| 主管领导审核签字 |
| --- |
| |

# 哈尔滨工业大学 2020 学年春季学期

# 软件构造　　试　题

| 题号 | 一 | 二 | 三 | 四 | 五 | 六 | 七 | 总分 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 得分 | | | | | | | | |
| 阅卷人 | | | | | | | | |

**片纸鉴心　诚信不败**

授课教师＿＿＿＿　密　姓名＿＿＿＿　封　学号＿＿＿＿　线　院系＿＿＿＿

**本试卷满分 100 分，折合 60%计入总成绩。**

**一　单项选择题（每个 2 分，共 32 分。请将全部答案填写至下表中，否则视为无效）**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| | | | | | | | |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| | | | | | | | |

**1　以下___是软件构造中的外部质量属性**

A　Line of Codes (LoC)　　B　Robustness

C　模块之间的耦合度　　D　代码符合Google的Java编码规范的程度

**2　以下针对Git的说法，正确的是____**

A　Git是分布式版本控制系统，可支持多个Git服务器和多个用户

B　Git的暂存区(Staging Area)是在本地磁盘上开辟的一段临时空间

C　一个commit节点可以有0个、1个、多个父亲commit节点

D　两个commit节点v1和v2，若文件f从v1到v2发生了变化，则Git仅存储f发生变化的代码行

**3　在当前commit上建立新分支314change并切换到该分支上工作，使用___指令**

A　`git commit -m "314change"`　　B　`git checkout -b 314change`

C　`git branch 314change`　　D　`git switch -c 314change`

**4　某Java程序需要用户输入密码进行登录。程序员在代码中使用一个char[]类型的变量pw存储密码而不使用String类型，以下说法不正确的是_____**

A　这种设计不合理，char[]是mutable的，很容易带来危险

B　String是immutable的，即使对其重新赋值，密码内容仍可能短暂存在于内存中，很容易带来危险

C　char[]是mutable的，在使用完密码后可以对其赋值为空，从而将敏感的密码内容从内存中消除掉，避免恶意攻击

D　若代码中不小心使用了类似于System.out.print("Contents:" + pw)的语句，只会打印出pw的内存地址而非其内容，不会造成密码泄露

**5** 以下代码执行之后，控制台输出的结果是____

```
Set<String> a = new HashSet<>();
String s = new String("xyz");
a.add(s);
s = s.substring(0, 1);
final Set<String> b = a;
b.add("xyz");
System.out.println(a.size() + " " + b.contains("x"));
```

A 代码无法通过静态类型检查　　B 0 false
C 1 false　　D 2 true

**6** 以下State类的RI是：只允许创建一个实例。以下说法正确的是___

```
class State{
    private static State instance;
    public static State getInstance(){
       if(instance == null)
          instance = new State();
        return instance;
    }
}
```

A 可以满足其RI
B 无法满足RI；需要在第2行static前面加上final才能满足RI
C 无法满足RI；需要给State类增加private State(){...}的方法才能满足RI，但多线程情况下仍有可能违反RI
D 在单线程情况下可以满足RI，在多线程情况下RI可能被违反

**7** 以下关于final的说法，不正确的是____

A ADT中某个final的方法，不能在子类型中被override
B 某个final的类，不能从其派生出子类型
C ADT的Rep中某个final的mutable变量，只能在该ADT的构造函数中为其赋值一次，其他方法中不能再改变其指向的内存地址
D 某个final的immutable变量，其值不能被修改

**8** 以下关于接口的说法，不正确的是____

A 接口中的方法不能用private关键词来修饰
B 接口中的方法不能用final关键词来修饰
C 接口中的方法不能用static关键词来修饰
D 接口定义中不能出现implements关键词

**9** 针对方法void m(Set<? super Integer> set)，以下___作为参数传递进去不是对它的合法调用（注：Number是Integer的父类型）

A HashSet<Object> set　　B Set<Integer> set
C Set<? extends Number> set　　D TreeSet<Number> set

**10** 针对以下equals()和hashCode()的说法，正确的是____

A Mutable类型的ADT不需要override这两个方法
B Java中的各种Collections类型（List, Set, Map等）实现的是行为等价性

C hashCode()中如果只包含“return 31;”语句，在功能实现上也是没有问题的

D 如果不override这两个方法，可以在spec中要求客户端不要在hash类型的集合类中使用该ADT

**11 以下代码输出的结果是____**

```
class X {
   public void abc(String a, int b) {
      System.out.print("xa ");
   }
}
class Y extends X {
   public void abc(String a) {
      System.out.print("ya ");
   }
   public void abc(String a, int b) {
      System.out.print("yb ");
   }
   public static void main(String[] args) {
      X x1 = new X();
      X x2 = new Y();
      Y y1 = new Y();
      x1.abc("", 1);
      x2.abc("", 1);
      y1.abc("");
      y1.abc("",1);
   }
}
```

A xa yb ya yb　　B xa ya yb yb

C xa xa ya yb　　D 代码无法通过静态类型检查

**12 以下针对设计模式的说法，不正确的是____**

A State模式相当于将ADT的状态管理功能delegate到其他ADT

B Factory method模式相当于将创建ADT的new操作从客户端代码delegate到了专门的ADT内部做了封装

C Decorator模式中只使用了一棵继承树，只使用继承关系，没有使用delegation

D Proxy模式利用了同一个ADT的不同子类型之间的delegation

**13 关于Java异常处理的说法，正确的是____**

A RuntimeException代表unchecked异常，其父类是Exception类

B 程序员若要自定义一个checked异常类，需直接继承自Exception类

C 如果某方法代码中包含了throw new xxException(...)这样的代码，则该方法的spec中一定会包含throws xxException

D 在try{...}catch{...}finally{...}结构中，如果try代码块里没有抛出任何异常，则catch和finally代码块不会被执行

**14 以下工具____不能直接用于帮助发现代码中存在的潜在错误**

A JUnit　　B SpotBugs　　C EclEmma　　D Eclipse Debugger

**15 以下___不是用于降低ADT的客户端程序员开发代价的软件构造方法**

A 在ADT内部设计多个overload方法，提供不同的参数列表
B 对ADT的方法设计强度更弱的spec，让客户端使用该方法的代价降低
C 让客户端创建抽象类型的对象，对客户端隐藏具体子类型
D 使用Façade设计模式对多个ADT的多个方法的调用逻辑进行封装

**16 以下关于多线程的说法，正确的是____**

A 只要ADT的rep里有mutable类型的变量，它在多线程场景下就无法做到thread safe
B 虽然ADT的observer方法内只涉及对rep的读操作，仍可能与其他方法产生竞争，必要时仍需对其加锁才可做到线程安全
C 一个thread safe的ADT，客户端在任何多线程场景下使用它都会做到线程安全
D 若某代码段需对两个共享对象加锁，可以使用synchronized(lock1 & lock2)的形式

二（**6分**）某方法m需要一个参数String credit，它要满足的条件是“[1,10]范围内的正整数，或正整数后面带有小数点0.5”。例如，"1"、"1.5"、"2"、"3.5"、"5"、"9.5"、"10"都是合法的参数值输入，但"0"、"0.5"、“03.5”、"6.3"、"7.0"、"10.5"都不是合法的参数值输入。请写出该方法最开始部分用assert对pre-condition进行合法性检测的Java代码。

```
public void m(String credit) {









   ...//此处开始为m所完成的常规业务逻辑
}
```

三（**28分**）某ADT的代码如下所示。

```
class Poem {
   public String title;
   public String author;
   private List<String> lines = new ArrayList<>();
   private Date date;

   // AF: 代表一首诗，包含四个属性
   //     title为诗的题目
   //     author为诗的作者
   //     lines为诗的带有次序的文本行
   //     date为诗的发表日期

   public Poem(String t, String a, List<String> l, Date d) {
      title = t;
```

```
        author = a;
        lines = l;
        date = d;
    }

    public void addOneLine(String newLine) {
        lines.add(newLine);
    }

    public Poem plagiarize(String newAuthor) {
        return new Poem(title, newAuthor, lines, date);
    }

    public String getAuthor() {
        return author;
    }

    public List<String> getSomeLines(int start, int end) {
        List<String> some = new ArrayList<>();
        for (int i = start; i < end; i++)
            some.add(lines.get(i));
        return some;
    }

    public List<String> getAllLines() {
        return lines;
    }
}
```

密

封

**1（5分）**请找出上述代码中存在表示泄露的代码行，在对应的右侧给出修改（写出新代码或者给出简要修改说明），使之避免表示泄露。

**2（5分）**方法getSomeLines(...)返回诗的第start行到第end行的文本。请为该方法设计spec，要充分考虑健壮性。必要时可修改/扩展参数、返回值类型、异常等。中英文均可。

线

```
/**
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 */
public List<String> getSomeLines(int start, int end)
```

**3（6分）**针对方法getSomeLines(...)和上一题中你设计的spec，使用等价类划分和边界值方法为其设计测试用例。假设在你的JUnit测试类中已提前构建好了一个Poem对象，其内容如下。

```
Author:   HITFC
Title:    HIT's 100th Anniversary
Lines:    Harbin 2020
          Love HIT
          Forever
Date:     2020-06-07
```

给出你的testing strategy描述，然后给出各测试用例的输入参数取值、期望输出、所覆盖的等价类或边界值。注意：此为黑盒测试，故无需参考上述给出的方法实现代码。

```
// Testing strategy（中英文均可）
//
//
//
//
//
//
//
//以下填写test cases，每行一个（无需填写poem对象的取值）
```

**4（6分）**针对经你修改后不存在表示泄露的代码，某客户端程序如下所示。请绘制出该程序运行之后的snapshot diagram（无需考虑JVM的垃圾回收）。

```
final List<String> lines = new ArrayList<>();
lines.add("Happpy");
lines.add("Birthday");
Poem p = new Poem("100th", "HIT", lines, new Date());
p.addOneLine("Love HIT");
List<String> some = p.getSomeLines(0, 1);
```

密

封

Snapshot Diagram

**5（6分）**针对经你修改后不存在表示泄露的代码，请继续对其修改，使该ADT在多线程环境下使用时能够做到thread safe。逐点写出你的修改策略，无需给出修改后的全部代码。

线

**四（16分）**某个ADT的rep如下所示。

```
class Encryption {
   String ss;
   int[] cs;
}
```

该ADT的AF和RI有以下三种设计方案：

方案1

```
// AF(cs,ss) = 一个加密字符串es
//     (1) es.length() == ss.length()
//     (2) 对任意0<=j<es.length()，如果j在cs中出现，
//         则es.charAt(j)='*'；否则，es.charAt(j)=ss.charAt(j)
//     例如：ss="Hello", cs=[1,3]，那么es="H*l*o";
// RI:
//     cs中不包含重复的数，且cs.length<=ss.length()
//     对任意的合法下标i, 0<=cs[i]<ss.length()
```

方案2

```
// AF(cs,ss) = 一个加密字符串es
//     es是对ss中的第j位连续重复cs[j]次得到的。如果cs[j]=0，
//     那么ss中的第j位不在es中出现。0<=j<ss.length()。
//     例如：ss="HIT", cs=[1,0,3]，则es="HTTT"
//     即第0位的'H'重复1次，第1位的'I'不出现，第2位的'T'重复3次
// RI:
//     ss.length() == cs.length
//     对任意的合法下标i, cs[i]>=0
```

方案3

```
// AF(cs,ss)= 一个加密字符串es
//     es是对ss中的第j位连续重复cs[j]次得到的。如果cs[j]=0，
//     那么ss中的第j位不在es中出现。0<=j<ss.length()。
// RI:
//     ss.length() == cs.length
//     对任意的合法下标i, cs[i]==0或1
```

**1（4分）**下表第1列给出了该ADT的ss和cs（R空间中的取值），请给出其在不同的AF/RI方案下映射得到的A空间的值。若ss和cs取值不合法，可填写“非法rep”。

方案1：

| | |
|---|---|
| AF("HITFC",[1,3]) | |
| AF("HITFC",[1,0,1,0,1]) | |
| AF("HITFC",[3,0,5,9,1]) | |

方案2：

| | |
|---|---|
| AF("",[]) | |

| | |
|---|---|
| AF("HITFC",[1,0,1,0,1]) | |
| AF("HITFC",[3,0,5,1,1]) | |

方案3：

| | |
|---|---|
| AF("HITFC",[1,3,1,1,0]) | |
| AF("HITFC",[1,0,1,1,0]) | |

**2（4分）**该ADT有以下方法的实现，该方法返回加密后的字符串的长度。

```
public int getLength() {
    int length = 0;
    for (int i = 0; i < cs.length; i++)
        length += cs[i];
    return length;
}
```

那么该getLength()方法符合以上1/2/3三个AF/RI方案中的哪个（些）？_____

**3（8分）**考虑该ADT的构造函数如下所示。

```
public Encryption(String ss, int[] cs) {
    this.ss = ss;
    this.cs = cs;
}
```

在上述AF/RI的方案1、2、3下，该构造函数的spec会有很大差异。假设你已有了三种方案下该构造函数的spec，且各spec中均要求输入参数ss和cs要满足各自的RI。请比较这三个spec的pre-condition之间的强弱关系，并简要说明理由。

密 封 线

————————————以下第五题和第六题，二选一作答即可————————————

**五（10分）**某ADT的功能是随机选择一首诗中的5行，以日志形式输出文本，其代码如下所示。

```
class Recorder {

   private static final NUMBER = 5;
   private static final Logger log = Logger.getLogger("Recorder");
   private Poem poem; //该类型参见第三题中的Poem

   public void recording() {
      poem = new Poem(...); //此处忽略了参数
      log.addHandler(new ConsoleHandler());
      log.setLevel(Level.INFO);

      Random r = new Random();
      List<String> lines = poem.getAllLines();
      for (int i = 0; i < NUMBER; i++) {
         String line = lines.get(r.nextInt(lines.size()));
         if(line.length() >= 10) //长度超过10才可被日志输出
            log.info(line);
      }
   }
}
```

后续要扩展新功能：在不同场景下，诗歌对象(poem)可从不同的源头读入（例如磁盘文件、用户键盘输入、某网络地址等）、输出日志的渠道可从控制台输出扩展到其他输出（例如写入磁盘文件、写入某个网络地址等）、随机选择诗歌文本行的时候，文本行的长度限制也可自由配置。

请使用template method设计模式，对上述代码进行改造，以支持上述功能扩展。写出你进行代码改造的基本思路描述，无需写出具体代码。

密 封 线

六（**10分**）以下给出了名为Student的ADT及其代码。

```
class Student {

    private String name;
    private Map<String, Integer> scores = new HashMap<>();
    //AF: name为学生名字；scores的key为学习内容，value为所获成绩
    //RI: scores中的value值范围为[0,100]

    public Student(String name) {
        this.name = name;
    }

    //根据学生所获得的成绩，计算总成绩
    public int getFinalScore() {
        int total = 0;
        for (String content : scores.keySet())
            total += scores.get(content);
        return total/scores.size();
    }

    //具体的学习动作，并获得分数
    public void getScore(String content) {
        System.out.println("我在通过听课学习" + content);
        int score = ...; //这里实现了具体听课学习的过程，返回所得到的成绩
        scores.put(content, score);
    }

    public static void main(String[] args) {
        Student s = new Student("张三");
        s.getScore("计算机系统");
        s.getScore("软件构造");
        System.out.println("总成绩：" + s.getFinalScore());
    }
}
```

目前代码只支持听课学习（见getScore()方法第1-2行）。现在学校鼓励学生通过网上MOOC自学来获得分数，而MOOC自学过程与听课学习过程不同，目前的getScore()代码无法支持它，将来也无法扩展到新的学习方式。

为此，拟修改Student类的设计，通过delegation机制将getScore()中的学习动作委派给接口Study（代码见下方）的子类型，学生可自由选择听课学习或MOOC自学来获得成绩。

请简要写出最恰当的delegation机制，对已有代码做出修改，使之可支持这两种学习方式，并可灵活扩展其他学习方式（例如通过竞赛获得成绩、通过与导师做科研获得成绩等）。

```
interface Study {
    /**
     * @param content 学习内容
     * @return 学习之后获得的分数，[0,100]范围内的整数
     */
    int study(String content);
}
```

七（**10分**）分析以下ADT的代码，判断两个类是否符合LSP。若不符合，请指出所有造成违反LSP的地方，并逐条说明理由。

```
class Graph<L> {

    protected Set<L> nodes = new HashSet<>();
    protected Map<L, Set<L>> edges = new HashMap<>();
    // AF: 表示一个有向图，nodes是图的节点集合；edges中的key表示节点，
    //     value为从该key节点出发的边所指向的节点集合；L为节点类型的泛型参数
    // RI: edges的每个key，均在nodes中出现；
    //     edges的每个value中所包含的所有节点，均在nodes中出现
    //     对edges中的一个<key, value>，key不能在value中出现

    /**
     * @param nodes 一组节点
     * @param filter 过滤条件
     * @return nodes中满足filter条件的节点的数目
     */
    public Number statistics (List<L> nodes, String filter) {
        ...
    }
}
```

```
class LoopWeightedGraph<L> extends Graph<L> {

    private Map<L, Integer> weights = new HashMap();
    // AF: weights中的key表示节点，value表示该key节点的权重
    // RI: edges的每个key，均在nodes中出现；
    //     edges的每个value中所包含的所有节点，均在nodes中出现
    //     weights包含的元素数量 == nodes包含的元素数量

    /**
     * @param nodes 一组节点
     * @param filter 过滤条件，长度不超过20
     * @return nodes中满足filter条件的节点的数目
     * @throws 若nodes中存在重复节点
     */
    @Override
    public Integer statistics(ArrayList<L> nodes, String filter)
                                                    throws Exception {
        ...
    }

    public int getTotalWeight() {...}
}
```

是否违反LSP？若违反，请逐条说明理由。

密 封 线