第一章：java 语言概述

诞生于 1995，一种可移植的、跨平台的语言

特点：面向对象，可移植，健壮，安全，动态

Java SE 桌面开发和低端商务应用的解决方案。

Java EE 是以企业为环境而开发应用程序的解决方案。

Java ME 是致力于消费产品和嵌入式设备的最佳解决方案。

JVM Java 虚拟机 JDK 开发工具包 JRE Java 运行环境 IDE 集成开发环境

.java 文件是源文件，通过 javac 命令编译后生成.class 文件；.class 文件是字码结文件，即.java 文件编译后的代码。

第二章：数据类型，运算符，表达式和语句

标识符：字母是区分大小写的，不能是 true, false, null（尽管它们不是关键字）

关键字：被赋予特定意义，abstract, continue, for, new, switch, default, goto*…

8 种基本数据类型可分为 4 大类型: 逻辑类型：boolean

字符类型 char 整数类型：byte(1), short(2), int(4), long 浮点类型：float(7), double

字符类型: Uincode'\uxxx' 2 个字节，16 位，最高位不是符号位 范围：0~2^16-1

（char 的最高位不是符号位，有可能超出 short 的取值范围，转化用 int）

定义数组 int intArray[ ][ ] = { {1,2}, {2,3}, {4,5} }; int a[ ][ ] = new int[2][3];

Instanceof 左对象，右类 boolean f = rectangleOne instanceof Rect;

第三章：类与对象

面向对象编程主要有三个特性：封装，继承，多态

成员变量的类型，对象接口也可以， 类内有效与书写的先后位置无关

构造方法：不能被 static、final、synchronized、abstract 和 native 修饰

Java 具有"垃圾收集"机制，Java 的运行环境周期地检测某个实体是否已不再被任何对象所引用，如果发现，就释放该实体占有的内存。

Static：静态变量既可以通过某个对象访问也可以直接通过类名访问，不可 this

Final：修饰的成员变量不占用内存，必须要初始化。有 static final a;

静态方法(类方法)只能调用该类的静态方法用静态变量 main 方法也是静态方法

在 Java 中，方法的所有参数都是"传值"的,方法中参数变量的值是调用者指定的值的拷贝。方法如果改变参数的值，不会影响向参数"传值"的变量的值.而当传的是引用类型数据 包括对象、数组、接口（interface）。当参数是引用类型时，"传值"传递的是变量的引用。

使用 import，增加编译时间，但不会影响程序运行的性能。

访问权限：在与类 A 同 package 的另外一个类 B 中，可以访问对象 a 的以下成员 Friendly (or default), protected, public

在类 A 的子类 B 中(不同 package)，可以访问对象 a 的以下成员

protected, public

第四章：继承（extends）

子类与父类：不支持多重继承，继承的变量和方法按照访问权限,例如不在同一个包里，子类不能继承父类的 friendly

子类不继承父类的构造方法，可采用 super()来表示

final 类不能被继承，final 方法 不能被重写

abstract 类不能用 new 运算符创建对象，必须产生其子类，由子类创建对象。

如果 abstract 类的类体中有 abstract 方法，只允许声明，而不允许实现；而该类的非 abstract 子类必须实现 abstract 方法，即重写 (override) 父类的 abstract 方法。

接口 interface+接口的名字不许提供方法的实现，接口也可以被继承 (extend)

方法 public abstract int g(int x,int y);

接口中的常量用 public static final 来修饰，但可以省略

接口中的方法用 public abstract 来修饰，但可以省略

在实现接口中的方法时，一定要用 public 来修饰，不可以省略

接口回调：接口回调是多态的另一种体现。把子类创建的对象的引用放到一个父类的对象中时，得到该对象的一个上转型对象

内部类：类体中不可以声明静态变量（类变量）和静态方法（类方法）

匿名类：一定是内部类，主要用途就是向方法的参数传值

Java 泛型的主要目的是可以建立具有类型安全的数据结构，如链表（LinkedList）、散列映射（HashMap）等数据结构。

第五章 字符串

字符串类 String 表示一个 UTF-16 格式的字符串

public int length() 获取一个字符串的长度

public static int parseInt | Byte Short Long Float Double(String s)

public String valueOf(byte b)

public String toString(){} 返回该类输出的字符串

public byte[] getBytes()：将当前字符串转化为一个字节数组

StringBuffer 类：能创建可修改的字符串序列

第八章 输入输出流

java.io 中有 4 个重要的 abstract class

InputStream（字节输入流）OutputStream（字节输出流）

Reader（字符输入流） Writer（字符输出流）

File read2 = new File("C:\\Users\\slark ","userTotTime.txt"); //文件路径

Buffer append 方法：可以将其它 Java 类型数据转化为字符串后

char charAt(int index) void setCharAt(int index, char ch)

StringBuffer（多线程）是线程安全的 StringBuilder（效率高）不是线程安全的

线程安全是指多个线程操作同一个对象不会出现问题

```
StringTokenizer fenxi = new StringTokenizer(s,".");
for(int i=0; fenxi.hasMoreTokens(); i++)    {String str = fenxi.nextToken();}
String cost = "市话费: 176.89 元，长途费: 187.98 元，网络费: 928.66 元";
Scanner scanner = new Scanner(cost);
scanner.useDelimiter("[^0123456789.]+");
    while(scanner.hasNext()){
    try{            double price = scanner.nextDouble();
                    System.out.println(price);}
    catch(InputMismatchException exp)
    {               String t = scanner.next(); }}
```

.任意字符 \\d 0-9 \\D非数字 \\s 空格类\\S 非~\\w可标识符\\W

X？0|1 X{n}恰好出现 n 次 X{n,}至少 n 次 X{n,m} n-m 次

"[159]ABC" "1ABC" 、"5ABC"和"9ABC"都是行

[^abc]：代表除了 a, b, c 以外的任何字符[a-d]：代表 a 至 d 中的任何一个

```
Pattern p;                  Matcher m;
p = Pattern.compile("\\d+"); m = p.matcher("2008 年 08 月 08 日");
  while(m.find()) { String str = m.group();}
```

第六章，泛型和集合

返回一个整数，0 表示一月，1 表示二月 calendar.get(Calendar.MONTH);

求相差日期 calendar.set(1931,8,18); long timeOne = calendar.getTimeInMillis();

LinkedList<E>：add() indexof() <E>,get(index)

HashSet<E> add() clear() remove() contain()

Iterator<Student> iter = tempSet.iterator(); while(iter.hasNext()){}

HashMap<K,V>: get(Object key)

```
class StudentComparator implements Comparator
    int score;
    public int compare(Object o1, Object o2){
        return ( ((Student)o1).score - ((Student)o1).score);}}
  main(){Arrays.sort(students, new StudentComparator());}
class A{
    public int compareTo(Object o) {return (this.score - stu.score);}
```

Stack<E> push() pop() intValue()

第七章 线程

线程是比进程更小的执行单位。一个进程在其执行过程中，可以产生多个线程，形成多条执行线索，动态概念

JVM 在主线程和其他线程之间轮流切换，保证每个线程都有机会使用 CPU 资源

四种状态： 新建 运行 中断 死亡

Thread 的子类创建线程 写一个类继承 Thread 重写 public void run 方法 主线程 new 一个类，然后 start 就可以

实现 Runnable 接口的类的实例，然后直接 Thread(Runnable target) start 就可

if(Thread.currentThread().getName().equals(name1)){}是里面的一个好语句

try{ Thread.sleep(1000); }catch(InterruptedException e) {}使得都能调用

一个已经运行的线程在没有进入死亡状态时，不要再给线程分配实体。

如果实现 runnable 类里面有 Thread 变量，可以用 thread1.interrupt();唤醒

线程同步是指多个线程要执行一个 synchronized 修饰的方法，如果一个线程 A 在占有 CPU 资源期间，使得 synchronized 方法被调用执行，那么在该 synchronized 方法返回之前（即 synchronized 方法调用执行完毕之前），其他占有 CPU 资源的线程一旦调用这个 synchronized 方法就会引起堵塞，堵塞的线程要一直等到堵塞的原因消除（ 即 synchronized 方法返回）

public synchronized void saveOrTake(int number){}

使用 wait()方法可以中断方法的执行，使本线程等待，暂时让出 CPU 的使用权，并允许其它线程使用这个同步方法。其它线程如果在使用这个同步方法时不需要等待，那么它使用完这个同步方法的同时，应当用 notifyAll()方法通知所有由于使用这个同步方法而处于等待的线程结束等待。

线程联合：B.join()如果线程 A 在占有 CPU 资源期间一旦联合 B 线程，那么 A 将立刻中断执行，一直等到它联合的线程 B 执行完毕，A 线程再重新排队等待

守护线程：当程序中的所有用户线程都已结束运行时，即使守护线程的 run()方法中还有需要执行的语句，守护线程也立刻结束运行

thread.setDaemon(true);

```
setVisible(true);    validate();
setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
public void actionPerformed(ActionEvent e) {e.getSource() == translate1 }
```

第十章 URL 网络编程

URL 对象通常包含最基本的三部分信息：协议、地址、资源。

第九章 图形用户界面设计

java.awt 包中的类创建的组件习惯上称为重组件。创建一个按钮组件时，都有一个相应的本地组件在为它工作，即显示主件和处理主件事件，该本地主件称为它的同位体。javax.swing 包为我们提供了更加丰富的、功能强大的组件，称为 Swing 组件，其中大部分组件是轻组件，没有同位体，而是把与显示组件有关的许多工作和处理组件事件的工作交给相应的 UI 代表来完成。 这些 UI 代表是用 Java 语言编写的类，这些类被增加到 Java 的运行环境中，因此组件的外观不依赖平台，不仅在不同平台上的外观是相同的，而且与重组件相比有更高的性能。JComponent 类的子类都是轻组件 JFrame, JApplet, JDialog 都是重组件，即有同位体的组件。这样，窗口（JFrame）、小应用程序（Java Applet）、对话框（JDialog）可以和操作系统交互信息。轻组件必须在这些容器中绘制自己，习惯上称这些容器为 Swing 的底层容器。对于 JFrame 窗口，默认布局是 BorderLayout 布局 FlowLayout、BorderLayout、CardLayout、GridLayout 布局 null 布局 p.setLayout(null);

setBounds(int a, int b, int width, int height) JButton next = new JButton("确定(5s)"); //确认按钮 Font style = new Font("宋体", Font.BOLD, 24); //宋体 24 号 JLabel A, B, C, D, question, tips, rate, fin; //框架中的文字， 用 JLabel JTextField inputans; //输入框 setTitle(s); setLayout(null); translate1.setFont(style); setBounds(500, 250, 500, 300); //给整个窗口设置位置大小

```
try {
    DatagramSocket serverSocket = new DatagramSocket(8888); // 创建服务器端套接字
    System.out.println("服务器已启动...");
    while (true) {
        byte[] receiveData = new byte[1024];
        DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
        serverSocket.receive(receivePacket); // 接收客户端消息
        String clientMessage = new String(receivePacket.getData()).trim(); // 解析接收到的消息
        System.out.println("[收到客户端消息]" + clientMessage);
        byte[] responseData = clientMessage.getBytes();
        DatagramPacket responsePacket = new DatagramPacket(responseData, responseData.length,
                receivePacket.getAddress(), 8888);
        serverSocket.send(responsePacket); // 发送响应消息给客户端
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

```
DatagramSocket clientSocket = new DatagramSocket(8888); // 创建客户端套接字
Scanner scanner = new Scanner(System.in);
Thread receive = new Thread(() -> {
    while (true) {
        byte[] receiveData = new byte[1024];
        DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
        try {
            clientSocket.receive(receivePacket); // 接收服务器响应
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        String serverResponse = new String(receivePacket.getData()).trim(); // 解析服务器响应
        System.out.println("[收到消息]" + serverResponse);
    }
});
receive.start();
while (true) {
    String message = "A: " + scanner.nextLine();
    byte[] sendData = message.getBytes();
    DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, InetAddress.getLocalHost(), 88
88);
    clientSocket.send(sendPacket); // 发送消息给服务器
}
```

```
int serverPort = 12345; // 服务器的端口号
try {// 创建ServerSocket对象，指定服务器端口号
    ServerSocket serverSocket = new ServerSocket(serverPort);
    System.out.println("Server is listening on port " + serverPort);
    while (true) {// 等待客户端连接
        Socket clientSocket = serverSocket.accept();
        System.out.println("New client connected: " + clientSocket.getInetAddress().getHostAddress());
        // 获取输入流和输出流
        BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
        String message = in.readLine(); // 从客户端接收数据
        System.out.println("Client message: " + message);
        out.println("Hello, Client!");              // 发送数据到客户端
        clientSocket.close();                  // 关闭连接
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

```
String serverHost = "127.0.0.1"; // 服务器的IP地址
int serverPort = 12345; // 服务器的端口号
try {
    // 创建Socket对象，指定服务器的IP地址和端口号
    Socket socket = new Socket(serverHost, serverPort);
    // 获取输入流和输出流
    BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
    // 发送数据到服务器
    out.println("Hello, Server!");
    // 从服务器接收数据
    String response = in.readLine();
    System.out.println("Server response: " + response);
    // 关闭连接
    socket.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

```
public synchronized void printMonth() {
    for (int i = 0; i < 12; i++) {
        System.out.print(months[i]);
        notify();
        if (i < 11) {
            try {
                wait();
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

```
class PrintMonth implements Runnable {
    Print print;

    PrintMonth(Print print) {
        this.print = print;
    }

    @Override
    public void run() {
        print.printMonth();
    }
}
```

```
public static void main(String[] args) {
    Print print = new Print();
    Thread number = new Thread(new PrintNumber(print));
    Thread month = new Thread(new PrintMonth(print));
    month.start();
    number.start();
}
```

```
public static void main(String[] args) {
    ExecutorService executor= Executors.newFixedThreadPool(3);
    executor.execute(new PrintChar('a',100));
    executor.execute(new PrintChar('b',100));
    executor.execute(new PrintNum(100));
    executor.shutdown();
}
```

```
class COVID19Comparator implements Comparator<Country> {
    @Override
    public int compare(Country c1, Country c2) {
        return Integer.compare(c1.getCOVID19(), c2.getCOVID19());
    }
}
    public static void main(String[] args) {
        Map< Country,String> countryMap = new TreeMap<>(new COVID19Comparator());
        countryMap.put(new Country("美国", 20932750, 44918565), "美国");
        for (Country country : countryMap.keySet()) {
            country.show();
        }
    }
```

```
class Country implements Comparable<Country> {
    @Override
    public int compareTo(Country o) {
        return Integer.compare(this.COVID19, o.COVID19);
    }
}
        TreeMap<Country, String> countryMap = new TreeMap<>();
```

```
Set<String> A = new HashSet<>();
A.add("张三");
A.add("李四");
Set<String> B = new HashSet<>();
B.add("李四");
B.add("王五");
Set<String> AB = new HashSet<>(A);
AB.retainAll(B);
System.out.println("A社团的人：" + A);
System.out.println("B社团的人：" + B);
System.out.println("同时是A和B社团的人：" + AB);
```

```
public static void main(String[] args) {
    String filePath = "About5ZU.txt";        // 读取文件路径
    Map<String, Integer> wordCountMap = new HashMap<>();        // 创建HashMap用于存储单词及其出现次数
    try {
        File file = new File(filePath);            // 打开文件
        Scanner scanner = new Scanner(file);
        while (scanner.hasNextLine()) {            // 逐行读取文件内容
            String line = scanner.nextLine();
            String[] words = line.split("[^a-zA-Z]+");    // 使用正则表达式分割行成单词数组
            for (String word : words) {                // 遍历单词数组，统计每个单词的出现次数
                if (!word.isEmpty()) {
                    String lowercaseWord = word.toLowerCase();   // 如果是大写则转换为小写
                    // 更新单词出现次数
                    wordCountMap.put(lowercaseWord, wordCountMap.getOrDefault(lowercaseWord, 0) + 1);
                }
            }
        }
        scanner.close();            // 关闭文件
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    List<Map.Entry<String, Integer>> wordCountList = new ArrayList<>(wordCountMap.entrySet());
    wordCountList.sort(Map.Entry.comparingByValue(Comparator.reverseOrder()));        // 根据出现次数从多到少排序单词
    int count = 0;        // 输出出现次数最多的50个英文单词
    for (Map.Entry<String, Integer> entry : wordCountList) {
        System.out.print(entry.getKey() + " ");
        count++;
        if (count % 10 == 0) {
            System.out.println();
        }
        if (count >= 50) {
            break;
        }
    }
} |
```

```
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        for (int i = 0; i < 10; i++) { // 循环连续测试10次
            String input = scanner.nextLine();
            StringBuilder uppercase = new StringBuilder(); // 用于存储大写字母
            StringBuilder lowercase = new StringBuilder(); // 用于存储小写字母
            StringBuilder digits = new StringBuilder(); // 用于存储数字
            for (int j = 0; j < input.length(); j++) {
                char ch = input.charAt(j);
                if (Character.isUpperCase(ch)) {
                    uppercase.append(ch);
                } else if (Character.isLowerCase(ch)) {
                    lowercase.append(ch);
                } else if (Character.isDigit(ch)) {
                    digits.append(ch);
                }
            }
            System.out.println("大写字母: " + uppercase.toString() +
" 小写字母: " + lowercase.toString() + " 数字: " + digits.toString());
        }
    }
```

```
public class Test {
    public static void main(String[] args) {
        // 判断字符串是否匹配正则表达式"ABC "
        System.out.println("Hi, ABC, good".matches("ABC "));
        // 判断字符串中是否包含"ABC"
        System.out.println("Hi, ABC, good".matches(".*ABC.*"));
        // 将字符串中的",;"替换为"#"
        System.out.println("A,B;C".replaceAll(",;", "#"));
        // 将字符串中的","或";"替换为"#"
        System.out.println("A,B;C".replaceAll("[,;]", "#"));
        // 使用正则表达式"[,;]"将字符串分割成多个部分
        String[] tokens = "A,B;C".split("[,;]");
        for (int i = 0; i < tokens.length; i++)
            System.out.println(tokens[i] + " ");
    }
}
```