Java语言 课程AI总结

第1周（课件）

课程概况与开发环境

- **课程安排与考核**
  - **内容规划**：涵盖Java基础、面向对象、常用类、集合、GUI、线程、数据库连接及网络编程等8讲内容 。
  - **考核方式**：理论成绩（作业40% + 期末60%）与实验成绩（平时60% + 小组40%）相结合 。
- **Java语言特性与对比**
  - **特性**：具有跨平台性（一次编译，到处运行）、面向对象、安全性、多线程等特点 。
  - **语言对比**：相比C++的手动内存管理，Java拥有自动垃圾回收（GC）；相比Python的解释执行，Java编译成字节码由JVM执行，性能较好且类型系统为静态强类型 。
- **运行机制与环境**
  - **JDK/JRE/JVM**：
    - **JVM**（Java虚拟机）：模拟计算机，实现跨平台运行的核心 。
    - **JRE**（Java运行环境）：包含JVM和核心类库，运行程序必需 。
    - **JDK**（Java开发工具包）：包含JRE和开发工具（如javac），用于开发 。
  - **环境配置**：需配置 `JAVA_HOME` 和 `Path` 环境变量以在命令行运行Java 。
- **开发工具 (Eclipse)**
  - Eclipse 是一个基于插件的开源集成开发环境（IDE），不自带JRE，需先安装JDK 。
  - **程序结构**：Java源文件包含包声明（package）、引入声明（import）和类定义（class）。 `public` 类名必须与文件名一致 。

Java 语法基础 (第1-2章)

- **数据类型**
  - **基本数据类型**（8种）：
    - 整数： `byte` (1字节), `short` (2字节), `int` (4字节, 默认), `long` (8字节, 需加L后缀) 。
    - 浮点： `float` (4字节, 需加F后缀), `double` (8字节, 默认) 。
    - 字符： `char` (2字节, Unicode编码) 。
    - 布尔： `boolean` (仅 `true` 或 `false`) 。
  - **引用数据类型**：包括数组、类、接口 。
- **类型转换**

    - **自动转换**：低级到高级（如 `byte -> int -> double`）自动进行。
    - **强制转换**：高级到低级需显式转换（如 `(byte) intVar`），可能导致精度丢失或溢出。
- **变量与常量**
    - 使用 `final` 关键字声明常量。
    - **标识符**：由字母、下划线、美元符号及数字组成，数字不能开头，区分大小写。
- **运算符**
    - 包含算术、关系、逻辑、位运算符及条件运算符（三目 `?:`）。
    - **注意**：整数除法（如 `10/3`）结果为整数，需精确结果应使用浮点数。

程序流程控制 (第2章)

- **选择结构**
    - **if-else**：适合处理范围和复杂的逻辑判断。
    - **switch**：适用于离散值的判断，支持 `byte`，`short`，`int`，`char`，`String` (JDK7+) 和枚举。需注意 `break` 防止穿透现象。
- **循环结构**
    - **for**：适合已知循环次数的场景。
    - **while**：先判断后执行。
    - **do-while**：先执行后判断，至少执行一次。
- **跳转语句**
    - `break`：终止当前循环或switch块。
    - `continue`：跳过本次循环剩余部分，进入下一次迭代。

数组 (第3章)

- **定义与初始化**
    - 数组是相同类型数据的有序集合，长度固定。
    - **静态初始化**：`int[] a = {1, 2, 3};`。
    - **动态初始化**：`int[] a = new int[10];` 系统赋默认值。
- **数组操作**
    - **访问**：使用索引（下标从0开始），`array.length` 获取长度。
    - **内存机制**：数组变量存储在栈（Stack），指向堆（Heap）中的连续内存空间。
- **算法应用**
    - **冒泡排序**：相邻元素两两比较，大的下沉小的上浮。
    - **二分查找**：针对有序数组的高效查找算法。

输入输出与常用类

- **键盘输入**

- 使用 `java.util.Scanner` 类。
- 常用方法：`nextInt()` (整数)，`nextFloat()` (浮点数)，`nextLine()` (字符串) 。

典型案例与作业分析

- **案例分析**
  - **购物方案（穷举法）**：利用多层嵌套 `for` 循环遍历所有可能的商品组合，满足总价要求 。
  - **找零问题（贪心思想）**：优先使用大面额硬币以减少硬币数量 。
  - **水仙花数**：利用取余 `%` 和除法 `/` 分离各位数字，计算立方和 。
  - **6174问题（卡普耶卡常数）**：将四位数数字重排求最大/小值之差，循环直至结果为6174 。
- **常见作业错误**
  - **循环控制**：在 `for` 循环中错误修改循环变量导致死循环或逻辑错误 。
  - **浮点数精度**：`float` 与 `double` 运算可能出现微小误差（如 `3.0999...`） 。
  - **自增/自减**：`k++` (先用后加) 与 `++k` (先加后用) 的区别 。

实验要求 (实验1)

- **目标**：掌握JDK环境配置，基本数据类型、运算符、流程控制及数组的操作 。
- **具体任务**：
  i. 声明并初始化各种基本数据类型，观察类型转换 。
  ii. 使用算术、关系、逻辑运算符进行计算和逻辑判断 。
  iii. 使用 `if-else` , `switch` , `for` , `while` 实现具体逻辑（如星期判断、求和、密码验证） 。
  iv. 数组操作：随机赋值、求最大最小值、冒泡排序及二分查找 。

第2周

第2周课程内容总结

本周课程主要围绕 Java 面向对象编程的核心思想（封装、继承、多态）以及异常处理机制展开，并辅以具体的实验任务来巩固理论知识。

第4章：面向对象（一）

本章重点介绍了面向对象的基础概念、类的定义与封装。

- **面向对象三大特征**：封装性、继承性、多态性 。
- **类与对象**
  - **类**：封装对象属性和行为的载体，是对象的蓝图。
  - **对象**：类的具体实例，拥有独立的内存空间 。

- **方法（Method）**
  - **构造方法**：必须与类名相同，无返回类型。若未显式定义，系统提供默认无参构造方法 。
  - **重载（Overload）**：同名方法通过不同的参数列表（个数或类型）区分，包括成员方法重载和构造方法重载 。
- **关键字**
  - **`this`**：指向当前对象本身，可用于调用成员变量、方法或本类的其他构造方法（需放首行）。
  - **`static`**：修饰静态成员（属性/方法），属于类而非具体对象，被所有对象共享，可通过类名直接访问。静态方法中不能使用 `this` 。
- **内部类**
  - 分为成员内部类、静态内部类、方法内部类和匿名内部类 。
- **访问权限**
  - Java 提供了 4 种访问权限修饰符： `private` （本类）、 `default` （本包）、 `protected` （本包及子类）、 `public` （任何地方） 。

第5章：面向对象（二）

本章深入讲解了继承、抽象、接口及多态等高级特性。

- **继承（Inheritance）**
  - 使用 `extends` 关键字，Java 仅支持单继承，但支持多层继承 。
  - **成员变量隐藏**：子类定义了与父类同名的变量。
  - **方法重写（Override）**：子类修改父类的方法，要求方法名、参数列表、返回类型相同 。
  - **`super` 关键字**：用于引用父类的成员变量、方法或构造方法 。
  - **`final` 关键字**：修饰的类不可继承，方法不可重写，变量为常量 。
- **抽象类与接口**
  - **抽象类（`abstract class`）**：可包含抽象方法（无方法体），不能实例化，强迫子类实现特定行为 。
  - **接口（`interface`）**：全局常量和公共抽象方法的集合。类通过 `implements` 实现接口，支持多重实现（模拟多继承） 。
- **多态（Polymorphism）**
  - **定义**：同一操作作用于不同对象产生不同结果。实现条件：继承、重写、父类引用指向子类对象 。
  - **转型**：
    - **向上转型**（子转父）：隐式，自动进行。
    - **向下转型**（父转子）：显式，需强制类型转换，建议配合 `instanceof` 判断 。
  - **`Object` 类**：所有类的父类，常用方法 `toString()` 和 `equals()` 。

第6章：异常（Exception）

本章介绍了 Java 的错误处理机制。

- **异常体系**
  - `Throwable` 是所有异常的基类，分为 `Error`（系统错误，免检）和 `Exception`（可捕获异常）。
  - `RuntimeException`（运行时异常）通常由逻辑错误引起（如空指针、数组越界），为免检异常。
- **异常处理**
  - **捕获**：`try-catch-finally` 结构。`finally` 块通常用于资源释放，无论是否发生异常都会执行。
  - **抛出**：
    - `throw`：在方法体内部抛出具体的异常对象。
    - `throws`：在方法声明处声明可能抛出的异常类型。
- **自定义异常**：通过继承 `Exception` 或 `RuntimeException` 创建特定业务的异常类。

实验2：面向对象编程实践

本次实验包含两个主要设计任务：

1. **校园建筑用地规划（60分）**
   - **目标**：基于面向对象思想设计校园建筑管理系统。
   - **设计**：
     - 抽象基类 `Construction`：包含共有属性（面积、位置等）和抽象方法 `displayInfo()`。
     - 子类（`Ground`，`TeachingBuilding` 等）：继承基类并实现具体逻辑，如食堂特有的 `serveFood()`。
     - **功能**：包含高德坐标（GCJ02）到 WGS84 的坐标转换算法实现。
2. **邮件地址合法检测（40分）**
   - **目标**：编写工具类 `EmailValidator` 验证邮箱格式。
   - **规则**：验证 `@` 符号数量与位置、本地部分与域名部分的长度限制（总长<254）、特殊字符（点号位置）等。
   - **异常**：检测到不合法时抛出自定义异常 `EmailValidationException` 并提示具体错误原因（如“邮箱地址必须包含@符号”）。

第3周

本周重点包括 **Java常用系统类（第7章）** 和 **I/O流（第9章）** 的理论学习，以及配套的作业讲解和地理目标实验数据。

第7章：Java常用系统类

本章主要介绍了 Java 开发中高频使用的系统核心类，帮助开发者更高效地处理数据和系统操作。

- **基本类型包装器（Wrapper Classes）**
  - 为8种基本数据类型提供了对应的引用类型（如 `int -> Integer`，`char -> Character`）。
  - **装箱与拆箱**：实现了基本类型与引用类型之间的自动转换（Auto-boxing/unboxing），便于在集合等只能存储对象的场景中使用。
- **字符串类（String）**
  - `String` 表示不可变的字符序列，一旦创建不可修改。
  - **字符串常量池**：JVM 对字符串常量的存储优化机制。
  - **常用方法**：包括 `charAt(int index)`（获取指定位置字符）、`compareTo(String anotherString)`（字符串比较）等。
- **其他常用类**
  - **System与Runtime类**：涉及系统级操作和运行时环境交互。
  - **Math与Random类**：提供数学运算和随机数生成功能。
  - **日期类**：处理日期和时间的相关操作。

第9章：I/O流（Input/Output）

本章讲解了 Java 中处理数据传输的核心机制，位于 `java.io` 包中。

- **流的分类**
  - **按数据单位**：
    - **字节流**（Byte Stream）：以字节（8bit）为单位，适用于二进制文件（图片、音频等）。顶级父类为 `InputStream` 和 `OutputStream`。
    - **字符流**（Character Stream）：以字符为单位，适用于文本数据。
  - **按传输方向**：
    - **输入流**（Input）：仅读取数据。
    - **输出流**（Output）：仅写入数据。
- **核心类与操作**
  - **文件字节流**：`FileInputStream`（读文件）和 `FileOutputStream`（写文件）是处理文件I/O最常用的字节流类。
  - **File类**：用于文件和目录路径名的抽象表示（如创建、删除文件，不涉及文件内容读写）。
  - **RandomAccessFile**：支持对文件的随机访问（读写）。

作业讲解（第7、9章）

针对第7章和第9章的课后作业进行了讲评，明确了关键知识点的考察方向。

- **第7章作业**
  - **选择题答案**：C, B, B。
  - **填空题重点**：涉及 `Date` 类、 `Calendar` 类、 `DateFormat` 类；字符串反转操作（如 `dlrowolleh` ）；时间单位（毫秒/ms）。
- **第9章作业**
  - **选择题答案**：B, D, A, C。
  - **填空题重点**：
    - 顶级父类： `InputStream` ， `OutputStream` 。
    - 流分类：字节流、字符流。
    - 文件操作类： `File` 。

实验3：地理目标数据

本次实验提供了具体的地理目标数据集，预计用于练习对象的创建、集合管理或I/O流读取解析。

- **数据内容**：包含大量具体的地理位置信息，每条记录包含以下字段：
  - **编号/学号**：如 `632032023001` 。
  - **姓名**：如 `苏方正` 。
  - **地址/名称**：如 `福建师范大学旗山校区花香园餐厅` 。
  - **类型**：如 `餐饮服务;中餐厅` 、 `科教文化服务;学校` 。
  - **经纬度**：具体的经度（Longitude）和纬度（Latitude）数值。
  - **分配地址**：具体的详细位置描述。

第4周

第8章 集合类 (Collections Framework)

本章主要介绍 Java 中的集合框架，用于存储和操作对象组。与数组不同，集合的长度是可变的，且可以存储任意类型的对象。

1. 集合框架概述

Java 在 `java.util` 包中提供了一套集合框架，主要分为单列集合（Collection）和双列集合（Map）。

- **Collection 接口**：单列集合的根接口，定义了添加、删除、清空、判断是否为空等通用方法。
  - **List**：元素**有序**、**可重复**。
  - **Set**：元素**无序**、**不可重复**。
  - **Queue**：模拟队列，先进先出 (FIFO)。

2. Collection 接口及其子接口

- **List 接口**
  - 特点：有序（存取顺序一致）、允许重复元素、支持通过索引访问。
  - **ArrayList**：最常用的实现类，内部封装了长度可变的数组。查询快，增删慢（需移动元素）。
  - **LinkedList**：基于链表实现。增删快，查询慢。
  - **Vector**：早期的线程安全列表，现在较少使用。
- **Set 接口**
  - 特点：无序、不允许重复元素。添加重复元素时返回 `false` 。
  - **HashSet**：基于哈希表实现，存取速度快。依赖 `hashCode()` 和 `equals()` 方法保证元素唯一性。
  - **TreeSet**：基于红黑树实现，可以对元素进行排序（自然排序或比较器排序）。
- **Queue 接口**
  - 实现类如 `ArrayDeque` ， `PriorityQueue` 。

3. Map 接口

Map 用于存储键值对 (Key-Value)。

- **HashMap**：基于哈希表，Key 不可重复，无序。
- **Hashtable**：古老的线程安全 Map，Key 和 Value 都不能为 null。
- **TreeMap**：基于红黑树，可按 Key 排序。
- **LinkedHashMap**：维护插入顺序。

4. 集合的遍历与操作

- **Iterator (迭代器)**：用于遍历集合的标准方式，提供 `hasNext()` 和 `next()` 方法。
- **增强 for 循环 (foreach)**：简化遍历语法的语法糖。
- **元素相等性判断**：集合中的 `contains()` ， `remove()` 等方法依赖对象的 `equals()` 方法。

5. 工具类 (Tools)

- **Collections**：操作集合的工具类，提供排序、查找、随机打乱（shuffle）等静态方法。
- **Arrays**：操作数组的工具类，提供排序、搜索、转 List 等方法。

6. 集合转换

- 数组与集合之间的相互转换（如 `Arrays.asList()` 和 `Collection.toArray()` ）。

补充：Java 键盘输入 (Scanner)

本周资料中包含了一份关于 `Scanner` 类的补充材料，虽然之前已介绍过，但在此作为基础回顾：

- **类**： `java.util.Scanner`
- **用法**： `Scanner s = new Scanner(System.in);`
- **常用方法**：
  - `nextLine()`：读取字符串。
  - `nextInt()`：读取整数。
  - `nextFloat()`：读取浮点数。

第6周

我为您总结了本周关于 **Java GUI（图形用户界面）** 的学习内容。本周主要围绕 **Swing** 库的使用、容器、组件、布局管理以及事件处理机制展开。

第10章 GUI (图形用户界面)

1. Swing 概述

- **定义**：Swing 是 Java 提供的一套轻量级 GUI 组件库，基于 AWT（Abstract Window Toolkit）构建，但提供了更丰富的组件和更强的跨平台一致性（可插拔的外观风格）。
- **与 AWT 对比**：
  - AWT 是重量级组件（依赖本地系统），Swing 是轻量级组件（纯 Java 实现）。
  - Swing 组件名通常在 AWT 组件名前加 `J`（如 `JButton` vs `Button`）。
- **组件层级**：主要分为顶层容器（如 `JFrame`，`JDialog`）和中间容器（如 `JPanel`）及基本组件。

2. Swing 容器

容器是承载其他组件的载体，分为顶层容器和中间容器。

- **JFrame (框架)**：
  - Swing 程序的主窗口，有边框和标题栏。
  - 默认布局管理器为 `BorderLayout`。
  - 需设置关闭操作： `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`。
- **JDialog (对话框)**：
  - 用于弹出的临时窗口，可设置为模态（Modal）或非模态。
  - 通常依赖于一个父窗口（Owner）。
- **JPanel (面板)**：
  - 中间容器，不能独立存在，必须添加到顶层窗口中。

    - 默认布局管理器为 `FlowLayout`。
- **JScrollPane (滚动面板)**：
    - 带滚动条的容器，通常用于显示内容较多的组件（如大文本域或列表）。只能包含一个直接子组件。

3. Swing 常用组件

- **文本组件**：
    - `JTextField`：单行文本框。
    - `JPasswordField`：密码框，输入内容以特定符号（如 `*`）显示。
    - `JTextArea`：多行文本域，可设置行数和列数。
    - 通用方法：`getText()`，`setText()`，`setEditable()`。
- **标签组件**：
    - `JLabel`：用于显示文本或图片，仅供展示，不能编辑。
- **按钮组件**：
    - `JButton`：普通按钮，用于触发动作。
    - `JRadioButton`：单选按钮，需配合 `ButtonGroup` 使用以实现互斥选择。
    - `JCheckBox`：复选框，支持多选。
- **下拉框与列表**：
    - `JComboBox`：下拉列表框，分为可编辑和不可编辑两种模式。
    - `JTree`：树形结构组件，用于展示层级数据（如文件目录）。
- **菜单组件**：
    - **下拉式菜单**：由 `JMenuBar`（菜单栏）、`JMenu`（菜单）、`JMenuItem`（菜单项）组成。
    - **弹出式菜单**：`JPopupMenu`，通常通过鼠标右键触发。

4. 布局管理器

Swing 提供了多种布局管理器来控制组件在容器中的排列方式。

- **FlowLayout (流式布局)**：`JPanel` 的默认布局。组件从左到右排列，一行满了自动换行。
- **BorderLayout (边界布局)**：`JFrame` 的默认布局。将容器分为 东、西、南、北、中 5 个区域。
- **GridLayout (网格布局)**：将容器划分为固定行数和列数的网格，所有组件大小一致。
- **GridBagLayout (网格包布局)**：最灵活但也最复杂，允许组件跨越多个网格，大小可以不同。
- **CardLayout (卡片布局)**：组件像卡片一样层叠，一次只显示一个（常用于向导界面）。
- **绝对布局 (Null Layout)**：通过 `setLayout(null)` 取消布局管理器，使用 `setBounds()` 精确指定组件坐标和大小。

5. 事件处理

Swing 的交互通过事件处理机制实现。

- **三大要素**：
  - **事件源 (Event Source)**：产生事件的组件（如按钮）。
  - **事件对象 (Event)**：封装了事件的具体信息（如鼠标点击的位置）。
  - **监听器 (Listener)**：负责处理事件的接口（需实现其中的方法）。
- **常见事件**：
  - **窗体事件**：窗口打开、关闭、激活等。
  - **鼠标事件**：点击、移动、进入、离开。
  - **键盘事件**：按键按下、释放、输入。
  - **动作事件 (ActionEvent)**：点击按钮、选择菜单项等高层语义事件。

实验内容 (推测)

根据文件名 `GUI实验5-1.mp4` 和 `GUI实验5-2.mp4`，本周实验课可能包含两个主要部分，预计涵盖：

1. 搭建基本的 GUI 界面（使用 `JFrame`，`JPanel`）。
2. 实现特定的交互功能（如按钮点击响应、数据输入与校验）。

第7周

本周课程主要涵盖了 **第11章 线程** 的核心内容，并补充了 **第10章 GUI绘图** 的相关知识。

第11章 线程 (Multithreading)

本章深入讲解了 Java 多线程编程的原理、实现方式及并发控制。

1. 线程概述与机制

- **进程与线程**：
  - **进程**：操作系统中独立执行的程序，是资源分配的基本单位 。
  - **线程**：进程中更小的执行单位，负责执行具体的任务，实现并发 。
- **创建线程的三种方式**：
  i. **继承 `Thread` 类**：重写 `run()` 方法，调用 `start()` 启动。
  ii. **实现 `Runnable` 接口**：重写 `run()` 方法，将实例传给 `Thread` 对象启动（支持多继承，较灵活）。
  iii. **实现 `Callable` 接口**：重写 `call()` 方法，配合 `FutureTask` 使用，支持返回值和抛出异常 。

2. 线程的生命周期与调度

- **生命周期**：包含新建 (New)、就绪 (Runnable)、运行 (Running)、阻塞 (Blocked) 和死亡 (Terminated) 五种状态 。

- **线程调度控制**：
  - **优先级**：使用 `setPriority(1-10)` 设置， `MAX_PRIORITY=10` ， `NORM_PRIORITY=5` ， `MIN_PRIORITY=1` 。
  - **休眠 (`sleep`)**：让当前线程暂停指定毫秒数，进入阻塞状态，不释放锁 。
  - **让步 (`yield`)**：暂停当前线程，转为就绪状态，让同优先级线程有机会执行 。
  - **插队 (`join`)**：在线程 A 中调用线程 B 的 `join()` ，A 会被阻塞直到 B 执行完毕 。
  - **后台线程 (Daemon)**：为其他线程提供服务（如 GC），通过 `setDaemon(true)` 设置，当前台线程全部结束时，后台线程会自动终止 。

3. 线程同步与通信

- **线程安全问题**：多线程并发访问共享资源（如卖票）可能导致数据不一致 。
- **同步机制 (`synchronized`)**：
  - **同步代码块**： `synchronized(lock) { ... }` 。
  - **同步方法**：在方法声明中添加 `synchronized` 关键字 。
- **死锁**：多个线程互相持有对方需要的资源且不释放，导致程序僵死 。
- **线程通信**：
  - 利用 `wait()` （等待）、 `notify()` / `notifyAll()` （唤醒）机制协调线程运行。
  - **经典案例**：生产者-消费者模式（Producer-Consumer），通过缓冲区协调生产与消费速度 。

第10章 GUI 绘图 (Graphics)

本部分是对 GUI 编程的补充，重点在于自定义绘图。

- **绘图基础**：
  - 通过继承 `JPanel` 并重写 `paintComponent(Graphics g)` 方法进行绘制。
  - 需先调用 `super.paintComponent(g)` 清除背景 。
- **常用绘图方法** (`java.awt.Graphics` / `Graphics2D`)：
  - **基本图形**： `drawLine` (直线)， `drawRect` / `fillRect` (矩形)， `drawOval` / `fillOval` (圆/椭圆)， `drawArc` (圆弧)。
  - **文本与图片**： `drawString` (字符串)， `drawImage` (图片)。
  - **样式设置**： `setColor` (颜色)， `setFont` (字体)。
- **应用场景**：可用于开发自定义控件、数据可视化图表或简单的游戏画面。

第8周

小组汇报。