**第2天的课后作业**

**基本概念**

1. 运行以下代码，解释二者的区别

public class Main {

public static void main(String[] args) {

Person p = new Person();

int n = 15; // n的值为15

p.setAge(n); // 传入n的值

System.out.println(p.getAge()); // 15

n = 20; // n的值改为20

System.out.println(p.getAge()); // 15还是20?

}

}

class Person {

private int age;

public int getAge() {

return this.age;

}

public void setAge(int age) {

this.age = age;

}

}

public class Main {

public static void main(String[] args) {

Person p = new Person();

String[] fullname = new String[] { "Homer", "Simpson" };

p.setName(fullname); // 传入fullname数组

System.out.println(p.getName()); // "Homer Simpson"

fullname[0] = "Bart"; // fullname数组的第一个元素修改为"Bart"

System.out.println(p.getName()); // "Homer Simpson"还是"Bart Simpson"?

}

}

class Person {

private String[] name;

public String getName() {

return this.name[0] + " " + this.name[1];

}

public void setName(String[] name) {

this.name = name;

}

}

代碼2每次調用成員函数都會對其重新定義，而代碼1不會。因此代碼1的輸出結果為15;代碼2輸出結果為Bart Simpson。

1. 简述面向对象程序语言的3大特性。

1．封装性，隐藏对象的属性和实现细节，仅对外公开接口，控制在程序中属性的读取和修改的访问级别。

2. 继承性，就是子类继承父类的属性和行为，使得子类对象具有与父类相同的属性、相同的行为。子类可以直接访问父类中的非私有的属性和行为。

3. 多态性，同一个对象调用一个相同的方法得到不同的结果。

1. 什么是方法重载？

方法重载是指在一个类中定义多个同名的方法，但要求每个方法具有不同的参数的类型或参数的个数。

1. 简述抽象类和接口的异同。

1.一个类只能继承一个抽象类,一个类可以实现多个接口。

2.接口中不能含有静态代码块和静态方法,而抽象类可以。

1. 为什么Java程序在发生错误时，是抛出异常对象，而不是终止程序的执行？

程序在发生异常的时候就不会中断，我们可以对异常进行捕获，然后改变程序执行的流程。

**编程操作（必须做）**

1. 在聊天软件中，发送方发送消息时，遇到网络超时后就会自动重发，因此，接收方可能会收到重复的消息，在显示给用户看的时候，需要首先去重。请练习使用Set去除重复的消息：

import java.util.\*;

public class Main {

public static void main(String[] args) {

List<Message> received = List.of(

new Message(1, "Hello!"),

new Message(2, "发工资了吗？"),

new Message(2, "发工资了吗？"),

new Message(3, "去哪吃饭？"),

new Message(3, "去哪吃饭？"),

new Message(4, "Bye")

);

List<Message> displayMessages = process(received);

for (Message message : displayMessages) {

System.out.println(message.text);

}

}

static List<Message> process(List<Message> received) {

// TODO: 按sequence去除重复消息

return received;

}

}

class Message {

public final int sequence;

public final String text;

public Message(int sequence, String text) {

this.sequence = sequence;

this.text = text;

}

}

**import** java.util.\*;

**public** **class** Day2 {

**public** **static** **void** main(String[] args) {

List<Message> received = Arrays.*asList*(

**new** Message(1, "Hello!"),

**new** Message(2, "发工资了吗？"),

**new** Message(2, "发工资了吗？"),

**new** Message(3, "去哪吃饭？"),

**new** Message(3, "去哪吃饭？"),

**new** Message(4, "Bye")

);

List<Message> displayMessages = *process*(received);

**for** (Message message : displayMessages) {

System.***out***.println(message.text);

}

}

**static** List<Message> process(List<Message> received) {

// **TODO**: 按sequence去除重复消息

Set tem = **new** HashSet<>();

List <Message> unique = **new** ArrayList<>();

**int** i = 0;

**for** (i=0;i<received.size();i++) {

**if**(tem.add(received.get(i).sequence))

{

unique.add(received.get(i));

}

}

// return new ArrayList<>(uniqueMessages);

//return uniqueMessages;

**return** unique;

}

}

**class** Message {

**public** **final** **int** sequence;

**public** **final** String text;

**public** Message(**int** sequence, String text) {

**this**.sequence = sequence;

**this**.text = text;

}

}

1. 应用继承和多态的思想，编写动物类，成员方法是动物叫声。写三个具体的类（猫、狗、羊）， 它们都是动物类的子类，并重写父类的成员方法。编写测试类，随机产生三种具体动物，调用叫声这个方法。

（1）定义一个父类Animal类

属性：kind(种类)

方法：创建带参（kind为参数）构造方法

创建cry():void方法

（2）编写三个具体的子类Cat类、Dog类、Sheep类，分别重写父类中的cry() 方法，输出信息分别为

Cat类：小猫的叫声：喵喵喵~~~

Dog类：小狗的叫声：汪汪汪~~~

Sheep类：小羊的叫声：咩咩咩~~~

（3）编写测试类，首先生成长度为5的父类对象数组，然后通过循环依次向数组中存入数据，现设定存储规则为：

a) 每次随机产生一个0~2的正整数

b) 若数值为 0，则生成一个Cat类的对象，存入数组

c) 若数值为 1，则生成一个 Dog 类的对象，存入数组

d) 若数值为 2，则生成一个 Sheep 类的对象，存入数组

最后循环输出数组成员，并分别调用 cry() 方法。

**import** java.util.Scanner;

**public** **class** Day2 {

**public** **static** **void** main(String[] args) {

**int** i = 0;

**int** j = 0;

Animal[] animals = **new** Animal[5];

**for** (i = 0; i < 5; i++) {

j = (**int**) (Math.*random*() \* (2 - 0) + 1);

**if** (j == 0) {

animals[i] = **new** Cat();

} **else** **if** (j == 1) {

animals[i] = **new** Dog();

} **else** **if** (j == 2) {

animals[i] = **new** Sheep();

}

System.***out***.println(j);

}

**for** (Animal animal : animals) {

animal.cry();

}

}

}

**class** Animal {

**public** String kind;

**public** String getKind() {

**return** kind;

}

**public** Animal(String kind) {

**this**.kind = kind;

}

**public** **void** cry() {

System.***out***.println("动物的叫声");

}

}

**class** Cat **extends** Animal {

**public** Cat() {

**super**("猫");

}

**public** **void** cry() {

System.***out***.print("小猫的叫声：喵喵喵~~~\n");

}

}

**class** Dog **extends** Animal {

**public** Dog() {

**super**("狗");

}

**public** **void** cry() {

System.***out***.print("小狗的叫声：汪汪汪~~~\n");

}

}

**class** Sheep **extends** Animal {

**public** Sheep() {

**super**("羊");

}

**public** **void** cry() {

System.***out***.print("小羊的叫声：咩咩咩~~~\n");

}

}

1. 网络编程题目

要求：

（1）运行以下代码，要求展示相关命令行窗口。

（2）修改上述代码，在服务器端实现字符串反转。

服务器端

**import** java.net.\*;

**import** java.nio.charset.StandardCharsets;

**import** java.io.\*;

**public** **class** Server {

**public** **static** **void** main(String[] args) **throws** IOException {

ServerSocket ss = **new** ServerSocket(6666); // 监听指定端口

System.***out***.println("server is running...");

**for** (;;) {

Socket sock = ss.accept();

System.***out***.println("connected from " + sock.getRemoteSocketAddress());

Thread t = **new** Handler(sock);

t.start();

}

}

}

**class** Handler **extends** Thread {

Socket sock;

**public** Handler(Socket sock) {

**this**.sock = sock;

}

@Override

**public** **void** run() {

**try** (InputStream input = **this**.sock.getInputStream()) {

**try** (OutputStream output = **this**.sock.getOutputStream()) {

handle(input, output);

}

} **catch** (Exception e) {

**try** {

**this**.sock.close();

} **catch** (IOException ioe) {

}

System.***out***.println("client disconnected.");

}

}

**private** **void** handle(InputStream input, OutputStream output) **throws** IOException {

BufferedWriter writer = **new** BufferedWriter(**new** OutputStreamWriter(output, StandardCharsets.***UTF\_8***));

BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(input, StandardCharsets.***UTF\_8***));

writer.write("hello\n");

writer.flush();

**for** (;;) {

String s = reader.readLine();

**if** (s.equals("bye")) {

writer.write("bye\n");

writer.flush();

**break**;

}

writer.write("ok: " + s + "\n");

writer.flush();

}

}

}

客户端

**import** java.net.\*;

**import** java.nio.charset.StandardCharsets;

**import** java.util.Scanner;

**import** java.io.\*;

**public** **class** Client {

**public** **static** **void** main(String[] args) **throws** IOException {

Socket sock = **new** Socket("localhost", 6666); // 连接指定服务器和端口

**try** (InputStream input = sock.getInputStream()) {

**try** (OutputStream output = sock.getOutputStream()) {

*handle*(input, output);

}

}

sock.close();

System.***out***.println("disconnected.");

}

**private** **static** **void** handle(InputStream input, OutputStream output) **throws** IOException {

BufferedWriter writer = **new** BufferedWriter(**new** OutputStreamWriter(output, StandardCharsets.***UTF\_8***));

BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(input, StandardCharsets.***UTF\_8***));

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.println("[server] " + reader.readLine());

**for** (;;) {

System.***out***.print(">>> "); // 打印提示

String s = scanner.nextLine(); // 读取一行输入

writer.write(s);

writer.newLine();

writer.flush();

String resp = reader.readLine();

System.***out***.println("<<< " + resp);

**if** (resp.equals("bye")) {

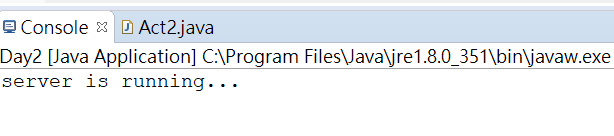
**break**;

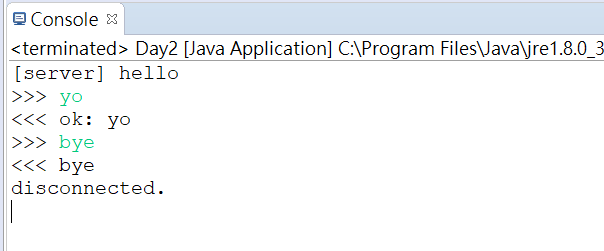
}

}

}

}





服务器端

**import** java.net.\*;

**import** java.nio.charset.StandardCharsets;

**import** java.io.\*;

**public** **class** Day2 {

**public** **static** **void** main(String[] args) **throws** IOException {

ServerSocket ss = **new** ServerSocket(6668); // 监听指定端口

System.***out***.println("server is running...");

**for** (;;) {

Socket sock = ss.accept();

System.***out***.println("connected from " + sock.getRemoteSocketAddress());

Thread t = **new** Handler(sock);

t.start();

}

}

}

**class** Handler **extends** Thread {

Socket sock;

**public** Handler(Socket sock) {

**this**.sock = sock;

}

@Override

**public** **void** run() {

**try** (InputStream input = **this**.sock.getInputStream()) {

**try** (OutputStream output = **this**.sock.getOutputStream()) {

handle(input, output);

}

} **catch** (Exception e) {

**try** {

**this**.sock.close();

} **catch** (IOException ioe) {

}

System.***out***.println("client disconnected.");

}

}

**private** **void** handle(InputStream input, OutputStream output) **throws** IOException {

BufferedWriter writer = **new** BufferedWriter(**new** OutputStreamWriter(output, StandardCharsets.***UTF\_8***));

BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(input, StandardCharsets.***UTF\_8***));

writer.write("hello\n");

writer.flush();

**for** (;;) {

String s = reader.readLine();

**if** (s.equals("bye")) {

writer.write("bye\n");

writer.flush();

**break**;

}

// 将用户输入的字符串反转

String reversedString = **new** StringBuilder(s).reverse().toString();

writer.write( reversedString + ":ko "+"\n"); // 将反转后的字符串发送给客户端

writer.flush();

}

}

}

客户端

**import** java.net.\*;

**import** java.nio.charset.StandardCharsets;

**import** java.util.Scanner;

**import** java.io.\*;

**public** **class** Day2 {

**public** **static** **void** main(String[] args) **throws** IOException {

Socket sock = **new** Socket("localhost", 6668); // 连接指定服务器和端口

**try** (InputStream input = sock.getInputStream()) {

**try** (OutputStream output = sock.getOutputStream()) {

*handle*(input, output);

}

}

sock.close();

System.***out***.println("disconnected.");

}

**private** **static** **void** handle(InputStream input, OutputStream output) **throws** IOException {

BufferedWriter writer = **new** BufferedWriter(**new** OutputStreamWriter(output, StandardCharsets.***UTF\_8***));

BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(input, StandardCharsets.***UTF\_8***));

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.println("[server] " + reader.readLine());

**for** (;;) {

System.***out***.print(">>> "); // 打印提示

String s = scanner.nextLine(); // 读取一行输入

// 将用户输入的字符串反转

String reverseString = **new** StringBuilder(s).reverse().toString();

writer.write(reverseString);

writer.newLine();

writer.flush();

String resp = reader.readLine();

// 将服务器回应的字符串反转

String reverseResp = **new** StringBuilder(resp).reverse().toString();

System.***out***.println("<<< " + reverseResp);

**if** (reverseResp.equals("bye")) { // 将 "bye" 反转为 "eyb" 进行比较

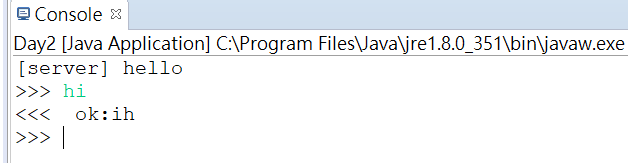
**break**;

}

}

}

}



**编程操作（选做）**

1. 编写一个USB接口程序，模拟计算机启动过程和关闭过程中药加载鼠标、键盘、麦克风等USB设备，具体要求如下：

（1）定了一个接口USB，包含两个抽象方法turnOn()和turnOff()方法，分别用于表示USB设备的启动和关闭；

（2）编写鼠标Mouse、键盘Keyboard、麦克风Mic类，实现接口中的turnOn()和turnOff()方法，方法中显示”XX设备启动了”或”XX设备关闭了”即可；

（3）编写计算机类Computer，要求：

* + 1. 属性：计算机最多能支持USB设备个数nMax和存储所连接USB设置的数组devices
    2. 添加USB设备的方法add(USB usb)，功能为遍历所有插槽，如果有空闲就添加一个USB设置
    3. 模拟开机启动USB设备的powerOn()方法，功能为遍历所有USB插槽，如果连接USB设备，则启动USB设备，然后显示“计算机开机成功”
    4. 模拟关机关闭USB设备的powerOff()方法，功能为遍历所有USB插槽，如果连接USB设备，则关闭USB设备，然后显示“计算机关机成功”

（4）编写测试类，要求建立计算机对象，建立鼠标、键盘、麦克风对象，并添加到计算机中，启动计算机，关闭计算机。