽象类
6. 向上向下类型转换
   1. 父类转子类 向下类型转换 不安全
   2. 子类转父类 向上类型转换 安全
   3. 如果发生多态，那么总是安全
7. 重载 重写 重定义
   1. 重载 函数重载
      1. 同一个作用域 函数名相同
      2. 参数 个数、类型、顺序不同满足条件
      3. 返回值不可以作为重载条件
   2. 重写
      1. 继承关系
      2. 父类中有虚函数
      3. 子类可以重写父类中的虚函数，返回值、函数名、参数类别都一致
   3. 重定义 （隐藏）
      1. 继承关系
      2. 非虚函数 子类重新定义 父类中同名的成员函数