**基础题**

### 练习一：线程池概念

问题：

请描述什么是线程池。

答：

线程池：其实就是一个容纳多个线程的容器，其中的线程可以反复使用，省去了频繁创建线程对象的操作，无需反复创建线程而消耗过多资源。

### 练习二：线程池优点

问题：

请描述合理利用线程池能够带来的三个好处。

答：

1. 降低资源消耗。减少了创建和销毁线程的次数，每个工作线程都可以被重复利用，可执行多个任务。

2. 提高响应速度。当任务到达时，任务可以不需要的等到线程创建就能立即执行。

3. 提高线程的可管理性。可以根据系统的承受能力，调整线程池中工作线线程的数目，防止因为消耗过多的内存，而把服务器累趴下(每个线程需要大约1MB内存，线程开的越多，消耗的内存也就越大，最后死机)。

### 练习三：Lambda表达式

问题：

请列举Lambda语法的省略规则

答：

在Lambda标准格式的基础上，使用省略写法的规则为：

1. 小括号内参数的类型可以省略；
2. 如果小括号内有且仅有一个参，则小括号可以省略；
3. 如果大括号内有且仅有一个语句，则无论是否有返回值，都可以省略大括号、return关键字及语句分号。

### 练习四：Lambda表达式

问题：

请列举Lambda表达式的3个组成部分，并解释说明。

答：

Lambda标准格式 Lambda省去面向对象的条条框框，格式由3个部分组成：一些参数，一个箭头，一段代码。

Lambda表达式的标准格式为：格式说明：

1. 小括号内的语法与传统方法参数列表一致：无参数则留空；多个参数则用逗号分隔。
2. ->是新引入的语法格式，代表指向动作。
3. 大括号内的语法与传统方法体要求基本一致。

### 练习五：Lambda表达式

问题：

请描述Lambda的使用前提

答：

Lambda的语法非常简洁，完全没有面向对象复杂的束缚。但是使用时有几个问题需要特别注意：

1. 使用Lambda必须具有接口，且要求接口中有且仅有一个抽象方法。无论是JDK内置的 Runnable 、 Comparator 接口还是自定义的接口，只有当接口中的抽象方法存在且唯一时，才可以使用Lambda。

2. 使用Lambda必须具有上下文推断。也就是方法的参数或局部变量类型必须为Lambda对应的接口类型，才能使用Lambda作为该接口的实例。

### 练习六：多线程

问题：

请在指定位置插入代码实现打印输出1-99。

|  |
| --- |
| **public class** Test06 { **public int start** = 1; **public int end** = 99; **public static void** main (String[] args) { **new** Test06().method();  } **public void** method() { *//请在此处插入代码，实现功能* Thread t = **new** Thread( a );  t.start();  } } |

答：

|  |
| --- |
| **public class** Test06 { **public int start** = 1; **public int end** = 99;  **public static void** main(String[] args) { **new** Test06().method();  }  **public void** method() { *//插入代码处* Runnable **a** = () -> {  **for** (**int** i = **start**; i <**end**; i++) {  System.***out***.println(i);  }  };  Thread t = **new** Thread(a);  t.start();  } } |
|  |

### 练习七：多线程

问题：给出以下代码，请问该程序的运行结果是什么?如有问题，请说明原因。

|  |
| --- |
| **public class** Test07**implements** Runnable { **public static void** main(String[] args) {  Thread t = **new** Thread(**new** Test07());  t.start();  }  **public void** run(**int** num) { **for** (**int** i = 0; i < num; i++) {  System.***out***.println(i);  }  } } |

答：

在编译时期就会报错

Test类没有重写Runnable接口中的run()方法

public void run(int num)不是Runnable接口中的run()方法。

**注意：Runnable接口中的run()方法，参数列表为空，不带参数。**

### 练习八：线程池练习

问题：

使用线程池创建多线程。模拟同学找老师学习Java。

1. 创建线程池对象，包含2个线程。从线程池中获取线程对象，然后调用MyRunnable中的run()。
2. 在MyRunnable实现类中，首先在控制台打印需求，“我需要一个老师”。模拟需要2秒钟时间老师可以过来指导学生，并在控制台打印老师的姓名。最后，在控制台打印“教我java,教完后，老师回到了办公室”；

答：

*/\*  
 \*1.Runnable实现类代码：  
 \*/*

|  |
| --- |
| **import** java.util.concurrent.ExecutorService; **import** java.util.concurrent.Executors;  **public class** MyRunnable **implements** Runnable { @Override **public void** run() {  System.***out***.println(**"我要一个老师"**); **try** {  Thread.*sleep*(2000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println(**"老师来了"**+Thread.*currentThread*().getName());  System.***out***.println(**"****教我java,教完后，老师回到了办公室"**);  } } |

*/\*  
 \*2.线程池测试类：  
 \*/*

|  |
| --- |
| **import** java.util.concurrent.ExecutorService; **import** java.util.concurrent.Executors;  **public class** ThreadPoolDemo { **public static void** main(String[] args) { *// 创建线程池对象* ExecutorService service = Executors.*newFixedThreadPool*(2);*//包含2个线程对象 // 创建Runnable实例对象* MyRunnable r = **new** MyRunnable(); *//自己创建线程对象的方式 // Thread t = new Thread(r); // t.start(); ‐‐‐>调用MyRunnable中的run() // 从线程池中获取线程对象,然后调用MyRunnable中的run()* service.submit(r); *// 再获取个线程对象，调用MyRunnable中的run()* service.submit(r);  service.submit(r); *// 注意：submit方法调用结束后，程序并不终止，是因为线程池控制了线程的关闭。 // 将使用完的线程又归还到了线程池中 // 关闭线程池 //service.shutdown();* } } |

### 练习九：Lambda（无参无返回）练习

问题：

给定一个导演 Director接口，内含唯一的抽象方法makeMovie，且无参数、无返回值，使用lambda表达式在Test中完成调用。

|  |
| --- |
| **public interface** Director { **void** makeMovie(); } |

在下面的代码中，请使用Lambda的省略格式调用 invokeDirect 方法，打印输出“导演拍电影啦！”字样：

|  |
| --- |
| **public class Test**09InvokeDirect { **public static void** main(String[] args) { *//* ***TODO 请使用Lambda【省略格式】调用invokeDirect方法***  }  **private static void** invokeDirect(Director director) {  director.makeMovie();  } } |

答：

* **使用Lambda【省略格式】调用invokeDirect方法**

|  |
| --- |
| **public class Test**09InvokeDirect { **public static void** main(String[] args) { *// 2.****在此使用Lambda【省略格式】调用invokeDirect方法*** *invokeDirect*(() -> System.***out***.println(**"导演拍电影啦！"**));  }  **private static void** invokeDirect(Director director) {  director.makeMovie();  } } |

### 练习十：Lambda（有参有返回）练习

问题：

给定一个计算器 Calculator 接口，内含抽象方法 calc (减法)，其功能是可以将两个数字进行相减，并返回差值。使用Lambda表达式在Test中完成调用。

|  |
| --- |
| **public interface** Calculator { **int** calc(**int** a, **int** b); } |

在下面的代码中，请分别使用Lambda的标准格式及省略格式调用 invokeCalc 方法，完成130和120的相减计算：

|  |
| --- |
| **public class Test10**InvokeCalc { **public static void** main(String[] args) { *//* ***TODO 请分别使用Lambda【标准格式】及【省略格式】调用invokeCalc方法来计算130-120的结果ß***   }  **private static void** invokeCalc(**int** a, **int** b, Calculator calculator) { **int** result = calculator.calc(a, b);  System.***out***.println(**"结果是："** + result);  } } |

答：

* **1.使用Lambda【标准格式】调用invokeCalc方法来计算130-120的结果ß**

|  |
| --- |
| **public class Test10**InvokeCalc { **public static void** main(String[] args) { *// 1.****在此使用Lambda【标准格式】调用invokeCalc方法来计算130-120的结果ß*** *invokeCalc*(130, 120, (**int** a, **int** b) -> { **return** a - b;  });  }  **private static void** invokeCalc(**int** a, **int** b, Calculator calculator) { **int** result = calculator.calc(a, b);  System.***out***.println(**"结果是："** + result);  } } |

* **2.使用Lambda【省略格式】调用invokeCalc方法来计算130-120的结果ß**

|  |
| --- |
| **public class Test10**InvokeCalc { **public static void** main(String[] args) { *// 2.****在此使用Lambda【省略格式】调用invokeCalc方法来计算130-120的结果ß*** *invokeCalc*(130, 120, (a, b) -> a - b);  }  **private static void** invokeCalc(**int** a, **int** b, Calculator calculator) { **int** result = calculator.calc(a, b);  System.***out***.println(**"结果是："** + result);  } } |