2.1 类和对象

2.1.2 面向对象编程的概念

先以面向对象的思想进行分析，然后使用面向对象的编程语言进行表达。

2.1.3 类和对象的概念

对象：在java语言中对象体现为内存空间的一块存储区域。

类：在java语言中体现为一种引用数据类型，里面包含成员变量和成员方法。

2.1.6 执行流程和内存分析

当运行`java xxxx`时，java会把编译后的字节码文件加载到内存当中（准确来说是加载到`方法区`）。

main方法会在`栈`中开辟一块存储空间。new关键字会在`堆`中开辟一块存储空间。

2.1.7 类的定义

```
class 类名{
  返回值类型 成员方法名(形参列表){
    成员方法体;
  }
}
```

形参列表用于定义方法，将方法体外的数据带到方法体内。

实参列表是对形参列表初始化。

2.1.15 可变长参数

看作一维数组使用即可，因为参数类型不能变

```
class 类名{
  返回值类型 成员方法名(参数类型... 参数名){
    成员方法体;
  }
}
```

2.1.22 参数传递的注意事项

JVM会为每个方法的调用在栈中分配一个对应的空间，这个空间称为栈帧。一个栈帧对应一个正在调用的方法，栈帧存放了该方法的参数，局部变量等。

基本数据类型传的是值；引用数据类型传的是引用地址；

2.2 方法和封装

2.2.4 重载的概念和体现形式

1. 方法名相同
2. 参数列表不同
    - 体现在参数的类型，顺序，数量上
    - 和参数的变量名无关
3. 返回类型无法区分方法的重载

2.2.8 this关键字的基本概念

1. 构造方法中的`this`代表当前正在构造的对象
2. 成员方法中的`this`代表当前正在调用的对象
3. `this`的本质是当前类型的引用变量

2.2.21 封装的实现

1. 私有化成员变量，使用`private`修饰，该成员变量只能在当前类的内部使用
2. 提供公有的`get`和`set`方法，并进行合法值的判断
3. 在公有的的构造方法中调用`set`方法进行合理值的判断

2.2.23 JavaBean

1. JavaBean是一种Java语言写成的可重用组件，其他Java类可以通过反射机制发现和操作这些JavaBean的属性
2. JavaBean本质上符合以下标准的Java类
    - 类是公有的
    - 有一个无参的公有的构造器
    - 有属性，且具有对应的`get`和`set`方法

2.3 static关键字和继承

2.3.2 static关键字的基本概念

1. 用`static`关键字修饰成员变量表示静态的含义
2. 此时成员变量/方法由对象层面提升为类层面，也就是整个类只有一份并被所有对象共享

3. 该成员变量/方法随着类的加载准备就绪，与是否创建对象无关。

2.3.3 static关键字的使用

1. 在非静态成员方法中既能访问非静态的成员又能访问静态的成员
2. 在静态成员方法中只能访问静态成员不能直接访问非静态成员，但可以通过创建引用

2.3.4 构造块和静态代码块

构造块：类内方法外用{ }括起来的代码

1. 构造块是先于构造方法体执行的
2. 当需要在执行构造方法体之前做一些准备工作，则将准备工作的相关代码放到构造块当中，比如对成员变量统一的初始化
3. 构造块随着对象的创建

```
public class XXX{
  {
    //构造块
  }

  //构造方法
  XXX(){

  }
}
```

静态代码块：

1. 先于构造块执行
2. 当需要在构造块之前随着类的加载做一些准备工作时，则将准备工作的代码放到静态构造块当中，比如数据库的驱动包等
3. 静态代码块随着类的加载而准备就绪

```
public class XXX{
  static {
    //静态代码块
  }

  //构造方法
  XXX(){

  }
}
```

2.3.6 Singleton

```
public class Singleton(){
  //因为构造方法是私有的，只能从类内构建对象，而且注意要加static。为了防止用户通过
`Singleton.singleton = null`修改对象，我们还       要在变量前加private
  private static Singleton singleton = new Singleton();

  //私有化构造方法，防止从外面构造对象
  private Singleton(){}

  //向外提供接口
  public static Singleton getInstance(){
    return singleton
  }
}
```

2.3.7 单例设计模式

1. 私有化构造方法，使用`private`关键字去修饰
2. 声明本类类型的引用指向本类类型的对象，并使用`private static`关键字去修饰
3. 提供公有的get方法负责把对象返回出去，并使用`public static`关键字去修饰

单例设计模式有两种设计方法：

1. 懒汉式

   ```
   public class Singleton(){

     private static Singleton singleton = null;

     private Singleton(){}

     public static Singleton getInstance(){
       if(singleton == null){
         singleton = new Singleton();
       }
       return singleton;
     }
   }
   ```

2. 饿汉式

```
public class Singleton(){

  private static Singleton singleton = new Singleton();

  private Singleton(){}

  public static Singleton getInstance(){
    return singleton
  }
}
```

2.3.12 继承的特点

1. 子类不能继承*父类的构造方法（构造方法名要和类名相同）*和私有成员方法，但私有成员变量可以被继承只是不能直接访问
2. 不论使用任何方式构造子类的对象时，都会自动调用父类的无参构造方法，来初始化从父类中继承的成员变量，相当于在第一行加上`super()`。**必须是第一行**。

注意⚠️：

1. 构造方法不能被继承，只是被调用
2. 每个子类的构造方法都会自动调用父类的无参构造方法。如果父类没有无参构造方法，子类的构造方法就必须调用父类的有参构造方法，否则会报错
3. `super()`必须在第一行。如果没写，系统会自动添加父类的无参构造方法

2.3.16 方法重写的原则

1. 方法名，形参列表，返回值相同
2. 访问权限不能变小，可以相等或变大
3. 不能抛出更大的异常

2.3.23 权限修饰符和包的定义

修饰符	本类	同一个包中的类	子类	其他类
public	可以访问	可以访问	可以访问	可以访问
protected	可以访问	可以访问	可以访问	不能访问
默认	可以访问	可以访问	不能访问	不能访问
private	可以访问	不能访问	不能访问	不能访问

1. public修饰的可以在任何位置使用
2. private修饰的只能在本类使用

2.3.24 final关键字

1. final关键字修饰的类，该类不能被继承
2. final关键字修饰的方法不能被重写，但可以被继承
3. final关键字修饰变量，变量必须初始化，不能被重写。通常用`public static final`关键字来修饰常量

2.4 多态和特殊类

2.4.1 多态的概念和语法格式

多态主要指同一种事物表现出的多种形态。

父类的引用指向子类的对象。`父类类型 引用变量名 = new 子类类型()`

1. 父类的引用可以调用父类独有的方法
2. 父类的引用不可以调用子类独有的方法
3. 如果子类重写父类的方法，编译期间会调用父类的，运行阶段会调用子类的（动态绑定）
4. 对于静态方法，调用的都是父类的

注意⚠️：

可以用哪些方法是由父类决定的，具体用哪个方法是由子类决定的。

2.4.8 多态的实际意义

屏蔽了不同子类的差异性实现通用的编程带来不同的效果。可以让公共父类作为方法的形参

```
public class Shape(){
```

```
  public void show(){

  }
}

public class Circle extends Shape(){
  @Override
  public void show(){

  }
}

public class Rectangle extends Shape(){
  @Override
  public void show(){

  }
}

public class ShapeTest(){

  public static void draw(Shape s){
    s.show();
  }

  public static void main(String[] args){
    ShapeTest.draw(new Circle());
    ShapeTest.draw(new Rectangle());
  }
}
```

2.4.9 抽象方法和抽象类的概念

抽象方法：

1. 不能具体实现的方法（没有方法体）
2. 用abstract关键字修饰
3. `访问权限 abstract 返回值类型 方法名();`

抽象类：

1. 不能具体实例化的类（不能创建对象）
    - 可以有成员变量，成员方法，构造方法
    - 可以有抽象方法也可以有抽象方法
    - 有抽象方法的类必须是抽象类
2. 用abstract关键字修饰

2.4.10 抽象类的实际意义

抽象类的意义在于**被继承**

一个类继承抽象类，要么重写抽象方法，要么把自身变成抽象类

2.4.13 笔试考点

注意⚠️：

1. private不能修饰abstract
2. final不能修饰abstract
3. static不能修饰abstract

2.4.14 接口的基本概念

1. 所有方法都是抽象的
    - 从java8开始，可以用default关键字来修饰非抽象方法，防止牵一发而动全身。可以重写也可以不用重写

      ```
      public interface hunter{
        //
        public default void hunt(){

        }
      }
      ```

    - 也可以增加静态方法
    - 从java9开始，在interface中可以使用private方法。
2. 只能有常量
3. 用interface修饰
4. 不能实例化

2.4.15 接口的实际意义

弥补不支持多继承的不足

2.4.16 类和接口的关系

类和接口之间的关系

名称	关键字	关系
类和类之间的关系	使用extends关键字表达继承关系	支持单继承
类和接口之间的关系	使用implements关键字表达实现关系	支持多实现
接口和接口之间的关系	使用extends关键字表达继承关系	支持多继承

2.4.17 抽象类和接口的区别

抽象类	接口
class	interface
extends	implements
单继承	多实现
可以有构造方法	没有构造方法
可以有成员变量	没有成员变量
可以有成员方法	只能有抽象方法，新特性除外

2.5 特殊类

2.5.1 内部类的基本概念

一个类（内部类）写在一个类（外部类）的内部并为这个类单独服务。内部类可以访问外部类的私有成员。

普通内部类：直接将一个类的定义放在另一个类的类体中

静态内部类：用static关键字修饰的内部类，隶属于类层级

局部内部类：直接将类的定义放在方法体时

匿名内部类：

2.5.2 普通内部类

```
访问修饰符 class Outer{
  访问修饰符 class Inner{

  }
}
```

2.5.3 普通内部类的使用

```
Outer outer = new Outer();
Outer.Inner inner = outer.new Inner();
```

2.5.4 静态内部类

```
访问修饰符 class Outer{
  访问修饰符 static class Inner{

  }
}
```

```
Outer.Inner inner = new Outer.Inner();
```

2.5.6 局部内部类

定义在方法体中，只在方法体中有效

```
访问修饰符 class Outer{
  访问修饰符 返回值类型 成员方法名(形参列表){
    class Inner{

    }
  }
}
```

2.5.7 局部内部类的使用方法

1. 只能在方法内使用
2. 可以在方法内直接创建对象
3. 不能使用访问修饰符和static
4. 可以使用外部方法的局部变量，但必须是final或理解成final

2.5.8 回调模式的概念和编程

如果一个方法的参数是一个接口类型，则在调用该方法是，需要创建并传递一个实现此接口类型的对象。在方法运行时会调用到参数中所实现的方法。

接口类型的引用指向实现类的对象。

2.5.9 匿名内部类的使用

在接口类型作为形参的时使用匿名内部类

格式：`接口/父类类型 引用变量名 = new 接口/父类类型(){ 方法的重写 }`

2.5.10 枚举类的概念

```
public class Direction{
  private final String desc;

  public static final Direction UP = new Direction("向上");
  public static final Direction DOWN = new Direction("向下");
  public static final Direction LEFT = new Direction("向左");
  public static final Direction RIGHT = new Direction("向右");

  private Direction(String desc){
    this.desc = desc;
  }
}
```

2.5.11 枚举类型的定义

`enum`关键字来修饰枚举类型。

```
public enum Direction{
  //枚举类型要求枚举值必须放在枚举类型的最前面，默认使用public static final来修饰
  UP("向上"),DOWN("向下"),LEFT("向左"),RIGHT("向右");

  private final String desc;

  //可以自定义构造方法，但必须是private
  private Direction(String desc){
    this.desc = desc;
  }
}
```

2.5.13 Enum类的概念和常用方法

所有枚举都继承自`java.lang.Enum`

1. static T[] values(): 获取所有对象
2. String toString(): 获取当前对象名称
3. int ordinal(): 获取索引位置
4. static T valueof(String str): 将字符串转换成枚举对象
5. int compareTo(E o): 比较两个枚举对象定义时的顺序

2.5.15 枚举类型实现接口的方式

可以只实现一次，所有对象共同使用；也可以每个对象单独实现

2.5.17 注解的定义和使用

Annotation -- 标签，对代码特殊说明，检查

```
访问修饰符 @interface 注解名称{
  注解成员;
}
```

1. 继承`java.lang.annotation.Annotation`
2. 注解中只有成员变量，没有成员方法。成员变量以“无形参的方法”形式来声明，方法名为成员变量名，其返回值为成员变量的类型

   ```
   //类型只能为八种基本数据类型，String类型，Class类型，enum类型以及Annotation类型
   public String value();
   ```

2.5.18 元注解的概念和@Retention的使用

元注解用于修饰其他注解

主要有：

1. @Retention：应用到一个注解上用于说明该注解的生命周期
   - RetentionPolicy.SOURCE：在源码中有效
   - RetentionPolicy.CLASS：在字节码文件中有效，默认方式
   - RetentionPolicy.RUNTIME：在运行时有效
2. @Documented
3. @Target
4. @Inherited
5. @Repeatable

2.5.19 @Documented

使用javadoc工具可以从源码中提取类，方法，成员等注释形成一个帮助文档，但该工具是不包括注解内容的

@Documented用于指定该注解将被javadoc工具提取成文档，必须是运行时有效（RetentionPolicy.RUNTIME）

2.5.20 @Target和@Inherited

@Target用于指定被修饰的注解能用于哪些元素的修饰

ElementType.ANNOTATION_TYPE	可以给一个注解进行注解
ElementType.CONSTRUCTOR	可以给构造方法进行注解
ElementType.FIELD	可以给属性进行注解
ElementType.LOCAL_VARIABLE	可以给局部变量进行注解
ElementType.METHOD	可以给方法进行注解
ElementType.PACKAGE	可以给一个包进行注解
ElementType.PARAMETER	可以给一个方法内的参数进行注解
ElementType.TYPE	可以给类型进行注解，比如类、接口、枚举

@Inherited

2.5.21 @Repeatable

表示自然可重复的含义

2.5.22 常见的预制注解

预制注解就是java语言自身提供的注解

@author	标明开发该类模块的作者，多个作者之间使用,分割
@version	标明该类模块的版本
@see	参考转向，也就是相关主题
@since	从哪个版本开始增加的
@param	对方法中某参数的说明，如果没有参数就不能写
@return	对方法返回值的说明，如果方法的返回值类型是void就不能写
@exception	对方法可能抛出的异常进行说明

@Override	限定重写父类方法, 该注解只能用于方法
@Deprecated	用于表示所修饰的元素(类, 方法等)已过时
@SuppressWarnings	抑制编译器警告

2.6 作业

END