主管领导审核签字

哈尔滨工业大学 2021-2022 学年春季学期

软件构造　　试　题

题号	一	二		三							总分
		1	2	1	2	3	4	5	6	7	
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＿＿＿＿

卷面总分 100 分，按得分 60%计入课程总成绩。

第一部分　单项选择题（每题 3 分，共 45 分。请将答案填入下表，否则不计分）

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

1. Stack trace 和 code churn 在软件构造三维度八视图中的共性特征是_____

   A　都是 component-level view　　B　都是 period view

   C　都是 run-time view　　D　无共性

2. 某程序员目前在 master 分支上工作，HEAD 指向 master 分支上的某个 commit 节点。在执行以下 git 指令后，HEAD 指向了____分支

```
git checkout -b test
...程序员修改某些代码
git add *
git commit -m "first"
git checkout master
...程序员修改某些代码
git add *
git commit -m "second"
git merge test
```

   A　test　　B　master　　C　first　　D　second

3. 以下关于 git 的说法，不正确的是____

A 一个 commit 可以有 0 个、1 个、2 个父亲 commit

B 程序员对某个文件做了重命名但没有修改文件内容，`git` 仍会看作是删除了一个文件、新建了一个文件

C 一个本地 git 仓库可以设置多个远程 git 仓库并与它们进行同步

D git 管理的软件配置项是代码行，即 git 管理变化的基本单元是代码行

4. 关于 immutability 和 mutability 的说法，不恰当的是____

A 客户端构造得到的 `Set/Map/List` 对象并不一定都是 mutable 的

B 一个 immutable 的 ADT，也可能会包含 mutator 方法

C 声明变量时，在前面加上 `final` 关键字，可保证该变量是 immutable 的以提高安全性

D 使用 immutable 类型可以降低程序蕴含 bug 的风险，但可能导致时空性能变差

5. 两个方法具有“行为等价性”，以下说法不恰当的是_____

A 站在客户端的视角看，它们提供完全相同的功能

B 站在客户端的视角看，它们可能展现出不同的性能

C 站在开发者的视角看，它们可能采用不同的异常处理策略

D 站在开发者的视角看，它们可能采用不同的数据结构和算法

6. 如果修改了某个方法的 spec 使之变弱了，那么正确的说法是___

A 如果用椭圆面积表示 spec 的强度，那么该方法的椭圆面积减小了

B Client 调用该方法的代价变小了，即调用时对传入该方法的参数可以做更少的检查

C 程序员实现该 spec 时的难度降低了，自由度升高了

D 程序员实现该 spec 的方式更少了

7. 关于 ADT 的 AF 和 RI，以下说法不正确的是____

A 即使两个 ADT 有相同的 rep 和相同的 AF，它们的 RI 未必相同

B 若 ADT 的某个方法返回一个 mutable 的对象，并不一定表明该 ADT 产生了表示泄露

C 如果 R 中的两个值被 AF 映射为 A 中的同一个值，那么这两个值所代表的对象是等价的

D `AF` 和 `RI` 以注释的形式写在代码中，后续可使用特定工具来生成 `Java` 文档对外发布

8. 关于 class 和 interface 的说法，不准确的是_____

A 一个接口可以 `extends` 一个或多个其他接口

B 一个类可以同时 `extends` 另一个类和 `implements` 一个乃至多个接口

C 一个类 `implements` 了一个接口，意味着该类必须要 `override` 接口中定义的所有方法

D 一个类除了实现其 `implements` 的接口中的方法，还可以增加新的方法

9. 有两个 `Java` 类 `X`、`Y`，二者的关系是 `Y extends X`；二者内部都有 `hit(X x)`方法，且具有相同的返回值类型；`Y` 中还有一个 `hit(Y y)`方法。假如客户端中有以下代码：

```
X a = new X();
Y b = new Y();
```

```
X c = new Y();
```

那么以下___可以通过编译器的静态检查

A `a.hit(b)` B `b.hit(a)` C `c.hit(b)` D 以上都对

10. 针对第 9 题中给出的前提条件，以下说法正确的是____

A 类 Y 中的 `hit(X x)`是对类 X 中的 `hit(X x)`的合法 overload

B 类 Y 中的 `hit(X x)`是对类 Y 中的 `hit(Y y)`的合法 overriding

C 类 Y 中的 `hit(X x)`是对类 X 中的 `hit(X x)`的合法 overriding

D 类 Y 中的 `hit(Y y)`是对类 X 中的 `hit(X x)`的合法 overriding

11. 以下说法不正确的是___

A `LinkedList<Object>`是 `List<Object>`的子类型

B `List<Integer>`是 `List<Object>`的子类型

C 如果定义了一个类型为 `Object[]`的数组 a，那么 a 中既可以存储 `Integer` 类型的元素，也可以存储类型为 `List` 类型的元素

D 子类型若重写父类型中的方法，重写后可以抛出与父类型方法不一样的异常

12. 两个 ADT A 和 B，A 希望在方法 `hit()`中委派某些功能给类 B 的实例。为了在二者之间建立 delegation 关系，以下最能灵活适应未来可能变化的方案是____

A 为 A 的构造方法增加参数 `B b`，在构造方法的代码中将 b 存储于 A 的 rep

B 在 A 的 rep 中增加一个属性 `B b = new SubTypeB(…)`，其中 `SubTypeB` 是 B 的某个子类型

C 给 A 增加一个 `public` 方法 `setDelegation(B b)`，在该方法代码中将 b 存储于 A 的 rep

D 在 `hit()`的内部代码中用 `B b = new SubTypeB(…)`的形式建立委派关系

13. 以下____设计模式不是通过 delegation 机制实现的

A Factory Method B Visitor

C Template Method D Iterator

14. 某个 ADT A，要在客户端实现对其 rep 中某个复杂数据结构的遍历，以下___是最恰当的实现机制

A A 提供一个 observer 方法，该方法给客户端返回该复杂数据结构的对象，由客户端自行对其进行遍历

B 将 A 实现 `Iterable` 接口，为此需要 override 该接口的 `iterator()`方法，返回一个可对该结构进行遍历的 `Iterator` 对象，客户端用此进行遍历

C 为降低客户端的遍历难度，为 A 增加 `hasNext()`和 `next()`方法，分别用于判定遍历是否结束、取下一个元素，客户端用这两个方法进行遍历

D 为了避免表示泄露，不能允许客户端遍历 A 的 rep 中的任何数据结构

15. 某方法的 spec 中规定了 precondition，以下做法不正确的是____

A 该方法代码中无需检查 precondition 是否违反，因为这是客户端的责任

B 在方法开始时对 precondition 进行检查，若违反则抛出一个异常

C 在该方法所在 ADT 的 RI 中加入该 precondition，并在 `checkRep()`中对 precondition 进行检查

D 在方法开始时对 precondition 进行检查，若违反则将原因输出到控制台提醒用户，并返回 `null`

第二部分 设计题 (15 分)

Java 的 `List` 接口中有一个方法 `subList()`，其 spec 摘录如下：

返回当前 `List` 对象的一个子列表（类型仍然为 `List`），该子列表包含元素的起始位置是当前 `List` 对象中下标为 `fromIndex` 的元素，结束位置是当前 `List` 对象中下标为 `toIndex-1` 的元素。如果 `fromIndex == toIndex`，那么返回的子列表为空。返回的子列表中的元素并不是对原列表中的元素的复制，而是指向当前 `List` 对象中的相应元素，这意味着，对返回的子列表中的元素的修改会同时反映到当前 `List` 对象，反之亦然。

`@param` `fromIndex` 标识子列表在当前 `List` 对象中的起始位置（包含该位置）

`@param` `toIndex` 标识子列表在当前 `List` 对象中的结束位置（不包含该位置）

`@return` 当前 `List` 对象的子列表，按相同次序包含了当前 `List` 对象中下标处于 `[fromIndex, toIndex)`范围内的元素

`@throws` 如果`(fromIndex < 0 || toIndex > size || fromIndex > toIndex)`，则抛出 `IndexOutOfBoundsException`，`size` 为当前 `List` 包含元素数量

```
List<E> subList(int fromIndex, int toIndex)
```

1.(9 分) 针对所给出的 `spec`，为 `List<Integer>`的 `subList()`方法设计测试用例。每个测试用例的输入应包括：当前 `List` 对象、`fromIndex`、`toIndex`。也要给出期望输出结果。

院系_____ 学号_____ 姓名_____ 授课教师_____

密 封 线

2. (6 分) 某客户端代码如下所示，请绘制出该代码全部执行后的 snapshot diagram，无需绘制中间过程。

```
List<String> list = new LinkedList<>();
list.add("a");
list.add("b");
list.add("c");
List<String> sub = list.subList(1, 3);
list.set(1, "d");
sub.remove(1);
```

第三部分 综合题 (40 分)

设计一套 ADT，实现对现实中各类“会议”的有效抽象。要支持客户端的功能包括：

- 根据会议名称、会议日期、其他附加信息，创建一个会议
- 参会者加入会议
- 参会者在会议中发言
- 参会者离开会议
- 将某些参会者从会议移除

为此设计了名为 `Person` 的 ADT 表示参会者、名为 `Conference` 的 ADT 表示会议，类 `CommonConference` 是接口 `Conference` 的一个实现类。请阅读试卷后附代码，完成以下题目。

1. (5 分) 检查各 ADT 的代码中是否存在表示泄露的情况？请简要写出原因和修改措施。同类型的表示泄露可以合并到一行填写。以下所留空行较多，无需一定要全部填满。

所属 ADT	代码行号	表示泄露的原因	如何修改以避免表示泄露

2. (5 分) 请根据类 `Person` 的 AF 检查第 13-22 行的代码，判断是否正确实现。若有错误，如何修改为正确的实现？

3. (5 分) 针对接口 Conference 中的各个方法，写出它们的分类（Creator, Producer, Observer, Mutator）

方法	分类
create()	
join()	
speak()	
leave()	
participants()	
remove()	
copy()	

4. (5 分) 针对类 CommonConference 的 RI，写出其 checkRep()的代码。

5. (5 分) 客户端希望使用 Collections.*sort*(participants,...)形式的代码对一个 List<Person>类型的对象 participants 进行按 ID 由高到低的排序（见 Client.java 第 16 行）。但当前所给出的代码中无法支持该功能。请简要描述一种设计思路，实现上述需求，并说明 Client.java 第 16 行如何修改。

院系_____ 学号_____ 姓名_____ 授课教师_____ 密 封 线

6. (8 分) 在 Conference 和 CommonConference 基础上，考虑两类特别的会议：线下会议和线上会议。与当前 CommonConference 的内部 rep 和方法实现相比，线下会议需增加一个新属性（“地点”）且不能移除参会者，线上会议需增加一个新属性（“腾讯会议号”），其 remove()行为需要增加限定“没有发过言的参会者不能被从会议移除”。考虑 OCP 和 LSP 原则，对当前 ADT 设计进行扩展，简要描述你的扩展思路，并阐述客户端代码如何使用这两类会议。

7. (7 分) 从健壮性的角度检查代码，判断 Client.java 中的代码哪些地方会异常终止执行？简要说明分别会以何种症状终止，以及造成终止的原因。如何改进现有代码可提高客户端程序执行的健壮性？为此需要对客户端代码和各 ADT 代码做什么修改？

Person.java

```
1  public class Person {
2      public String name;
3      public String ID;

4      //AF: name为参会者姓名，ID为身份证号，客户端使用ID唯一标识参会者
5      //RI: true

6      public Person(String name, String ID) {
7          this.name = name;
8          this.ID = ID;
9      }

10     public String getID() {
11         return this.ID;
12     }

13     @Override
14     public int hashCode() {
15         return 31 + ID.hashCode() + name.hashCode();
16     }

17     @Override
18     public boolean equals(Person other) {
19         if (ID.equals(other.ID) && name.equals(other.name))
20             return true;
21         return false;
22     }
23 }
```

Conference.java

```
1  public interface Conference {
2      /**
3       * 创建一个会议
4       * @param name 会议名称，长度不超过30，只能包含字母、数字和空格，且空
5       * 格不能出现在最开始和最末尾，且不能有连续多个空格出现
6       * @param date 会议日期
7       * @return 一个会议对象
8       */
9      static Conference create(String name, Date date) {
10         return new CommonConference(name, date);
11     }

12     /**
```

```
13      * 参会者发言
14      * @param p 发言的参会者名字，需要出现在当前参会者清单当中
15      * @param mins 发言时长（分钟），mins>0
16      * @throws 如果p未出现在参会者清单当中，则抛出该异常
17      */
18     void speak(Person p, int mins) throws IllegalArgumentException;

19     /**
20      * 新的参会者加入会议
21      * @param p 新参会者
22      * @throws 如果p已经在会议中，则抛出该异常
23      */
24     void join(Person p) throws IllegalArgumentException;

25     /**
26      * 参会者离开会议
27      * @param p 离开会议的参会者
28      * @throws 如果p之前没有在参会者清单中，则抛出该异常
29      */
30     void leave(Person p) throws IllegalArgumentException;

31     /**
32      * 获得参会者名单
33      * @return 所有参会者名单
34      */
35     List<Person> participants();

36     /**
37      *从参会者清单中移除参会人，被移除的参会人的ID以prefix开头
38      * @param prefix 需要移除的ID前缀
39      */
40     void remove(String prefix);

41     /**
42      * 复制一个新会议
43      * @param days 本次会议的days天之后作为新会议的日期，days>0
44      * @return 一个新会议，与当前会议名称相同，且在当前会议日期days天之后
45      */
46     Conference copy(int days);
47 }
```

CommonConference.java

```
1  public class CommonConference implements Conference {

2      public String name;
```

```
3     public Date date;
4     public List<Person> participants = new ArrayList<>();
5     public Map<Person, Integer> records = new LinkedHashMap<>();

6     // AF: name 为会议名称，date 为会议举办日期，participants 为当前在会议中
      //     的参会者，其中每个成员是一个参会者
      //     records 为发言记录，key 为发言的参会者，value 为发言时长，按添加
      //     次序存储（LinkedHashMap 具有按插入次序存储的能力）

7     // RI: name 长度不超过 30，只能包含字母、数字和空格，且空格不能出现在最开
      //     始和最末尾，且不能有连续多个空格出现
      //     participants 中不能出现 ID 相同的参会者

8     public CommonConference(String name, Date date) {
9        this.name = name;
10       this.date = date;
11    }

12    @Override
13    public void speak(Person p, int mins)
                                  throws IllegalArgumentException {
14       if (!participants.contains(p) || mins <= 0)
15          throw new IllegalArgumentException("不是合法的发言者");
16       else
17          records.put(p, mins);
18    }

19    @Override
20    public void join(Person p) throws IllegalArgumentException {
21       if (participants.contains(p))
22          throw new IllegalArgumentException("已在会议中");
23       participants.add(p);
24    }

25    @Override
26    public void leave(Person p) throws IllegalArgumentException {
27       if (!participants.contains(p))
28          throw new IllegalArgumentException("不在会议中");
29       participants.remove(p);
30    }

31    @Override
32    public List<Person> participants() {
33       return this.participants;
34    }

35    @Override
```

```
36      public void remove(String prefix) {
37          for (Person p : participants) {
38              if (p.getID().startsWith(prefix))
39                  participants.remove(p);
40          }
41      }

42      @Override
43      public Conference copy(int days) {
44          Calendar now = Calendar.getInstance();
45          now.setTime(this.date);
46          now.set(Calendar.DATE, now.get(Calendar.DATE) + days);
47          return new CommonConference(this.name, now.getTime());
48      }
49  }
```

Client.java

```
1   public class Client {

2       public static void main(String[] args) {
3           Person p1 = new Person("Zhang", "100");
4           Person p2 = new Person("Li",    "200");
5           Person p3 = new Person("Wang",  "300");

6           Conference conf = Conference.create("HITFC 2022", new Date());
7           conf.join(p1);
8           conf.join(p2);
9           conf.join(p3);
10          conf.speak(p1, 10);
11          conf.join(new Person("Zhao", "101"));
12          conf.leave(p1);
13          conf.speak(new Person("Feng","201"), 15);
14          conf.remove("10");

15          List<Person> participants = conf.participants();
16          Collections.sort(participants, /* TODO */);

17          Conference anotherConf = conf.copy(-1);
18      }
19  }
```