chapter 3 面向对象的程序设计

> OOP
>
> Object Oriented Programming

1. 类、域、方法、对象
   - 类 `class` / `type`
     - 抽象的概念
     - 对世界上万事万物的一种抽象和概括
     - 分类的依据
       - 有相同的特征或属性
       - 有相同的行为或功能
     - 类 = 属性 + 功能
     - `类 = 域 + 方法`
     - 类是创建对象的模板
     - 类的定义

       ```
       访问限定修饰符 class 类名
       [<类型参数列表 extends 父类型名 & 父类型名 & ...>]
       [extends 父类名]
       [implents 接口名列表] {
           // 类体
       }
       ```

   - （成员）域 `field`
     - 一类事物共有的特征或属性
     - 域和方法常常紧密关联
     - 域的定义

       > 等同于变量的定义
       >
       > 直接定义在类体中

       ```
       访问限定修饰符 类型名 域名;
       ```

   - 成员方法 `method`
     - 一类事物共有的行为或功能
     - 方法的定义

       ```
       访问限定修饰符 返回类型 方法名([参数列表]) [throws 受检异常类列表] {
           // 方法体
       ```

```
}
```

- 参数列表

> 用 `,` 间隔开多个参数

- 形式参数 `formal parameter`
- 实际参数 `actual parameter`
  - `void` 方法没有返回值 `return;`
  - `return` 返回控制
  - 构造方法（构造器） `constructor`
  - 与类同名
  - 没有返回类型
  - 初始化成员域
  - 类的默认构造方法
    - 没有参数
    - 没有方法体

> 初始化类的域为默认值
> 1. 布尔类型默认值 `false`
> 2. 基本数值类型 `0` 或 `0.0`
> 3. 引用数据类型 `null`

  - 如果显式声明的构造方法，默认构造方法将不存在
  - `this` 指代当前类
  - 构造器块

```
{
// ...
}
```

- 对象/实例/实例对象 `instance`
  - 具体的
  - 类来生成
  - `new` 创建或生成一个类的对象，总是调用类的构造方法
  - `.` 调用域或方法

2. 继承 `inheritance`
   - `extends` 扩展
   - `子类`（`sub class`）`extends`（`继承`）`父类`（`super class`）
     - 子类拥有父类的域和方法

> 注意父类成员的访问限定修饰符
> 1. 父类的 `default` 成员，只有同包子类可继承
> 2. 父类的 `private` 成员，不能继承

    - 子类可以有自己的域和方法
- `super` 指代父类
- 父类有无参构造方法
    - 子类可以隐式调用父类的无参构造方法
- 父类没有无参构造方法
    - 子类必须显式调用父类的有参构造方法
    - 子类显式调用父类构造方法时，必须在其构造方法的第一行
    - 类加载顺序

> 父类静态代码块内容 - 子类静态代码块内容 - 父类构造器块 - 父类构造器 - 子类构造器块 - 子类构造器

```
public class Parent {

  {
      System.out.println("parent constructor block");
  }

  static {
      System.out.println("parent static block");
  }

  public Parent() {
      System.out.println("parent constructor");
  }
}

class Child extends Parent {

  {
      System.out.println("child constructor block");
  }

  static {
      System.out.println("child static block");
  }

  public Child() {
      System.out.println("child constructor");
  }
}

class Test {
  public static void main(String[] args) {
      new Child();
  }
}

/*
```

```
parent static block
child static block
parent constructor block
parent constructor
child constructor block
child constructor
*/
```

- Java 语言里，类是单继承的（接口可以多实现）
- `instanceof` 判断对象是否是类的实例

3. 多态 `polymorphism`
   - 静态多态性
     - 同一个类内部（也可以发生于子类和父类之间）
     - 同名方法的重载 `overload`
     - 参数不同
       - 类型不一样
       - 数量不一样
       - 顺序不一样
     - 类的构造方法之间都是重载
   - 动态多态性
     - 子类和父类之间
     - 子类 重写 `overwrite` / 覆盖 `override` 了父类的方法（有相同的声明，访问权限可扩大）
     - 子类是否调用、子类在何处调用父类的方法根据方法的定义和需求确定
4. 包 `package`
   - 管理代码的目录结构
   - 通常都是公司或组织或学校域名的反写
   - 包名都是小写字母
   - 所有的类都要放在包里
   - `package` 打包语句，必须是类里的第一行代码
   - `import` 导包语句
   - Java 的包
     - java.lang Java 的语言包，使用这个包的类不需要导入 `common sense`
   - `FQN` - Fully Qualified Name 全限定名
5. 封装 `encapsulation`
   - 访问限定修饰符
     - 类的访问限定修饰符
       - `default class` 只有同包的其他类可访问
       - `public class` 外包的类可访问，需导入，常用
     - 类成员的访问限定修饰符
       - `private` 私有的，只有同一个类内部可访问
       - `default` 默认的，只有同一个包内部可访问
       - `protected` 受保护的，外包中子类的实例对象可访问
       - `public` 共有的，外包可访问

- 成员域一般都是私有的

> 隐藏内部的数据

- 成员方法一般都是公有的

> 为外部提供访问的接口

- 封装的含义

6. `abstract`
   - 抽象：抽取“像”的部分
   - 可以修饰类和方法
   - 抽象类
     - 抽象的类不能实例化
     - 抽象的类是用来被扩展的
     - 抽象类的子类必须实现抽象类中所有的抽象方法
   - 抽象方法
     - 抽象的方法没有实现
     - 抽象的方法必须声明在抽象类中
     - 在抽象类的子类中被实现
7. `final`
   - 终态
   - 可以修饰类、域和方法
   - 终态的类不能再被子类化
   - 终态的域
     - 只能在声明时或构造方法中被初始化
     - 初始化之后值不能再被修改
   - 终态的方法不能在子类中被重写
   - 静态并终态的域
     - 只能在声明时初始化
     - 初始化之后值不能再被修改
     - 常量，都是大写字母，单词之间用下划线 `_` 分隔
8. `static`
   - 静态
   - 可以修饰域和方法
   - 静态的成员隶属于`类对象`
   - 静态方法中只能直接引用静态成员
   - 静态方法中不能使用 `this` 和 `super`
   - 方法中不能定义静态变量
   - 静态导入（JDK 1.5+）

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

   - 静态块 `static block`

> 静态块在类加载时自动执行一次, 之后不再执行

`java`
`static {`
`// ...`
`}`

9. 接口 `interface`
    - 与类处于同一个级别
    - 接口的定义

```
访问限定修饰符 interface 接口名
[<类型参数列表 extends 父类型名 & 父类型名 & ...>]
[extends 父接口名列表] {
    // 接口体
}
```

    - 接口可以定义域和方法
        - 接口的域都是公有常量
        - 接口的方法都是公有抽象方法
    - 接口没有构造方法，不能实例化
    - 接口是用来被实现的 `implements`
    - 类实现接口必须实现接口的所有抽象方法
    - Java 中，一个类可以实现多个接口
    - 抽象类与接口之间的联系和区别
        - 相同点
            - 都不能实例化
            - 都可以定义抽象方法
            - 对于他们的子类 / 实现类做了限制和约束
        - 不同点
            - 接口的域都是公有常量，抽象类都可以
            - 接口没有构造方法，抽象类有
            - 接口不能定义非抽象方法，抽象类可以
            - Java 语言的类只能继承一个抽象类，但可以实现多个接口
    - 接口本身可以扩展 `extends` 多个父接口
10. 变量作用域范围
    - 变量（方法）产生作用的有效范围
        - 类作用域范围
            - 类的起始 `{` 到类的终止 `}`
            - 类的域和方法
        - 块 `block` 作用域范围
            - 从变量声明之处，到当前块结束之处
            - 方法中的局部变量 `local variable`，方法的参数，循环的变量
    - 方法内的局部变量可以覆盖同名的域
11. 参数传递方式

- 值传递
  - 传递参数的值
  - 方法的参数是基本数据类型 `primitives`
  - 方法中的改变不影响实际参数 `references`
- “引用传递”
  - 传递参数的地址（也是值传递）

    > Is Java “pass-by-reference” or “pass-by-value”?

  - 方法的参数是引用数据类型
  - 方法中的改变会影响实际参数
  - 注意：String类型以及基本数据类型的封装类是特例（还是值传递）
    - String `immutable`
    - primitive Wrapper