面向对象期末复习

一、类与对象

1. 类：
   - 一个模板，描述一类对象的行为和状态
2. 对象
   - 类的实例，有自己的参数，可以调用类中的方法
3. 实例变量，又称成员变量
   - 每个对象的属性
   - 每个对象的实例变量的值可以不同
   - 调用格式：`<对象名>.<变量名>`
   - has-a relation：一个类可以把其他类的对象作为实例变量
4. 静态变量
   - 静态变量由这个类的所有成员共享
   - 调用格式：`<类名>.<静态变量名>` 或 `<任意对象名>.<静态变量名>`
5. 实例方法，又称成员方法：
   - 调用格式：`<对象名>.<方法名>`，同类中可省略`<对象名>`。
   - 必须在创建对象后才能使用
   - 每个对象的实例方法运行结果可能不同
6. 静态方法，又称类方法：
   - 调用格式：`<类名>.<静态方法名>`，同类中可省略`<类名>`。
   - 可以在没有创建对象时调用
   - 主方法为静态方法
   - 静态方法内部不可调用实例变量和实例方法，因此不能出现任何 `this` `super` 关键字

	可调用静态变量、方法	**可调用实例变量、方法**
静态方法	√	
实例方法	√	√

7. 构造器
   - 可以重载：一个类中可以有多个方法签名不同的构造器
   - 默认构造器：
     - 没有在类中显式创建构造器时，编译器会自动加一个默认构造器
     - 默认构造器无传入参数
       将基本类型初始化为默认值
       将引用类型初始化为`null`
     - 只要类中有自己写的构造器，无论是有参还是无参，都不会再生成默认构造器。
   - 枚举里的构造器必须 private 或者 no-modifier

二、继承

1. is-a 关系：

- `D extends A` ⟺ `D is a A`

2. 子类可以重写父类中的方法
3. 范围修饰符
   - 子类继承父类的修饰符后可以改大，不能改小

	本类中	同一个包下	这个类的子类	所有地方
public	√	√	√	√
protected	√	√	√	
no modifier	√	√	同包中子类可访问 不同包子类不可访问	
private	√			

4. `this`关键字
   - this只能表示本类的对象，方法被谁调用，`this`就表示谁
   - 传入参数与本类实例变量有相同名字时，需要用`this.<变量名>`表示实例变量
   - 构造器中使用`this(...)`来调用本类的另一个构造器，且该`this`语句必须是这个构造器的第一条语句
5. `super`关键字
   - 调用父类中的方法：`super.<方法名>`
   - 调用父类构造器：
     - 创建一个子类对象时，必须先创建一个父类对象
     - 如果没有显式写`super`，则默认调用父类的无参构造器
     - 显式写：`super(...)`调用父类构造器，且该语句必须是子类构造器的第一条语句
   - constructor chain：
     - Object ← A ← B ← D ← E
     - 所有类都是`Object`的子类
     - 想要新建子类对象，必须先新建一个父类对象
     - 因此无论想要新建哪个类的对象，最开始被创建的一定是`Object`对象

三、多态

1. 三个存在条件
   - 子类继承父类（也可以是接口、抽象类）
   - 子类重写父类中的方法
   - 父类引用指向子类对象
   - 例：`Fu fu = new Zi();`
     `Fu`被称为“引用类型”
     `fu`被称为“引用”
     `new Zi()`被称为“对象”
2. 动静绑定：
   - 动态绑定：编译阶段不知道调用哪个方法，运行时才知道
   - 静态绑定：在编译阶段就知道调用什么方法/变量
3. 实例方法：动态绑定
   先去父类中检查是否有该实例方法。
   如果父类中没有此方法：编译错误

如果父类中有此方法：调用子类中的该方法。
如果子类重写，则调用重写后的方法。
如果子类没有重写，则调用从父类继承来的方法。

4. 静态方法：静态绑定
   - `final` `static` `private` 方法都是静态绑定，始终调用等号左边的父类中的方法。
5. 实例、静态变量
   - 由等号左边的父类决定
   - 继承中子类可以继承父类中的变量，但多态中只看父类
6. 向下转型
   - `instanceof` 关键字：用于检查一个引用是否是某个类的对象
     语法：`if (obj instanceof Fu) {...}`
     如果 `obj` 是 `Fu` 的对象则为 `true`，否则为 `false`
   - 先用`instanceof`检查一个引用是否指向正确的对象，再向下转型

四、`final`关键字

1. `final`变量：
   - 必须被赋值
   - 被赋值后无法更改
   - 要么声明变量时赋值，要么在所有构造器里赋值
2. `final`方法：父类中的 final 方法无法在子类中被重写
3. `final`类
   - `final` 类无法被继承
   - 一旦写好这个类后就无法改变

五、抽象类

1. 抽象类不能用于实例化对象
2. 一般使用方法：子类继承抽象类，重写所有抽象方法。用抽象类的非抽象子类实例化对象。
3. 抽象方法不能为 private
4. 有抽象方法一定是抽象类，抽象类不一定有抽象方法。
5. 构造器和静态方法不能声明为 abstract
6. 抽象类可以作为多态中的父类
7. 可通过`<抽象类名><静态方法名>`调用抽象类中的静态方法
8. 抽象类不能用 final 声明

六、接口

1. 变量必须是 public static final，必须被初始化
2. 方法必须是 public abstract，
3. 接口没有构造器，不能被实例化
4. 一个类可以 implements 任意多个接口，只能 extends 一个父类
5. 接口可以作为多态中的父类
6. 接口里可以有具象方法，但是没学过，考试不考。

七、其他

1. package 语句：放在 .java 文件首行，用于声明文件位置
2. 枚举：一种很高级的常量
    - 每个变量都是 final 和 static
    - 枚举对象的构造器必须为 private 或者 no modifier
    - 枚举是一种类，可以重载构造器
3. 泛型
    - 3.1：泛型参数
      需要满足变量命名规则（字母数字下划线美刀），理论上可以用任何合法字符串作为变量。但是有几个约定俗成的名字：
        - T：引用数据类型，不能是八大基本类型
        - E：集合中存放的元素
    - 泛型参数是数据类型的代称，运行时会变成传入的数据类型。
    - 只能表示引用类型，无法表示基本数据类型。
    - 3.2：泛型方法
        - 命名规范及举例

方法声明	是否合法	说明
`public static <T> void printArray(T[] array)`	合法	所有关键字都出现
`<T> void printArray(T[]) array`	合法	默认范围修饰符，实例方法
`<T> static printArray(T[] array)`	不合法	`<T>` 必须在 `static` 和范围修饰符 之后
`<T> void printArray(int[] array)`	合法	可以声明泛型参数但不使用
`<K,V> void printPair(K[] arrayK, V arrayV)`	合法	多个泛型参数用逗号分隔
`<hdfjka> void printArray(hdfjka[] array)`	合法	泛型参数命名符合标识符规则

        - 下面的代码中，传进来的数组是什么引用类型，执行的时候 T 就变成什么类型

```
public static <T> void printArray(T[] array) {
    for (T element : array)
        System.out.printf("%s ", element);
    System.out.println();
}
```

        - Bounded Type Parameter
            - 尖括号中为标记符的限制条件，只有符合条件的类型才能替代标识符
            - 尖括号里的 `extends` 实际上表示 `extends 或 implements`
        - 下面的代码中，只有使用了 Comparable 接口的引用类型才能放入 T

```
public static <T extends Comparable<T>> void printArray(T[] array) {
    for (T element : array)
        System.out.printf("%s ", element);
    System.out.println();
}
```

    - 3.3：泛型类

- 在类名后面添加了类型参数声明
- 写类中的代码时可以用 T

```
public class Box<T> {

    private T t;

    public void add(T t) {
        this.t = t;
    }

    public T get() {
        return t;
    }

}
```

- 3.4：向上转型：

  假设 G 为泛型类，`Child extends Father`。**G<Child> 不是 G<Father> 的子类**.

```
//可以正常运行
String s = "1234";
Object o = s;

//可以正常运行
ArrayList<String> list1 = new ArrayList<>();
List<String> list2 = list1;

//报错
List<String> strList = new ArrayList<>();
List<Object> objList = strList;
//strList 不是 objList 的子类
```