一、 java面向过程编程

1. 核心要素

- 数据结构： 原生类型、对象类型、数组类型、集合类型
- 方法调用：访问性、返回类型、方法参数、异常等
- 执行流程：赋值、逻辑、迭代（循环）、递归等。

基本类型 从小到大

byte(8) char(16) short(16) int(32) long(64)

注意 long 或者 double可能会出现线程不安全的问题。

访问性从高到底 （java8）

public : all

protected : 集成+同包

default： 同包

private： 当前类

java9之后支持模块化，增强了访问性的封装。

Reference引用。其实java还有FinalReference 引用，这个类是一个同包才能访问的：

这个就和Object类中的finalize有关，finalize是一个protected方法，必须继承或者同包访问。

二、java面向对象基础

1. 基本特性

- 封装
- 派生
- 多态

2. 基础

设计模式：

- Gof 23 设计模式：构建、结构、行为

- 方法设计：名称、访问性、参数、返回类型、异常
- 泛型设计：类级别、方法级别
- 异常设计：传播性、层次性

方法设计中的重点：
- 单元：一个类或者一组类（组件）。
    - 类采用名称结构
        - 动词过去式 + 名词
        - 动词ing+ 名词
        - 形容词 + 名词
- 执行： 某个方法
    - 方法命名
        - 方法命名动词
        - 方法参数名词

异常设计：
- 异常
    - 根（顶级）异常 （不推荐）
        - Throwable
            - checked类型 Exception
            - 非checkd异常： RuntimeException
            - Error： 不常用
        - jdk 1.4之后的 StackTraceElement
            - 添加异常原因 cause
            - 反模式：吞并异常
            - 注意方法性能消耗： Throwable中构造函数 调用的fillInStackTrace()方法，这里可以去 -（1）避免异常栈调用深度（JVM参数控制深度，物理层面）
                -（2）logback日志框架控制堆栈输出深度（逻辑屏蔽）

泛型设计
- java泛型属于编译时处理，运行时擦除。
- 同名方法 参数两个List是被视为一样的 不是方法的重载，会编译不通过
- T 和 ？ 作为泛型的区别：

三、 java函数式编程基础

函数式包含：

- lambda表达式
- 默认方法
- 方法引用

1. 匿名内置类

- 使用场景

java是一门面向对象的静态语言，其封装性能屏蔽数据结构的细节，从而更加关注模块的功能性。其静态性也确保了java强类型的特性。随着模块功能的提升，伴随而来的是复杂度的增加，代码的语义清晰依赖于开发人员抽象和命名类或方法的能力。尽管编程思想和设计模式能够促使编程风格趋于统一，然后很多业务属于面向过程的方式，这与面向对象编程在一定程度 上存在一些冲突。java编程为了解决这个问题，引入了匿名内部类的方案。

- 匿名内部类典型场景

java Event / Listener

java Concurrent

Spring Template

- 匿名内部类的特性：

（1）无需名称

（2）声明范围：

- static block
- 实例 block
- 静态方法或实例方法中
- 构造函数中

（3）类的全名称：${package}.${declared_class}.${num}

（4）虽然无需名称，但是其实字节码中是根据定义顺序生成子类。

可以javap -v 看一下对应的字节码。

- 匿名内部类的特点：

基于多态
实现类无需名称
允许多个抽象方法

2. lambda表达式

基本特点：
（1）流程编排清晰
（2）函数类型编程
（3）改善代码臃肿
（4）兼容接口升级

实现手段：
（1）@FunctionInterface接口
（2）Lambda语法
（3）方法引用
（4）接口default方法实现

3. 接口默认方法
使用场景：
当接口升级时，添加了新的抽象方法，此时基于老接口的实现类必然会遇到编译问题。默认方法的出现解决了这个问题。同时也能为实现类提供默认或者样板实现，减少实现类的负担，无需再使用Adapter实现。

四、问题：

1. 泛型中的？和 T
T一般用在接口定义上，？ 一般用在方法参数上。
T 通常作为一个范围约束，可以约束上下界

2. Throwlable类中的构造函数加入的一个参数 writableStackTrace
Throwable中构造函数 调用的fillInStackTrace()方法 ， 这里可以去
- （1）避免异常栈调用深度（JVM参数控制深度，物理层面）
（2）logback日志框架控制堆栈输出深度（逻辑屏蔽）

还有个不错的选择，Throwable提供了构造函数中参数writableStackTrace， 如果是false，就不会调用fillInStackTrace 方法。

```
55          */
56      @   protected Throwable(String message, Throwable cause,
57                              boolean enableSuppression,
58                              boolean writableStackTrace) {
59              if (writableStackTrace) {
60                  fillInStackTrace();
61              } else {
62                  stackTrace = null;
63              }
64              detailMessage = message;
65              this.cause = cause;
66              if (!enableSuppression)
67                  suppressedExceptions = null;
68          }
```

3. 泛型的类型

（1）就是一个T , 代表Object，适用于任何类型

（2）上限 <T extends Serializable>

（3）下限 <T super > 比如在HashMap中 的 computeXXX方法参数中用到。

4. try catch 后不要直接 printStackTrace。

printStackTrace()方法会导致异常堆栈输出到标准错误流中（System.err），会出现比较慢的抢占的情况。

5. Refrence的应用场景

强、软、弱、幻象、finalReference引用几种，分别和GC的阶段有关。

6. 泛型为啥不能执行基本类型

这是Java设置的限制，希望泛型面向对象而非原生类型。