第2章 Java SE 核心 API

内容导读

本章重点是异常处理、多线程基础、I/O 系统、网络编程、Java 反射机制、JVM 性能调优（JVM 内存结构剖析、GC 分析及调优、JVM 内存参数优化）、Java 泛型及 JDK（Java 开发工具包）新特性。本章要求初步具备面向对象设计和编程的能力，掌握基本的 JVM 优化策略，熟练掌握 Java SE 核心内容（特别是 I/O 和多线程）。

2.1 常用 API

API 指应用程序编程接口。有经验的开发人员知道，厂商一定会提供一些用于实现某种功能的 Java 类。其实，这些 Java 类就是厂商提供给应用程序编程的接口，类中定义了各种方法，把这些类称为 API。本章涉及的 Java API 指的就是 JDK 中提供的各种功能的 Java 类、接口及其对应的方法等。

2.1.1 字符串

本节主要的考核点是 Java 常用的字符串 API，如 String、StringBuffer、StringBuilder。重点掌握这三类 API 的特点、作用和区别，以及 StringTokenizer 的基本知识点。

1. 关于以下程序段，正确的说法是（　　）。

```
1.String s1 = "abc" + "def";
2.String s2 = new String(s1);
3.if (s1 == s2)
4.  System.out.println("== succeeded");
5.if (s1.equals(s2))
6.  System.out.println(".equals() succeeded");
```

A. 行 4 与行 6 都将执行
B. 行 4 执行，行 6 不执行
C. 行 6 执行，行 4 不执行
D. 行 4、行 6 都不执行

解释：String 重写了 Object 的 equals() 方法，重写的这个方法用于比较字符串的内容。而“==”比较的是内存地址，对于字符串来说，一般用 equals() 比较两个字符串的内容是否一样，而用“==”比较两个字符串变量是否指向同一个字符串值。s2 是新创建的对象，与 s1 的内存地址是不同的，因此执行行 6。

答案：C

2. 在 Java 中，字符串由 java.lang.String 和（　　）定义。

A. java.io.StringChar
B. java.io.StringBuffer
C. java.lang.StringChar
D. java.lang.StringBuffer

解释：在 Java 中，字符串是作为对象出现的，由 java.lang.String 和 java.lang.StringBuffer 及 java.lang.StringBuilder 定义，分别用来处理长度不变和长度可变的字符串，这三个类都被定义为 final。

答案：D

3. 以下代码将打印（　　）。

```
public static void main(String[] args) {
    String classFile = "com. jd. ".
replaceA11(".", "/") + "MyClass.class";
    System.out.println(classFile);
}
```

A. com.jd
B. com/jd/MyClass.class
C. ///////MyClass.class
D. com.jd.MyClass

解释：replaceAll() 方法的第一个参数是一个正则表达式，而“.”在正则表达式中表示任何字符，所以会把前面字符串的所有字符都替换成“/”。如果想替换的只是“.”，就要写成“\.”。

答案：C

4. 若有 byte[] src,dst;，下面（　　）程序能够正确地实现 GBK 编码字节流到 UTF-8 编码字节流的转换。

A. dst=String.fromBytes(src,"GBK").getBytes("UTF-8")
B. dst=new String(src,"GBK").getBytes("UTF-8")

C. dst=new String("GBK",src).getBytes()

D. dst=String.encode(String.decode(src,"GBK"), "UTF-8")

解释：选 B 选项，先通过 GBK 编码字节流还原字符串，在该字符串正确的基础上得到 UTF-8 编码字节流所对应的字节串。

答案：B

5. Java 中将 ISO8859-1 字符串转成 GB2312 编码的语句是（　　）。

A. new String(String.getBytes("ISO8859-1"),GB2312)

B. new String(String.getBytes("GB2312"), ISO8859-1)

C. new String(String.getBytes("ISO8859-1"))

D. new String(String.getBytes("GB2312"))

解释：这里“ISO8859-1”是一个普通字符串，指的是一种编码方式，不要被它迷惑了。String.getBytes("ISO8859-1") 表示获取这个字符串的 byte 数组。new String(String.getBytes("ISO8859-1"),GB2312) 使上面的字符数组按照 GB2312 编码成新的字符串。

答案：A

6. String 与 StringBuffer 的区别是（　　）。

A. String 是不可变的对象，StringBuffer 是可以再编辑的

B. String 用于创建字符串常量

C. StringBuffer 用于创建字符串变量

D. 以上说法都对

解释：String 是不可变类型，StringBuffer 是可变类型，String 可以进行常量赋值，如 String str="123"，也可以使用 new 运算操作创建对象。StringBuffer 只能通过 new 运算创建对象。

答案：D

7. 下列 Java 程序的输出结果为（　　）。

```
public class Example {
    String str = new String("hello");
    char[] ch = { 'a', 'b' };
    public static void main(String args[]) {
        Example ex = new Example();
        ex.change(ex.str, ex.ch);
        System.out.print(ex.str + " and ");
        Sytem.out.print(ex.ch);
    }
    public void change(String str, char ch[]) {
        str = "test ok";
        ch[0] = 'c';
    }
}
```

A. hello and ab　　B. hello and cb

C. hello and a　　D. test ok and ab

解释：String 类是 final 类型的，不能继承和修改这个类。str="test ok"，其实是隐含地让 Java 生成一个新的 String 对象，与原来的“Hello”没有任何关系，当方法结束时，str 作用结束，输出的还是“Hello”。char ch[] 是传递引用，修改了原内容。

答案：B

8. 下面程序的运行结果是（　　）。

```
String str1 = "hello";
String str2 = "he" + new String("llo");
System.err.println(str1 == str2);
```

A. true　　B. false

C. exception　　D. 无输出

解释：“==”用于比较对象的引用是否一致（实际上就是比较两者的 hash code，通过各自的 hashCode() 方法生成），str1 和 str2 尽管内容相同，但却是两个对象，所以为 false。

答案：B

9. StringBuffer 类对象创建之后可以再修改和变动。(　　)(2017 年，德邦)

解释：String 对象不可变，StringBuffer 对象可变。例如，String str = "aa"; str = "aa"+"bb"，此时 str 的值为“aabb”，但是“aabb”不是在开始的字符串“aa”后面直接连接的“bb”，而是又新生成了字符串“aabb”，字符串“aa”一旦被初始化，它的值不可能再改变了。StringBuffer strb = StringBuffer("aa"); strb.append("bb")，此时的 strb 的值也为“aabb”，但是“aabb”是直接在开始的字符串“aa”后面连接的“bb”，并没有生成新的字符串。

答案：正确

2.1.2 日期时间

本节主要考核 Java 常用的日期 API，如 Date、SimpleDateFormat、Formatter、Calendar、LocalDateTime、Duration、Instant、DateTimeFormatter。重点掌握这 8 种 API 的特点与作用，了解获取时间戳的常用方法与方法效率。

1. 下面将 Calendar 转换为 Date 的代码是否正确？(　　)

```
Calendar cal=Calendar.getInstance();
Date date=cal.getTime();
```

解释：Calendar 对象的 getTime() 返回的是一个 Date 对象。

答案：正确

2. 运行下面的程序段（2008 年为闰年，2 月有 29 天），控制台的输出结果为(　　)。

```
Calendar c = Calendar.getInstance();
c.set(Calendar.YEAR,2008);
c.set(Calendar.MONTH,1);
c.set(Calendar.DATE,32);
SimpleDateFormat sdf=new SimpleDateFormat
("yyyy/M/dd");
System.out.println(sdf.format(c.
getTime()));
```

A. 2008/1/01　　B. 2008/3/03

C. 2008/2/1　　D. 2008/3/01

解释：因为 2008 年是闰年，所以 2 月有 29 日，而 32 日已经超出了 2 月的日期了，所以顺延到下一个月，此处 2 月 32 日相当于 2 月 29 日 +3 天，此时是 3 月 3 日。

YYYY 和 yyyy 的区别：YYYY 是 week-based-year，当天所在的周属于的年份，一周从周日开始，到周六结束；只要本周跨年了，本周就算入下一年。yyyy 是自然年。

HH 和 hh 的区别：HH 是 24 小时制，hh 是 12 小时制。MM 和 M、dd 和 d、HH 和 H、mm 和 m、ss 和 s 都是相同的，区别为是否有前导零：M、d、H、m、s 表示非零开始；MM、dd、HH、mm、ss 表示从零开始。

答案：B

3. 以下代码中(　　)可以实现功能：启动线程不断打印当前时间，每隔 1s 打印一次，时间格式如 2019-12-29 09:09:09。

```
class TimerPrinter extends TimerTask {
< 插入代码 >}
```

A.

```
public void run() {
    DateFormat sdf = new DateFormat
    (" yyyy--MM--dd hh:mm:ss ");
    System.out.println(sdf.format(new
    Date()));
}
public static void main(String args[]) {
     Timer t = new Timer();
```

```
    t.schedule(new TimerPrinter(), 0, 1000);
}
```

B.

```
public void run() {
   DateFormat sdf = new DateFormat
   (" yyyy·-MM-dd hh:mm:ss");
    System.out.println(sdf.format(new Date()));
}
public static void main(String args[]) {
    Timer t = new Timer();
    t.schedule(new TimerPrinter(), 0, 1);
}
```

C.

```
public void run() {
   SimpleDateFormat sdf = new
   SimpleDateFormat(" yyyy-MM-dd hh:mm:ss");
   System.out.println(sdf.format(new
   Date()));
}
public static void main(String args[]) {
    Timer t = new Timer();
    t.schedule(new TimerPrinter(), 0, 1000);
}
```

D.

```
public void run() {
   SimpleDateFormat sdf = new
   SimpleDateFormat(" yyyy--MM-dd hh:mm:ss");
   System.out.println(sdf.format(new
   Date()));
}
public static void main(String args[]) {
    Timer t = new Timer();
    t.schedule(new TimerPrinter(), 0, 1);
}
```

解释：本题的关键性语句是 SimpleDateFormat sdf = new SimpleDateFormat(" yyyy-MM-dd hh:mm:ss")，因此要考虑表达式的书写。2019-12-29 09:09:09 应该选择 yyyy-MM-dd hh:mm:ss。

答案：C

4. Calendar 类中的 DAY_OF_WEEK 可以获取（　　）。

A. 年中的某一天

B. 月中的某一天

C. 星期中的某一天

D. 月中的最后一天

解释：本题考查记忆型知识点。从字面也很容易猜到其作用。

答案：C

2.1.3 System（in/out）

System.in 是标准输入流；System.out 是标准输出流。System.in 字节输入流对应的 IO 设备为键盘；System.out 对应的 IO 设备为控制台。本节重点掌握 System.in 与 System.out 的使用，输入流 / 输出流及其包含的多个流的概念、特征与常用方法。

阅读以下程序，如果输出结果的第二行为 bb=a，那么第一行的输出是（　　）。

```
public class A{
    public static void main(String args[]){
        char a = 'm';
        int i=100;
        int j=97;
        int aa=a+i;
        System.out.println("aa="+aa);
        char bb=(char)j;
        System.out.println("bb="+bb);
    }
}
```

A. aa=1　　B. aa=204
C. aa=m　　D. aa=209

解释：bb=a 意味着 int 型的 97 对应 char 型的 a，m 是第 13 位字母，m-a=12，12+97 对应 char 型的 m，aa=12+97+100=209。

答案：D

2.1.4 自动装箱和拆箱

在 Java 中，基本类型与其对应封装类之间能够自动进行转换，其本质是 Java 的自动装箱（auto-boxing）和拆箱过程（auto-unboxing）。其中，装箱是指将基本类型数据值转换成对应的封装类对象，即将栈中的数据封装成对象存放到堆中的过程；拆箱是装箱的反过程，是将封装的对象转换成基本类型数据值，即将堆中的数据值存放到栈中的过程。本节重点掌握装箱与拆箱的异同，以及装箱和拆箱与封装的联系。

1. 在 JDK 1.5 之后，下列 Java 程序的输出结果为（　　）。（2016 年，阿里巴巴）

```
int i=0;
Integer j = new Integer(0);
System.out.println(i==j);
System.out.println(j.equals(i));
```

A. true，false　　B. true，true
C. false，true　　D. false，false

解释：对于引用类型数据，“==”直接比较两个对象的内存地址，如果相等，则说明这两个引用实际上是指向同一个对象地址。equals() 比较的是对象的内容。对于基本数据的封装类型（Byte、Short、Character、Integer、Float、Double、Long、Boolean），除了 Float 和 Double 之外，其他的都是实现了常量池的，因此对于这些数据类型而言，一般也可以直接通过“==”来判断是否相等。

答案：B

2. 以下对拆箱和装箱的描述错误的是（　　）。
A. Java 中的基本数据类型有以下几种：String、int、char、byte、short、double、long、float
B. 装箱是基本数据类型转为包装类
C. 拆箱是由包装类转为基本数据类型
D. 以上说法都不对

解释：Java 中的基本数据类型有 8 种，不包括 String，应该是 byte、short、int、long、float、double、char、boolean。

答案：A

3. 以下关于装箱和拆箱的说法错误的是（　　）。
A. 装箱是指将基本类型数据值转换成对应的封装类对象
B. 装箱是将栈中的数据封装成对象存放到堆中的过程
C. 拆箱是指将封装的对象转换成基本类型数据值
D. 拆箱是指将基本类型数据值转换成对应的封装类对象

解释：基本类型与其对应封装类之间能够自动进行转换，其本质是 Java 的自动装箱和拆箱过程。

装箱是指将基本类型数据值转换成对应的封装类对象，即将栈中的数据封装成对象存放到堆中的过程。所以 D 选项错误。

拆箱是装箱的反过程，是指将封装的对象转换成基本类型数据值，即将堆中的数据值存放到栈中的过程。

答案：D

4. Java 中拆箱是指将基本数据类型的对象转

为引用数据类型。(　　)

解释：拆箱是装箱的反过程，是指将封装的对象转换成基本类型数据值，即将堆中的数据值存放到栈中的过程。

答案：错误

2.2 集合 API

本节主要考核 Java 中常用的集合 API。从大的分类来说，集合 API 分成 Collection 和 Map 两种。其中，Collection 包含三种常用集合，分别是 Set、List、Queue；Map 包含两种常用集合，分别是 HashMap、TreeMap。这些集合 API 都需要掌握，涉及内容有各种集合的常用方法、优缺点，以及各种集合之间的区别等。

2.2.1 Collection

本节主要考核 Collection 的相关概念。Collection 可以分为 Set、List、Queue 三种类型。Map 是不属于 Collection 的，Map 是一个独立的数据结构。Collection 和 Map 在实现上又互相依赖。本节需要掌握 Collection 的基本方法、接口及迭代器的使用。

1. 没有实现或继承 Collection 接口的是(　　)。

A. List　　B. Vector

C. Iterator　　D. Set

解释：A 选项，List 接口的定义为 public interface List<E> extends Collection<E>。

B 选项，Vector 的定义为 public class Vector<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable。Vector 实现了 List 接口，自然实现了 Collection 接口。

C 选项，Iterator 接口未实现 Collection 接口。

D 选项，public interface Set<E> extends Collection<E>，Set 接口继承自 Collection 接口。

答案：C

2. 运行以下代码将打印出(　　)。

```
public static void main(String[] args) {
    List list1 = new ArrayList();
    list1.add(0);
    List list2 = list1;
    System.out.println(list1.get(0)
    instanceof Integer);
    System.out.println(list2.get(0)
    instanceof Integer);
}
```

A. 编译错误　　B. true true

C. true false　　D. false false

解释：Collection 类型的集合（ArrayList、LinkedList）只能装入对象类型的数据，该题中装入了 0，是一个基本类型，但是 JDK 5 以后提供了自动装箱与自动拆箱，所以 int 类型自动装箱变为 Integer 类型，编译能够正常通过。将 list1 的引用赋值给 list2，那么 list1 和 list2 都将指向同一个堆内存空间。instanceof 是 Java 中的关键字，用于判断一个对象是否属于某个特定类的实例，并且返回 boolean 型的返回值。显然，list1.get(0) 和 list2.get(0) 都属于 Integer 的实例。

答案：B

3. 下面有关 List 接口、Set 接口和 Map 接口的描述，错误的是(　　)。

A. 它们都继承自 Collection 接口

B. List 是有序的 Collection，使用此接口能够精确地控制每个元素的插入位置

C. Set 是一种不包含重复元素的 Collection

D. Map 提供 key 到 value 的映射。一个 Map 中不能包含相同的 key，每个 key

只能映射一个 value

解释：Map 接口和 Collection 接口是同一等级的。

答案：A

4. 下面直接继承自 Collection 接口的接口有（　　）。（2017 年，德邦）

A. List　　B. Map

C. Set　　D. Iterator

解释：直接继承自 Collection 接口的接口有 Set、List、Queue。

答案：AC

2.2.2 泛型和增强泛型

Java 泛型（generics）是 JDK 5 中引入的一个新特性，泛型提供了编译时类型安全监测机制，该机制允许程序员在编译时监测非法的类型。使用泛型机制编写的程序代码要比那些杂乱地使用 Object 变量，然后再进行强制类型转换的代码具有更好的安全性和可读性。泛型对集合类尤其有用。例如，ArrayList 就是一个无处不在的集合类。本节重点掌握泛型的使用（具体指泛型类、泛型接口和泛型方法这三种常用的使用方式），以及无边界通配符的使用与含义。

1. 集合框架中的泛型有什么优点？

参考答案：Javase 1.5 引入了泛型，所有的集合接口和实现都大量地使用它。使用泛型主要有以下优点。

（1）泛型可以为集合提供一个可以容纳的对象类型，因此，如果添加其他类型的任何元素，会在编译时报错。这避免了在运行时出现 ClassCastException 错误，因为在编译时就会出现报错信息。

（2）泛型使得代码整洁，不需要使用显式转换和 instanceof 操作符。

（3）泛型给运行时带来好处，因为不会产生类型检查的字节码指令。

2. 泛型中的无边界通配符是（　　）。

A. <? extends T>　　B. <? super T>

C. <T extends ?>　　D. <?>

解释：通配符类型如下。

- 无边界通配符：<?>，使用无边界通配符可以让泛型接收任意类型的数据。
- 上边界通配符：<？ extends 具体类型 >，使用固定上边界的通配符的泛型可以接收指定类型及其所有子类类型的数据，这里的指定类型可以是类，也可以是接口。
- 下边界通配符：<? super 具体类型 >，所有固定下边界的通配符的泛型可以接收指定类型及其所有超类类型的数据。

答案：D

3. 创建一个只能存放 String 的泛型 ArrayList 的语句是（　　）。

A. ArrayList<int> al=new ArrayList<int>()

B. ArrayList<String> al=new ArrayList<String>()

C. ArrayList al=new ArrayList<String>()

D. ArrayList<String> al =new List<String>()

解　释：List<String> list = new ArrayList <String>();

两个 String 其实只有第一个起作用，另外，从 JDK 7 开始还支持 List<String>list = new ArrayList<>() 这种写法。List<String> list = new ArrayList<String>() 的第一个 String 就是告诉编译器，List 中存储的是 String 对象，也就是起类型检查的作用，之后编译器会擦除泛型占位符，以保证兼容以前的代码。

答案：B

4. 以下关于泛型的说法错误的是（　　）。

A. 泛型是 JDK 5 出现的新特性

B. 泛型是一种安全机制

C. 使用泛型避免了强制类型转换

D. 使用泛型必须进行强制类型转换

解释：用了泛型之后就不要强制转换。使用了泛型后就会在编译期进行类型检测。例如，对于 List<String>，如果再往 List 中添加不是 String 的对象，就会在编译期报错，因此在运行期就注定了从 List 里取出来的必须是 String，当然就不必进行强制转换。

答案：D

5. 以下关于 Java 中泛型的说法错误的有（　　）。

A. List<? extends T> 为可以接收任何继承自 T 类型的 List

B. 方法可以返回泛型类型

C. 可以把 List<String> 传递给一个接收 List<Object> 参数的方法

D. Array 中可以用泛型

解释：C 选项，泛型是确定元素的类型，所以，一旦确定就不能传入其他类型。D 选项，Java 中的数组不支持泛型，集合却支持泛型。

答案：CD

6. 在开发中，泛型简化了编程、提高了开发效率，以下说法正确的有（　　）。（2017 年，京东）

A. 泛型是程序设计语言的一种特性，所有语言均有

B. 泛型类是引用类型

C. 泛型可以加强类型安全和减少类型转换的次数

D. Java 泛型可以让程序运行速度因为类型转换次数减少而加快

解释：A 选项，泛型是程序设计语言的一种特性，允许程序员在强类型编程语言中编写代码时定义一些可变部分，那些部分在使用前必须确定。各种程序设计语言和其编译器、运行环境对泛型的支持均不一样。D 选项，Java 泛型的参数只可以代表类，不能代表个别对象。由于 Java 泛型的类型参数之实际类型在编译时会被消除，所以无法在运行时得知其类型参数的类型。Java 编译器在编译泛型时会自动加入类型转换的编码，故运行速度不会因为使用泛型而加快。

答案：BC

2.2.3 List/Vector/Stack

List 接口是有序的队列。Vector 可以实现可增长的对象数组。Stack 是 Vector 的一个子类，它实现一个标准的后进先出的栈。本节要熟练掌握 List、Vector、Stack 这三个集合的特征，基于 Java 语言的常用语法、常用方法，以及各集合间的区别。

1. ArrayList list = new ArrayList(20) 中的 list 扩容（　　）次。

A. 0　　B. 1　　C. 2　　D. 3

解释：ArrayList 的构造方法有以下三个。

（1）ArrayList()，生成一个初始容量为 10 的空列表。

（2）ArrayList(Collection<? extends E> c)，生成一个包含指定 Collection 的元素的列表，长度为 c。

（3）ArrayList(int initialCapacity)，生成一个具有指定初始容量的空列表。

这里调用的是第三个构造方法，直接初始化成大小为 20 的列表，没有扩容，所以选择 A 选项。

答案：A

2. ArrayList 和 Vector 的主要区别是（　　）。

A. Vector 与 ArrayList 一样，也是通过数组实现的，不同的是 Vector 支持线程的同步

B. Vector 与 ArrayList 一样，也是通过数组实现的，不同的是 ArrayList 支持线程的同步

C. Vector 通过链表结构存储数据，ArrayList 通过数组存储数据

D. 上述说法都不正确

解释：Vector 支持线程的同步，也就是内部加锁的，但是效率较低。

答案：A

3. 以下（　　）类是非线程安全的。

A. Vector　　B. ArrayList

C. StringBuffer　　D. Properties

解释：A 选项，Vector 相当于一个线程安全的 List。B 选项，ArrayList 是非线程安全的，其对应的线程安全类是 Vector。C 选项，StringBuffer 是线程安全的，相当于一个线程安全的 StringBuilder。D 选项，Properties 实现了 Map 接口，是线程安全的。

答案：B

4. 下列关于 Arraylist、Vector、LinkedList 的说法不正确的是（　　）。

A. ArrayList 和 Vector 都是使用数组方式存储数据，此数组的元素个数大于实际存储的元素个数，以便增加和插入元素

B. ArrayList 和 Vector 都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢

C. Vector 由于使用了 synchronized 方法（线程安全），通常性能上较 ArrayList 强

D. LinkedList 使用双向链表实现存储，按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快

解释：ArrayList 和 Vector 都是使用数组方式存储数据，此数组的元素个数大于实际存储的元素个数，以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢，Vector 由于使用了 synchronized 方法（线程安全），通常性能上较 ArrayList 差，而 LinkedList 使用双向链表实现存储，按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快。

答案：C

5. 针对下面选项中的几种集合结构，（　　）在增加成员时可能使得原有成员的存储位置发生变化。

A. List　　B. Set

C. Map　　D. Vector

解释：Vector 是集合架构中最常见的容器，它是一种顺序容器，支持随机访问。Vector 是一块连续分配的内存，从数据安排的角度来讲，与数组极其相似，不同之处是，数组是静态分配空间，一旦分配了空间的大小，就不可再改变了；Vector 是动态分配空间，随着元素的不断插入，会按照自身的一套机制不断扩充自身的容量，按照容器现在容量的一倍进行增长。Vector 容器分配的是一块连续的内存空间，每次容器的增长，并不是在原有连续的内存空间后再进行简单的叠加，而是重新申请一块更大的新内存，并把现有容器中的元素逐个复制过去，然后销毁旧的内存。这时原有指向旧内存空间的迭代器已经失效，所以在操作容器时，迭代器要及时更新。

答案：D

2.2.4 Set/HashSet

Set 集合的特点是无序、不可重复。Set 集合中常用的子类有 HashSet、TreeSet、LinkedHashSet。本节要熟练掌握 Set 集合的常用子类、各子类的特征与常用方法，以及各子类间的联系与区别。

1. 下列关于容器集合类的说法正确的是（　　）。

A. LinkedList 继承自 List

B. AbstractSet 继承自 Set

C. HashSet 继承自 AbstractSet

D. WeakMap 继承自 HashMap

解释：A 选项，LinkedList 类是实现了 List 接口，而不是继承。B 选项，AbstractSet 类实现 Set 接口。C 选项，HashSet 继承自 AbstractSet 类，同时也实现 Set。

答案：C

2. 下面 Set 类中（　　）是有序的。

A. LinkedHashSet　　B. TreeSet

C. HashSet　　D. AbstractSet

解释：A 选项，LinkedHashSet 继承于 HashSet，是由可预知迭代顺序的 Set 接口的哈希表和链接列表实现。

B 选项，TreeSet 使用二叉树的原理对新 add() 的对象按照指定的顺序排序（升序、降序），每增加一个对象都会进行排序，将对象插入二叉树指定的位置。

C 选项，HashSet 存储元素并不是按照存入时的顺序（和 List 显然不同），而是按照哈希值（hashCode）来存的，所以取数据也是按照哈希值来取。

D 选项，AbstractSet 是一个抽象类，它继承于 AbstractCollection。AbstractCollection 实现了 Set 中的绝大部分方法，为 Set 的实现类提供了便利。

答案：B

2.2.5 hashCode()

在 Java 中，hashCode() 方法是 Object 类中定义的方法，返回值为 int 型，根据一定的规则将与对象相关的信息（如对象的存储地址、对象的字段等）映射成一个数值，这个数值称为散列值（也称为哈希码）。本节要掌握 hashCode() 方法的定义与作用。

1. 能否使用任何类的对象作为 Map 的 key？

参考答案：可以使用任何类的对象作为 Map 的 key，然而在使用它们之前，需要考虑以下几点。

（1）如果类重写了 equals() 方法，它也应该重写 hashCode() 方法。

（2）类的所有实例需要遵循与 equals() 和 hashCode() 方法相关的规则，即每一个参与确定两个对象是否相等（equals）的属性，也应该在 hashCode() 方法中考虑到。

（3）为保证更好的性能，用户自定义 key 类的最佳方法是使之为不可变（immutable）的，这样，hashCode() 生成的值可以被缓存起来。不可变的类也可以确保 hashCode() 和 equals() 在未来不会改变，这样就可以解决与可变相关的问题了。

2. hashCode() 方法和 equals() 方法有何重要性？

参考答案：HashMap 使用 Key 对象的 hashCode() 和 equals() 方法来决定 key-value 对的索引。当试着从 HashMap 中获取值时，也会用到这些方法。如果这些方法没有被正确地实现，在这种情况下，两个不同 Key 也许会产生相同的 hashCode() 和 equals() 输出，HashMap 将会认为它们是相同的，然后新加入的元素会覆盖已经在 Map 中的元素，而

非把这两个对象 key 存储到不同的地方。同样地，所有不允许存储重复数据的集合类都使用 hashCode() 和 equals() 去查找重复，所以正确实现它们非常重要。在比较两个对象是否相等时，会调用 equals() 方法进行比较，equals() 通常是通过比较两者的 hashCode() 产生的哈希码是否相等来确定两个对象是否相等。

2.2.6 Collections

Collections 是集合类的一个工具类、帮助类，其中提供了一系列静态方法，用于对集合中的元素进行排序、搜索及线程安全等各种操作。本节主要考核 Collections 概念及集合类中各种操作的实际使用。

1. Java 中提供的对数组进行操作的工具类为（　　）。

A. Collections　　B. Scanner

C. Date　　D. Arrays

解释：Java 中操作数组的工具类是 Arrays 类。

答案：D

2. 下面关于 Collection 和 Collections 的区别的说法正确的是（　　）。

A. Collections 是集合顶层接口

B. Collection 是针对 Collections 集合操作的工具类

C. List、Set、Map 都继承自 Collection 接口

D. Collections 是针对 Collection 集合操作的工具类

解释：java.util.Collection 是一个集合框架的父接口。它提供对集合对象进行基本操作的通用接口方法。Collection 接口在 Java 类库中有很多具体的实现类。Collection 接口的意义是为各种具体的集合提供最大化的统一操作方式。java.util.Collections 是一个包装类。它包含各种有关集合操作（如搜索、排序等）的静态方法。Collections 类不能实例化，就像一个工具类，服务于 Java 的 Collection 框架。

答案：D

3. 关于 HashMap、Hashtable、Collections.synchronizedMap、ConcurrentHashMap，以下说法正确的有（　　）。

A. 多线程环境下使用 Hashtable 和 Collections.synchronizedMap 实现同步的效率差别不大

B. ConcurrentHashMap 是使用重入锁实现同步的

C. Hashtable 是在 HashMap 的基础上，为方法加上了 synchronized 关键字实现同步的

D. 多线程环境下使用 ConcurrentHashMap 和 Collections.synchronizedMap 实现同步的效率差别不大

解释：B 选项，ConcurrentHashMap 允许多个线程修改操作并发进行，其关键在于使用了锁分离技术。它使用多个锁来控制对哈希表的不同部分进行的修改。ConcurrentHashMap 内部使用段（Segment）来表示这些不用的部分，每个段其实就是一个小的哈希表，它们有自己的锁。只要多个修改操作发生在不用的段上，它们就可以并发进行。有些方法需要跨段，如 size() 和 containsValue()，它们可能需要锁定整张表，而不仅仅是某个段。D 选项，synchronizedMap 有可能造成资源冲突（某些线程等待较长时间），ConcurrentHashMap 不会抛出 ConcurrentModificationException，即使一个线程在遍历的同时，另一个线程也尝试进行修改。所以两者在多线程时，效率相差较大。

答案：AC

2.2.7 Map

Map（映射）提供的是键－值对（key-value）映射，而 Collection 是集合接口，提供的是一组数据。注意区分 Map 的各种子集合，并掌握 Map 及其子集合的使用与常用方法。

1. 在 Java 的 Hashtable、Vector、LinkedList、TreeSet 四者中，Hashtable 和（　　）是线程安全的。

A. Vector　　B. TreeSet

C. LinkedList　　D. 以上都是

解释：Hashtable 是线程安全的 Map；Vector 是线程安全的 ArrayList；TreeSet 和 LinkedList 都不是线程安全的。

答案：A

2. 为什么 Map 接口不继承 Collection 接口？

参考答案：尽管 Map 接口和它的实现也是广义的集合框架的一部分，但 Map 不是集合，集合也不是 Map。因此，Map 继承 Collection 毫无意义，反之亦然。Map 包含 key-value 对，它提供抽取 key 或 value 列表集合的方法，但是它不适用“一组对象”规范。

3. 以下关于 HashMap 和 Hashtable 的说法正确的有（　　）。（2017 年，德邦）

A. 都实现了 Map 接口

B. Hashtable 类不是同步的，而 HashMap 类是同步的

C. Hashtable 不允许 null 键或值

D. HashMap 不允许 null 键或值

解释：HashMap 和 Hashtable 有以下区别。

（1）继承的父类不同。Hashtable 继承的是 Dictionary 类，HashMap 继承的是 AbstractMap 类。

（2）Hashtable 中的方法是同步的，而 HashMap 中的方法是非同步的。在多线程并发的环境下，可以直接使用 Hashtable，但是要使用 HashMap，就要自己增加同步处理。

（3）Hashtable 中的键和值都可以出现空值。当 get() 方法返回 null 值时，既可能表示 HashMap 中没有该键，也可能表示 HashMap 中有 key 为 null；可以有一个或多个键所对应的 key 为 null。因此，在 HashMap 中不能由 get() 方法来判断 HashMap 中是否存在某个键，而应该用 containsKey() 方法来判断。

答案：AC

4. 在 Java 中，关于 HashMap 类的描述，以下说法中错误的是（　　）。（2017 年，完美世界）

A. HashMap 能够保证其中元素的顺序

B. HashMap 允许将 null 用作值

C. HashMap 允许将 null 用作键

D. HashMap 使用键－值的形式保存数据

解释：HashMap 的底层是由数组加链表实现的，对于每一个 key 值，都需要通过对象的 hashCode() 来计算哈希值，然后通过哈希值来确定顺序，并不是按照加入顺序来存放的，因此可以认为是无序的。故 A 选项错误。

答案：A

2.2.8 Stream

Java 8 API 添加了一个新的对象，称为流——Stream，可以以一种声明的方式处理数据。Stream 使用一种类似 SQL 语句以从数据库查询数据的直观方式，来提供对 Java 集合进行高效、遍历的聚合操作和大批量数据的操作。本节要重点掌握 Stream 的特征与分类，以及各类 Stream 的概念、特征、常用方法与适用条件。特别注意这里的 Stream 和 IO 中的 Stream 是完

全不同的，不要把这两者混淆了。

1. 以下属于 Java 8 中新引入的 API 的 Stream 所在的包是（　　）。

A. java.io.stream　　B. java.io.streams
C. java.util.streams　　D. java.util.stream

解释：Java 8 中新引入的用于处理集合数据的 API 位于 java.util.stream 包下。注意不要将它和输入流 / 输出流混淆。

为区分这两者，如果无特殊说明，本书中将 java.util.stream 下的 Stream 称为“流式操作”，而将输入流 / 输出流按照传统称为“流”。

答案：D

2. Java 8 中的流式操作可以分成（　　）。

A. Terminal 类型　　B. Intermediate 类型
C. Traditional 类型　　D. Parallel 类型

解释：Java 8 流式操作分为 Intermediate 和 Terminal 两种类型，两者组成数据流的管道（pipeline）。一个 Stream 包含一个数据源、若干个 Intermediate 类型操作和一个 Terminal 类型操作。Intermediate 操作用于对 Stream 中的数据进行处理和转换，Terminal 类型的操作用于得到最终结果。

答案：AB

3. Java 8 流式操作中的 filter() 作用于（　　）。

A. Predicate　　B. Interface
C. Class　　D. Method

解释：Predicate 也是和 Stream 一样在 Java 8 中引入的新特性。它是一个函数式接口，位于 java.util.function 包下，可以用 Lambda 表达式作为参数。filter() 方法的参数是一个 Predicate 类型的数据。

答案：A

4. Java 8 流式操作中的 map() 作用于（　　）。

A. Predicate　　B. Interface
C. Class　　D. Function

解释：map() 的参数是 Function。

答案：D

5. Java 8 流式操作中的 forEach() 作用于（　　）。

A. 方法
B. 消费者（Consumer）
C. 生产者（Producer）
D. Predicate

解释：forEach() 的参数是 Consumer。

答案：B

6. Java 8 中的管道（pipeline）是一系列（　　）操作。

A. 多线程　　B. 同步
C. 连续　　D. 流（stream）

答案：D

7. 在 Java 8 中，获取对象集合的源（source）的方法是（　　）。

A. 通过 stream() 方法
B. 通过 obtain() 方法
C. 通过 obtainSource() 方法
D. 以上都是

解释：通过集合对象的 stream() 方法可以得到对象的源。

答案：A

8. 以下代码的输出结果是（　　）类型数据。

```
class Main{
    public static void main(String [] args)
    {
```

```
        final String str = "test";
        str.chars().forEach(ch->System.
        out.println(ch));
    }
}
```

A. 字符　　B. 字符串
C. 整数　　D. 浮点数

解释：str.chars() 得到的是一个 IntPipeline 类型的对象，在这个对象上使用 forEach() 得到的是整数类型的数据。

答案：C

2.3 异常

Java 中的异常（Exception）又称为例外，是一个在程序执行期间发生的事件，它中断正在执行程序的正常指令流。为了能够及时有效地处理程序中的运行错误，必须使用异常类。Java 异常强制用户考虑程序的健壮性和安全性。异常处理不应用来控制程序的正常流程，其主要作用是捕获程序在运行时发生的异常并进行相应处理。本节要求重点掌握异常分类及其子类的概念、特征、异同与联系，以及 try-catch-finally 的异常处理语句。

2.3.1 基础

在 Java 中，所有异常类都是内置类 Throwable 的子类，即 Throwable 类位于异常类层次结构的顶层。Throwable 类下有两个异常分支 Exception 类和 Error 类。本节重点考查 Throwable 类及其子类，以及 Java 的异常处理机制。

1. 关于 Java 的异常处理机制的叙述正确的有（　　）。（2017 年，搜狐）

A. 无论程序是否发生错误及捕捉到异常情况，都会执行 finally 部分
B. 当 try 部分的程序发生异常时，才会执行 catch 部分的程序
C. 当 catch 部分捕捉到异常情况时，才会执行 finally 部分
D. 其他选项都不正确

解释：只要程序不退出，finally 部分都会被执行。但 catch 部分只有 try 中出现异常时才会执行。

答案：AB

2. 下面有关 Java 异常类的描述，说法正确的有（　　）。（2016 年，中国电信）

A. 异常的继承结构：基类为 Throwable，用 Error 和 Exception 实现 Throwable，RuntimeException 和 IOException 等继承 Exception
B. 非 RuntimeException 一般是外部错误（不考虑 Error 的情况下），其必须在当前类被 try{}catch{} 语句块捕获
C. Error 类体系描述了 Java 运行系统中的内部错误及资源耗尽的情形，Error 不需要捕捉
D. RuntimeException 体系包括错误的类型转换、数组越界访问和试图访问空指针等，必须被 try{}catch{} 语句块捕获

解释：Throwable 类有以下子类。

（1）Exception（异常）：是程序本身可以处理的异常。

（2）Error（错误）：是程序无法处理的错误。这些错误表示故障发生于虚拟机自身或者发生在虚拟机试图执行应用时，一般不需要程序处理。

（3）检查异常（编译器要求必须处置的异常）：除了 Error、RuntimeException 及其子类以外，其他的 Exception 类及其子类都属于可查异常。

这种异常的特点是 Java 编译器会检查它，也就是说，当程序中可能出现这类异常，要么用 try-catch 语句捕获它，要么用 throws 子句声明抛出它，否则编译不会通过。

（4）非检查异常（编译器不要求处置的异常）：包括运行时异常（RuntimeException 与其子类）和错误（Error）。

所以，A、B、C 选项正确。RuntimeException 体系包括错误的类型转换、数组越界访问和试图访问空指针等，D 选项前半句对，错在后半句，应是非 RuntimeException 必须被 try{}catch{} 语句块捕获。

答案：ABC

3. 以下关于 Java 中 finally 块中的代码，描述正确的是（　　）。（2017 年，美团）

A. finally 块中的代码也可以在 return 后执行
B. 异常没有发生时才被执行
C. 若 try 块后没有 catch 块时，finally 块中的代码才会执行
D. 异常发生时才被执行

解释：finally 块的语句在 try 或 catch 中的 return 语句执行之后返回之前执行，且 finally 块中的修改语句不能影响 try 或 catch 中 return 语句已经确定的返回值，若 finally 块中也有 return 语句，则覆盖 try 或 catch 中的 return 语句直接返回。

答案：A

4. 下列关于异常的说法，正确的是（　　）。（2017 年，乐视）

A. RuntimeException 及其子类的异常可以不做处理
B. catch 段中的语句不允许再次出现异常
C. 在方法定义中以 throws 标识出的异常，在调用该方法中的方法必须处理
D. 程序中所有的可能出现的异常必须在 catch 中捕获，否则将引起编译错误

解释：RuntimeException 及其子类的异常可以不必处理，属于 Unchecked Exception，A 选项正确。B 选项，允许出现异常，如果 catch 异常，可执行 rollback。C 选项，throws 用来修饰一个方法，表示该方法如果产生异常的话，就不在本方法中捕获，而是丢弃给调用此方法的对象处理，如 public int get() throws Exception{}。D 选项，出现的异常可以抛出或捕获。

答案：A

5. 对于非运行时异常，程序中一般可以不做处理，由 Java 虚拟机自动进行处理。（　　）（2017 年，德邦）

解释：Java 异常都继承自 Throwable 类，Throwable 类的子类有 Error 和 Exception，其中 Exception 又分为运行时异常和编译时异常。编译时异常是防范性质的异常，需要显式处理。运行时异常是程序员问题造成的，并不强制进行显式处理。

答案：错误

2.3.2 抛出异常

当在定义的方法中不确定对异常应该如何处理，或者想将处理异常的权限交给方法调用者时，需要用 throws 和 throw 将异常抛出来。本节重点考核 throws 和 throw 的异同，以及各自的概念、特征与适用情境。

1. 下列有关 Java 异常处理的叙述中正确的有（　　）。（2017 年，凤凰网）

A. finally 是为确保一段代码不管是否捕获异常都会被执行的一段代码
B. throws 用来声明一个成员方法可能抛出的各种非运行时异常情况
C. final 可以用于声明属性和方法，分别表示属性的不可变及方法的不可继承
D. throw 用于明确地抛出一个异常情况

解释：final 关键字可以用于修饰类、方法、变量。用于修饰方法时，表示该方法不可在子类中覆盖，但子类中还是可以继承它。

答案：ABD

2. 下面代码输出的结果是（　　）。（2017 年，美团）

```
public class NULL {
    public static void print() {
        System.out.println("MTDP");
    }
    public static void main(String[] args) {
        try {
            ((NULL) null).print();
        } catch (NullPointerException e) {
System.out.println("NullPointerException");
        }
    }
}
```

A. NullPointerException
B. MTDP
C. 都不输出
D. 无法正常编译

解释：null 可以被强制转换为任意对象，因此可以调用 print() 方法输出 MTDP。

答案：B

3. 下列异常中可以使用 throw 抛出的有（　　）。（2019 年，第四范式）

A. Error　　B. Throwable
C. Exception　　D. RuntimeException

解释：可以使用 throw 抛出 Throwable 及其子类的异常，而 Error/Exception 和 RuntimeException 都是 Throwable 类的子类，所以都可以。

相关异常类继承的实现关系如下：

```
java.lang.Object
    java.lang.Throwable
        java.lang.Error
        java.lang.Exception
        java.lang.Exception.RuntimeException
```

答案：ABCD

2.3.3 捕获异常

Java 提供了 try（尝试）、catch（捕捉）、finally（最终）这三个关键字来处理异常。捕获异常是指用 catch 来捕获 try 语句块中发生的异常，是异常处理机制的一部分。本节主要考查 catch 的概念、特征及适用情境，以及 catch 涉及的 Java 异常处理机制。

1. 以下描述不正确的是（　　）。（2017 年，斐讯）

A. try 块不可以省略
B. 可以使用多重 catch 块
C. finally 块可以省略
D. catch 块和 finally 块可以同时省略

解释：在 try-catch-finally 中，catch 块和 finally 语句块可以省略其中一个，但不可将两者都省略。

答案：D

2. 在 Java 语言中，下列（　　）是异常处理的出口。

A. try{...} 子句　　B. catch{...} 子句
C. finally{...} 子句　　D. 以上说法都不对

解释：本题考查记忆型知识点。

答案：C

3. 关于以下方法调用描述正确的有（　　）。

```
private static final List<String> list=
```

```
 new ArrayList<>();
public static String test(String j){
    int i = 1, s = 1, f = 1, a = 1, b =
1,c = 1,d = 1,e = 1;
     list.add(new String("1111111111111
1111111111111111"));
    return test(s+i+f+a+b+c+d+e+"");
}
```

A. 一定会发生“OutOfMemoryError: Java heap space”

B. 一定会发生“StackOverflowError”

C. 一定会发生“OutOfMemoryError:Java heap space”与“StackOverflowError”

D. 当发生内存溢出错误时不需要用 try-catch 来捕获，需检查代码及 JVM 参数配置的合理性

解释：StackOverflowError 原因在于无限调用递归方法，方法是以栈帧的形式存在于虚拟机栈内存中，一直创建栈帧，导致栈溢出。代码中的递归无结束条件，所以会报错，B 选项正确。

OutOfMemoryError：Java 堆用于存储对象实例，只要不断地创建对象，并且保证 GCRoots 到对象之间有可达路径，以避免垃圾回收机制来清除这些对象，在对象数量到达最大堆的容量限制后就会产生内存溢出异常。所以一直调用 new String() 并不会造成堆内存溢出，也不会报错，A、C 选项错误。

Java 异常可以分为 Checked Exception（检查异常）和 Unchecked Exception（非检查异常），检查异常（不包括 RuntimeException 及其子类）必须使用 try-catch 或者 throws 处理，而 Error 属于非检查异常，非检查异常可以使用 try-catch 捕获，但是没必要，D 选项正确。

答案：BD

2.3.4 自定义异常

当项目出现未被 Java 描述并封装成对象的问题时，Java 对这些特有的问题按照 Java 的问题封装思想进行自定义异常封装。在 Java 中要想创建自定义异常，需要继承 Throwable 类或者其子类 Exception。本节重点考查自定义异常的父类、概念、特征、基于 Java 语言的常用语法及适用情境。

1. 自定义的异常类可以从（　　）类继承。

A. Error

B. AWTError

C. VirtualMachineError

D. Exception 及其子类

解释：Java 语言中的 Throwable 类分为 Error 和 Exception 两个子类。自定义的异常类从 Exception 及其子类继承。

答案：D

2. 以下对自定义异常的描述正确的是（　　）。

A. 自定义异常必须继承自 Exception

B. 自定义异常可以继承自 Error

C. 自定义异常可以更加明确地定位异常出错的位置和给出详细的出错信息

D. 程序中已经提供了丰富的异常类，使用自定义异常没有意义

解释：Java 语言中的 Throwable 类分为 Error 和 Exception 两个子类。自定义的异常类是从 Exception 及其子类继承的，A、B 选项错误。JDK 中只是提供了常见的一些异常行为的类，为了更准确地描述异常行为，许多时候需要自定义异常，以此保证程序可以准确捕获异常，所以 D 选项错误。

答案：C

3. 下列关于自定义异常类的描述有误的是（　　）。

A. 定义异常类时通常需要提供两种构造

方法：无参数的构造方法和带一个字符串的构造器，这个字符串将作为该异常对象的详细说明（也就是异常对象 getMessage() 方法的返回值）

B. 在选择抛出什么异常时，应该选择合适的异常类，从而可以明确描述该异常情况

C. 通常情况下，程序很少会自行抛出系统异常，因为异常的类名通常包含了该异常的有用信息

D. 用户自定义异常都应该继承 Exception 基类，如果希望自定义 Runtime 异常，也应该继承 Exception 基类，利用 Exception 的 printStackTrace() 方法可以追踪异常出现的执行堆栈情况，并可以此找到异常的源头

解释：定义的异常类是从 Exception 及其子类继承的。

答案：D

4. 给出以下代码，该程序的运行结果是（　　）。

```
class Example {
    Public static void main (String args[]){
        System.out.println(3.0/0);
    }
}
```

A. 编译失败

B. 运行期异常

C. java.lang.ArithmeticException 异常抛出

D. 打印输出 Infinity

解释：在 Java 中有三个特殊的浮点型的数值：正无穷大、负无穷大、NaN，这三种数值用来表示出错或溢出的情况。Java 中存在除 0 异常，但是 0.0（是 double 型）并不是 0，所以除以 0.0 并不报错，而计算负数的平方根会得到 NaN。

答案：D

2.4 线程

Java 为多线程编程提供了内置的支持。多线程可以让程序员编写高效率的程序来达到充分利用 CPU 的目的。本节主要考核线程的生命周期、线程与进程的概念及异同、线程同步、线程间通信、线程死锁、线程控制（如挂起、停止和恢复）等。

2.4.1 线程概述

一个线程指的是进程中一个单一顺序的控制流，一个进程中可以并发多个线程，每条线程并行执行不同的任务。一个线程不能独立地存在，它必须是进程的一部分。一个进程一直运行，直到所有的非守护线程都结束运行后才能结束。本节主要考核线程的创建，线程与进程各自的概念、特征、常用方法，以及两者之间的联系与异同。

1. 在 Java 语言中编写一个多线程的程序，可以使用的方法有（　　）。（2017 年，德邦）

A. 扩展类 Thread

B. 实现 Runnable 接口

C. 扩展类 Runnable

D. 实现 Thread 接口

解释：实现线程的方法如下。

（1）继承 Thread 类（覆盖它的 run() 方法）。

（2）实现 Runnable 接口（实现接口的 run() 方法）。

除了上面两种方法外，还可以使用 ExecutorService、Callable、Future 等实现多线程。

答案：AB

2. 关于线程和进程，下列描述不正确的是（　　）。（2016 年，阿里巴巴）

A. 在一个进程的线程共享堆区，而进程中的线程各自维持自己的堆栈

B. 进程的隔离性要好于线程

C. 线程在资源消耗上通常要比进程轻量

D. 不同进程间不会共享逻辑地址空间

解释：进程是具有一定独立功能的程序关于某个数据集合的一次运行活动，是系统进行资源分配和调度的一个独立单位。

线程是进程的一个实体，是 CPU 调度和分配的基本单位，它是比进程更小的能独立运行的基本单位。线程自己基本上不拥有系统资源，只拥有一点在运行中必不可少的资源（如程序计数器，一组寄存器和栈，非共享），但是它可以与同属一个进程的其他的线程共享进程所拥有的全部资源。

进程与线程有以下区别。

（1）包含关系：一个程序至少有一个进程，一个进程至少有一个线程。

（2）内存共享：进程在执行过程中拥有独立的内存单元（一个进程崩溃后，在保护模式下不会对其他进程产生影响）；多个线程共享进程提供的内存（拥有自己的私有栈空间，只是作为运行需要的极少内存），从而极大地提高程序的运行效率，但一个线程死掉就等于整个进程死掉。所以多进程的程序要比多线程的程序健壮。

（3）执行过程：进程独立执行；线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

（4）从逻辑角度来看，多线程的意义在于一个应用程序中有多个执行部分可以同时执行。但操作系统并没有将多个线程看作多个独立的应用来实现进程的调度、管理及资源分配。这是进程和线程的重要区别。

答案：D

3. 下面代码输出的结果是（　　）。（2017 年，美团）

```
public static void main(String args[]) {
    Thread t = new Thread() {
        public void run() {
            print();
        }
    }
    t.run();
    System.out.print("MT");
}
static void print() {
    System.out.print("DP");
}
```

A. DPMT

B. MTDP

C. MTDP 和 DPMT 都有可能

D. 都不输出

解释：start() 方法用来启动一个线程，当调用 start() 方法后，系统才会开启一个新的线程，进而调用 run() 方法来执行任务，而单独调用 run() 方法跟调用普通方法是一样的，已经失去线程的特性了。因此在启动一个线程时一定要使用 start() 方法，而不是 run() 方法。如果是多线程的情况（即用 t.start()，而不是 t.run()），MTDP 和 DPMT 都有可能，现在用的是 t.run()，因此只有一个 main 线程，是单线程，所以顺序执行，输出的是 DPMT。

答案：A

4. 线程启动调用的方法的是（　　）。

A. init()　　B. start()

C. run()　　D. main()

解释：启动一个线程是调用 start() 方法，使线程所代表的虚拟处理机处于可运行状态，这意

味着它可以由 JVM 调度并执行，但并不是线程就会立即运行。run() 方法是线程启动后要进行回调（callback）的方法。

答案：B

5. 以下说法错误的有（　　）。（2017 年，乐视）

A. Java 线程类的优先级相同

B. Thread 和 Runnable 接口没有区别

C. 若一个类继承了某个类，只能使用 Runnable 实现线程

D. 其他选项均不正确

解释：Java 线程类的优先级是有可能不同的。例如，守护（daemon）线程的优先级就低于普通线程。

Thread 实现了 Runnable 接口，是一个类，不是接口。在使用过程中，因为 Thread 是类，所以如果一个类继承了 Thread，就不能集成其他类了，此时一般推荐使用 Runnable 接口。除此之外，Thread 中还定义了 start() 方法，可以用它启动一个线程。

实现线程的两个常用方法如下。

（1）继承 Thread 类，重写（Overwrite）run() 方法，缺点是一旦继承了 Thread 类就不能继承其他类了。

（2）实现 Runnable 接口，调用 run() 方法。

答案：ABC

2.4.2 线程的生命周期

线程是一个动态执行的过程，它也有一个从创建到死亡的过程。本节主要考核线程从创建到死亡的整个生命周期，线程生命周期中状态的转换及转换线程状态的常用方法。

1. 在一个线程中调用 sleep(1000) 方法，将使该线程在（　　）时间后获得对 CPU 的控制（假设睡眠过程中不会有其他事件唤醒该线程）。（2019 年，第四范式）

A. 正好 1000ms

B. 少于 1000ms

C. 大于等于 1000ms

D. 不一定

解释：在程序休眠（sleep）1000ms 之后线程进入就绪态，在这种状态下，需要检查现在是否有资源允许这个线程继续运行，如果条件不满足，则需要等待，如果现在有资源，则立即执行。所以它至少在 1000ms 之后才有可能获得对 CPU 的控制。

答案：C

2. 在 Java 线程状态转换时，下列转换不可能发生的有（　　）。（2017 年，完美世界）

A. 初始态→运行态

B. 就绪态→运行态

C. 阻塞态→运行态

D. 运行态→就绪态

解释：在 Java 线程状态转换时，初始态→运行态、阻塞态→运行态。这两种转换不可能发生。

答案：AC

3. 编译并运行下面的程序时，会发生（　　）。（2017 年，德邦）

```
public class Bground extends Thread{
    public static void main(String argv[]){
        Bground b = new Bground();
        b.run();
    }
    public void start(){
        for(int i=0;i<10;i++){
           System.out.println("Value of
           i = "+i);
        }
```

```
    }
}
```

A. 编译错误，指明 run() 方法没有定义
B. 运行错误，指明 run() 方法没有定义
C. 编译通过并输出 0~9
D. 编译通过，但无输出

解释：对于线程而言，start() 方法是让线程从初始状态变成就绪状态。run() 方法才是执行体的入口。但是在 Thread 中，run() 方法是一个空方法，没有具体实现。Bground 类继承了 Thread 类，但是没有覆盖 Thread 类的 run() 方法，因此调用 run() 方法肯定无输出。
答案：D

4. 下列可以终止当前线程的运行的是（　　）。（2017 年，凤凰网）

A. 当一个优先级高的线程进入就绪状态时
B. 当该线程调用 sleep() 方法时
C. 当创建一个新线程时
D. 抛出一个异常时

解释：抛出异常，线程终止。
答案：D

2.4.3 多线程、锁和死锁

Java 给多线程编程提供了内置的支持，但多线程可能会带来内存泄漏、程序不可控等问题，因此需要使用 Java 的锁机制来控制并发代码产生的问题。本节主要考核多线程编程的概念、特征与实际应用，以及 Java 锁机制中 synchronized 和 volatile 的作用。

1. Java 中(　　)关键字可以对对象加互斥锁。（2017 年，斐讯）

A. transient　　B. synchronized
C. serialize　　D. static

解释：本题考查记忆型知识点。
答案：B

2. 下面对多线程和多进程编程描述正确的有（　　）。（2017 年，美团）

A. 线程的数据交换更快，因为它们在同一地址空间内
B. 线程因为有自己的独立栈空间且共享数据，不利于资源管理和保护
C. 在多进程中，子进程可以获得父进程的所有堆和栈的数据
D. 进程比线程更健壮，但是进程比线程更容易杀掉

答案：ACD

3. 下列说法正确的有（　　）。（2017 年，完美世界）

A. 因为直接调用 Thread 对象的 run() 方法会报异常，所以应该使用 start() 方法来开启一个线程
B. 一个进程是一个独立的运行环境，可以看作一个程序或一个应用。而线程是在进程中执行的一个任务。Java 运行环境是一个包含不同的类和程序的单一进程。线程可以称为轻量级进程。线程需要较少的资源来创建和驻留在进程中，并且可以共享进程中的资源
C. synchronized 可以解决可见性问题，volatile 可以解决原子性问题
D. ThreadLocal 用于创建线程的本地变量，该变量是线程之间不共享的

解释：volatile 与 synchronized 的区别如下。
- volatile 本质是在提示 JVM 当前变量在寄存器中的值是不确定的，需要从主存中读取；

synchronized 则是锁定当前变量，只有当前线程可以访问该变量，其他线程被阻塞。

- volatile 仅能用于变量级别，而 synchronized 可以用于变量或方法。
- volatile 仅能实现变量的修改可见性，但不具备原子特性，而 synchronized 可以保证变量的修改可见性和原子性。
- volatile 不会造成线程的阻塞，而 synchronized 可能会造成线程的阻塞。
- volatile 标记的变量不会被编译器优化，而 synchronized 标记的变量可以被编译器优化。

答案：BD

4. 假设 a 是一个由线程 1 和线程 2 共享的初始值为 0 的全局变量，则线程 1 和线程 2 同时执行下面的代码，最终 a 的结果不可能是（　　）。（2017 年，今日头条）

```
boolean isOdd = false;
for (int i = 1; i <= 2; ++i) {
        if (i % 2 == 1)
    isOdd = true;
        else
    isOdd = false;
        a += i * (isOdd ? 1 : -1);
}
```

A. –1　　B. –2　　C. 0　　D. 1

解释：

假设两线程为 A、B，设有三种情况。

（1）A、B 不并发：此时相当于两个方法顺序执行。A 执行完后 a=-1，B 使用 -1 作为 a 的初值，B 执行完后 a=-2。

（2）A、B 完全并发：此时读写冲突，相当于只有一个线程对 a 的读写最终生效，方法只执行了一次。此时 a=-1。

（3）A、B部分并发：假设A先进行第一次读写，得到a=1；之后A的读写被B覆盖了。B使用1作为a的初值，B执行完后a=0。

根据以上分析，a 的值不可能是 1。

答案：D

2.4.4 Lock

在 Java 并发编程中，锁有两种实现：隐式锁（synchronized）和显式锁（Lock）。本节主要考核 Lock 的概念、特征、基于 Java 语言的常用语法与适用情境，以及 synchronized 和 Lock 的区别。

1. 关于 synchronized 和 java.util.concurrent.locks.Lock 的比较，以下描述正确的有（　　）。

A. Lock 不能完成 synchronized 所实现的所有功能

B. synchronized 会自动释放锁

C. Lock 一定要求程序员手工释放，并且必须在 finally 中释放

D. Lock 有比 synchronized 更精确的线程语义和更好的性能

解释：Lock 能完成 synchronized 所实现的所有功能，A 选项错误。与 synchronized 比较，Lock 有比 synchronized 更精确的线程语义和更好的性能。synchronized 会自动释放锁，而 Lock 一定要求程序员手工释放，并且必须在 finally 中释放。Lock 还有更强大的功能。例如，它的 tryLock() 方法可以以非阻塞方式获得锁。

答案：BCD

2. 以下可用来实现线程间通知和唤醒的有（　　）。

A. Object.wait()/notify()/notifyAll()

B. ReentrantLock.wait()/notify()/notifyAll()

C. Condition.await()/signal()/signalAll()

D. Thread.wait()/notify()/notifyAll()

解释：wait()、notify() 和 notifyAll() 是 Object 类中的方法。调用某个对象的 wait() 方法能让当前线程阻塞，并且当前线程必须拥有此对象的锁。调用某个对象的 notify() 方法能够唤醒一个正在等待这个对象的锁的线程，如果有多个线程都在等待这个对象的锁，则只能唤醒其中一个线程。调用 notifyAll() 方法能够唤醒所有正在等待这个对象的锁的线程，即让这些线程都进入就绪状态，等待获取资源运行。Condition 是在 Java 1.5 中才出现的，它用来替代传统的 Object 的 wait()、notify() 实现线程间的协作，相比使用 Object 的 wait()、notify()，使用 Condition 的 await()、signal() 这种方式实现线程间协作更加安全和高效。因此通常推荐使用 Condition。

答案：AC

3. 下列关于 Java 多线程并发控制机制的叙述中，错误的是（　　）。

A. Java 中对共享数据操作的并发控制采用加锁技术

B. 线程之间的交互，提倡采用 suspend()/resume() 方法

C. 共享数据的访问权限都必须定义为 private

D. Java 中没有提供检测与避免死锁的专门机制，但应用程序员可以采用某些策略防止死锁的发生

解释：本题考查多线程的并发控制机制。Java 中对共享数据操作的并发控制采用传统的加锁技术，也就是给对象加锁，A 选项正确。线程之间的交互，提倡采用 wait() 和 notify()/notifyAll() 方法，这三个方法是 Object 类的方法，是实现线程通信的方法。不提倡使用 suspend() 方法和 resume() 方法，它们容易造成死锁，所以 B 选项错误。共享数据的访问权限都必须定义为 private，不能为 public 或其他，C 选项正确。Java 中没有提供检测与避免死锁的专门机制，因此完全由程序进行控制，应用程序员可以采用某些策略防止死锁的发生，D 选项正确。

答案：B

4. 在多线程同步中，对象加锁应该注意（　　）。

A. 返还对象的锁

B. 用 synchronized 保护的共享数据必须是私有的

C. Java 中对象加锁具有可重用性

D. 以上都对

解释：在多线程同步中，对象加锁应该注意的是，一定要返还对象的锁，用 synchronized 保护的共享数据必须是私有的，对象加锁具有可重用性。所以选 D 选项。

答案：D

2.4.5 线程池

Java 中使用 ThreadGroup 类来代表线程组，表示一组线程的集合，可以对一批线程和线程组进行管理。线程池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，无须因为反复创建线程而消耗过多资源。本节主要考核线程组与线程池的概念、特征、适用情境与实际应用，以及线程池管理的常用方法。

1. 在 Java 中，管理线程组的类是（　　）。

A. java.lang.ThreadGroup

B. java.lang.Thread

C. java.lang.Runnable

D. java.lang.Object

解释：Java 语言将一组线程定义为线程组，再将线程组作为一个对象进行统一的处理和维护，

线程组由 java.lang.ThreadGroup 类实现。

答案：A

2. 线程池的目的是复用线程，以提高响应速度，减少系统创建和销毁线程开销。（　　）

解释：本题属于记忆型知识点的考查。

答案：正确

3. Java 线程池中 submit() 和 execute() 方法的区别有（　　）。

A. submit() 方法可以向线程池提交任务

B. execute() 方法的返回类型是 void，它定义在 Executor 接口中

C. submit() 方法可以返回持有计算结果的 Future 对象，它定义在 ExecutorService 接口中

D. 以上都对

解释：两个方法都可以向线程池提交任务，execute() 方法的返回类型是 void，它定义在 Executor 接口中，而 submit() 方法可以返回持有计算结果的 Future 对象，它定义在 ExecutorService 接口中，它扩展了 Executor 接口，其他线程池类，如 ThreadPoolExecutor 和 ScheduledThreadPoolExecutor 都有这些方法。所以选 B、C 选项。

答案：BC

4. 关于 sleep() 和 wait()，以下描述错误的一项是（　　）。

A. sleep() 是线程类（Thread）的方法，wait() 是 Object 类的方法

B. sleep() 不释放对象锁，wait() 放弃对象锁

C. sleep() 暂停线程，但监控状态仍然保持，结束后会自动恢复

D. wait() 进入等待锁定池，只针对此对象发出 notify() 方法后获取对象锁，并进入运行状态

解释：针对此对象发出 notify() 方法后获取对象锁并进入就绪状态，而不是运行状态。另外，针对此对象发出 notifyAll() 方法后也可能获取对象锁并进入就绪状态，而不是运行状态。

答案：D

2.5 文件操作和 I/O 流

流代表任何有能力产出数据的数据源对象，或者有能力接收数据的接收端对象。流的本质是数据传输，根据数据传输特性将流抽象为各种类，方便更直观地进行数据操作。本节主要考查 I/O 流的分类，各类 I/O 流的概念、特征、适用情境与常用方法，各类 I/O 流之间的联系与区别。

2.5.1 文件和目录操作

Java 中 File 文件类以抽象的方式代表文件名和目录路径名。本节主要考核 File 类的概念、特征、适用情境与常用方法，File 类中进行文件和目录的创建、文件的查找和文件的删除等操作的语法与用法。

1. Java 对文件类提供了许多操作方法，能获得文件对象父路径名的方法是（　　）。

A. getAbsolutePath()

B. getParentFile()

C. getAbsoluteFile()

D. getName()

解释：本题考查 File 类的基本知识。File 类是通过文件名列表来描述一个文件对象的属性，通过 File 类提供的方法，可以获得文件的名称、长度、所有路径等信息，还可以改变文件名称、删除文件等。

答案：B

2. 在 File 类提供的方法中，用于创建目录的方法是（　　）。

A. mkdir()　　B. mkdirs()

C. list()　　D. listRoots()

解释：本题属于记忆型知识点考查。

答案：A

3. 下列说法中，正确的是（　　）。

A. 类 FileInputStream 和 FileOutputStream 用于文件 I/O 处理，用其所提供的方法可以打开本地主机上的文件，并进行顺序读 / 写

B. 通过类 File 的实例或者一个表示文件名的字符串可以生成文件 I/O 流，在生成流对象的同时，文件被打开，但不能进行文件读 / 写

C. 对于 InputStream 和 OutputStream 来说，其实例都是非顺序访问流，即只能顺序读 / 写

D. 当从标准输入流读取数据时，从键盘输入的数据被直接输入程序中

解释：文件被打开时，可以进行读写操作，所以 B 选项错误。当 InputStream/OutputStream 配合 RandomAccessFile 时，可以不按顺序访问文件内容，所以 C 选项错误。当从标准输入流读取数据时，从键盘所输入的数据被缓冲，按 Enter 键时，程序才会得到输入数据，D 选项错误。

答案：A

2.5.2 字节流和字符流

在 io 包（java.io）中，流分为两种：字节流与字符流。字节流有 InputStream、OutputStream，字符流有 Reader、Writer。本节主要考核字节流与字符流的概念、特征、适用情境与常用方法，以及两者之间的联系与区别。

1. 过滤字节流输出都是（　　）抽象类的子类。（2017 年，德邦）

解释：本题考查记忆型知识点。

答案：FilterOutputStream

2. 能向内存直接写入数据的流是（　　）。

A. FileOutputStream

B. FileInputStream

C. ByteArrayOutputStream

D. ByteArrayInputStream

解释：本题考查 Java 的内存读写。在 java.io 中，还提供了 ByteArrayInputStream、ByteArrayOutputStream 和 StringBufferInputStream 类可直接访问内存，它们是 InputStream 和 OutputStream 的子类。用 ByteArrayOutputStream 可向字节数组写入数据；用 ByteArrayInputStream 可从字节数组中读取数据。

答案：D

3. Java 中类 ObjectOutputStream 支持对象的写操作，这是一种字节流，它的直接父类是（　　）。

A. Writer　　B. DataOutput

C. OutputStream　　D. ObjectOutput

解释：ObjectOutputStream 是字节流，所有的字节输出流都是 OutputStream 抽象类的子类。ObjectOutputStream 既继承了 OutputStream 抽象类，又实现了 ObjectOutput 接口，Java 用接口技术代替双重继承。

答案：C

4. 在读字符文件 Employee.dat 时，使用该文件作为参数的类是（　　）。

A. BufferedReader

B. DataInputStream
C. DataOutputStream
D. FileInputStream

解释：本题考查java.io包中的字符输入流。Java的I/O输出包括字节流、文件流、对象流等，要注意区分不同流使用的不同类。字符类输入流都是抽象InputStreamReader及其子类FileReader、BufferedReader等。A选项中的BufferedReader类是把缓冲技术用于字符输入流，提高了字符传送的效率，但它不能处理文件流。B选项中的DataInputStream类用于处理字节流，实现了DataInput接口，不能处理文件流。C选项中的DataOutputStream类实现了DataOutput接口，不能处理文件流。D选项中的FileInputStream类可以对一个磁盘文件涉及的数据进行处理，满足题目要求。

答案：D

5. 在程序读入字符文件时，能够以该文件作为直接参数的类是（　　）。

A. FileReader
B. BufferedReader
C. FileInputStream
D. ObjectInputStream

解释：FileInputStream是字节输入流。ObjectInputStream用于对象串行化时从对象流中读取对象。所以C选项和D选项都不是本题的答案。A选项和B选项的FileReader和BufferedReader都是字符类输入流。但是FileReader的参数是读入的文件，而BufferedReader的参数是FileReader流的一个对象。

答案：A

6. 下列叙述中，错误的是（　　）。

A. 所有的字节流都从InputStream类继承
B. 所有的字节输出流都从OutputStream类继承
C. 所有的字符流都从OutputStreamWriter类继承
D. 所有的字符输入流都从Reader类继承

解释：字符类输出流的各个类都是抽象类Writer的子类，其中包括PrintWriter类、OutputStreamWriter类及其子类FileWriter、BufferWriter、CharArrayWriter、PipedWriter、StringWriter等。OutputStreamWriter类只是Writer类的一个子类，所以C选项的说法并不全面。

答案：C

2.5.3 转换流

字节流和字符流可以进行相互转换。OutputStreamWriter将字节输出流变为字符输出流，InputStreamReader将字节输入流变为字符输入流。本节主要考核转换流的概念、特征、适用情境与常用方法。

1. 下面语句能够正确地创建一个InputStreamReader的实例的是（　　）。（2019年，第四范式）

A. new InputStreamReader(new FileInputStream("data"))
B. new InputStreamReader(new FileReader("data"))
C. new InputStreamReader(new BufferedReader("data"))
D. new InputStreamReader("data")

解释：InputStreamReader将字节输入流转换为字符输入流，是字节流通向字符流的桥梁。它的构造方法如下。

（1）InputStreamReader(InputStream in)：用

于构造一个默认编码集的 InputStreamReader 类。

（2）InputStreamReader(InputStream in,String charsetName)：用于构造一个指定编码集的 InputStreamReader 类。

InputStreamReader 的主要方法如下。

（1）int read()：读取单个字符。

（2）int read(char []cbuf)：将读取到的字符存到数组中，返回读取的字符数。

答案：A

2. 下面的流中，（　　）是字节流通向字符流的桥梁。

A. InputStreamReader
B. OutputStreamWriter
C. LineNumberReader
D. ObjectInputStream

解释：InputStreamReader 是字节流通向字符流的桥梁。

答案：A

3. 下列有关转换流的说法，错误的是（　　）。

A. OutputStreamWriter 是字符流通向字节流的桥梁
B. 可以指定字节流和字符流之间转换的字符集
C. InputStreamReader 使用了缓冲区技术
D. OutputStreamWriter 是 OutputStream 的子类

解释：InputStreamReader 没有使用缓冲区技术，C 选项错误。

答案：C

4. 下列关于转换流的说法不正确的是（　　）。

A. InputStreamReader 和 OutputStreamWriter 都是转换流
B. InputStreamReader 是字符流通向字节流的桥梁
C. 转换流可以在创建对象的时候指定编码集
D. 在需要使用字符流时，可以用转换流把字节流转换成字符流

解释：字符流通向字节流的桥梁是 OutputStreamWriter。

答案：B

5. FileWriter 类直接继承的类是（　　）。

A. OutputStreamWriter
B. Writer
C. BufferedWriter
D. InputStreamReader

解释：FileWriter 类从 OutputStreamWriter 类继承而来。

答案：A

6. InputStreamReader 是转换流，可以将字节流转换成字符流，是字符流与字节流之间的桥梁。它的实现使用的设计模式是（　　）。

A. 工厂模式　　B. 装饰者模式
C. 适配器模式　　D. 代理模式

解释：java.io 流主要运用了两个设计模式，适配器模式和装饰者模式。

适配器模式：例如，InputStreamReader 和 OutputStreamWriter 做了 InputStream 与 OutputStream 字节流类到 Reader/Writer 之间的转换。

装饰者模式（无处不在）：例如，Buffered InputStream bis = new BufferedIn putStream (new FileInputStream())。

答案：C