知识小料

「电计 2203 班」周常规知识整理共享

第 31 期

日期：2024-6-16

学科：高级程序设计语言（C++）

C++ 考试选手应知应会的 Java 知识。

1 Java 语言概述

	Java	C++
开发方法	纯面向对象	面向对象
执行原理	解释型语言	编译型语言
有无指针	无	有
异常处理	出色	一般
执行效率	较慢	较快

表 1: Java 与 C++ 的主要对比

结构化程序设计方法的主要原则：自顶向下、逐步求精、模块化。

JVM,JRE,JDK 名词辨析

- JVM（Java 虚拟机）——是 Java 语言运行的虚拟机；
- JRE（包含 JVM 和核心库）——是 Java 语言运行的环境；
- JDK（包含 JRE 和开发工具）——是 Java 开发人员使用的工具；
- 总结起来，JDK 包含 JRE，JRE 包含 JVM。

Java 程序的执行流程

- 编写程序——程序员编写代码，得到源文件，目标为 .java 文件；
- 编译文件——用编译器编译源文件，得到字节码文件，目标为 .class 文件；
- 解释文件——用 JVM（Java 虚拟机）解释字节码文件，得到结果。

Java 标识符——类名、变量名

- **允许**——字母、数字、下划线 (_)、美元 ($) 组成
- **不允许**——数字开头
- **不允许**——包含 @、# 等特殊字符和空格
- **不允许**——与各种关键字重名

原则上类名以大写字母开头，对象名、函数名以小写字母开头。
示例：变量名如 a_b$c、a1 正确；如 a_b@c、1a、goto [1]错误。
关键字举例：

- true、false——小写才是关键字
- implements、extends——用于类的继承和接口调用
- var、null——C++ 没有而 Java 特有的

2 类与对象·基本

类 是对象的抽象化，对象 是类的具体化，实例 是对象的别名。

2.1 类的基本定义

类的基本定义格式：(成员变量 + 构造器 + 方法)

```
1  public class Person {
2      String name; //类的成员变量
3      int age;
4      Person(String name, int age){ //构造器
5          this.name = name;
6          this.age = age;
7      }
8      void say(String content){ //类的方法
9          System.out.println(content);
10     }
11 }
```

[1]其实 Java 语言中有 goto 这个关键字，但是这个指令不允许使用。

2.2 构造器

构造器就是一种特殊的方法，用于创建对象。比如上面例子的

`Person(String name, int age)`

就是构造器。

在主方法[2]中，可以使用 `Person p = new Person("XiaoMing", 19);` 这样的语句来构造一个对象 p。

其他规则

- 构造器名须与类名相同。
- 构造器名不要指定返回值。
- 如果没有指定构造器，则系统会默认提供一个空的构造器。

2.3 对象访问与 this 的使用

对于上述的 p 对象，可以用点号操作符来访问它的成员变量与方法。

`p.age = 20;` 或者 `p.say("Hello");`

在构造器中，`this` 总是指向调用该方法的对象。比如刚刚的例子，构造器中的 `this` 指向的就是 p，构造的过程即表示把 p 的成员变量 `p.name` 赋值为 `name`，把另一个成员变量 `p.age` 赋值为 `age`。

2.4 变量

用一个例子捋清楚这些变量：

```
1  public class Person{
2      String name;  //成员变量·实例变量
3      static int age; //成员变量·类变量
4      Person(String newName, int newAge){ //局部变量·形参
5          this.name = newName;
6          {
7              int x = 1; //局部变量·方法局部（也是代码块局部）
8              this.age = newAge + 1;
9          }
10     }
```

[2]在 Java 中，「函数」一般称作「方法」。

图 1: 变量分类图

```
11 }
```

其他规则

1. 对于成员变量，以下的访问是允许的：
   - 实例.实例变量——p.name
   - 实例.类变量——p.age
   - 类.类变量——Person.age
2. 对于成员变量，初始值是系统会默认分配的。对于局部变量，必须明确初始化。
3. 类变量的作用域是整个类，实例变量的作用域仅限于其实例。

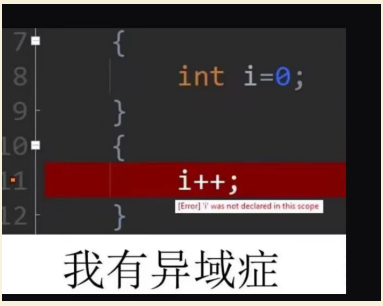

图 2: 代码块局部变量在离开其作用域后立即销毁

2.5 方法

- 未使用 static 修饰的方法只能拿实例调用，不能拿类调用。
- 使用了 static 修饰的方法既能拿实例调用，也能拿类调用。

```
class Person{
    //成员变量与构造器的定义省略
    public void sing() { ... } //无static，只能实例调用
    public static void run() { ... } //有static，实例和类都能调用
}
public class Test{
    public static void main(String [] args){
        Person p = new Person("小明",19);
        p.sing();   //实例调用
        p.run();    //实例调用
        Person.run(); //类调用
    }
}
```

3 类与对象·封装、继承、多态

封装、继承、多态，即是面向对象的三个特征。

3.1 封装

封装是一种将抽象性接口细节包装、隐藏起来的方式。通过访问控制符，可以增加程序安全性。

访问控制符：public > protected > default > private

比如说，public 修饰的方法、变量可以在全局范围内访问，private修饰的方法、变量仅限于类内访问。下列的例子中，要修改对象 p 的信息，不可以直接访问 private 修饰的 p.name 和 p.age，而是可以用 public 修饰的方法 setInformation 来实现。

```
class Person{
    private String name; //私有的成员变量
    private int age;
    public Person(String name, int age){ //公有的构造器
        this.name = name;
```

```
        this.age = age;
    }
    public void setInformation(String name, int age){ //公有的方法
        this.name = name;
        this.age = age;
    }
}
public class Test{
    public static void main(String [] args){
        Person p = new Person("小明", 19); //构造器是public的，所以在Person类外
            可以使用
//      p.age = 20; //这两个成员变量都是private的，不可以类外访问，所以会出错
//      p.name = "小红";
        p.setInformation("小红", 20); //这个方法是public的，可以类外访问
    }
}
```

3.2 继承

Java 也有继承的操作，不过 一个类至多只能继承一个父类 ，即不支持多继承。

子类 Son 继承父类 Father 的变量 x，同时也有其新的变量 y。子类能够拥有父类全部的属性、方法（除非是 `private`）。

```
class Father{ //父类的定义
    int x;
    Father(int newX){
        this.x = newX;
    }
}
class Son extends Father{ //子类的定义
    int y;
    Son(int newX, int newY){
        super(newX); //super关键字用于调用父类的构造器（方法、属性等）
        this.y = newY;
    }
}
```

3.3 多态

在 C++ 中，多态性主要体现为函数与运算符的重载。在 Java 中，多态性则主要体现为函数的重载（Overload）与重写（Override）。

- 重载（Overload）——多个同名但是形参列表不同的方法，就是方法重载。
- 重写（Override）——用子类的某个方法覆盖父类的同名方法，就是方法重写。

比如，以下的例子实现了构造器的重载。（类似于 C++ 的构造函数和「拷贝构造函数」的关系）

```
class Person{
    String name;
    int age;
    Person(String name, int age){ //带String,int类的两个参数的构造器
        this.name = name;
        this.age = age;
    }
    Person(Person p){ //带Person类的一个参数的构造器
        this.name = p.name;
        this.age = p.age;
    }
}
```

再比如，以下的例子实现了 run 方法的重写。

```
class Animal{
    void run() { System.out.println("动物的run方法"); }
}
class Dog extends Animal{
    void run() { System.out.println("狗狗的run方法"); }
}
public class Test{
    public static void main(String [] args){
        Dog d = new Dog();
        d.run(); //执行狗狗的run方法，该方法重写了父类动物的run方法
    }
}
```

4 类与对象・特殊的类

4.1 单例类

如果一个类始终只能创建一个对象，那就是单例类。

4.2 终稿类

如果一个类不能被继承，那就是终稿类。用`final`修饰。

典型的终稿类是 `String` 类。

终稿变量（`final` 修饰的变量）是一经初始化就不可修改的变量。

4.3 抽象类

如果一个类不能直接创建对象(不能被实例化),那就是抽象类。用`abstract`修饰。

这种类存在的意义在于提供一个模板，用子类继承抽象类，从而实现各种具体的功能。

如果一个类含有抽象方法，那整个类都必须用`abstract`修饰。

4.4 接口

接口是抽象类的一种，用`interface` 来定义接口，用 `implements`来调用接口。

一个类可以实现多个接口，这使得 Java 克服了不能多继承的困难。

4.5 内部类

把一个类放在另一个类的内部进行定义，那就是内部类。

如果把一个类放在某个方法的内部进行定义，那就是局部内部类。

4.6 Java 的垃圾回收机制

当对象永久性地失去引用时，系统就会回收其所占的资源。而在回收任何对象之前，总会先调用它的 `finalize` 方法以试图「救活」它，如果能「救活」则该对象继续运行，如果「救不活」那就只能回收资源了。

4.7 交付包（.jar）

Java 归档文件（JAR）是将若干个 `.class` 文件压缩、归档并交付给用户使用的交付包，扩展名为 `.jar`。

.jar 文件装的是.class 文件，可以用压缩软件来解压缩。