类和对象

公告

今天学习内容：

- 类和对象
  - 类和对象定义/如何识别类/类之间的关系
  - 类的组成
    - 域变量
    - 构造器
    - 方法
    - 修饰符：访问权限控制/static 修饰符/final 修饰符
    - 代码块
    - 包
  - 创建对象
  - 应用练习
  - 常见面试问题
- 课后练习

类和对象定义

Java 是一门面向对象的程序设计语言，它把一切都看作是对象，Java 源代码都是以类的形式来组织。

首先我们来看一下类和对象的概念：

- 类

  类是构造对象的模版或者蓝图，它定义了一类对象的状态和行为，从形式来看是将数据和行为封装放在一个类里。类中的数据称为实例域（instance field），操作数据的过程称为方法（method）。

- 对象

  对象是类的一个实例，有状态和行为。每个类实例（对象）都有一组特定的实例域值，这些值的集合就是这个对象的当前状态（state）。

例如：人是一个对象，他的状态有：姓名、性别、年龄、身高、体重等；他的行为有：走路、吃饭、睡觉、工作、学习等。而具体的每一个人则为该类的对象（object）。

根据上述分析，定义人这个类如下：

```
public class Person {
    private String name;
    private Integer sex;
    private Integer age;
    private Integer height;
    private Integer weight;

    public void walking() {}
```

```
    public void eat() {}

    public void gotoBed() {}

    public void work() {}

    public void study() {}
}
```

而每一个人，也就是根据类 `Person` 构造出来的对象则是该类的实例，例如下边构造了 person1、person2、person3 三个实例：

```
Person person1 = new Person();
Person person2 = new Person();
Person person3 = new Person();
```

如何识别类

如何进行面向对象编程（OOP），我们通常是从设计类开始，然后再向类中添加属性和方法。

那么怎么来识别类？有一个简单的规则：在分析问题的过程中寻找名词，而方法对应着动词。

例如：我们来分析订单处理系统，有这样一些名词：

- 商品（Item）
- 订单（Order）
- 送货地址（Shipping address）
- 付款（Payment）
- 账户（Account）

接下来分析一下有哪些动词：

- 添加购物车
- 提交订单
- 支付订单
- 取消订单

对于这些动词：添加、提交、支付、取消，我们还要标识出完成这些动作相对应的对象。

类之间的关系

类之间最常见关系有：

- 依赖（`uses-a`）

  一个类 A 的方法操作了另一个类 B 的对象，我们就说 A 依赖于 B。例如：订单对象 Order 需要访问账户 Account 对象来看该客户是否可参与活动、是否有会员折扣等，因此 Order 依赖于 Account。而商品（Item）对象和客户账户无关。

- 聚合（has-a）

  类 A 的对象包含类 B 的对象，我们称之为聚合。例如：一个 Order 订单对象包含一个或多个 Item 商品对象，因此它们是聚合关系。

- 继承（is-a）

  继承关系表示：类 A 从类 B 扩展而来，类 A 不但包含从类 B 继承的属性和方法，还拥有一些额外的属性和方法。

类的组成

Java 类主要由以下 7 部分组成：

- 包定义
- 包导入
- 域（Field）/成员变量（Member variable）
  - 实例变量，不以 static 修饰的变量
  - 类变量，以 static 修饰的变量
- 方法（method）：对象的行为
- 构造方法：用于对象的实例化
- 内部类（inner class）：在类中声明的其他类
- 代码块
  - 实例块
  - 静态块

下边我们来定义一个类，包含上述 5 个组成部分：

```
public class JavaClass {
    // 类变量
    private static String classField;

    // 实例变量
    private String field;

    // 实例代码块
    {
        field = "实例变量";
    }

    // 静态代码块
    static {
        classField = "类变量";
    }

    // 构造方法
    public JavaClass() {}

    // 方法
    public void method() {}
```

```
    // 内部类
    class InnerClass {
    }
}
```

域变量

在类中，方法之外定义的变量我们称为域变量，也叫成员变量，它分为两种：

- 实例变量，不以 static 修饰的变量
- 类变量，也可以称为静态变量或静态域，以 static 修饰的变量

而方法中的变量我们称为局部变量。

```
public class Variable {
    // 类变量 or 静态变量 or 静态域
    static int count = 0;

    // 实例变量
    int index = 0;

    public void method() {
        // 局部变量
        int i = 0;
    }
}
```

构造器

要使用对象，就必须先构造对象，并指定其初始状态。Java 使用构造器（constructor）构造新实例。

构造器是一种特殊的方法，用来构造并初始化对象。构造器的名字和类名相同，并且没有返回值。构造器是伴随 new 操作符的执行而被调用。

一个类可以有一个或多个构造器，它们的名字都和类名相同，但是参数个数或类型不一样（这种特征也叫重载 overloading）。

我们在编写一个类时，如果没有编写构造器，那么系统会提供一个无参数的构造器；而如果编写了构造器，则系统就不会再提供无参数的构造器。

通过 new 定义赋值的变量也叫对象变量，一个对象变量并没有实际包含一个对象，而仅仅引用一个对象。（Java 中所有变量的值都是存储在另一个地方的一个对象的引用，所有 Java 对象存储在堆中）。

当有多个构造器时，可以 `this` 来调用另一个构造器，并且调用语句只能出现在第一句。

```
public class ClassConstrutor {
    private int n1;
    private int n2;
```

```
    public ClassConstrutor() {}

    public ClassConstrutor(int n1) {
        this();
        this.n1 = n1;
    }

    public ClassConstrutor(int n1, int n2) {
        this(n1);
        this.n2 = n2;
    }
}
```

综上所述，Java 的类构造器有如下一些特征：

- 构造器和类同名
- 每个类可以有 1 个或多个构造器，但是参数个数或类型不一样
- 构造器没有返回值
- 构造器总是伴随 new 操作一起被调用
- 编写类时未编写构造器时，系统提供一个无参数的构造器

方法

方法即对象的行为，用于操作对象以及存取它们的实例域。方法也可以称为函数，一个方法的定义如下：

```
(访问权限修饰符) (修饰符) 返回值数据类型 方法名(形式参数列表) {
    语句;
    return (返回值);
}
```

如果方法没有返回值则使用 `void` 来定义返回值类型。

方法参数

Java 语言总是采用按值调用（call by value），方法得到的是所有参数值的一个拷贝，方法是不能修改传递给他的任何参数变量的内容。方法参数总共有两种类型：

1. 基本数据类型
2. 对象引用

一个方法不能修改一个基本数据类型的参数，也不能修改引用类型参数的指向，但是却可以修改引用类型参数指向的对象的值。

```
private void changePrimitiveValue(int n) {
    n = 200;
}

private void changeReferenceValue(StringBuilder sb) {
    sb.append("New");
```

```
    sb = new StringBuilder();
    sb.append("NewValue");
}

int n = 100;
System.out.println(n);
changePrimitiveValue(n);
System.out.println(n);

StringBuilder sb = new StringBuilder();
sb.append("this is a string.");
System.out.println(sb.toString());
changeReferenceValue(sb);
System.out.println(sb.toString());
```

隐式参数 `this`，Java 编译器在编译时会将对象自己放在第一个参数上，我们称之为隐式参数；第二个参数开始才是位于方法名后边括号中的参数，这些是显示参数。使用关键字 `this` 来表示第一个隐式参数，因此可以在方法内使用 `this.` 来访问对象自己的其他成员变量或方法。

当没有使用 `this.` 前缀来访问一个变量时，首先看是否存在该名字的局部变量，如果不存在则再去看该对象是否存在该名字的成员变量。

```
private int numb = 1;

private void print(int numb) {
    System.out.println(numb);
    System.out.println(this.numb);
}

@Test
public void testPrint() {
    this.print(2);
}
```

可变参数

当我们需要传同类型的一组参数，但是却不知道参数个数，这时可以使用可变参数语法：`参数类型... 变量名`。

一个方法只能有一个可变参数，并且可变参数只能是最后一个。可变参数变量我们可以当成是一个数组来使用。

```
private void print(int... numbs) {
    if (numbs == null) {
        System.out.println("param is null");
        return;
    }
    if (numbs.length == 0) {
        System.out.println("参数个数为0");
```

```
            return;
        }
        System.out.println("共" + numbs.length + "个参数");
        for (int numb : numbs) {
            System.out.println(numb);
        }
    }

    this.print();
    this.print(null);
    this.print(1);
    this.print(1, 2, 3);
```

方法重载

类构造器可以重载，普通方法一样可以重载。重载（overloading）指的是方法名相同，但是方法的参数类型或个数不同。不能根据返回值类型来区分重载，为什么？看如下代码定义了 2 个返回值类型不同但是方法名字和参数个数/类型相同的方法，如果调用的时候直接调用而没有将结果赋值给一个变量，那么编译器就不知道该调用哪一个方法了。

```
int max(int n1, int n2);
long max(int n1, int n2);
max(1, 3);
```

方法名和参数个数/类型，我们称为方法的签名（signature）。方法的签名必须唯一，方法返回值类型不是方法签名的一部分。

修饰符

Java 语言提供了一些修饰符，用来定义类、成员变量和方法，它放在语句的最前端，主要分为以下两类：

- 访问修饰符
- 非访问修饰符

访问权限控制

在 Java 中，通过访问控制修饰符来限定对类、成员变量和方法的访问，Java 支持 4 种访问权限：

修饰符	说明	当前类	同包	子类	不同包	备注
public	公开	√	√	√	√	可用于类、接口、成员变量、方法
protected	保护	√	√	√	×	可用于成员变量、方法
default	默认	√	√	×	×	可用于类、接口、成员变量、方法
private	私有	√	×	×	×	可用于成员变量、方法

static 修饰符

使用 `static` 修饰符定义的成员变量和方法称为静态域与静态方法，使用类名来访问，也可以使用类的实例变量名来访问。

类的所有实例对象共享一个变量，在静态方法中可以访问静态域变量，但是不能访问非静态的域变量。

```
public class ClassStaticPrefix {
    public static int count;
    public int index;

    public ClassStaticPrefix() {
        count++;
    }

    public static void setCount(int count) {
        ClassStaticPrefix.count = count;
    }

    public static void main(String[] args) {
        ClassStaticPrefix obj1 = new ClassStaticPrefix();
        System.out.println(obj1.count);
        ClassStaticPrefix obj2 = new ClassStaticPrefix();
        System.out.println(ClassStaticPrefix.count);
    }
}
```

final 修饰符

使用 `final` 定义的成员变量，在构建对象时必须进行初始化，并且在后面的操作中，不能够再对它进行修改。

final 修饰符通常用于基本类型域，或不可变类的域（如果类中的每个方法都不会改变其对象，这种类就是不可变的类）。

使用 final 定义的成员变量，我们也称为常量，常量通常全部大写，字母和字母之间使用下划线连接。

```
public class ClassFinalPrefix {
    public static final int MAX_INDEX = 1000;
    public final int index;

    public ClassFinalPrefix(int index) {
        this.index = index;
    }
}
```

代码块

定义一个类时，在类里面允许使用大括号括起来一段代码来对对象进行初始化，这个代码块也可以称为初始化块（initialization block）。一个类中可以包含多个代码块，当构造类的实例时，这些代码块按照从上到下的顺序进行执行。

使用 `static` 修饰的代码块，代码块内只能访问静态域，不能访问非静态域，我们也称之为静态初始化块。静态初始化块当类被第一次调用时执行一次。

```
public class CodeBlock {
    public static final int MAX_INDEX;
    public final int fromIndex;

    static {
        System.out.println("执行 static 代码块");
        MAX_INDEX = 1000;
    }

    {
        System.out.println("执行初始化代码块");
        this.fromIndex = 1;
    }

    public static void main(String[] args) {
        System.out.println(CodeBlock.MAX_INDEX);

        CodeBlock codeBlock = new CodeBlock();
        System.out.println(codeBlock.fromIndex);

        CodeBlock codeBlock2 = new CodeBlock();
        System.out.println(codeBlock2.fromIndex);
    }
}
```

包

Java 中使用包（package）来将类组织起来，包+类名必须具有唯一性。当一个类中需要访问其他包名下的类时，需要使用包+类名的方式，否则编译器不知道去哪里加载这个类。例如：

```
java.util.Date date = new java.util.Date();
```

这种写法显然很繁琐，为了简化写法，我们可以使用 `import` 关键字在类定义之前导入要使用的类，这样在类中需要使用的时候直接使用类名即可，而不用每次都加上包前缀。例如：

```
import java.util.Date;

Date date = new Date();
```

我们要导入同一个包下的多个类时，我们可以使用星号。例如：`import java.util.*` 表示导入 java.util 包下的所有类。注意：星号每次只能导入一个包下的所有类，而不能导入它的子包下的其它类。

对于静态域和静态方法，我们在导入的时候可以再加上 static 修饰符，表示静态导入，这样就可以直接使用域变量或者方法名，而不用加类名前缀。例如：

```
import static java.lang.System.out;

out.println("hello, world.");
```

类路径： Java 中，类是存储在文件系统的子目录中，而类存储在文件系统中的目录我们称为类路径，类的路径必须和包名匹配。

注意： javac 编译器总是在当前的目录中查找文件，但 JVM 仅在类路径（CLASSPATH）中有 “.” 目录时才查看当前目录。默认的类路径包含 “.” 目录，但如果自己设置了类路径而忘记加入 “.” 目录时，程序可以通过编译，但不能运行。

创建对象

我们通常使用 new 操作符来创建一个对象，在创建对象的同时，对域变量进行初始化，有如下 4 种初始化方式：

- 在声明域变量的同时对变量进行赋值
- 在初始化代码块中对域变量进行赋值
- 在构造器方法中对域变量进行赋值
- 对于未进行赋值的域变量，系统会自动初始化为默认值（基本类型为二进制的 0，引用类型为 null）

对静态域的初始化，有如下 2 中方式：

- 在声明静态域变量的同时对变量进行赋值
- 在静态初始化代码块中对静态域变量进行赋值

注意： 对于使用了 `final` 修饰符的(静态)域变量，必须进行显式的赋值初始化。

我们来看一个示例：

```
public class CreateObjectTest {
    static String className = "创建对象测试";
    static String staticFieldNotSet;
    final static String finalStaticField;

    int index = -1;

    {
        System.out.println("执行第一个初始化块: " + index);
        index = 1;
        System.out.println("  赋值后: " + index);
    }

    static {
        System.out.println("执行第一个静态初始化块");
        finalStaticField = "final 修饰的静态域必须进行显式的赋值初始化";
```

```
    }

    {
        System.out.println("执行第二个初始化块: " + index);
        index = 2;
        System.out.println("  赋值后: " + index);
    }

    static {
        System.out.println("执行第二个静态初始化块");
    }


    public CreateObjectTest() {
        System.out.println("执行构造器");

        this.index = 3;
        className = "创建对象测试 - 构造器赋值";
    }

    public static void main(String[] args) {
        new CreateObjectTest();
    }

    {
        System.out.println("执行第三个初始化块");
    }

    static {
        System.out.println("执行第三个静态初始化块");
    }

    {
        System.out.println("执行第四个初始化块");
    }

    static {
        System.out.println("执行第四个静态初始化块");
    }
}
```

通过示例执行结果，我们可以判断出通过 new 构造一个对象时，代码执行顺序如下：

- 按从上到下的顺序执行静态域变量的声明以及赋值语句、静态初始化代码块
- 按从上到下的顺序执行域变量的声明以及赋值语句、初始化代码块
- 执行构造器方法

对象析构和 finalize 方法

Java 语言有自动垃圾回收器，不需要人工回收内存，所以 Java 不支持析构器。

我们可以为任何一个类添加 finalize 方法，该方法在垃圾回收器清除对象之前调用。因为不知道垃圾回收器什么时候进行回收，因此在实际应用中不要依赖使用 finalize 方法回收任何短缺的资源。

如果确实需要自己进行回收处理的，通常使用 `Runtime.getRuntime().addShutdownHook()` 方法添加关闭钩（shutdown hook）。

常见面试问题

类和对象的联系和区别

类是构造对象的模版或者蓝图，是现实世界中对某一事物的描述，它定义了一类对象的状态和行为；而对象是类的一个实例，有状态和行为。比如：人是一个类，而每一个具体的人张三、李四等就是一个个对象，也即是人这个类的一个个实例。

类的组成

一个类主要由下边 7 部分组成：

- 包定义
- 包导入
- 域（Field）/成员变量（Member variable）
- 方法（method）：对象的行为
- 构造方法：用于对象的实例化
- 内部类（inner class）：在类中声明的其他类
- 代码块

可以根据前边课件中的内容在分别扩展阐述一下

什么是重载、方法签名

重载（overloading）只是一个类中可以多个名字相同的方法，但这些方法的参数个数或类型不同。方法签名是指方法名和参数列表，而方法返回值类型不是方法签名的一部分，一个类中的方法签名必须唯一。

类的构造器方法有哪些特征?

- 构造器方法名和类名相同，并且没有返回值（void 也不能加）
- 构造器方法仅伴随 new 操作符而被调用，不能被继承、覆盖。
- 类定义时，如果没有编写构造器方法则系统会提供一个无参数的默认构造器
- 构造器方法可以被重载

在类的实例中为什么可以使用 this 来表示对象自身?

Java 编译器在编译时会将对象自身当作方法的第一个参数传递，而方法括号内的参数放在第二个参数及以后。这第一个参数我们称为隐式参数，用 `this` 表示，因此我们可以在方法内使用 this 来当作对象自身。

课后练习

1. 自己编写练习一下课程中的示例和练习。