# Java程 序 设 计

## 第 6 讲：方法设计、JVM 栈帧与参数值传递机制

厦门大学嘉庚学院

XIAMEN UNIVERSITY TAN KAH KEE COLLEGE

CONTENTS

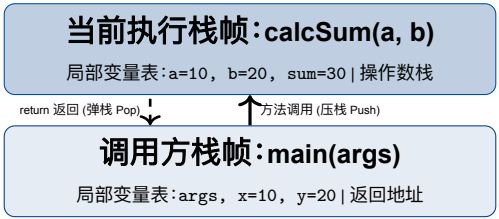
# 方法的定义与结构解构

**【知识要点】方法声明五大要素**

[修饰符] 返回值类型 方法名(形参列表) [throws 异常] { 方法体 }

- **修饰符**:如 `public static`;
- **返回值类型**:若无返回值必须显式声明为 `void`;
- **形参列表**:方法调用时接收外部传入数据的占位局部变量。

# JVM 线程栈帧（Stack Frame）结构图解



# 核心定论：Java 中只有值传递 (Pass-by-Value)



### 基本类型传参 (传数值副本)

- 传递的是栈中数值的一份**独立拷贝**；
- 方法内部对形参的任何修改，**绝不影响**外部实参变量！

### 引用类型传参 (传地址副本)

- 传递的是指向堆内存对象的**地址引用拷贝**；
- 通过引用修改对象内部属性，**外部同步生效**！

# 随堂概念抢答 1:参数值传递轨迹预测

**【随堂抢答·概念检测】问题:执行以下代码后,控制台输出的内容是?**

```java
public static void change(int x, int[] arr) {
    x = 999;
    arr[0] = 999;
}
public static void main(String[] args) {
    int num = 10;
    int[] list = {10, 20};
    change(num, list);
    System.out.printf("num=%d, list[0]=%d\n", num, list[0]);
}
```

A. num=10, list[0]=10

B. num=999, list[0]=999

C. num=10, list[0]=999

D. num=999, list[0]=10

**【知识要点】正确答案:【C 】**

**原理解析**:

- num 是基本类型,change 方法内部修改的是独立形参副本,实参 num 仍为 10;
- list 是引用类型,实参与形参指向堆内存中同一块连续数组空间,通过 arr[0]=999 修改物理堆内存数据,实参同步变为 999!

# 方法重载(Overload)三要素

**【知识要点】重载充要条件**

在同一个类中,方法名相同,且形参列表**必须不同**(满足以下任一即可):

1. 参数个数不同;
2. 参数数据类型不同;
3. 不同类型参数的先后顺序不同。

**注意**:仅返回值类型不同或修饰符不同,**不能构成重载,编译报错**!

# 随堂找茬 1:伪重载导致编译报错

**【随堂找茬·避坑挑错】看看以下代码为什么会发生编译报错?**

```java
public class Calculator {
    // 方法 1
    public int compute(int a, int b) { return a + b; }

    // 编译报错: Method compute(int, int) is already defined!
    public double compute(int a, int b) { return (double)(a + b); }
}
```

**【知识要点】避坑指南**

调用时如 `compute(1, 2)`,JVM 无法依据返回值推断应调用哪一个方法!返回值类型不能作为重载的区分依据!

## 【代码演练】规范方法文档编写

```java
/**
 * 计算传感器温度校验和
 * @param data 原始遥测数据字节数组
 * @param length 有效载荷长度
 * @return 8位无符号累加和校验码
 * @throws IllegalArgumentException 当数据为空时抛出
 */
public static byte calculateCrc(byte[] data, int length) {
    if (data == null) throw new IllegalArgumentException("Data is null");
    // 计算逻辑...
    return 0;
}
```

# 递归调用机制与基线条件 (Base Case)



**【知识要点】递归两大支柱**

1. **基线条件 (Base Case)**：递归终止的出口，防止无休止无限调用；
2. **递归推进 (Recursive Step)**：将大问题拆解为规模更小的子问题，向基线条件收敛推进。

```java
public static long factorial(int n) {
    if (n <= 1) return 1; // 基线出口
    return n * factorial(n - 1); // 递归推进
}
```

# 随堂找茬 2：缺少基线条件导致 StackOverflowError

**【随堂找茬·避坑挑错】看看以下代码为什么会导致 JVM 瞬间崩溃?**

```java
public static int sum(int n) {
    // 致命错误：遗漏了 if (n <= 1) return 1; 基线终止条件！
    return n + sum(n - 1);
}
```

**【知识要点】JVM 栈内存溢出原理解析**

缺少基线条件会导致方法无限压栈，快速耗尽 JVM 线程栈内存空间（默认 1MB），抛出致命 `java.lang.StackOverflowError` 导致进程崩溃！

# 随堂抢答 2：重载方法调用优先级匹配

**【随堂抢答·概念检测】问题：类中定义了 print(int x) 与 print(double x)，调用 print(5) 会执行哪一个？**

A. 优先精确匹配调用 `print(int x)`

B. 随机调用一个

C. 优先调用 `print(double x)`

D. 产生编译歧义报错

**【知识要点】正确答案：【A 】**

**原理解析**：

- JVM 在编译期依据最精确匹配原则（Most Specific Match）选择方法；
- 字面量 5 是 `int` 类型，存在精确匹配的 `print(int)`，因此立即绑定该方法；
- 只有当不存在 `int` 版本时，才会发生自动类型提升调用 `print(double)`。

# 随堂实操练笔 1：值传递与引用传递验证

**【随堂实操·动手练笔】任务 1：编写 ValuePassDemo.java**

编写 `ValuePassDemo.java`，实现：

1. 编写交换两个基本整型的方法 `swap(int a, int b)`，并在 main 中验证其为何无法实现交换；
2. 编写交换数组前两项的方法 `swapArray(int[] arr)`，验证其成功交换的原因。

# 随堂实操练笔 2：重载计算器设计

## 【随堂实操·动手练笔】任务 2：编写 MethodOverloadDemo.java

编写 `MethodOverloadDemo.java`，实现：

1. 提供重载的 `add(int a, int b)`；
2. 提供重载的 `add(double a, double b)`；
3. 提供重载的 `add(int a, int b, int c)`；
4. 编写标准的 Javadoc 文档注释（`@param`，`@return`）。

# 工程视野与模块化封装

**【工程视野】高内聚低耦合与系统工程**

方法是复用与封装的最小细胞;“单一职责、高内聚低耦合”,体现大型复杂工程的分工协作与治理理念。

**【实战技巧】代码审查与辅助开发**

**实战建议**:利用 AI 为方法一键生成规范 Javadoc 注释,并生成覆盖边界极端值的单元测试(JUnit)。

**【随堂实操与课后思考】课后思考与巩固**

- **练习**:编写递归与循环两种方式实现斐波那契数列,对比执行耗时。
- **思考**:深度递归中,如何将递归算法改写为显式栈迭代循环以防栈溢出?

# 可变长参数 (Varargs) 机制与规范

**【知识要点】语法：数据类型... 参数名**

- 本质是由编译器自动包装为对应类型的数组；
- **铁律**：一个方法中最多只能有**一个**可变长参数，且**必须位于形参列表的最末尾**！

```java
public static int sum(String prefix, int... numbers) {
    int total = 0;
    for (int n : numbers) total += n;
    return total;
}
```

# 递归调用栈深度与尾递归优化思考

**【知识要点】JVM 栈内存与递归深度限制**

递归调用每一层都会创建独立的栈帧压入线程栈；当递归深度过大（如数万层）时，会耗尽栈空间触发 `StackOverflowError`！工业级开发中对于大深度问题，应使用显式循环或数据结构代替隐式递归。

# 随堂抢答 3：可变长参数位置规则

**【随堂抢答·概念检测】问题：以下哪个方法签名在编译时会直接报错？**

A. `public static void log(String tag, int... vals)`

B. `public static void log(int... vals, String tag)`

C. `public static void log(int... vals)`

D. `public static void log(double d, int... vals)`

# 随堂抢答 3：答案与深度解析

**【知识要点】正确答案：【B 】**

**原理解析**：

- 可变长参数必须放在形参列表的最末尾；
- 选项 B 将 `int... vals` 放在首位，导致编译器无法确定可变参数的结束边界，直接报编译语法错误！

# 课程互动与讲师主页

## 在线主页与课程资源

- **个人主页**：
  `https://matnoble.top`
- **课程专栏**：
  `https://matnoble.top/courses/`
- **GitHub 仓库**：
  `https://github.com/MatNoble`

## 思考：Questions & Answers

**“学贵善疑，逻辑筑基”**

感谢大家的专注聆听！

课件最新版本与补充资料将持续同步至课程主页。

**现在进入自由提问与交流时间。**