以下是你的Word文档的优化：

**学习笔记**

**目录**

1. 学习方法
2. 快捷键
3. Java基础语法学习总结
   1. Java的开发环境
   2. 变量和基本数据类型
   3. 运算符和分支语句
   4. 简单输入和循环结构
   5. 数组
   6. 方法

**1. 学习方法**

**理论知识**

* 记笔记

**编码能力**

* 代码逻辑
* 熟悉API- 查文档 – GPT
* 编程能力

**编码思想**

* 面向对象编程思想

**2. 快捷键**

* ctrl + h：查看该类的子类
* ctrl + F12：查看该类的方法
* ctrl + n：进入搜索界面
* alt+ enter：万能报错和快捷
* ctrl + alt + L：格式化页面
* ctrl + alt + T：包裹功能快的快捷的操作
* ctrl + D：复制当前行
* ctrl + R：执行当前类代码

**3. Java基础语法学习总结**

**3.1 Java的开发环境**

名词解释

* **JVM** (Java Virtual Machine)：用来加载和运行 **.class** 文件。
* **JRE** (Java Runtime Environment)：包含JVM和系统类库。
* **JDK** (Java Development Kit)：JRE + 编译、运行等命令工具。

**3.2 变量和基本数据类型**

变量

* 用来存储数据的，命名规则：0-9a-zA-Z\_(\w)&
* 推荐使用小驼峰命名规范

基本数据类型

* byte、short、char、int、long、float、double、boolean
* 默认整数声明为**int**型，默认小数为**double**型
* 转换规则：大的可以存储小的，但整数和分数不能直接转换存储

**1.1.7 类和对象**

**类**

* **Class（类）**：类是一种类似于哲学的概念，它可以表示任何东西。Java是一门面向类的编程语言，即万物皆对象。类是对象的抽象化，符合我们对于现实世界中某个物品的认知。例如，苹果是水果，水果是东西，将类具象化就是这个水果是苹果。
* **编译期和运行期**：在编译期，Java文件被 javac 编译成 .class 文件；在运行期，.class 文件被 JVM 加载并转换为二进制字节码存储在内存中。JVM会对加载进来的字节码数据进行抽象化，从而产生了类的概念。
* **反射**：反射常用于框架的设计，它允许在运行时动态地创建对象、调用方法、获取字段等操作。通过读取指定文件中的类名来创建对象，实现了代码的灵活性和复用性。

**对象**

* **对象**：类的具象化实现，起源于生活。在Java中，对象是整个程序的灵魂，所有方法的构造、属性的创建、方法的调用都是以类为基础的。Java真正将对象贯彻到底，使得编程更加直观和灵活。

**补充说明**

类定义了对象的结构和行为，而对象则是类的实例化。在面向对象编程中，通过创建类来定义对象的特征和行为，并通过实例化这些类来创建具体的对象，从而实现程序的功能。

**1.1.8 继承、super、重载和重写**

**继承**

* **继承（extends）**：子类可以在权限允许的范围内继承父类，实现类的复用和扩展。子类可以访问父类的属性和方法，并且可以扩展自己的独特方法和属性。

**Super**

* **Super（超级）**：super关键字用于在子类的构造函数第一行调用父类的构造方法或访问父类的成员方法或变量。

**重载**

* **重载（Overload）**：方法重载指在同一个类中可以存在多个同名方法，但它们的参数列表不同。重载提高了代码的可读性和灵活性。

**重写**

* **重写（Override）**：方法重写是指子类重新定义父类中已经定义的方法，以改变或扩展其行为。

**Final**

* **Final**：final修饰的变量不能被改变，修饰的方法不能被重写，修饰的类不能被继承。

使用条件：

* **继承**：子类应该是父类的特殊化。如果子类与父类之间存在明显的 "is-a" 关系，则可以使用继承。
* **Super**：在子类中需要调用父类的构造方法或成员变量时使用。
* **重载**：当你希望在同一个类中拥有多个同名方法，但参数不同的情况下使用。
* **重写**：当子类需要修改或者扩展父类中的方法行为时使用，实现多态性。

**1.1.9 抽象类、抽象方法、接口、引用类型数组**

**抽象类**

* **抽象类（Abstract Class）**：抽象类是一种不能被实例化的类，它存在的目的是为了被继承。抽象类可以包含抽象方法和非抽象方法。抽象方法是没有具体实现的方法，它只有声明，而非抽象方法是有具体实现的方法。子类继承抽象类时，必须实现父类中的所有抽象方法，否则子类也必须声明为抽象类。抽象类通过 abstract 关键字声明。
* **应用场景**：当一个类具有共同的属性和方法，但其中一些方法没有具体实现时，可以将这个类声明为抽象类，以便子类来实现这些方法，从而实现代码的复用。

**抽象方法**

* **抽象方法（Abstract Method）**：抽象方法是没有具体实现的方法，它只有方法的声明，而没有方法体。抽象方法必须在抽象类中声明，而抽象类可以有零个或多个抽象方法。子类继承抽象类时，必须实现父类中的所有抽象方法，否则子类也必须声明为抽象类。
* **应用场景**：当一个类中有一些方法需要在子类中具体实现时，可以将这些方法声明为抽象方法，从而强制子类实现这些方法，提高了代码的可变性。

**接口**

* **接口（Interface）**：接口是一种特殊的抽象类，它定义了一组抽象方法和常量。接口中的方法默认是抽象的，不需要使用 abstract 关键字声明。类可以实现接口，并且必须实现接口中定义的所有方法。接口可以用于描述对象的行为，实现多态性和代码的组织。
* **应用场景**：当需要定义一组抽象方法来描述对象的行为时，可以使用接口。如果一个类需要实现多个行为，可以实现多个接口。

**引用类型数组**

* **引用类型数组**：引用类型数组是指数组中的元素是对象的引用，而不是对象本身。在Java中，数组可以存储对象的引用，而不是对象的实例。这意味着数组中的每个元素都可以引用一个对象，但并不包含对象的实际数据。需要注意的是，数组的长度是固定的，一旦创建就无法改变。
* **应用场景**：当需要存储对象的引用，并且需要根据索引来访问这些对象时，可以使用引用类型数组。

使用条件：

* **抽象类**：当一个类具有共同的属性和方法，但其中一些方法没有具体实现时，可以将这个类声明为抽象类。
* **抽象方法**：当一个类中有一些方法需要在子类中具体实现时，可以将这些方法声明为抽象方法，从而强制子类实现这些方法。
* **接口**：当需要定义一组抽象方法来描述对象的行为时，可以使用接口。如果一个类需要实现多个行为，可以实现多个接口。
* **引用类型数组**：当需要存储对象的引用，并且需要根据索引来访问这些对象时，可以使用引用类型数组。数组的长度是固定的，所以需要提前知道存储的元素数量。

抽象类、抽象方法、接口以及引用类型数组是Java面向对象编程中常用的概念，它们能够提高代码的可重用性、灵活性和组织性。

**1.2.1 多态、内部类**

**多态**

* **多态（Polymorphism）**：多态是面向对象编程中的一个重要概念，它允许同一操作作用于不同的对象，产生不同的行为。具体来说，多态性有两种形式：编译时多态和运行时多态。编译时多态发生在编译时期，主要是通过方法重载实现的；而运行时多态则发生在运行时期，主要是通过方法重写和接口实现实现的。
* **应用场景**：适用于需要处理不同类型对象的情况，通过统一的接口对不同的对象进行操作，提高了代码的灵活性和可维护性。

**内部类**

* **内部类（Inner Class）**：内部类是定义在另一个类内部的类。内部类可以访问包含它的外部类的成员，包括私有成员。内部类可以分为静态内部类、成员内部类、局部内部类和匿名内部类四种类型。
* **应用场景**：当一个类只在一个类内部使用，并且它对外部类的成员有访问需求时，可以考虑使用内部类。内部类可以帮助组织代码、隐藏实现细节，并且可以更轻松地访问外部类的成员。

**1.2.2 访问权限控制，static的使用**

**访问权限控制**

在Java中，有四种访问权限控制修饰符：

* **Private（私有的）**：Private关键字用于限制成员（字段或方法）的访问权限，仅允许在声明它们的类中使用。私有成员无法被该类的外部访问。
* **Default（默认的）**：缺省访问修饰符不是default关键字，而是不使用任何访问修饰符。这种修饰符允许成员在同一包中的其他类中访问，但对于不在同一包中的类来说是不可见的。
* **Protected（受保护的）**：Protected关键字用于限制成员的访问权限，允许在声明它们的类、同一包内的其他类以及该类的子类中使用。受保护成员对于不在同一包中的其他类来说是不可见的。
* **Public（公共的）**：Public关键字用于指定成员的访问权限，允许在任何地方都能访问该成员。公共成员对所有类都是可见的。

**Static**

* **Static（静态的）**：静态关键字用于指定成员（字段、方法、代码块）属于类而不是实例，因此它们只会被类的单一副本所持有，而不是每个实例都有自己的副本。静态成员在类加载时被初始化，并且可以通过类名直接访问。
* **Static final常量**：应用率高，必须声明同时初始化，常常通过类名点来访问，不能被改变。建议：常量名所有字母都大写，多个单词之间用\_分隔。编译器在编译时，会将常量直接替换为具体的数，效率高。何时用：在程序运行过程中数据永远不变，并且经常使用。

**使用条件**

* **Private、Default、Protected和Public**：根据访问权限的需求选择合适的访问修饰符，确保代码的安全性和可维护性。
* **Static**：当需要将成员与类关联，而不是与实例关联时，使用静态修饰符。

**1.2.3 Enum**

**枚举类型的定义**

在Java中，可以通过关键字 enum 来定义枚举类型。枚举类型的每个值都是枚举类型的一个实例。下面是一个简单的枚举类型的定义示例：

enum Weekday {

MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY

}}

枚举类型的应用

枚举类型在Java中有以下几种应用：

* **作为常量集合**：枚举类型可以用于表示一组常量，例如表示一周中的每一天。
* **类型安全**：枚举类型在编译时就确定了所有可能的取值，从而避免了使用常量或字符串字面量时可能出现的拼写错误或逻辑错误。
* **易于理解和维护**：枚举类型可以提高代码的可读性，因为它明确地列出了所有可能的选项。
* **用于switch语句**：枚举类型可以与 switch 语句一起使用，使代码更加清晰和易于理解。

**示例**

下面是一个使用枚举类型的例子，表示一周中的每一天，并输出它们的顺序：

enum Weekday {

MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY

}

public class EnumExample {

public static void main(String[] args) {

for (Weekday day : Weekday.values()) {

System.out.println(day + " is day number " + (day.ordinal() + 1) + " of the week.");

}

}

}

输出结果为：

MONDAY is day number 1 of the week.

TUESDAY is day number 2 of the week.

WEDNESDAY is day number 3 of the week.

THURSDAY is day number 4 of the week.

FRIDAY is day number 5 of the week.

SATURDAY is day number 6 of the week.

SUNDAY is day number 7 of the week.

## Java基础语法学习总结

### 1.1.1 Java的开发环境

- \*\*编译期\*\*：写好的 `.Java` 文件被 `javac` 编译成 `.class` 文件

- \*\*运行期\*\*：`.class` 文件被JVM加载（转换为二进制字节码存储在内存中）并运行

- \*\*特点\*\*：一次编译，到处使用

#### 名词解释

- \*\*JVM\*\* (Java Virtual Machine)：用来加载和运行 `.class` 文件。

- \*\*JRE\*\* (Java Runtime Environment)：最小的运行环境，JVM + 系统类库 (`java.lang.\*`, `java.util`, `java.io.\*`, `java.net.\*`)

- \*\*JDK\*\* (Java Delevopment Kit)：Java开发工具包，最小的开发环境，JRE + 编译，运行等命令工具。

### 1.1.2 变量和基本数据类型

#### 变量

- 用来存储数据的，命名规则：0-9a-zA-Z\_(\w)&，不能以数字开头，命名推荐：小驼峰命名规范

#### 基本数据类型

- \*\*Byte\*\*：1

- \*\*short\*\*：2

- \*\*char\*\* (无符号)：2

- \*\*int\*\*：4

- \*\*long\*\*：8

- \*\*float\*\*：4

- \*\*double\*\*：8

- \*\*boolean\*\*：1

### 1.1.3 运算符和分支语句

#### 运算符

- \*\*算术\*\*：+，-，\*，/，%，++，--

- \*\*关系\*\*：>、<、>=、<=、==、!=

- \*\*逻辑\*\*：&&、||、!

- \*\*赋值\*\*：=、+=、-=、\*=、/=、%=

### 1.1.4 简单输入和循环结构

#### 分支结构(下)

- \*\*switch...case\*\* 结构：多条路

- 优点：效率高、结构清晰

- 缺点：只能对整数判断相等

- `break`：跳出switch

- switch中数据的类型：byte, short, int, char, String, 枚举

#### 循环

- 反复多次执行一段相同或相似的代码

- \*\*循环三要素\*\*：

1. 循环变量的初始化

2. 循环的条件（以循环变量为基础）

3. 循环变量的改变

- \*\*循环结构\*\*：

- `while` 结构：先判断后执行，有可能一次都不执行

- `do...while` 结构：先执行后判断，至少执行一次

- `for` 结构：应用率最高，适合与次数相关的

### 1.1.5 数组

- 数组用于存储数据，每个元素的地址是连续的

- 数组名为第一个元素的地址，下标从1开始

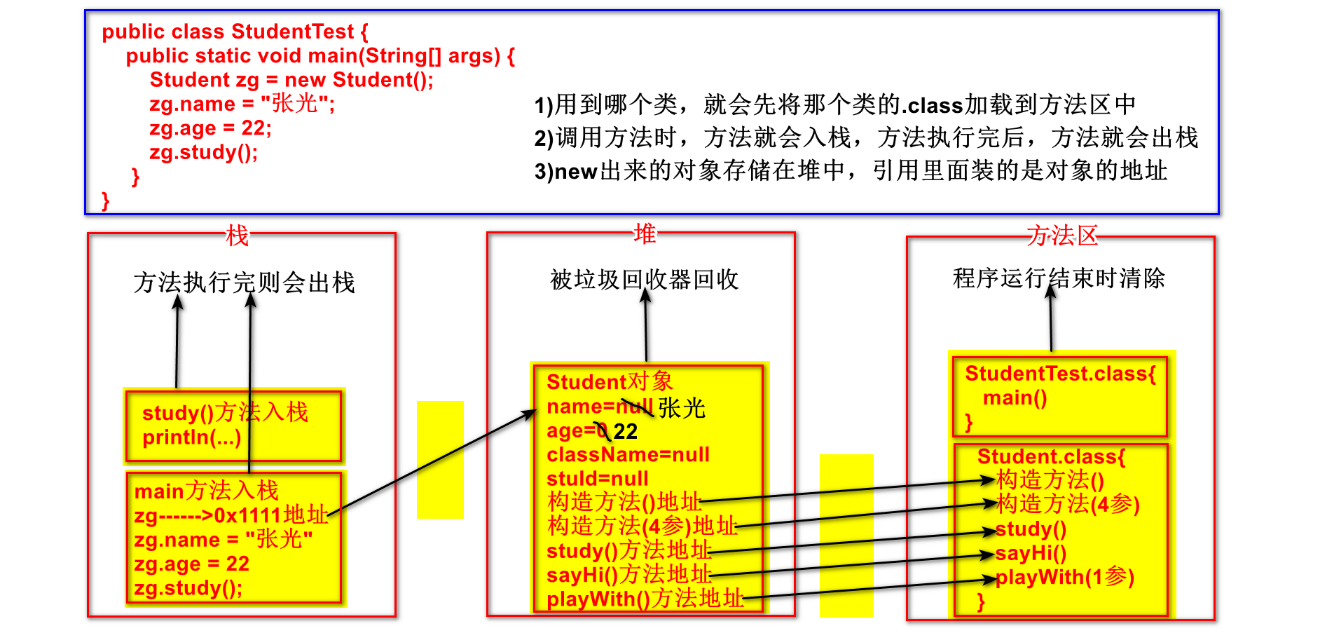
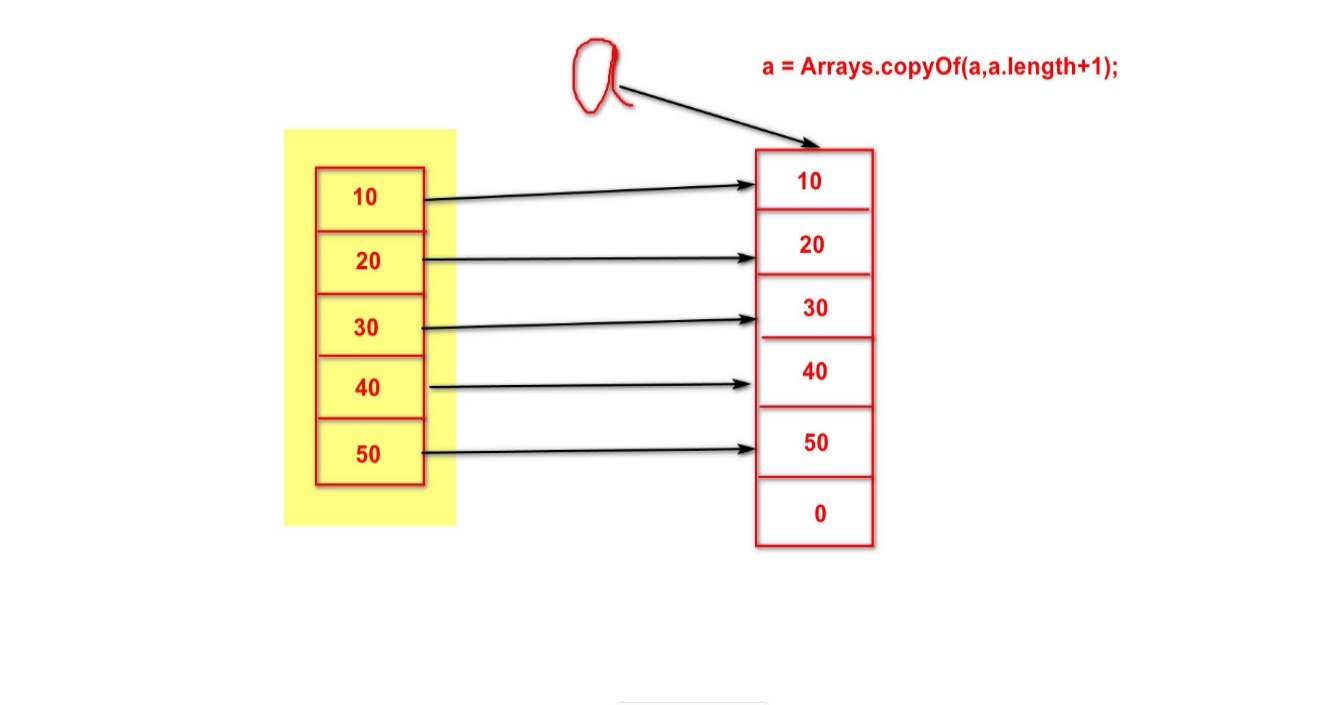
- 遍历数组、数组拷贝、数组扩容、排序等操作

### 1.1.6 方法

- 方法用于封装特定的业务逻辑功能

- 方法调用和返回值

- 方法重载



1.1.7类和对象

类：

译为Class,这个类是一种类似于哲学的概念，你可以说它是任何东西，java是一门Oriented Class的编程语言，万物皆对象，类即是对象的抽象化，符合现实中我们人类对于某个物品的认知，例如：苹果是水果，水果是东西，将类具象化就是这个水果是苹果，在.java文件被javac编译成.class文件,.class通过JVM加载到内存中去，把.class文件变成2进制存储在内存中,而在内存中又对加载进来的2进制数据进行抽象，从而产生了类的类的说法，而对于其中的相同的属性和行为又会被再一次的抽象化为类，以此来存储到内存中，产生了反射的概念，反射常用于框架的设计，因为有时候我们无法直接主管的创建对象，通过读取指定文件中的名字来创建对象，这时我们只能使用到反射技术，这种框架集设计其实也是方便了后期的代码的复用性和改动，只需要改动配置文件中的信息就可以实现想要的效果。面对一些无法直接调用的属性和行为，我们可以也可以通过反射实现调用。

对象：

类的具象化实现，万物皆对象，起源于生活，以人的思想而产生了对象的概念，例如我们的母语，从来都是以主语来实现，没有主语的参与，整个对话会变得没有逻辑，主语就类似于对象，从来都是主语的动作或属性，而java中也是以对象为整个程序的灵魂，不管是方法的构造，属性的创建，方法的调用，都是以类为基础的，可以说java真正的将对象贯彻到底了。

GPT:  
当谈到面向对象编程（OOP）时，"类"和"对象"是两个基本概念。下面是对它们的详细说明：

**类（Class）**： 类是面向对象编程的基本构建单元之一。它是一种用户定义的数据类型，用于表示某种实体或抽象概念。类可以包含属性（也称为字段或成员变量）和方法（也称为函数或成员函数）。属性描述了类的特征或状态，而方法定义了类的行为或功能。类可以看作是对象的蓝图或模板，它定义了对象的结构和行为。

**对象（Object）**： 对象是类的实例。它是类的具体化，代表了现实世界中的一个实体或一个概念。每个对象都具有自己的状态（通过类的属性来表示）和行为（通过类的方法来实现）。在程序中，对象被用来执行特定的任务，与其他对象交互，或者表示系统中的实体。

简而言之，类定义了对象的结构和行为，而对象则是类的实例化。类是抽象的，描述了一类对象的共同特征和行为；而对象是具体的，代表了这个类中的一个具体实例。在面向对象编程中，我们通过创建类来定义对象的特征和行为，并通过实例化这些类来创建具体的对象，从而实现程序的功能。

1.1.8继承(extends),super重载(overload),重写(override),final

继承：

一个类(derived class)可以在权限允许的范围内继承另外一个类(base class)，一个类只允许有一个base class,即单根继承，derived class 可以访问 base class(super), 而base class 不能访问 derived class,该类为派生类,derived class extends base class 权限内(private default protected public)可以得到的属性和方法，同时derived class 可以扩展出自己独特的方法和属性以满足业务的个性需求，也是java 三大特性之一(封装，继承，多态)。同时，该派生类进行实例化时必须得先构建父类？，默认调用base class的无参构造方法，然后向下依次调用构造方法，如果base class 不存在 无参构造方法，则 derived class 必须在自己的构造方法中显示的调用父类的构造方法，super(),只能放在构造方法的第一行。

应用—derived class 需要拥有base class 属性和方法以及独特的方法，实现多形态。

Super:

译为超级。

应用：在derived class 的构造函数第一行，在derived class 的其他中调用base class 被隐藏或重写的方法。

Overload:

在本类中,或继承自父类的方法中，当创建的方法的方法名原来拥有的方法相同时，参数列表与原来的放方法不同时，就会产生重载，当调用该方法时，编译器会根据传递的参数将相应的方法和对应的类绑定1同1不同。

Override:

在extends(扩展)时，derived class 获得base class的方法，可以对base class 方法进行覆盖，

条件：方法名相同，参数列表相同，返回值相同，derived class 重写的方法权限大于等于base class 的全限，重写的方法抛出的异常不能比base class 的方法范围大。

final：

修饰变量：变量不能被改变 修饰方法：方法不能被重写 修饰类：类不能被继承

GPT：

继承（extends）： 继承是面向对象编程中的一个重要概念，它允许一个类（子类）基于另一个类（父类）来创建。子类继承了父类的属性和方法，并且可以增加自己的属性和方法，或者修改父类的方法行为。继承通过使用 extends 关键字实现。子类可以继承父类的非私有属性和方法，但不能继承父类的私有成员。

super： super 是一个关键字，用于引用父类的成员变量、构造方法或方法。在子类中，如果要调用父类的构造方法或成员变量，可以使用 super 关键字。在方法重写时，也可以使用 super 调用父类的方法。

重载（Overload）： 方法重载是指在同一个类中可以存在多个同名方法，但它们的参数列表不同（参数类型、参数个数或参数顺序）。重载的方法拥有相同的名称，但是不同的参数列表，这样可以根据传递的参数类型和个数来调用不同的方法。重载可以提高代码的可读性和灵活性。

重写（Override）： 方法重写是指子类重新定义父类中已经定义的方法。在子类中，方法的名称、返回类型和参数列表必须与父类中的方法相同，这样才能称为方法重写。通过重写，子类可以改变或扩展父类方法的行为，实现多态性。

使用条件：

继承：子类应该是父类的特殊化。如果子类与父类之间存在明显的 "is-a" 关系，则可以使用继承。

super：在子类中需要调用父类的构造方法或成员变量时使用。

重载：当你希望在同一个类中拥有多个同名方法，但参数不同的情况下使用。

重写：当子类需要修改或者扩展父类中的方法行为时使用，实现多态性。

继承、super、重载和重写是面向对象编程中非常重要的概念，它们使得代码更加模块化、灵活和可维护。

1.1.9抽象类,抽象方法,接口,引用类型数组

抽象类：

Abstract class,不能够new(),可以声明抽象方法,属性随意,也可以没有抽象方法，其他类要想继承该类，得实现它得抽象方法才能声明为类，不然只能为抽象类。复用代码。

抽象方法：

只有方法的声明，没有方法体，abstract 方法名()，提高编程的可变性。

接口：

里面声明的属性都是(public static final) 常量，声明的方法有抽象方法public abstract 方法名()，Java后面的版本又可以使用public static 返回类型 方法名(){}，public default 返回类型 方法名(){} 。

作用：作为一个方法的扩展，部分具有的功能，直接实现该接口，保证方法已被实现，提高编程的可变性。

引用类型数组：

数组的原名为存储的类型的起始地址，数组中存储的是存储数据的地址。

GPT：

抽象类（Abstract Class）： 抽象类是一种不能被实例化的类，它存在的目的是为了被继承。抽象类可以包含抽象方法和非抽象方法。抽象方法是没有具体实现的方法，它只有声明，而非抽象方法是有具体实现的方法。子类继承抽象类时，必须实现父类中的所有抽象方法，否则子类也必须声明为抽象类。抽象类通过 abstract 关键字声明。

抽象方法（Abstract Method）： 抽象方法是没有具体实现的方法，它只有方法的声明，而没有方法体。抽象方法必须在抽象类中声明，而抽象类可以有零个或多个抽象方法。子类继承抽象类时，必须实现父类中的所有抽象方法，否则子类也必须声明为抽象类。

接口（Interface）： 接口是一种特殊的抽象类，它定义了一组抽象方法和常量。接口中的方法默认是抽象的，不需要使用 abstract 关键字声明。类可以实现接口，并且必须实现接口中定义的所有方法。接口可以用于描述对象的行为，实现多态性和代码的组织。

引用类型数组： 引用类型数组是指数组中的元素是对象的引用，而不是对象本身。在Java中，数组可以存储对象的引用，而不是对象的实例。这意味着数组中的每个元素都可以引用一个对象，但并不包含对象的实际数据。需要注意的是，数组的长度是固定的，一旦创建就无法改变。

使用条件：

抽象类：当一个类具有共同的属性和方法，但其中一些方法没有具体实现时，可以将这个类声明为抽象类。

抽象方法：当一个类中有一些方法需要在子类中具体实现时，可以将这些方法声明为抽象方法，从而强制子类实现这些方法。

接口：当需要定义一组抽象方法来描述对象的行为时，可以使用接口。如果一个类需要实现多个行为，可以实现多个接口。

引用类型数组：当需要存储对象的引用，并且需要根据索引来访问这些对象时，可以使用引用类型数组。数组的长度是固定的，所以需要提前知道存储的元素数量。

抽象类、抽象方法、接口以及引用类型数组是Java面向对象编程中常用的概念，它们能够提高代码的可重用性、灵活性和组织性。

窗体顶端

1.2.1多态,内部类

多种形态，由于extends 和 implements 产生了不同的形态，多种形态可以进行向上造型和向下造型，向上造型可以提高代码的复用性，向下造型则是为了调用向上造型丢失的Field 或 method,为了恢复原状，在向下造型过程最后先使用a instanceof b 判断 a 是不是 b 的实例，再让 a强转为b。 强转发送异常会抛出ClassCastException的异常。

成员内部类：

内部类是在类中声明的类，也可以在方法中声明类(局部类，作用范围仅在方法体中），内部类的可见性由外部类来决定，InnerClass 可以调用 OuterClass 的属性和方法(使用的是外部类名.this)，其他类想要使用内部类，必须通过外部类。何时用：若A类(Baby)只让B类(Mama)用，并且A类(Baby)还想访问B类(Mama)的成员时，可以设计成员内部类，内部类的名称为 外部类型$外部类名.class，分类：成员内部类，静态内部类，

匿名内部类，是没有名字的，直接去实现了抽象类的抽象方法或接口，采用向上造型，此derived class没有名字，常用于函数式接口的实现，采用Lambda表达式简写代码。

package和import：

package：声明包 作用：避免类的命名冲突 规定：同包中的类不能同名，但不同包中的类可以同名。

类的全称：包名.类名。包名常常有层次结构 建议：包名所有字母都小写

import：导入类 同包中的类可以直接访问，但不同包中的类不能直接访问，若想访问：

先import导入类，再访问类--------------建议 类的全称------------------------------太繁琐、不建议

GPT：

多态（Polymorphism）： 多态是面向对象编程中的一个重要概念，它允许同一操作作用于不同的对象，产生不同的行为。具体来说，多态性有两种形式：编译时多态和运行时多态。编译时多态发生在编译时期，主要是通过方法重载实现的；而运行时多态则发生在运行时期，主要是通过方法重写和接口实现实现的。

内部类（Inner Class）： 内部类是定义在另一个类内部的类。内部类可以访问包含它的外部类的成员，包括私有成员。内部类可以分为静态内部类、成员内部类、局部内部类和匿名内部类四种类型。

下面是Java代码示例：

// 多态示例

class Animal {

public void sound() {

System.out.println("Animal makes a sound");

}

}

class Dog extends Animal {

@Override

public void sound() {

System.out.println("Dog barks");

}

}

class Cat extends Animal {

@Override

public void sound() {

System.out.println("Cat meows");

}

}

public class PolymorphismExample {

public static void main(String[] args) {

Animal animal1 = new Dog(); // 多态：父类引用指向子类对象

Animal animal2 = new Cat(); // 多态：父类引用指向子类对象

animal1.sound(); // 输出 "Dog barks"

animal2.sound(); // 输出 "Cat meows"

}

} javaCopy code

// 内部类示例

public class Outer {

private int outerField = 10;

// 成员内部类

class Inner {

public void display() {

System.out.println("Value of outerField in Inner class: " + outerField);

}

}

public void outerMethod() {

// 在外部类中创建内部类的实例并调用方法

Inner inner = new Inner();

inner.display();

}

public static void main(String[] args) {

Outer outer = new Outer();

outer.outerMethod();

}

}

使用条件：

多态：适用于需要处理不同类型对象的情况，通过统一的接口对不同的对象进行操作，提高了代码的灵活性和可维护性。

内部类：当一个类只在一个类内部使用，并且它对外部类的成员有访问需求时，可以考虑使用内部类。内部类可以帮助组织代码、隐藏实现细节，并且可以更轻松地访问外部类的成员。

1.2.2 访问权限控制，static的使用

访问权限控制：

Private 仅限本类中使用，default 缺省的，本类和同包类中可以使用--------------------------java不建议，protected 受保护的，本类，同包类和同包子类中可以使用，public到处用~。

详细：

Private（私有的）：

Private关键字用于限制成员（字段或方法）的访问权限，仅允许在声明它们的类中使用。私有成员无法被该类的外部访问。

缺省访问修饰符不是default关键字，而是不使用任何访问修饰符。这种修饰符允许成员在同一包中的其他类中访问，但对于不在同一包中的类来说是不可见的。

Protected关键字用于限制成员的访问权限，允许在声明它们的类、同一包内的其他类以及该类的子类中使用。受保护成员对于不在同一包中的其他类来说是不可见的。

Public关键字用于指定成员的访问权限，允许在任何地方都能访问该成员。公共成员对所有类都是可见的。

Static：

静态关键字用于指定成员（字段、方法、代码块）属于类而不是实例，因此它们只会被类的单一副本所持有，而不是每个实例都有自己的副本。静态成员在类加载时被初始化，并且可以通过类名直接访问。

static final常量：

应用率高 必须声明同时初始化 常常通过类名点来访问，不能被改变

建议：常量名所有字母都大写，多个单词之间用\_分隔 编译器在编译时，会将常量直接替换为具体的数，效率高 何时用：在程序运行过程中数据永远不变，并且经常使用

1.2.3enum

枚举类型的定义

在Java中，可以通过关键字 enum 来定义枚举类型。枚举类型的每个值都是枚举类型的一个实例。下面是一个简单的枚举类型的定义示例：

enum Weekday {

MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY

}

枚举类型的应用

作为常量集合：枚举类型可以用于表示一组常量，例如表示一周中的每一天。

类型安全：枚举类型在编译时就确定了所有可能的取值，从而避免了使用常量或字符串字面量时可能出现的拼写错误或逻辑错误。

易于理解和维护：枚举类型可以提高代码的可读性，因为它明确地列出了所有可能的选项。

用于switch语句：枚举类型可以与 switch 语句一起使用，使代码更加清晰和易于理解。

枚举类型的例子

下面是一个使用枚举类型的例子，表示一周中的每一天，并输出它们的顺序：

enum Weekday {

MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY

}

public class EnumExample {

public static void main(String[] args) {

for (Weekday day : Weekday.values()) {

System.out.println(day + " is day number " + (day.ordinal() + 1) + " of the week.");

}

}

}

输出结果为：

MONDAY is day number 1 of the week.

TUESDAY is day number 2 of the week.

WEDNESDAY is day number 3 of the week.

THURSDAY is day number 4 of the week.

FRIDAY is day number 5 of the week.

SATURDAY is day number 6 of the week.

SUNDAY is day number 7 of the week.