1. 静态变量/成员变量/局部变量：静态变量是类变量，共享一个值；成员变量是实例变量，每个对象有独立值；局部变量在方法或块中定义，只在该范围内有效。
2. Overload和Override：Overload是方法重载，同一类中用不同参数列表定义多个方法；Override是方法重写，子类提供与父类相同方法签名的新实现。
3. 访问权限：public可被任何类访问；protected可被同包和子类访问；default仅同包可访问；private仅本类可访问。
4. try/catch/finally：try包含可能抛出异常的代码；catch捕获并处理异常；finally无论是否发生异常都执行。
5. 继承/封装/多态：继承是子类继承父类属性和方法；封装是隐藏内部实现，只暴露必要接口；多态是同一操作作用于不同对象产生不同结果。
6. this/super：this指代当前对象，用于区分成员变量和局部变量；super指代父类对象，用于调用父类属性和方法。
7. 抽象类和接口：抽象类可包含抽象方法和具体方法，用于代码复用；接口只包含抽象方法，用于定义行为规范。

8. final关键字：用于声明常量、方法和类，分别表示值不可变、方法不可重写、类不可继承。
9. 实现多线程：继承Thread类、实现Runnable接口、使用ExecutorService。
10. 字节输入流/输出流：FileInputStream/FileOutputStream用于文件读写；BufferedInputStream/BufferedOutputStream提供缓冲功能；ObjectInputStream/ObjectOutputStream用于对象序列化与反序列化。
11. start/run方法：start方法启动新线程并执行run方法；run方法包含线程执行的代码。
12. 异常处理机制：通过try捕获异常，catch处理异常，保证程序稳定性；自定义异常类实现特定业务逻辑处理。
13. throw/throws关键字：throw用于显式抛出异常对象；throws用于声明方法可能抛出的异常类型。
14. 静态方法与非静态方法：静态方法属于类，可直接通过类名调用；非静态方法属于实例，需要通过对象调用。
15. 构造方法：用于初始化对象状态的方法，与类名相同且无返回值；特点是在创建对象时自动调用。
16. 类中类：内部类、静态内部类、局部内部类、匿名内部类；特点是可以访问外部类私有成员，实现代码封装和逻辑分组。

17. 代码块特点：静态代码块在类加载时执行一次；构造代码块在每次创建对象时执行；同步代码块用于多线程同步；普通代码块在方法或循环中定义局部变量。
18. 字节流/字符流读写：字节流以字节为单位处理数据，适用于任意类型文件；字符流以字符为单位处理数据，适用于文本文件。通过InputStream/OutputStream和Reader/Writer实现读写操作。
19. TCP/UDP网络编程：TCP是面向连接的协议，通过三次握手建立连接，保证数据可靠传输；UDP是无连接的协议，发送方将数据打包成数据报直接发送给接收方，不保证数据到达顺序和可靠性。
20. 向上转型/向下转型：向上转型是将子类对象赋值给父类引用；向下转型是将父类引用强制转换为子类类型。
21. 泛型：泛型是一种类型参数化的编程技术，允许在定义类、接口和方法时使用类型参数，提高代码复用性和类型安全性。

代码题：

1、

```
// 抽象类 Shape
abstract class Shape {
    // 抽象方法 getArea
    abstract double getArea();
}

// Rectangle 类从 Shape 类派生
class Rectangle extends Shape {
    private double length;
    private double width;

    // 构造函数
    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    // 实现 getArea 方法
    @Override
    double getArea() {
        return length * width;
    }
}

// Cube 类从 Rectangle 类派生（这不是最佳实践，但为了符合您的要求）
class Cube extends Rectangle {
    private double height;

    // 构造函数
    public Cube(double length, double width, double height) {
        super(length, width); // 调用父类构造函数
        this.height = height;
    }

    // 由于 Cube 是从 Rectangle 派生的，它继承了 getArea 方法，但这个方法只计算矩形的面积，
    // 而不是立方体的体积。因此，我们可能需要添加一个新方法来计算立方体的体积。
    public double getVolume() {
        return super.getArea() * height;
    }
}

// 主类
public class Main {
    public static void main(String[] args) {
        Rectangle rectangle = new Rectangle(5, 10);
        System.out.println("Rectangle Area: " + rectangle.getArea());
```

```
        Cube cube = new Cube(5, 10, 20);
        // 注意：调用 getArea() 对于 Cube 来说可能没有意义，因为它返回的是底面的面积，而不是立方体的体积。
        System.out.println("Cube Base Area (from getArea()): " + cube.getArea());
        System.out.println("Cube Volume: " + cube.getVolume());
    }
}
```

2、

```
public class PerfectNumbers {

    public static void main(String[] args) {
        int limit = 1000;

        for (int i = 1; i <= limit; i++) {
            if (isPerfect(i)) {
                System.out.println(i + " 是一个完数。");
            }
        }
    }

    public static boolean isPerfect(int number) {
        int sum = 0;

        for (int i = 1; i <= number / 2; i++) {
            if (number % i == 0) {
                sum += i;
            }
        }

        return sum == number;
    }
}
```

3、

```
import java.util.Arrays;

public class RandomShuffleArray {

    public static void main(String[] args) {
        int[] arr = new int[100];

        // 初始化数组为 1 到 100
        for (int i = 0; i < arr.length; i++) {
            arr[i] = i + 1;
        }

        // 使用 Math.random() 打乱数组
        for (int i = arr.length - 1; i > 0; i--) {
            int j = (int) (Math.random() * (i + 1));
```

```
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }

        // 打印打乱后的数组
        System.out.println(Arrays.toString(arr));
    }
}
```

4、

```
public class Main {
    public static void main(String[] args) {
        String originalString = "HelloWorld";
        String reversedString = reverseString(originalString);
        System.out.println(reversedString); // 输出: dlroWolleH
    }

    public static String reverseString(String input) {
        return new StringBuilder(input).reverse().toString();
    }
}
```

5、

```
import java.io.FileInputStream;
import java.io.IOException;

public class FileReadExample {

    public static void main(String[] args) {
        String filePath = "c:/test/A.java";
        FileInputStream fileInputStream = null;

        try {
            // 打开文件输入流
            fileInputStream = new FileInputStream(filePath);

            // 创建一个字节数组来存储读取的数据
            byte[] buffer = new byte[1024];
            int bytesRead;

            // 读取文件内容，并显示在屏幕上
            while ((bytesRead = fileInputStream.read(buffer)) != -1) {
                // 将字节转换为字符串
                String data = new String(buffer, 0, bytesRead);
                System.out.print(data);
            }

        } catch (IOException e) {
            // 处理文件读取过程中的异常
            e.printStackTrace();
```

```
            } finally {
                // 关闭文件输入流
                if (fileInputStream != null) {
                    try {
                        fileInputStream.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
}
```

6、

```
    // (1) 打印机的抽象类
    abstract class Printer {
        // 打印的抽象方法
        abstract void print();
}
    // (2) 彩色打印机类
    class ColorPrinter extends Printer {
        // 彩色打印机的打印方法
        @Override
        void print() {
            System.out.println("使用彩色打印机进行打印。");
        }
    }

    // (2) 黑白打印机类
    class BlackWhitePrinter extends Printer {
        // 黑白打印机的打印方法
        @Override
        void print() {
            System.out.println("使用黑白打印机进行打印。");
        }
}
    // (3) 测试类
    public class TestPrinter {
        public static void main(String[] args) {
            // 创建一个彩色打印机对象
            Printer colorPrinter = new ColorPrinter();
            // 使用彩色打印机打印
            colorPrinter.print();

            // 创建一个黑白打印机对象
            Printer blackWhitePrinter = new BlackWhitePrinter();
            // 使用黑白打印机打印
            blackWhitePrinter.print();
        }
}
```

7、
（1）创建一个 HashMap，键（Key）为字符类型（因为汉字也是字符），值（Value）为整型，用于存储每个汉字的出现次数。
（2）读取书中的文本内容。
（3）遍历文本中的每个字符。
（4）检查 HashMap 中是否已存在该字符：
　　如果存在，则获取当前计数并加 1。
　　如果不存在，则添加该字符到 HashMap 中，并设置计数为 1。
（5）遍历完成后，HashMap 将包含每个汉字及其出现次数。
8、

```
public class BadmintonMatchup {

    private static final String[] TEAM_A = {"a", "b", "c"};
    private static final String[] TEAM_B = {"x", "y", "z"};
    private String[] matchups = new String[3]; // 用于存储比赛结果的数组

    public static void main(String[] args) {
        BadmintonMatchup matcher = new BadmintonMatchup();
        if (matcher.findMatchups()) {
            matcher.printMatchups();
        } else {
            System.out.println("没有找到符合条件的比赛名单！");
        }
    }

    private boolean findMatchups() {
        return backtrack(0);
    }

    private boolean backtrack(int index) {
        // 所有位置都已填满，找到一个解
        if (index == 3) {
            return true;
        }

        // 尝试 TEAM_B 中的所有队员
        for (int i = 0; i < 3; i++) {
            if (isValid(index, i)) {
                // 放置队员
                matchups[index] = TEAM_B[i];

                // 递归填充下一个位置
                if (backtrack(index + 1)) {
                    return true; // 找到解，返回 true
                }

                // 回溯：撤销选择
                matchups[index] = null;
            }
```

```
        }

        // 没有找到解
        return false;
    }

    private boolean isValid(int indexA, int indexB) {
        // a 不和 x 比
        if (indexA == 0 && indexB == 0) {
            return false;
        }

        // c 不和 x, z 比
        if (indexA == 2 && (indexB == 0 || indexB == 2)) {
            return false;
        }

        // 检查是否已经有队员和 TEAM_B[indexB] 比赛
        for (int i = 0; i < indexA; i++) {
            if (matchups[i] != null && matchups[i].equals(TEAM_B[indexB])) {
                return false;
            }
        }

        return true;
    }

    private void printMatchups() {
        for (int i = 0; i < 3; i++) {
            System.out.println(TEAM_A[i] + " 对 " + matchups[i]);
        }
    }
}
```

9、

```
public class UpperCaseCounter {

    public static void main(String[] args) {
        // 示例字符串
        String inputString = "Hello World! This is a Test String.";

        // 调用方法计算大写字母个数
        int count = countUpperCaseLetters(inputString);

        // 输出结果
        System.out.println("The number of uppercase letters in the string is: " + count);
    }

    /**
     * 计算字符串中大写字母的个数
```

```
     *
     * @param str 要检查的字符串
     * @return 大写字母的个数
     */
    public static int countUpperCaseLetters(String str) {
        int count = 0;
        for (int i = 0; i < str.length(); i++) {
            char ch = str.charAt(i);
            if (Character.isUpperCase(ch)) {
                count++;
            }
        }
        return count;
    }
}
```

10、

```
public class SimpleEncryption {

    public static void main(String[] args) {
        String plaintext = "hello";
        String ciphertext = encrypt(plaintext);
        System.out.println("Encrypted text: " + ciphertext);
    }

    public static String encrypt(String plaintext) {
        StringBuilder ciphertext = new StringBuilder();
        for (char c : plaintext.toCharArray()) {
            if (Character.isLetter(c)) {
                char base = Character.isLowerCase(c) ? 'a' : 'A';
                c = (char) (((c - base + 5) % 26) + base); // 置换为字母表中其后的第 5 个字母
            }
            ciphertext.append(c);
        }
        return ciphertext.toString();
    }
}
```

11、

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("请输入第一个字符串:");
        String str1 = scanner.nextLine();

        System.out.println("请输入第二个字符串:");
        String str2 = scanner.nextLine();
```

```
        // 字符串拼接
        System.out.println("字符串拼接结果: " + concatenate(str1, str2));

        try {
            // 整数相加
            int int1 = Integer.parseInt(str1);
            int int2 = Integer.parseInt(str2);
            System.out.println("整数相加结果: " + addIntegers(int1, int2));
        } catch (NumberFormatException e) {
            System.out.println("输入的字符串无法转换为整数!");
        }

        try {
            // 浮点数相加
            double double1 = Double.parseDouble(str1);
            double double2 = Double.parseDouble(str2);
            System.out.println("浮点数相加结果: " + addDoubles(double1, double2));
        } catch (NumberFormatException e) {
            System.out.println("输入的字符串无法转换为浮点数!");
        }

        scanner.close();
    }

    // 字符串拼接函数
    public static String concatenate(String str1, String str2) {
        return str1 + str2;
    }

    // 整数相加函数
    public static int addIntegers(int int1, int int2) {
        return int1 + int2;
    }

    // 浮点数相加函数
    public static double addDoubles(double double1, double double2) {
        return double1 + double2;
    }
}
12、
public class Card {
    private String face; // 牌面值
    private String suit; // 花色

    // 构造方法
    public Card(String face, String suit) {
        this.face = face;
        this.suit = suit;
```

```
    }

    // 获取牌面值
    protected String getFace() {
        return face;
    }

    // 获取花色
    protected String getSuit() {
        return suit;
    }

    // 主方法，用于测试
    public static void main(String[] args) {
        Card redAce = new Card("A", "红桃");
        System.out.println("花色: " + redAce.getSuit());
        System.out.println("牌面值: " + redAce.getFace());
    }
}
public class StringCounter {

    // 计算子串在字符串中的出现次数
    public static int countSubstring(String str, String sub) {
        if (str == null || sub == null || str.length() == 0 || sub.length() == 0 || str.length() < sub.length()) {
            return 0;
        }

        int count = 0;
        int index = 0;

        while ((index = str.indexOf(sub, index)) != -1) {
            count++;
            index += sub.length();
        }

        return count;
    }

    // 主方法，用于测试
    public static void main(String[] args) {
        String str = "这是一个测试字符串，用于测试子串出现的次数";
        String sub = "测试";
        System.out.println("子串 \"" + sub + "\" 在字符串中出现了 " + countSubstring(str, sub) + " 次");
    }
}
13、
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
```

```
public class SumNumbersFromFile {

    public static void main(String[] args) {
        String filename = "numbers.txt";
        int sum = 0;

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] numbersInLine = line.split("\\s+"); // Split by whitespace
                for (String numberStr : numbersInLine) {
                    try {
                        int number = Integer.parseInt(numberStr);
                        sum += number;
                    } catch (NumberFormatException e) {
                        System.err.println("Invalid number format: " + numberStr);
                    }
                }
            }
            System.out.println("The sum of all numbers in " + filename + " is: " + sum);
        } catch (IOException e) {
            System.err.println("An error occurred while reading the file " + filename + ": " + e.getMessage());
        }
    }
}
```

14、

```
import java.math.BigInteger;

public class BinaryAdder {

    public static String addBinaryStrings(String x, String y) {
        BigInteger num1 = new BigInteger(x, 2); // 将二进制字符串转换为 BigInteger
        BigInteger num2 = new BigInteger(y, 2);

        BigInteger sum = num1.add(num2); // 计算两个 BigInteger 的和

        return sum.toString(2); // 将和转换回二进制字符串
    }

    public static void main(String[] args) {
        String x = "111";
        String y = "1";
        String result = addBinaryStrings(x, y);
        System.out.println(result); // 输出: 1000
    }
}
```