深度剖析对比学习法在 Java 语言教学中的应用模式

郑秀琴 王飞平

衢州职业技术学院，浙江衢州，324000；

摘要：在高职 Java 语言教学中，面对已掌握 Python 语言的学生，对比学习法可利用其知识基础、降低学习难度、提升教学效果。基于多年教学经验，从 Java 与 Python 核心特性对比出发，构建“知识迁移→概念辨析→实践强化”三位一体对比教学模式，对比 Java 语言核心模块，探讨其在各教学环节的应用路径，以教学案例验证有效性，为高职编程课教学改革提供可复制范式。

关键词：对比学习法；Java 教学；程序设计课程；知识迁移

DOI：10.69979/3041-0673.25.11.066

1 引言：对比学习法的教学价值与实践背景

1.1 高职 Java 教学的现实困境

高职学生在学习 Java 时常面临“双重挑战”：一方面，Java 语法的严谨性与 Python 的动态灵活性形成鲜明反差，导致学生因习惯 Python 的“宽松”语法而频繁出现编译错误；另一方面，高职学生抽象思维能力较弱，对面向对象编程、多线程等复杂概念理解困难，传统“孤立式”教学易引发学习挫败感。

1.2 对比学习法的理论依据

对比学习法基于认知心理学的“迁移理论”，通过建立新旧知识的关联点，帮助学生在辨析差异中构建新知识体系。对于已掌握 Python 的学生，Java 与 Python 在编程范式、控制结构、开发工具等方面存在大量可比点，具备天然的对比教学基础。

1.3 学生 Python 基础的双向影响

正迁移优势：Python 奠定的编程思维、数据结构认知可直接迁移至 Java 学习，降低入门门槛。

负迁移干扰：Python 的动态类型导致学生忽视 Java 静态类型的重要性；缩进式代码块与 Java 大括号语法易混淆，需针对性破除认知惯性。

2 核心语言特性对比：构建结构化知识图谱

2.1 语法规则：从“动态灵活”到“静态严谨”

2.1.1 变量与数据类型对比表

特性	Python	Java	教学对比策略
类型声明	动态推断，如：score = 90	显式声明，如：int score = 90;	类型错误对比：Python 运行时错误 vs Java 编译时错误，强调静态类型对程序健壮性的意义。
数据类型范围	自动扩容，如：int 无固定大小	固定范围，如：int 占 4 字节	用数轴图可视化 Java 基本类型取值范围 vs Python“无限精度”整数的局限性。
类型转换	隐式转换为主（str(10)）	显式强制转换（(int) 10.5）	类型转换错误对比：Python 自动截断 vs Java 编译报错，强化显式转换意识。

2.1.2 代码块结构对比

Python 依赖缩进，如：

```
if score < 60:
    print("不及格")
```

Java 依赖大括号，如：

```
if (score < 60) {
   System.out.println("未成年");}
```

教学策略：通过“缩进错误 vs 大括号缺失”的编译结果对比，感受 Java 语法的严谨性，配合 IDE 培养规范书写习惯。

2.2 面向对象编程：从“弱封装”到“强规范”

2.2.1 类定义与封装对比

Python 类的“弱封装”，如：

```
class Student:
    def __init__(self, name):
        self.name = name
        self._age = 0
```

Java 类的“强封装”，如：

```
publicclass Student {
    private String name;
    private int age;
    public Student(String name) {
        this.name = name;    }
```

```
    public void setAge(int age) {
        if (age > 0) this.age = age;    }
}
```

教学重点：通过“属性直接修改”对比实验，讲解封装的本质，强化“通过方法操作属性”的 Java 编程规范。

特性	Python（多继承）	Java（单继承+接口）	对比教学案例
继承语法	class Sub(Parent1, Parent2)	class Sub extends Parent implements Interface	设计“动物类”案例：Python 多继承实现 “会飞的鱼” vs Java 接口实现 “飞行动物”，解析多继承潜在的“钻石继承”问题。
多态机制	鸭子类型（动态绑定）	基于继承的静态多态（父类引用子类对象）	对比“调用不存在方法”的结果：Python 运行时报错 vs Java 编译期提示，强调 Java 编译时类型检查的优势。

2.3 异常处理：从“灵活捕获”到“分层控制”

2.3.1 异常处理流程对比

Python 统一捕获（推荐具体类型，允许通用捕获），如：

```
try:
    x = 1 / 0
except Exception as e:
    print(f"错误：{e}")
```

Java 强制分层（受检异常必须处理或声明抛出），如：

```
try {
    FileReader fr = new FileReader("test.txt");
} catch (FileNotFoundException e) {
    e.printStackTrace();}
```

教学突破点：通过“文件读取”案例对比，解释 Java 受检异常的设计目的，对比 Python 非受检异常的“事后处理”模式，培养学生“防御性编程”思维。

2.4 多线程编程：从“轻量化”到“系统化”

2.4.1 线程创建方式对比

Python 线程轻量化，如：

```
import threading
def task():
    print("Python 线程")
thread = threading.Thread(target=task)
thread.start()
```

Java 线程系统化，如：

```
public class JavaThread implements Runnable {
    public void run() {
        System.out.println("Java 线程");
    }
    public static void main(String[] args) {
        new Thread(new JavaThread()).start();    }
}
```

2.4.2 线程同步机制对比

Python 简单锁（Lock），如：

```
lock = threading.Lock()
with lock:
    shared_var += 1
```

Java 同步块（synchronized 关键字 + 显式锁 ReentrantLock）：

```
private static int sharedVar = 0;
private static final Object lock = new Object();
public static void increment() {
    synchronized (lock) {
        sharedVar++;  }
}
```

教学实践：通过“银行转账”并发场景，对比两种语言的线程安全实现，重点讲解 Java 内存可见性与线程协作机制，解析 Python 全局解释器锁对多线程性能的影响，凸显 Java 在高并发场景的优势。

3 对比学习法的教学应用模式构建

3.1 课程设计：分层递进的对比框架

3.1.1 三阶段对比教学模型

知识迁移阶段（第 1-2 周）：

建立“Python 已知→Java 未知”的映射关系，如用 Python 列表对比 Java 数组，Python 函数对比 Java 方法。

设计“语法转换”练习：将 Python 代码片段改写为 Java，重点关注类型声明、代码块结构差异。

组织“辩论赛”课堂活动：正方“Python 灵活好用” vs 反方“Java 严谨可靠”，通过观点碰撞深化概念理解。

实践强化阶段（第 14-16 周）：

设计“同功能双语言实现”项目，如学生信息管理系统：Python 版用字典存储数据，Java 版用类+数组/集合实现，对比两者在数据安全性、代码可维护性上的差异。

引入代码审查环节：要求学生交叉检查 Java 代码中的“Python 式惯性错误”，培养规范编程习惯。

3.2 课堂教学：差异化对比策略设计

3.2.1 语法对比：错误驱动式教学

设计“典型错误对比库”：收集学生常见错误，通过 IDE 报错信息对比，强化 Java 语法规则。

动态演示类型转换：用调试工具逐步执行 Python 和 Java 的类型转换代码，观察内存变化，理解静态类型语言的内存管理机制。

3.2.2 面向对象：场景化对比案例

封装性对比案例：模拟“学生成绩管理”场景，Python 版允许直接修改成绩，Java 版通过 setScore()方法校验成绩范围，通过具体场景演示封装的实际价值。

多态性对比案例：设计“图形绘制”程序，Python 版根据对象类型动态调用 draw()方法，Java 版通过抽象类 Shape 统一接口，子类重写 draw()方法，对比两种语言在扩展性上的差异。

3.3 实践环节：对比导向的项目设计

3.3.1 基础实验：语法差异验证

实验 1：变量作用域对比

Python 代码：

```
x = 10
def func():
    x = 20
    print(x)  # 局部变量
func()
print(x)  # 全局变量
```

Java 代码：

```
public class ScopeTest {
    static int x = 10;
    public static void func() {
        int x = 20;
        System.out.println(x);  //局部变量
    }
    public static void main(String[] args) {
        func();
        System.out.println(x);  //全局变量
    }
}
```

实验目标：通过相同逻辑的代码对比，理解两种语言在变量作用域解析上的一致性，消除“Java 作用域更复杂”的认知误区。

3.3.2 综合项目：同功能双语言实现

课程设计任务：学生信息管理系统

Python 版：使用字典存储学生信息，函数式编程实现增删改查，重点体现快速开发优势。

Java 版：使用 Student 类封装学生属性，ArrayList 存储数据，通过接口 StudentService 定义操作规范，实现面向对象的分层设计。

对比点：

数据安全性：Java 版通过 private 属性和校验方法防止非法数据，Python 版依赖开发者自觉。

代码扩展性：Java 版新增“继续教育学院学生”只需继承 Student 类，Python 版需修改多个函数，对比 OOP 对大型项目的支撑作用。

概念辨析阶段（第 3-13 周）：

针对核心难点，采用表格可视化差异。

特性	Python 实现	Java 实现	核心差异
封装	弱私有（_前缀）	强私有（private 修饰符）	编译期强制数据保护
继承	多继承	单继承 + 接口	避免菱形继承问题
多态	鸭子类型	父类引用子类对象	编译时类型检查

4 教学实施效果与反思

4.1 实证分析

两届同专业学生，分别采用传统教学法与对比教学法，通过对比，发现学生在知识掌握度、实践能力和学

习态度方面，均有显著的提升。

4.2 挑战与对策

1.负迁移强化问题：部分学生过度依赖 Python 经验，忽视 Java 特有规范。

对策：设计“负迁移专项训练”，针对高频错误编写对比练习题。

2.对比深度把控问题：过度对比可能导致学生混淆概念。

对策：建立“对比边界清单”，明确核心对比点，对非核心差异仅作简要说明，避免信息过载。

5 结论与展望

对比学习法在高职 Java 教学中构建了“以旧知引新知，以差异促理解”的有效路径，通过系统化的语言特性对比、分层递进的教学活动设计，显著提升了学生对 Java 严谨语法和面向对象编程的掌握程度，同时培养了跨语言编程思维。未来可进一步探索“对比学习法+可视化工具”的融合应用，以及基于虚拟仿真技术的跨语言编程环境开发，为高职学生提供更立体的对比学习体验。

参考文献

[1]李晶晶;王超;薛思敏.对比式教学法在程序设计类课程教学中的应用[J].计算机教育.2025(01):184-187.

[2]张伟.Java Web 应用开发中对比学习法的实践研究[J].职业教育研究.2024(24):85-88.

[3]苏绚;蔡缇萦.提升元认知的思维训练编程课程框架建构研究[J].佳木斯职业学院学报.2024,40(10):214-216.

[4]申雪萍,原仓周,邵兵.面向软件实践能力的Java程序设计课程教学改革探索[J].计算机教育.2024,(02):20-25.

[5]白雪.智慧教育背景下程序设计类课程混合式教学方法改革研究[J].信息系统工程.2024(01):169-172.

[6]易锋,马慧,何怀文,翁佩纯.对比教学法在程序设计基础课程中的应用实践研究[J].电脑知识与技术.2022(20):155-157.

作者简介：郑秀琴，（1968-），副教授、高级工程师，研究方向为高职编程课程教学改革、计算机网络与安全技术。