一、问：请简述子类不可以继承父类的哪些成员？使用继承有什么好处？多重继承关系中属性的初始化顺序是怎样的？

答：子类不可以继承父类的私有成员（private），而且如果父类的构造函数是私有的则子类也无法继承。使用继承的好处包括代码重用、减少重复代码和提高代码的可维护性。在多重继承关系中，属性的初始化顺序是从最顶层的父类开始依次向下初始化。

二、问：什么是方法重载？Override 和 Overload 的区别？

答：方法重载指在一个类中，方法名相同，参数列表不同的情况。Override 指子类重新定义父类的方法，而 Overload 指一个类中存在多个方法名相同但参数列表不同的方法。

三、问：简述 this 关键字的作用、super 关键字的作用。

答：在 Java 中，this 关键字用于引用当前对象，super 关键字用于访问父类的成员或构造方法。

四、问：什么情况下需要使用 static 关键字？

静态成员变量和方法属于类而不是对象，当需要在多个对象之间共享数据时，需要使用 static 关键字。

五、问：请简述什么是多态？实现多态三要素？instanceof 关键词的作用？在 Java 中如何实现多态？

答：多态是指同一操作作用于不同的对象上会产生不同的行为。实现多态的三要素包括继承、重写父类方法和父类引用指向子类对象。instanceof 关键词用于判断一个对象是否是某个类的实例。在 Java 中，多态可以通过父类引用指向子类对象来实现。

六、问：简述 final 关键字在类、方法和变量上的用途。

答：final 关键字在类上表示该类不可被继承，方法上表示该方法不可被重写，变量上表示该变量为常量，数值不可改变。

七、问：如何实现对象的封装？什么是装箱拆箱？

答：对象的封装是指将对象的状态和行为封装在一起，并对外部隐藏对象的内部细节。装箱指将基本数据类型转换为对应的包装类，拆箱指将包装类转换为基本数据类型。

八、问：请简述线程的五个状态？线程对象调用 start()方法和调用 run()方法的区别?

线程的五个状态包括新建状态、就绪状态、运行状态、阻塞状态和死亡状态。线程对象调用 start()方法会创建一个新的线程并执行相应的 run()方法，而直接调用 run()方法实际上是在当前线程中执行 run()方法，不会创建新的线程。

九、问：什么是面向对象编程的主要特点？简述类和对象的关系和区别?

面向对象编程的主要特点包括封装、继承和多态。类是对象的抽象，对象是类的实例化。类是对对象的抽象描述，而对象是类的实例。

十、问：集合类的理解 ArrayList 和 LinkedList 的联系和区别
答：ArrayList 和 LinkedList 都是集合类，区别在于 ArrayList 基于数组实现，支持随机访问，而 LinkedList 基于链表实现，支持快速插入和删除。

十一、问：在 Java 中，构造方法的作用是什么？
答：构造方法用于初始化对象，当创建对象时会自动调用构造方法进行初始化。

十二、问：什么是继承？有什么好处？
答：继承是指一个类获取另一个类的属性和方法的过程。继承的好处包括代码重用、提高代码的可维护性和扩展性。

十三、问：简述抽象类和接口的区别。
答：抽象类可以包含抽象方法和非抽象方法，而接口只能包含抽象方法。一个类只能继承一个抽象类，但可以实现多个接口。

十四、问：面向对象设计中的“单一职责原则”是什么意思？
答：单一职责原则指一个类只应该有一个引起它变化的原因，即一个类只负责一项职责。

十五、问：请举例说明什么是异常，Java 中，如何进行异常处理？finally 关键字的作用？
答：异常是程序在运行期间发生的错误，Java 中异常处理通过 try-catch-finally 块捕获和处理异常，finally 块中的代码会在 try 或 catch 块中的代码执行完毕后被执行。

十六、问：什么是对象的序列化？
答：对象的序列化是指将对象转换为字节序列的过程，以便保存在文件中或通过网络传输。

十七、问：解释一下 Java 的垃圾回收机制。
答：Java 的垃圾回收机制是通过 JVM 自动管理内存，在对象不再被引用时自动回收其占用的内存空间。

十八、问：请简述怎么定义枚举类型？写出 java 中的常见包装类（用中文写不给分）？
答：定义枚举类型可以使用 enum 关键字。Java 中的常见包装类包括 Integer、Long、Float、Double 等。

十九、问：请简述写出 I/O 流的四个基类，IO 流分成哪几个流，他们之间有什么区别？
答：I/O 流的四个基类包括 InputStream、OutputStream、Reader 和 Writer。IO 流分为字节流和字符流，字节流以字节为单位进行操作，字符流以字符为单位进行操作。

二十、问：请简述写出 I/O 流的四个基类，IO 流分成哪几个流，他们之间有什么区别？
答：Java 的异常机制是通过 Throwable 类和其子类来实现的，异常是指程序在运行期间发生的错误。throw 用于抛出一个异常，而 throws 用于声明一个方法可能抛出的异常。

编程：

1. 定义一个学生类`Student`，包含姓名（`String`类型）、年龄（`int`类型）、成绩（`double`类型）属性，以及获取和设置这些属性的方法，还要有一个打印学生信息的方法。（使用 List 集合保存 10 个 Student 对象。遍历集合打印学生信息）

```
import java.util.ArrayList;
import java.util.List;

// 定义 Student 类
public class Student {
    private String name;
    private int age;
    private double score;

    // 构造方法
    public Student(String name, int age, double score) {
        this.name = name;
        this.age = age;
        this.score = score;
    }

    // getter 和 setter 方法
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public double getScore() {
        return score;
    }

    public void setScore(double score) {
        this.score = score;
    }
```

```
    // 打印学生信息的方法
    public void printInfo() {
        System.out.println("姓名: " + name);
        System.out.println("年龄: " + age);
        System.out.println("成绩: " + score);
    }
}

// 主类
public class Main {
    public static void main(String[] args) {
        // 创建一个 List 集合
        List<Student> students = new ArrayList<>();

        // 添加 10 个 Student 对象到集合中
        students.add(new Student("张三", 18, 90.5));
        students.add(new Student("李四", 19, 88.0));
        students.add(new Student("王五", 20, 92.5));
        students.add(new Student("赵六", 21, 85.0));
        students.add(new Student("钱七", 22, 95.5));
        students.add(new Student("孙八", 23, 89.0));
        students.add(new Student("周九", 24, 91.5));
        students.add(new Student("吴十", 25, 87.0));
        students.add(new Student("郑十一", 26, 93.5));
        students.add(new Student("冯十二", 27, 86.0));

        // 遍历集合打印学生信息
        for (Student student : students) {
            student.printInfo();
            System.out.println(); // 打印空行分隔每个学生的信息
        }
    }
}
```

2. 定义一个名为“Student”的类，要求使用封装。包含姓名(name)和年龄(age)两个私有属性，以及 getter/setter 方法、一个全参构造方法和一个显示学生信息的方法。
设计一个圆类`Circle`，包含半径（`double`类型）属性，计算面积（返回`double`类型）和周长（返回`double`类型）的方法。

```
// Student.java
public class Student {
    // 私有属性
    private String name;
```

```
    private int age;

    // 全参构造方法
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // getter方法
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    // setter方法
    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    // 显示学生信息的方法
    public void displayInfo() {
        System.out.println("学生姓名: " + name);
        System.out.println("学生年龄: " + age);
    }
}

// Circle.java
public class Circle {
    // 半径属性
    private double radius;

    // 构造方法
    public Circle(double radius) {
        this.radius = radius;
    }

    // getter方法
```

```
    public double getRadius() {
        return radius;
    }

    // setter 方法
    public void setRadius(double radius) {
        this.radius = radius;
    }

    // 计算面积的方法
    public double calculateArea() {
        return Math.PI * radius * radius;
    }

    // 计算周长的方法
    public double calculatePerimeter() {
        return 2 * Math.PI * radius;
    }
}
```

3. 编写 Java 程序，实现键盘录入任意字符串，输出反转后的字符串

```
import java.util.Scanner;

public class ReverseString {
    public static void main(String[] args) {
        // 创建 Scanner 对象用于读取键盘输入
        Scanner scanner = new Scanner(System.in);

        System.out.println("请输入一个字符串:");
        // 读取用户输入的字符串
        String inputString = scanner.nextLine();

        // 调用自定义方法来反转字符串
        String reversedString = reverseString(inputString);

        // 输出反转后的字符串
        System.out.println("反转后的字符串是: " + reversedString);

        // 关闭 Scanner
        scanner.close();
    }

    // 自定义方法，用于反转字符串
```

```
    private static String reverseString(String str) {
        StringBuilder sb = new StringBuilder(str);
        return sb.reverse().toString();
    }
}
```

4. 编写一个矩形类`Rectangle`，有长（`double`类型）和宽（`double`类型）属性，计算面积（返回`double`类型）和周长（返回`double`类型）的方法。

```
public class Rectangle {
    private double length; // 长
    private double width;  // 宽

    // 构造方法，用于初始化长和宽
    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    // 获取长
    public double getLength() {
        return length;
    }

    // 设置长
    public void setLength(double length) {
        this.length = length;
    }

    // 获取宽
    public double getWidth() {
        return width;
    }

    // 设置宽
    public void setWidth(double width) {
        this.width = width;
    }

    // 计算面积的方法
    public double calculateArea() {
        return length * width;
    }
```

```
    // 计算周长的方法
    public double calculatePerimeter() {
        return 2 * (length + width);
    }
}
```

5. 创建一个动物类`Animal`，有叫声方法（返回`String`类型叫声描述），然后创建狗类`Dog`和猫类`Cat`继承动物类，实现各自不同的叫声。

```
// Animal.java
public abstract class Animal {
    // 抽象方法，由子类实现具体的叫声
    public abstract String makeSound();
}

// Dog.java
public class Dog extends Animal {
    @Override
    public String makeSound() {
        return "汪汪叫";
    }
}

// Cat.java
public class Cat extends Animal {
    @Override
    public String makeSound() {
        return "喵喵叫";
    }
}

// Main.java
public class Main {
    public static void main(String[] args) {
        Animal myDog = new Dog();
        Animal myCat = new Cat();

        System.out.println("狗的叫声：" + myDog.makeSound());
        System.out.println("猫的叫声：" + myCat.makeSound());
    }
}
```

6. 请编码实现动物世界的继承关系：
动物（Animal）具有行为：吃（eat）. 睡觉（sleep）。
动物包括：兔子（Rabbit），老虎（Tiger）。

这些动物吃的行为各不相同（兔子吃草，老虎吃肉）；但睡觉的行为是一致的。请通过继承实现以上需求，并编写测试类 AnimalTest 进行测试。

输出结果如下：

兔子吃草
睡觉
老虎吃肉
睡觉

```
// Animal.java
public abstract class Animal {
    // 抽象方法 eat，子类必须实现
    public abstract void eat();

    // 具体方法 sleep，所有动物共享
    public void sleep() {
        System.out.println("睡觉");
    }
}

// Rabbit.java
public class Rabbit extends Animal {
    @Override
    public void eat() {
        System.out.println("兔子吃草");
    }
}

// Tiger.java
public class Tiger extends Animal {
    @Override
    public void eat() {
        System.out.println("老虎吃肉");
    }
}

// AnimalTest.java
public class AnimalTest {
    public static void main(String[] args) {
        Animal rabbit = new Rabbit();
        Animal tiger = new Tiger();

        rabbit.eat();
        rabbit.sleep();
        tiger.eat();
        tiger.sleep();
```

```
    }
}
```

7. 设计一个 Shape 接口和它的两个实现类 Square 和 Circle，要求如下：
(1). Shape 接口中有一个抽象方法 area(),该方法接收一个 double 类型的参数，返回一个 double 类型的结果。
(2). Square 和 Circle 中实现了 Shape 接口中的 area()抽象方法，分别用于求正方形和圆形的面积并返回。
(3). 在测试类 Test 中创建 Square 和 Circle 对象，计算边长为 2 的正方形的面积和半径为 3 的圆形的面积，并输出结果，示例如下。
边长为 2 的正方形面积：4.0
半径为 3 的正方形面积: 28.259

```
// Shape 接口
public interface Shape {
    double area(double dimension);
}

// Square 类实现 Shape 接口
public class Square implements Shape {
    @Override
    public double area(double sideLength) {
        return sideLength * sideLength;
    }
}

// Circle 类实现 Shape 接口
public class Circle implements Shape {
    @Override
    public double area(double radius) {
        return Math.PI * radius * radius;
    }
}

// 测试类 Test
public class Test {
    public static void main(String[] args) {
        // 创建 Square 对象
        Square square = new Square();
        double squareArea = square.area(2);
        System.out.println("边长为 2 的正方形面积：" + squareArea);

        // 创建 Circle 对象
        Circle circle = new Circle();
        double circleArea = circle.area(3);
```

```
        System.out.println("半径为 3 的圆形面积: " + circleArea);
    }
}
```

8. 写一个交通信号灯类`TrafficLight`，模拟信号灯的红、黄、绿切换，有获取当前信号灯颜色的方法。

```
public class TrafficLight {
    private String color; // 当前信号灯的颜色

    // 定义颜色的枚举类型
    public enum LightColor {
        RED, YELLOW, GREEN
    }

    // 构造函数，初始化为红色
    public TrafficLight() {
        this.color = LightColor.RED.name();
    }

    // 设置信号灯的颜色
    public void setColor(String color) {
        this.color = color;
    }

    // 获取当前信号灯的颜色
    public String getColor() {
        return this.color;
    }

    // 模拟信号灯的切换过程
    public void switchLight() {
        switch (this.color) {
            case "RED":
                this.color = LightColor.GREEN.name();
                break;
            case "GREEN":
                this.color = LightColor.YELLOW.name();
                break;
            case "YELLOW":
                this.color = LightColor.RED.name();
                break;
        }
    }
}
```

```
// 使用示例
public class Main {
    public static void main(String[] args) {
        TrafficLight trafficLight = new TrafficLight();
        System.out.println("初始颜色: " + trafficLight.getColor());

        // 切换几次信号灯颜色
        for (int i = 0; i < 5; i++) {
            trafficLight.switchLight();
            System.out.println("切换后颜色: " + trafficLight.getColor());
        }
    }
}
```

9. 实现一个员工类`Employee`，包含工号（`String`类型）、姓名（`String`类型）、工资（`double`类型）属性，以及计算奖金的方法（奖金为工资的一定比例，参数为比例，返回`double`类型奖金金额）。

```
public class Employee {
    private String id; // 工号
    private String name; // 姓名
    private double salary; // 工资

    public Employee(String id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    // 计算奖金的方法，奖金为工资的一定比例
    public double calculateBonus(double ratio) {
        return salary * ratio;
    }
}
```

10. 编码实现以下功能：
(1). 定义员工 Emp 类，包括属性：姓名 name、年龄 age 和方法：抽象的自我介绍方法 introduce、构造方法；定义管理层类继承员工类包括特有属性：月薪 salary，并实现父类自我介绍方法。
(2). 编写测试类测试管理层类。11. 创建集合保存 5 个学生的姓名，姓名允许重复。遍历集合中所有姓名

```
import java.util.*;

// 定义员工类
abstract class Emp {
    protected String name;
    protected int age;

    // 构造方法
    public Emp(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // 抽象的自我介绍方法
    public abstract void introduce();
}

// 定义管理层类继承员工类
class Manager extends Emp {
    private double salary;
```

```
    // 构造方法
    public Manager(String name, int age, double salary) {
        super(name, age);
        this.salary = salary;
    }

    // 实现父类的自我介绍方法
    @Override
    public void introduce() {
        System.out.println("大家好，我叫" + name + "，今年" + age + "岁，我的月薪是" + salary +
"元。");
    }
}

// 测试类
public class TestManager {
    public static void main(String[] args) {
        // 创建管理层对象
        Manager manager1 = new Manager("张三", 30, 10000.0);
        Manager manager2 = new Manager("李四", 35, 12000.0);

        // 调用自我介绍方法
        manager1.introduce();
        manager2.introduce();

        // 创建集合保存 5 个学生的姓名
        List<String> studentNames = new ArrayList<>();
        studentNames.add("王五");
        studentNames.add("赵六");
        studentNames.add("孙七");
        studentNames.add("周八");
        studentNames.add("吴九");
        studentNames.add("王五"); // 姓名允许重复

        // 遍历集合中所有姓名
        for (String name : studentNames) {
            System.out.println("学生姓名: " + name);
        }
    }
}
```

12. 编写 Java 程序，键盘录入任意字符串，统计并输出该字符串中每个字符出现的次数

```
import java.util.HashMap;
import java.util.Map;
```

```
import java.util.Scanner;

public class CharacterCount {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("请输入一个字符串:");
        String input = scanner.nextLine();

        Map<Character, Integer> charCountMap = new HashMap<>();

        // 统计字符出现次数
        for (char c : input.toCharArray()) {
            charCountMap.put(c, charCountMap.getOrDefault(c, 0) + 1);
        }

        // 输出每个字符及其出现次数
        for (Map.Entry<Character, Integer> entry : charCountMap.entrySet()) {
            System.out.println("字符 '" + entry.getKey() + "' 出现了 " + entry.getValue() + " 次");
        }

        scanner.close();
    }
}
```

13. 写一个图书类`Book`，有书名（`String`类型）、作者（`String`类型）、价格（`double`类型）属性，以及打折方法（参数为折扣比例，返回打折后的价格）。

```
public class Book {
    private String title; // 书名
    private String author; // 作者
    private double price; // 价格

    // 构造方法
    public Book(String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }

    // getter 方法
    public String getTitle() {
        return title;
    }

    public String getAuthor() {
```

```
        return author;
    }

    public double getPrice() {
        return price;
    }

    // 打折方法
    public double discountedPrice(double discountRate) {
        // 计算打折后的价格
        double discountedPrice = price * (1 - discountRate);
        return discountedPrice;
    }
}
```

14. 构建一个手机类`Phone`，包含品牌（`String`类型）、内存（`int`类型）、颜色（`String`类型）等属性，以及查看手机信息的方法（返回包含所有属性信息的字符串）。

```
public class Phone {
    // 属性
    private String brand;
    private int memory;
    private String color;

    // 构造器
    public Phone(String brand, int memory, String color) {
        this.brand = brand;
        this.memory = memory;
        this.color = color;
    }

    // 获取品牌的方法
    public String getBrand() {
        return brand;
    }

    // 设置品牌的方法
    public void setBrand(String brand) {
        this.brand = brand;
    }

    // 获取内存的方法
    public int getMemory() {
        return memory;
    }
```

```
    // 设置内存的方法
    public void setMemory(int memory) {
        this.memory = memory;
    }

    // 获取颜色的方法
    public String getColor() {
        return color;
    }

    // 设置颜色的方法
    public void setColor(String color) {
        this.color = color;
    }

    // 查看手机信息的方法
    public String viewPhoneInfo() {
        return "Brand: " + brand + ", Memory: " + memory + "GB, Color: " + color;
    }
}
```

15. 实现一个银行账户类`BankAccount`，有账户号（`String`类型）、余额（`double`类型）属性，存钱（参数为存入金额）和取钱（参数为取出金额，需判断余额是否足够）的方法。

```
public class BankAccount {
    private String accountNumber; // 账户号
    private double balance;          // 余额

    public BankAccount(String accountNumber) {
        this.accountNumber = accountNumber;
        this.balance = 0.0; // 初始化余额为 0
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount; // 存钱操作
            System.out.println("成功存入 " + amount + " 元。");
        } else {
            System.out.println("存款金额必须大于 0。");
        }
    }

    public void withdraw(double amount) {
        if (amount > 0) {
```

```
            if (balance >= amount) {
                balance -= amount; // 取钱操作
                System.out.println("成功取出 " + amount + " 元。");
            } else {
                System.out.println("余额不足，无法完成取款操作。");
            }
        } else {
            System.out.println("取款金额必须大于 0。");
        }
    }

    public String getAccountNumber() {
        return accountNumber;
    }

    public double getBalance() {
        return balance;
    }
}
```

16. 设计一个形状抽象类`Shape`，有计算面积的抽象方法（返回`double`类型），然后由三角形类`Triangle`和正方形类`Square`继承实现，三角形类有底和高属性，正方形类有边长属性。

```
// 定义抽象类 Shape
public abstract class Shape {
    // 声明抽象方法 calculateArea()
    public abstract double calculateArea();
}

// 三角形类 Triangle 继承自 Shape
public class Triangle extends Shape {
    private double base; // 底
    private double height; // 高

    // 构造方法
    public Triangle(double base, double height) {
        this.base = base;
        this.height = height;
    }

    // 实现 calculateArea() 方法
    @Override
    public double calculateArea() {
        return 0.5 * base * height; // 三角形面积公式
```

```
    }
}

// 正方形类 Square 继承自 Shape
public class Square extends Shape {
    private double sideLength; // 边长

    // 构造方法
    public Square(double sideLength) {
        this.sideLength = sideLength;
    }

    // 实现 calculateArea() 方法
    @Override
    public double calculateArea() {
        return sideLength * sideLength; // 正方形面积公式
    }
}
```

17. 写一个成绩管理类`ScoreManager`，能够添加学生成绩（参数为学生姓名和成绩）、删除学生成绩（参数为学生姓名）、查询学生成绩（参数为学生姓名，返回成绩）。

```
import java.util.*;

public class ScoreManager {
    private Map<String, Integer> scoreMap;

    public ScoreManager() {
        this.scoreMap = new HashMap<>();
    }

    // 添加学生成绩
    public void addScore(String studentName, int score) {
        if (scoreMap.containsKey(studentName)) {
            System.out.println("学生 " + studentName + " 已存在，成绩更新为 " + score);
        } else {
            System.out.println("添加学生 " + studentName + " 的成绩：" + score);
        }
        scoreMap.put(studentName, score);
    }

    // 删除学生成绩
    public void removeScore(String studentName) {
        if (scoreMap.containsKey(studentName)) {
            System.out.println("删除学生 " + studentName + " 的成绩");
```

```
            scoreMap.remove(studentName);
        } else {
            System.out.println("学生 " + studentName + " 不存在");
        }
    }

    // 查询学生成绩
    public Integer queryScore(String studentName) {
        if (scoreMap.containsKey(studentName)) {
            System.out.println("查询学生 " + studentName + " 的成绩：" +
scoreMap.get(studentName));
            return scoreMap.get(studentName);
        } else {
            System.out.println("学生 " + studentName + " 不存在");
            return null;
        }
    }
}

// 测试代码
class Main {
    public static void main(String[] args) {
        ScoreManager manager = new ScoreManager();
        manager.addScore("Alice", 90);
        manager.addScore("Bob", 85);
        manager.queryScore("Alice");
        manager.queryScore("Charlie");
        manager.removeScore("Bob");
        manager.removeScore("Dave");
    }
}
```

18. 实现一个时间类`Time`，能够进行时间的加、减计算，并能以特定格式（如 HH:mm:ss）格式化输出时间。

```
public class Time {
    private int hours;
    private int minutes;
    private int seconds;

    public Time(int hours, int minutes, int seconds) {
        this.hours = hours;
        this.minutes = minutes;
        this.seconds = seconds;
        normalize(); // Ensure time is in valid range
```

```
}

public Time add(Time other) {
	int totalSeconds = this.toSeconds() + other.toSeconds();
	return fromSeconds(totalSeconds);
}

public Time subtract(Time other) {
	int totalSeconds = this.toSeconds() - other.toSeconds();
	if (totalSeconds < 0) {
		throw new IllegalArgumentException("Cannot have negative time");
	}
	return fromSeconds(totalSeconds);
}

public String format() {
	return String.format("%02d:%02d:%02d", hours, minutes, seconds);
}

private void normalize() {
	while (seconds >= 60) {
		seconds -= 60;
		minutes++;
	}
	while (minutes >= 60) {
		minutes -= 60;
		hours++;
	}
	while (hours >= 24) {
		hours -= 24;
	}
	while (seconds < 0) {
		seconds += 60;
		minutes--;
	}
	while (minutes < 0) {
		minutes += 60;
		hours--;
	}
	while (hours < 0) {
		hours += 24;
	}
}
```

```
    private int toSeconds() {
        return hours * 3600 + minutes * 60 + seconds;
    }

    private Time fromSeconds(int totalSeconds) {
        int secs = totalSeconds % 60;
        int mins = (totalSeconds / 60) % 60;
        int hrs = totalSeconds / 3600;
        return new Time(hrs, mins, secs);
    }
}
```

19. 设计一个水果类`Fruit`，有不同水果的子类，如苹果类`Apple`、香蕉类`Banana`等，每个子类有自己的特点（如颜色、口感等）。

```
// 定义一个抽象类 Fruit
public abstract class Fruit {
    // 定义一个抽象方法，用于描述水果的特点
    public abstract void describe();
}

// 苹果类 Apple，继承自 Fruit
public class Apple extends Fruit {
    private String color;
    private String taste;

    public Apple(String color, String taste) {
        this.color = color;
        this.taste = taste;
    }

    @Override
    public void describe() {
        System.out.println("This apple is " + color + " and tastes " + taste + ".");
    }
}

// 香蕉类 Banana，继承自 Fruit
public class Banana extends Fruit {
    private String color;
    private String taste;

    public Banana(String color, String taste) {
        this.color = color;
        this.taste = taste;
```

```
    }

    @Override
    public void describe() {
        System.out.println("This banana is " + color + " and tastes " + taste + ".");
    }
}

// 测试代码
public class Main {
    public static void main(String[] args) {
        Fruit apple = new Apple("red", "sweet");
        Fruit banana = new Banana("yellow", "creamy");

        apple.describe();
        banana.describe();
    }
}
```

20. 写一个计数器类`Counter`，能够计数（递增）并重置（置为 0）。

```
public class Counter {
    private int count; // 用于存储计数的值

    // 构造器，初始化计数器为 0
    public Counter() {
        this.count = 0;
    }

    // 方法：计数递增
    public void increment() {
        this.count++;
    }

    // 方法：获取当前计数值
    public int getCount() {
        return this.count;
    }

    // 方法：重置计数器为 0
    public void reset() {
        this.count = 0;
    }
}
```