

Java 语言实验指导书 1

二、实验内容与步骤

第一部分：基本数据类型与变量程序设计要求

声明 Java 所有的基本数据类型变量并初始化。

尝试进行类型转换（包括自动转换和强制转换），理解其规则和潜在风险。

```
public class DataTypeDemo {  
    public static void main(String[] args) {  
        // 1. 声明并初始化各基本数据类型变量（byte, short, int, long, float, double, char, boolean）  
        // 【请学生填写代码】  
        byte byteVar = 100;  
        short shortVar = 200;  
        int intVar = 1000;  
        long longVar = 100000L; // 注意 L 后缀  
        float floatVar = 3.14F; // 注意 F 后缀  
        double doubleVar = 3.1415926;  
        char charVar = 'A';  
        boolean booleanVar = true;  
  
        // 2. 输出各变量的值  
        // 【请学生填写代码】使用 System.out.println 输出每个变量  
  
        // 3. 类型转换实践  
        // 3.1 自动类型转换（隐式转换）：将一个小范围类型的值赋给一个大范围类型的变量  
        int autoConverted = byteVar; // 将 byte 赋值给 int  
        System.out.println("autoConverted: " + autoConverted);  
        // 【请学生填写代码】尝试将 int 赋值给 long，将 float 赋值给 double  
  
        // 3.2 强制类型转换（显式转换）：将一个大范围类型的值赋给一个小范围类型的变量  
        byte forcedConverted = (byte) intVar; // 将 int 强制转换为 byte  
        System.out.println("forcedConverted: " + forcedConverted);  
        // 【请学生填写代码】尝试将 long 强制转换为 int，将 double 强制转换为 float  
        // 4. 思考：如果将一个较大的 long 值（例如：long largeLong = 10000000000L;）强制转换为 short，结果会怎样？为什么？  
        long largeLong = 10000000000L;  
        short convertedShort = (short) largeLong;  
        System.out.println("convertedShort: " + convertedShort);  
    }  
}
```

观察各种类型转换的输出结果。强制转换 largeLong 为 short 后，输出值是你预期的吗？为什么？

结果会发生数据溢出（截断），得到的结果通常是一个与原数值完全不同的值（可能是负数或很小的正数）。

原因：long 类型占用 64 位，而 short 类型仅占用 16 位。当进行强制类型转换 (short) largeLong 时，Java 会直接截断高位字节，只保留最低的 16 位。由于 10000000000L 的二进制数值远超 16 位所能表示的范围 (-32768 到 32767)，截断后的二进制码被重新解释为 short 值，从而导致数据丢失和错误。

什么是自动类型转换？在什么情况下会发生？指将一个“小范围类型”的值赋给一个“大范围类型”的变量（例如 byte 赋值给 int），Java 编译器会自动处理，不会丢失精度。

什么是强制类型转换？它可能会带来什么风险（如数据溢出、精度丢失）？如何避免？

指将一个“大范围类型”的值赋给一个“小范围类型”的变量（例如 int 赋值给 byte），必须在代码中显式使用（类型）进行转换。可能会导致数据溢出（数值改变）或精度丢失（例如 double 转 int 会丢失小数部分）。在转换前先判断数值是否在目标类型的取值范围内。

float 和 double 变量在声明和初始化时，为什么要加 F 和 L 后缀？

在 Java 中，整数面值量默认是 int 类型，浮点数面值量默认是 double 类型。

L 后缀：当定义的 long 类型数值超过 int 的范围时，必须加 L 告知编译器这是一个 long 值，否则会报错。

F 后缀：既然浮点数默认为 double，如果直接将 3.14 赋给 float 变量，属于从大范围（double）到小范围（float）的转换，编译器会报错。因此必须加 F 显式声明这是个 float 类型。

第二部分：运算符与表达式

程序设计要求

使用算术运算符完成基本数学运算。

使用关系运算符比较两个数值，并输出布尔结果。

使用逻辑运算符组合多个布尔条件。

```
import java.util.Scanner;
public class OperatorDemo {
    public static void main(String[] args) {
        // 1. 算术运算符: +、-、*、/、%
        int a = 10;
        int b = 3;
        int sum = a + b;
        int difference = a - b;
        int product = a * b;
        // 【请学生填写代码】计算 a/b 和 a%b
        System.out.println("a / b = " + (a / b)); // 整数除法
        System.out.println("a % b = " + (a % b)); // 取模

        // 2. 关系运算符: >、<、>=、<=、==、!=
        // 【请学生填写代码】比较 a 和 b 的大小，输出 a > b, a == b, a != b 的结果
        System.out.println("a == b: " + (a == b));
        System.out.println("a != b: " + (a != b));

        // 3. 逻辑运算符: &&、||、!
        boolean x = true;
        boolean y = false;
        // 计算 x && y, x || y, !x 的值并输出
        System.out.println("x && y: " + (x && y));
        System.out.println("x || y: " + (x || y));
        System.out.println("!x: " + (!x));

        // 4. 综合应用：从键盘输入一个整数，判断其是否为偶数
        Scanner scanner = new Scanner(System.in);
        System.out.print("请输入一个整数: ");
        int num = scanner.nextInt();
        // 使用关系运算符和逻辑运算符判断 num 是否为偶数
        boolean isEven = (num % 2 == 0);
        System.out.println(num + " 是偶数吗? " + isEven);
        scanner.close(); }
    }
```

结果与知识点提问

10 / 3 的结果是 3 而不是 3.333...，为什么？如何得到更精确的结果？

原因：在 Java 中，两个整数 (int) 进行运算，结果依然是整数。10 和 3 都是整数，因此执行的是整数除法，小数部分会被直接舍弃（截断）。

解决方法将其中至少一个操作数转换为浮点类型。例如使用 10.0 / 3 或 (double)10 / 3。

关系运算符和逻辑运算符的运算结果是什么类型？

结果都是布尔类型：boolean (即 true 或 false)。

表达式 (num % 2 == 0) 是如何判断一个数是否为偶数的？

% 是取模（求余）运算符。num % 2 计算 num 除以 2 后的余数。如果余数为 0，说明该数能被 2 整除，即为偶数；如果余数为 1（对于正整数），则为奇数。

第三部分：程序流程控制

程序设计要求

使用 if-else 语句判断一个数的正负和奇偶性。使用 switch

语句根据数字输出对应的星期几。

使用 **for** 循环计算 1 到 100 的整数和。

使用 **while** 循环求 100 以内所有偶数的和。

使用 **do-while** 循环模拟简单的用户输入验证。 import

```
java.util.Scanner;
```

```
public class FlowControlDemo {
```

```
    public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);
```

```
// 1. if-else 语句
```

```
        System.out.print("请输入一个整数: ");
```

```
        int number = scanner.nextInt();
```

```
// 【请学生填写代码】判断 number 的正负
```

```
        if (number > 0) {
```

```
            System.out.println(number + " 是正数。"); } else if (number < 0) {
```

```
            System.out.println(number + " 是负数。"); } else {
```

```
            System.out.println(number + " 是零。"); }
```

```
// 【请学生填写代码】判断 number 的奇偶性
```

```
// 2. switch 语句
```

```
        System.out.print("请输入一个 1-7 之间的数字: ");
```

```
        int day = scanner.nextInt();
```

```
// 【请学生填写代码】使用 switch 根据 day 输出对应的星期几switch (day) {
```

```
    case 1:
```

```
        System.out.println("星期一");
```

```
        break;
```

```
    case 2:
```

```
        System.out.println("星期二");
```

```
        break;
```

```
// ... 补充其他情况default:
```

```
        System.out.println("输入错误！"); }
```

```
// 3. for 循环：计算 1 到 100 的整数和int sumFor = 0;
```

```
// 【请学生填写代码】使用for 循环计算 sumFor for (int i = 1; i <= 100; i++) {
```

```
    sumFor += i; }
```

```
System.out.println("1 到 100 的整数和 (for 循环): " + sumFor);
```

```
// 4. while 循环：求 100 以内所有偶数的和
```

```
int sumWhile = 0;
```

```
int j = 1;
```

```
// 【请学生填写代码】使用 while 循环计算 sumWhile while (j <= 100) {
```

```
    if (j % 2 == 0) {
```

```
        sumWhile += j;
```

```
    }
```

```
    j++; }
```

```
System.out.println("100 以内所有偶数的和 (while 循环): " + sumWhile);
```

```
// 5. do-while 循环：模拟输入验证，直到输入正确的密码为止
```

```
String correctPassword = "123456";
```

```
String inputPassword;
```

```
do {
```

```
    System.out.print("请输入密码: ");
```

```
    inputPassword = scanner.next();
```

```
    if (!inputPassword.equals(correctPassword)) {
```

```
        System.out.println("密码错误，请重新输入！");
```

```
    }
```

```
} while (!inputPassword.equals(correctPassword));
```

```
System.out.println("密码正确，登录成功！");
```

```
scanner.close(); }
```

}

结果与知识点提问

if-else 语句和 switch 语句在处理多分支选择时各有何特点？如何选择？

if-else：适合判断区间范围（如 `score > 60`）或复杂的布尔条件表达式。

switch：结构更清晰，适合判断具体的离散值（如特定的整数、字符、枚举或 String）

选择：如果是基于某个具体变量的等值匹配（如星期几、菜单选项），优先用 switch；如果是范围判断或多条件组合，使用 if-else。

for、while 和 do-while 三种循环语句的执行流程有何异同？do-while 循环至少会执行几次循环体？

for：适合循环次数已知的情况，结构紧凑（初始化、条件、迭代在一起）。

while：适合循环次数未知的情况，先判断条件，再执行循环体。

do-while：先执行一次循环体，再判断条件。

至少执行次数：do-while 循环至少执行 1 次 循环体。

在 switch 语句中，如果忘记写 break 关键字，会发生什么现象？这叫什么现象？

现象：程序会继续执行下一个 case 的代码，而不管下一个 case 的条件是否满足，直到遇到 break 或 switch 结束。

名称：这种现象被称为“Case 穿透”（Fall-through）。

循环语句中的循环变量（如 for 中的 i, while 中的 j）的作用是什么？如何避免死循环？

作用：用于控制循环的执行次数或终止条件。

避免死循环：必须确保在循环体内部更新循环变量（例如 `i++` 或 `j++`），使其最终能够导致循环条件变为 false。

第四部分：数组的应用

程序设计要求

声明、初始化一个整型数组。

使用循环遍历数组并输出每个元素。

找出数组中的最大值和最小值。

对数组元素进行升序排序（可使用冒泡排序）。

在排序后的数组中，使用二分查找算法查找指定元素。

```
import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;

public class ArrayDemo {
    public static void main(String[] args) {
        // 1. 数组的声明、动态初始化和赋值
        int[] array = new int[10]; // 声明一个长度为 10 的整型数组
        Random random = new Random();
        // 【请学生填写代码】使用 Random 类生成随机数，为数组的每个元素赋值（范围 1-100）
        for (int i = 0; i < array.length; i++) {
            array[i] = random.nextInt(100) + 1; // 生成 1-100 的随机数
        }
        // 2. 遍历数组并输出
        System.out.println("原始数组:");
        // 【请学生填写代码】使用 for 循环遍历并打印数组 for (int i = 0; i < array.length; i++) {
            System.out.print(array[i] + " ");
        }
        System.out.println(); // 换行
        // 3. 求数组的最大值和最小值
        int max = array[0];
        int min = array[0];
        // 【请学生填写代码】遍历数组，找出 max 和 min for (int i = 1; i < array.length; i++) {
            if (array[i] > max) {
                max = array[i];
            }
            if (array[i] < min) {
                min = array[i];
            }
        }
    }
}
```

```

System.out.println("最大值: " + max);
System.out.println("最小值: " + min);
// 4. 数组排序（冒泡排序）实现冒泡排序算法对数组进行升序排序for (int i = 0; i < array.length - 1; i++) {
    for (int j = 0; j < array.length - 1 - i; j++) {
        if (array[j] > array[j + 1]) {
            // 交换元素
            int temp = array[j];
            array[j] = array[j + 1];
            array[j + 1] = temp;
        }
    }
}
System.out.println("排序后的数组:");
System.out.println(Arrays.toString(array)); // 使用 Arrays.toString 快速输出

// 5. 二分查找（必须在有序数组上进行）
Scanner scanner = new Scanner(System.in);
System.out.print("请输入要在数组中查找的数字: ");
int target = scanner.nextInt();
int low = 0;
int high = array.length - 1;
int index = -1; // 初始化为-1， 表示未找到
// 实现二分查找算法
while (low <= high) {
    int mid = (low + high) / 2;
    if (array[mid] == target) {
        index = mid;
        break;
    } else if (array[mid] < target) {
        low = mid + 1;
    } else {
        high = mid - 1;
    }
}
if (index != -1) {
    System.out.println("找到元素 " + target + ", 其在数组中的索引为: " + index); } else {
    System.out.println("数组中未找到元素 " + target);
}
scanner.close();
}

```

结果与知识点提问

数组的索引（下标）是从什么数字开始的？访问 `array[array.length]` 会发生什么？冒泡排序算法的核心思想是什么？

起始索引：从 0 开始。

访问结果：会抛出 `ArrayIndexOutOfBoundsException`（数组下标越界异常）。因为长度为 N 的数组，最大合法索引是 `N-1`, `array.length` 已经超出了范围。

核心思想：通过重复遍历要排序的数组，依次比较相邻的两个元素。如果它们的顺序错误（如前大后小），就交换它们。这样每一轮遍历后，最大的元素就会像“气泡”一样“浮”到数组的末端。

二分查找算法的前提条件是什么？为什么？相比简单的顺序查找有何优势？

前提：数组必须是有序的（升序或降序）³⁴。原因：二分查找利用了数据的有序性，通过比较中间值来排除一半的搜索区间。如果是无序的，无法确定目标值在左边还是右边。优势：效率极高。查找次数呈对数级别 $\$O(\log n)\$$ ，而顺序查找是线性级别 $\$O(n)\$$ 。

Java 中数组的长度是固定的吗？如何获取数组的长度？

是的，数组一旦初始化，长度就固定不可变。通过数组的 `.length` 属性获取（注意不是方法，没有括号）

程序设计题 1：校园建筑用地规划

福建师范大学旗山校区存在多种类型的建筑用地。现要求根据分配的地理目标，采用面向对象的思想进行设计。

基类为 **Construction**（建筑用地），它具有所有建筑用地共有的属性和行为。

其下有多个具体子类，分别代表不同类型的用地：**Ground**（广场）、**TeachingBuilding**（教学楼）、**AcademicBuilding**（学院楼）和 **Canteen**（食堂）。

每种类型的建筑用地都有其独特的功能和计算标准。你需要完成以下任务：

1. **设计类结构**：设计一个抽象基类 **Construction** 和它的具体子类。
2. **实现方法**：在基类中定义抽象方法，并在子类中提供具体实现，展示多态性。
3. **创建对象并测试**：在 **main** 方法中创建不同子类的对象，统一管理并调用它们的方法。

具体要求如下：

1. **抽象类 (abstract class)**：
 - **Construction** 应设计为抽象类。
 - 它应包含所有用地共有的属性（如 **name**, **location**, **area**, **width**, **length**）。
 - 它应包含一个实现的坐标转换方法 **gcj02ToWGS84**, 高德坐标 GCJ02 见 excel, 代码见题末。
 - 它应包含一个抽象方法 **void displayInfo()**，用于显示用地详细信息，由子类实现。
2. **继承 (extends)**：
 - **Ground**, **TeachingBuilding**, **AcademicBuilding**, **Canteen** 都应继承自 **Construction** 类，且有自己特有的变量。
3. **构造方法 (constructor)**：
 - 父类和子类都应提供构造方法来初始化属性。
4. **方法重写 (@Override)**：
 - 每个子类必须重写父类的 **displayInfo()**方法，提供自身的具体实现。
 - 每个子类可以拥有自己特有的方法（如 **Canteen** 可以有一个 **serveFood()**方法）。
5. **多态**：
 - 在测试类中，存储不同子类的对象，如教室按座位分为不同类型。
 - 通过循环遍历数组，调用每个对象的 **displayInfo()**方法，演示运行时多态。

```
import java.util.ArrayList;
import java.util.List;
// 1. 抽象基类 Construction
abstract class Construction {
    protected String name;
    protected double area;
    protected double width;
    protected double length;
    // 原始高德坐标 (GCJ02)
    protected double gcjLat;
    protected double gcjLon;
    // 坐标转换常量
    private static final double PI = Math.PI;
    private static final double A = 6378245.0;
    private static final double EE = 0.00669342162296594323;
    public Construction(String name, double length, double width, double gcjLat, double gcjLon) {
        this.name = name;
        this.length = length;
        this.width = width;
        this.area = length * width;
        this.gcjLat = gcjLat;
        this.gcjLon = gcjLon;
    }
}
```

```

}

// 抽象方法: 显示详细信息
public abstract void displayInfo();

// 具体的坐标转换方法: GCJ02 -> WGS84
protected LatLng gcj02ToWGS84(double lat, double lon) {
    double dLat = transformLat(lon - 105.0, lat - 35.0);
    double dLon = transformLon(lon - 105.0, lat - 35.0);
    double radLat = lat / 180.0 * PI;
    double magic = Math.sin(radLat);
    magic = 1 - EE * magic * magic;
    double sqrtMagic = Math.sqrt(magic);

    dLat = (dLat * 180.0) / ((A * (1 - EE)) / (magic * sqrtMagic) * PI);
    dLon = (dLon * 180.0) / (A / sqrtMagic * Math.cos(radLat) * PI);

    double mgLat = lat + dLat;
    double mgLon = lon + dLon;

    return new LatLng(lat * 2 - mgLat, lon * 2 - mgLon);
}

private double transformLat(double x, double y) {
    double ret = -100.0 + 2.0 * x + 3.0 * y + 0.2 * y * y + 0.1 * x * y + 0.2 * Math.sqrt(Math.abs(x));
    ret += (20.0 * Math.sin(6.0 * x * PI) + 20.0 * Math.sin(2.0 * x * PI)) * 2.0 / 3.0;
    ret += (20.0 * Math.sin(y * PI) + 40.0 * Math.sin(y / 3.0 * PI)) * 2.0 / 3.0;
    ret += (160.0 * Math.sin(y / 12.0 * PI) + 320 * Math.sin(y * PI / 30.0)) * 2.0 / 3.0;
    return ret;
}

private double transformLon(double x, double y) {
    double ret = 300.0 + x + 2.0 * y + 0.1 * x * x + 0.1 * x * y + 0.1 * Math.sqrt(Math.abs(x));
    ret += (20.0 * Math.sin(6.0 * x * PI) + 20.0 * Math.sin(2.0 * x * PI)) * 2.0 / 3.0;
    ret += (20.0 * Math.sin(x * PI) + 40.0 * Math.sin(x / 3.0 * PI)) * 2.0 / 3.0;
    ret += (150.0 * Math.sin(x / 12.0 * PI) + 300.0 * Math.sin(x / 30.0 * PI)) * 2.0 / 3.0;
    return ret;
}

// 内部类: 坐标对象
public static class LatLng {
    private final double latitude;
    private final double longitude;

    public LatLng(double latitude, double longitude) {
        this.latitude = latitude;
        this.longitude = longitude;
    }

    @Override
    public String toString() {
        return String.format("latitude=% .6f, longitude=% .6f", latitude, longitude);
    }
}

// 2. 子类实现
class Ground extends Construction {
}

```

```
public Ground(String name, double length, double width, double lat, double lon) {
    super(name, length, width, lat, lon);
}

@Override
public void displayInfo() {
    LatLng wgs84 = gcj02ToWGS84(this.gcjLat, this.gcjLon);
    System.out.println("[广场] " + name + " | 面积: " + area + "平米");
    System.out.println("    高德坐标: " + gcjLat + ", " + gcjLon);
    System.out.println("    GPS 坐标: " + wgs84);
}
}

class TeachingBuilding extends Construction {
    private int floorCount; // 特有属性: 楼层数

    public TeachingBuilding(String name, double length, double width, double lat, double lon, int floorCount) {
        super(name, length, width, lat, lon);
        this.floorCount = floorCount;
    }

    @Override
    public void displayInfo() {
        LatLng wgs84 = gcj02ToWGS84(this.gcjLat, this.gcjLon);
        System.out.println("[教学楼] " + name + " | 楼层: " + floorCount + " | 总使用面积: " + (area * floorCount));
        System.out.println("    GPS 坐标: " + wgs84);
    }
}

class AcademicBuilding extends Construction {
    private String department; // 特有属性: 所属学院

    public AcademicBuilding(String name, double length, double width, double lat, double lon, String department) {
        super(name, length, width, lat, lon);
        this.department = department;
    }

    @Override
    public void displayInfo() {
        LatLng wgs84 = gcj02ToWGS84(this.gcjLat, this.gcjLon);
        System.out.println("[学院楼] " + name + " (" + department + ")");
        System.out.println("    GPS 坐标: " + wgs84);
    }
}

class Canteen extends Construction {
    public Canteen(String name, double length, double width, double lat, double lon) {
        super(name, length, width, lat, lon);
    }

    public void serveFood() {
        System.out.println("    -> " + name + " 正在供应午餐...");
    }

    @Override
    public void displayInfo() {
        LatLng wgs84 = gcj02ToWGS84(this.gcjLat, this.gcjLon);
        System.out.println("[食堂] " + name);
    }
}
```

```

        System.out.println("      GPS 坐标: " + wgs84);
        serveFood();
    }
}

// 3. 测试类 (包含 main 方法)
public class CampusPlanning {
    public static void main(String[] args) {
        List<Construction> campus = new ArrayList<>();

        // 创建对象 (使用示例坐标, 例如福师大附近)
        campus.add(new Ground("中心广场", 100, 100, 26.0231, 119.1831));
        campus.add(new TeachingBuilding("知明楼", 60, 20, 26.0245, 119.1845, 6));
        campus.add(new AcademicBuilding("计算机学院楼", 50, 25, 26.0255, 119.1855, "计信学院"));
        campus.add(new Canteen("桃苑餐厅", 40, 30, 26.0225, 119.1825));

        System.out.println("== 校园建筑用地规划信息 ===");
        // 多态遍历
        for (Construction c : campus) {
            c.displayInfo();
            System.out.println("-----");
        }
    }
}

```

程序设计题 2：邮件地址合法检测

一个合法规范的邮件地址需要符合特定的格式标准，下表列出了邮件地址需遵循的核心规范和示例。

规范要素	具体要求	正确示例	错误示例
基本结构	必须包含 本地部分 、 @符号 、 域部分 ，顺序固定且缺一不可	username@example.com	username.example.com(缺少@)
本地部分(@前)	可包含：字母(a-z, A-Z)、数字(0-9)、点(.)、下划线(_)、连字符(-)。 不能以点(.)或连字符(-)开头或结尾。 不能有两个或多个连续的点(..)。	user.name user_name user-name	.user@ user..name@ user-@

	长度一般不超过 64 个字符。		
@ 符号	必须有且只能有一个，且不能紧邻空格。	u@example.com	u@@example.com u@example.com
域部分	需符合域名规范，通常包含点(.)分隔的多级标签。	example.com sub.example.org	example(无点分隔) -
(@后)	标签可由字母、数字、连字符(-)组成，不能以连字符开头或结尾。 顶级域名（如.com、.cn）需有效存在。 长度不超过 253 个字符（通常） 。		example. com example.- com
总长度	整个邮箱地址总长通常不超过 254 个字符 。		
大小写	邮箱地址不区分大小写，但建议统一使用小写以避免不必要的麻烦。（可提示或忽略）	user@example.com	USER@EXAMPLE.COM(虽等效，但不建议全部大写)

试设计一个邮件规范验证工具类 `EmailValidator` 来验证邮件地址是否合法，继承 `Exception` 类，如用户输入的邮件地址合法，输出相应提示；如用户输入的邮件地址不合法，输出具体的不合法问题提示。

```

import java.util.ArrayList;
import java.util.List;

// 1. 抽象基类 Construction
abstract class Construction {
    protected String name;
    protected double area;
    protected double width;
    protected double length;
    // 原始高德坐标 (GCJ02)
    protected double gcjLat;
    protected double gcjLon;

    // 坐标转换常量
    private static final double PI = Math.PI;
    private static final double A = 6378245.0;
    private static final double EE = 0.00669342162296594323;

    public Construction(String name, double length, double width, double gcjLat, double gcjLon) {
        this.name = name;
        this.length = length;
        this.width = width;
        this.area = length * width;
        this.gcjLat = gcjLat;
        this.gcjLon = gcjLon;
    }

    // 抽象方法：显示详细信息
    public abstract void displayInfo();
}

```

```

// 具体的坐标转换方法: GCJ02 -> WGS84
protected LatLng gcj02ToWGS84(double lat, double lon) {
    double dLat = transformLat(lon - 105.0, lat - 35.0);
    double dLon = transformLon(lon - 105.0, lat - 35.0);
    double radLat = lat / 180.0 * PI;
    double magic = Math.sin(radLat);
    magic = 1 - EE * magic * magic;
    double sqrtMagic = Math.sqrt(magic);

    dLat = (dLat * 180.0) / ((A * (1 - EE)) / (magic * sqrtMagic) * PI);
    dLon = (dLon * 180.0) / (A / sqrtMagic * Math.cos(radLat) * PI);

    double mgLat = lat + dLat;
    double mgLon = lon + dLon;

    return new LatLng(lat * 2 - mgLat, lon * 2 - mgLon);
}

private double transformLat(double x, double y) {
    double ret = -100.0 + 2.0 * x + 3.0 * y + 0.2 * y * y + 0.1 * x * y + 0.2 * Math.sqrt(Math.abs(x));
    ret += (20.0 * Math.sin(6.0 * x * PI) + 20.0 * Math.sin(2.0 * x * PI)) * 2.0 / 3.0;
    ret += (20.0 * Math.sin(y * PI) + 40.0 * Math.sin(y / 3.0 * PI)) * 2.0 / 3.0;
    ret += (160.0 * Math.sin(y / 12.0 * PI) + 320 * Math.sin(y * PI / 30.0)) * 2.0 / 3.0;
    return ret;
}

private double transformLon(double x, double y) {
    double ret = 300.0 + x + 2.0 * y + 0.1 * x * x + 0.1 * x * y + 0.1 * Math.sqrt(Math.abs(x));
    ret += (20.0 * Math.sin(6.0 * x * PI) + 20.0 * Math.sin(2.0 * x * PI)) * 2.0 / 3.0;
    ret += (20.0 * Math.sin(x * PI) + 40.0 * Math.sin(x / 3.0 * PI)) * 2.0 / 3.0;
    ret += (150.0 * Math.sin(x / 12.0 * PI) + 300.0 * Math.sin(x / 30.0 * PI)) * 2.0 / 3.0;
    return ret;
}

// 内部类: 坐标对象
public static class LatLng {
    private final double latitude;
    private final double longitude;

    public LatLng(double latitude, double longitude) {
        this.latitude = latitude;
        this.longitude = longitude;
    }

    @Override
    public String toString() {
        return String.format("latitude=%.6f, longitude=%.6f", latitude, longitude);
    }
}
}

// 2. 子类实现
class Ground extends Construction {
    public Ground(String name, double length, double width, double lat, double lon) {
        super(name, length, width, lat, lon);
    }
}

```

```

@Override
public void displayInfo() {
    LatLng wgs84 = gcj02ToWGS84(this.gcjLat, this.gcjLon);
    System.out.println("[广场] " + name + " | 面积: " + area + "平米");
    System.out.println("    高德坐标: " + gcjLat + ", " + gcjLon);
    System.out.println("    GPS 坐标: " + wgs84);
}
}

class TeachingBuilding extends Construction {
    private int floorCount; // 特有属性: 楼层数

    public TeachingBuilding(String name, double length, double width, double lat, double lon, int floorCount) {
        super(name, length, width, lat, lon);
        this.floorCount = floorCount;
    }

    @Override
    public void displayInfo() {
        LatLng wgs84 = gcj02ToWGS84(this.gcjLat, this.gcjLon);
        System.out.println("[教学楼] " + name + " | 楼层: " + floorCount + " | 总使用面积: " + (area * floorCount));
        System.out.println("    GPS 坐标: " + wgs84);
    }
}

class AcademicBuilding extends Construction {
    private String department; // 特有属性: 所属学院

    public AcademicBuilding(String name, double length, double width, double lat, double lon, String department) {
        super(name, length, width, lat, lon);
        this.department = department;
    }

    @Override
    public void displayInfo() {
        LatLng wgs84 = gcj02ToWGS84(this.gcjLat, this.gcjLon);
        System.out.println("[学院楼] " + name + " (" + department + ")");
        System.out.println("    GPS 坐标: " + wgs84);
    }
}

class Canteen extends Construction {
    public Canteen(String name, double length, double width, double lat, double lon) {
        super(name, length, width, lat, lon);
    }

    public void serveFood() {
        System.out.println("    -> " + name + " 正在供应午餐...");
    }

    @Override
    public void displayInfo() {
        LatLng wgs84 = gcj02ToWGS84(this.gcjLat, this.gcjLon);
        System.out.println("[食堂] " + name);
        System.out.println("    GPS 坐标: " + wgs84);
        serveFood();
    }
}

```

```
// 3. 测试类 (包含 main 方法)
public class CampusPlanning {
    public static void main(String[] args) {
        List<Construction> campus = new ArrayList<>();
        // 创建对象 (使用示例坐标, 例如福师大附近)
        campus.add(new Ground("中心广场", 100, 100, 26.0231, 119.1831));
        campus.add(new TeachingBuilding("知明楼", 60, 20, 26.0245, 119.1845, 6));
        campus.add(new AcademicBuilding("计算机学院楼", 50, 25, 26.0255, 119.1855, "计信学院"));
        campus.add(new Canteen("桃苑餐厅", 40, 30, 26.0225, 119.1825));

        System.out.println("== 校园建筑用地规划信息 ==");
        // 多态遍历
        for (Construction c : campus) {
            c.displayInfo();
            System.out.println("-----");
        }
    }
}
```

```

### ### 程序设计题 2：邮件地址合法检测（异常处理与字符串操作）

按照题目要求，验证器通过抛出自定义异常（继承 `Exception`）来反馈错误信息。

\*\*文件：`EmailCheckDemo.java`\*\*

```
```java
import java.util.Scanner;

// 自定义异常类：用于捕获邮件格式错误
class EmailValidationException extends Exception {
    public EmailValidationException(String message) {
        super(message);
    }
}

// 邮件验证工具类
class EmailValidator {

    // 验证方法，如果非法则抛出异常
    public static void validate(String email) throws EmailValidationException {
        if (email == null || email.trim().isEmpty()) {
            throw new EmailValidationException("邮箱不能为空");
        }

        // 1. 检查 @ 符号
        int atIndex = email.indexOf('@');
        if (atIndex == -1) {
            throw new EmailValidationException("缺少 '@' 符号");
        }

        if (atIndex != email.lastIndexOf('@')) {
            throw new EmailValidationException("存在多个 '@' 符号");
        }
    }
}
```

```

    }

    if (email.contains(" ")) {
        throw new EmailValidationException("邮箱不能包含空格");
    }

    // 分割本地部分和域部分
    String localPart = email.substring(0, atIndex);
    String domainPart = email.substring(atIndex + 1);

    // 2. 验证本地部分
    validateLocalPart(localPart);

    // 3. 验证域部分
    validateDomainPart(domainPart);

    // 4. 长度检查 (简单检查总长度)
    if (email.length() > 254) {
        throw new EmailValidationException("邮箱总长度超过254个字符");
    }
}

private static void validateLocalPart(String local) throws EmailValidationException {
    if (local.isEmpty()) {
        throw new EmailValidationException("本地部分 (@前) 不能为空");
    }

    if (local.startsWith(".")) || local.endsWith(".")) {
        throw new EmailValidationException("本地部分不能以点(.)开头或结尾");
    }

    if (local.startsWith("-") || local.endsWith("-")) {
        throw new EmailValidationException("本地部分不能以连字符(-)开头或结尾");
    }

    if (local.contains("..")) {
        throw new EmailValidationException("本地部分不能包含连续的点(..)");
    }

    // 检查非法字符
    for (char c : local.toCharArray()) {
        if (!Character.isLetterOrDigit(c) && c != '.' && c != '_' && c != '-') {
            throw new EmailValidationException("本地部分包含非法字符: " + c);
        }
    }
}

private static void validateDomainPart(String domain) throws EmailValidationException {
    if (domain.isEmpty()) {
        throw new EmailValidationException("域部分 (@后) 不能为空");
    }

    if (!domain.contains(".")) {
        throw new EmailValidationException("域部分必须包含点(.)分隔的标签"); // example(无点)
    }

    if (domain.startsWith("-") || domain.endsWith("-")) {
        throw new EmailValidationException("域部分不能以连字符(-)开头或结尾");
    }

    // 简单的域名字符检查
    for (char c : domain.toCharArray()) {

```

```

        if (!Character.isLetterOrDigit(c) && c != '.' && c != '-') {
            throw new EmailValidationException("域部分包含非法字符: " + c);
        }
    }
}

// 主测试类
public class EmailCheckDemo {
    public static void main(String[] args) {
        // 测试用例数组
        String[] testEmails = {
            "username@example.com",      // 合法
            "user.name@example.com",     // 合法
            "user..name@example.com",    // 非法: 连续点
            ".user@example.com",         // 非法: 点开头
            "user@example",              // 非法: 域无点
            "user@@example.com",         // 非法: 两个@
            "user name@example.com"     // 非法: 包含空格
        };

        System.out.println("== 邮件地址合法性检测 ==");

        for (String email : testEmails) {
            System.out.print("检测: " + String.format("%-25s", email) + " -> ");
            try {
                EmailValidator.validate(email);
                System.out.println("✓ 合法");
            } catch (EmailValidationException e) {
                System.out.println("✗ 非法: " + e.getMessage());
            }
        }

        // 也可以开启手动输入测试
        /*
        Scanner scanner = new Scanner(System.in);
        System.out.print("\n请输入要检测的邮箱: ");
        String input = scanner.nextLine();
        try {
            EmailValidator.validate(input);
            System.out.println("结果: 输入合法! ");
        } catch (EmailValidationException e) {
            System.out.println("结果: 输入非法 (" + e.getMessage() + ")");
        }
        scanner.close();
        */
    }
}

```