

主管  
领导  
审核  
签字

软件构造 试 题

题号	一	二	三	四	五	六	七	八	九	总分
得分										
阅卷人										

片纸鉴心 诚信不败

本试卷满分 100 分，按 60%计入总成绩。

一 单项选择题（每题 2 分，共 30 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. 在软件构造“三维度八视图”中，Stack trace 和 Code churn 存在的共性是\_\_\_\_  
A 都是 Moment View                      B 都是 Run Time View  
C 都是 Code Level View                    D 二者无共性
2. 关于软件构造质量的说法，不恰当的是\_\_\_\_  
A 软件的内部质量指标会对外部质量指标产生影响  
B 可变数据类型比不可变数据类型更加高效、安全、节省内存  
C 正确性是最重要的软件构造质量指标，没有之一  
D 良好设计的委派关系比单纯使用继承关系更能提高软件的可维护性
3. 关于配置管理工具 Git 的说法，正确的是\_\_\_\_  
A 客户端安装的 Git 中，能够配置 1 个或多个远程仓库  
B Git 不允许将本地工作目录中发生变化的文件直接提交到本地 Git 仓库中  
C 若某文件在相邻的两个版本中发生了变化，新版本中只存储该文件发生变化的代码行，以节省存储空间  
D 在进行分支合并时，Git 可识别两个分支上做的不同修改并自动合并它们
4. 关于 Java 中关键字 final 的说法，不正确的是\_\_\_\_  
A 一个 final 的可变类型变量，一旦初始化了某个值，其值仍然可以再改变  
B 一个 final 的不可变类型变量，一旦初始化之后，其指向的内存地址就不

授课教师

姓名

学号

院系

---

能再改变

- C 一个 **final** 的类，无法被继承，即无法再派生子类型
  - D 一个 **final** 的方法，无法被其子类型 **overload**
5. 关于 ADT 的方法 spec 的说法，不正确的是\_\_\_\_
- A spec 作为客户端程序和 ADT 之间的防火墙，其描述中不应该出现 **rep** 的任何信息
  - B spec 的 **post-condition** 中可以包含对方法的参数的修改，但一般不提倡这么做
  - C 返回值为 **void** 的方法，其 spec 中不存在任何 **post-condition**
  - D Java 中的 spec 包含 **@param**、**@return**、**@throws** 三种合法的 **annotation**
6. 关于 ADT 的说法，不正确的是\_\_\_\_
- A **AF**、**RI**、**safety from rep exposure**、**testing strategy** 都不应暴露给客户端程序员
  - B 一个 ADT 的两个对象实例，若客户端看待它们是等价的，它们内部的 **rep** 取值也应一样
  - C 一个接口可以有多个实现类，它们的功能应完全等价，但性能可能存在差异
  - D 对 **mutable** 的 ADT 来说，为保证安全性，不应该存在任何表示泄露
7. 关于 ADT 的 **RI** 的说法，不正确的是\_\_\_\_
- A **RI** 可以为空，表示 **rep** 的任何取值都是合法的
  - B 在客户端使用 ADT 的对象实例的全过程中，**RI** 都应始终保持为真
  - C **checkRep()** 是任何 ADT 都要从 **Object** 类 **override** 的方法，以检查 **RI** 是否为真
  - D 具有相同 **rep** 和相同 **AF** 的两个 ADT，其 **RI** 未必相同
8. 关于 OOP 的说法，正确的是\_\_\_\_
- A **Overload** 只能发生在父类型和子类型的方法之间
  - B 两个 **override** 的方法的 spec 应严格保持不变
  - C 两个 **overload** 的方法的返回值或参数个数要有差异
  - D 参数化多态中，可以有多于一个的泛型参数
9. 针对 Java，以下说法正确的是\_\_\_\_
- A 若子类型中有些方法在父类型中不存在，则无法通过静态检查
  - B 接口中的方法只有 spec 但没有实现体，类中的方法均有实现体
  - C 一个类可以实现多个接口，一个接口可以扩展多个接口，一个类不可以继承多个类
  - D 两个不等价的对象，其调用 **hashCode()** 时返回的结果也应不等价
10. 类 A 中有一个方法 `public Object method(List<Number> str)`，类 B 是类 A 的子类型且二者符合 LSP，那么 B 中的以下\_\_\_\_方法是可以通过 Java 静态检查的合法 **override** 方法
- A `public Date method(List<Number> abc)`
  - B `public Date method(ArrayList<Double> str)`
  - C `private Object method(List<Number> abc)`
  - D `public Object method(List<Number> str) throws Exception`
11. 针对 OOP 中 **delegation** 机制的说法，最恰当的是\_\_\_\_
- A 类 X 将某功能委派给类 Y，则 X 的 **rep** 中需要加一个属性，其类型为 Y，用于存储该委派关系
  - B 委派关系发生在对象实例层面，可以动态建立起对象实例之间的委派关系
  - C 一个类的实例无法与具有相同父类型的实例之间建立委派关系

D 两个对象之间的委派关系一旦建立就无法解除,后续将持续使用该关系进行功能委派

12. \_\_\_\_不是符合 OCP 原则的合理设计

- A 在黑盒框架中,引入 **Plugin** 接口,将具体应用中需要定制的个性化功能委派给 **Plugin** 的子类型
- B 使用 **instanceof** 操作符判断一个对象的具体类型,然后使用 **if/else** 结构调用具体类型的具体操作
- C 客户端要调用不兼容的方法,可增加一个适配器类,在其中将客户端请求转换为原有类的方法调用
- D 客户端代码中要尽可能使用接口或抽象类来定义对象,避免直接使用具体类

13. 关于 Template Method 设计模式的说法,不正确的是\_\_\_\_

- A 该设计模式中没有使用 **delegation** 关系
- B 白盒框架与其遵循相同的设计思想
- C 父类型中的模板方法前面通常使用 **final** 关键字
- D 客户端使用遵循该模式的 ADT 时,需要 **new** 的是父类型而不是子类型

14. 在 Java 中, \_\_\_\_被称为 checked 异常

- A **RuntimeException** 的子类
- B 继承自 **Error** 的异常
- C 实现 **Throwable** 接口的异常
- D 不继承自 **RuntimeException** 的异常

15. Java 中 \_\_\_\_是用于在程序中执行断言的关键字/方法

- A **assert ...**
- B **assertEquals(...)**
- C **assertion ...**
- D **assertTrue(...)**

## 二 简答题 (25 分)

Java 的 **String** 类中提供了一个方法: **public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)**, 它将某个 **String** 对象代表的字符串中的某些连续字符复制到目标字符数组中:

(1) 要复制的第一个字符位于索引 **srcBegin** 处,要复制的最后一个字符位于索引 **srcEnd-1** 处,因此要复制的总字符数为 **srcEnd-srcBegin**。

(2) 将字符复制到 **dst** 的子数组中,从索引 **dstBegin** 开始,直到索引 **dstBegin+(srcEnd-srcBegin)-1** 结束。

参数说明:

- **srcBegin**: 要复制的字符串中第一个字符的索引
- **srcEnd**: 要复制的字符串中最后一个字符之后的索引
- **dst**: 目标数组
- **dstBegin**: 目标数组中的起始偏移量

如果以下任何一项为真,该方法执行后将抛出 **IndexOutOfBoundsException**:

- (1) **srcBegin < 0**
- (2) **srcBegin > srcEnd**
- (3) **srcEnd** 大于该字符串的长度

- 
- (4) `dst == null`
  - (5) `dstBegin < 0`
  - (6) `dstBegin+(srcEnd-srcBegin) > dst.length`

示例 1: 以下代码将字符串 `str` 中的前 5 个字符"Hello"复制到一个长度为 5 的字符数组 `dst` 中。它是从源字符串 `str` 的第 0 个位置开始复制的, 一直复制到第 5 个位置之前; 这些字符被复制到目标字符数组 `dst` 的第 0 个位置开始。因此代码打印输出结果将是"Hello"。

```
String str = "Hello World";
final char[] dst = new char[5];
str.getChars(0, 5, dst, 0);
System.out.println(dst);
```

示例 2: 以下代码试图将源字符串 `str` 中的前 5 个字符复制到一个长度为 3 的字符数组 `dst` 中。但由于目标数组长度不够, 因此将抛出 `IndexOutOfBoundsException` 异常。

```
String str = "Hello World";
final char[] dst = new char[3];
str.getChars(0, 5, dst, 0);
```

**1 (10 分)** 根据该方法的 `spec`, 利用等价类划分方法, 为其设计一组测试用例, 能够较完备的测试对该方法的各种合法和错误的使用。下表给出的行数较多, 无需全部填满。

序号	字符串	src Begin	src End	dst	dst Begin	方法执行后预期结果 (包括: <code>dst</code> 中的内容、是否抛出异常)
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						

授课教师

姓名

学号

院系

密

封

线

**2 (5 分)** 若在 spec 的 pre-condition 中增加对参数 `srcBegin` 和 `secEnd` 的约束条件:

`0<=srcBegin <= secEnd、srcEnd <= this.length()`, 并从后置条件抛出异常的说明中去掉前三个条件, 请判断修改后的 spec 相比于修改前的 spec 强度如何变化 (变强、变弱、不变、无法比较), 并说明原因。

**3 (5 分)** 针对示例 1 中给出的四行代码, 绘制其执行之后的 snapshot 图。

**4 (5 分)** `IndexOutOfBoundsException` 是一个 unchecked 异常。若客户端程序中调用了 `getChars()` 方法, 执行后抛出了 `IndexOutOfBoundsException`, 那么该客户端程序的程序员应如何应对才最符合编程惯例?

---

## 综合实战题（45 分）

以下是三个类（Game、BasketballGame、Ranking）和客户端的代码。客户端首先构造若干球队的集合，这些球队之间两两捉对比赛，根据比赛结果统计各支队球的总积分，形成排行榜。

```
//immutable
class Game {
    private final String ta, tb;
    private final LocalDateTime date;
    private final int sa, sb;

    //AF: 代表一场比赛，ta 和 tb 表示参加比赛的两支队伍，
    //    date 表示比赛的日期/时间，sa 和 sb 表示两队在本场比赛中的得分
    //RI: ta.equals(tb) == false, sa >= 0, sb >= 0

    public Game (String ta, String tb, LocalDateTime date, int sa, int sb) {
        this.ta = ta;          this.tb = tb;
        this.sa = sa;          this.sb = sb;
        this.date = date;
    }
    public String getTeamA()    { return ta; }
    public String getTeamB()    { return tb; }
    public int getScoreA()      { return sa; }
    public int getScoreB()      { return sb; }
}

class BasketballGame extends Game {

    // AF: 代表一场篮球比赛，不允许有平分，必须分出胜负
    // RI: sa != sb

    public BasketballGame(String ta, String tb, LocalDateTime date, int sa, int sb){
        super(ta, tb, date, sa, sb);
    }
}

//mutable
class Ranking {
    private final Set<String> teams;
    private final Set<Game> games = new HashSet<>();
    private final Map<String, Integer> scores = new HashMap<>();
    private final Map<String, Integer> played = new HashMap<>();

    // AF: 代表一系列比赛后各支队伍的排行榜，包含以下信息：
    //    teams 表示参与排行的所有队伍；
    //    games 表示所有已经完成且被加入当前排行榜的比赛；
    //    scores 中的 key 表示队伍，value 表示该队伍获得的总积分；
    //    played 中的 key 表示队伍，value 表示该队伍已经比赛的场次
    // RI: TODO
}
```

```
/**
 * 构造一个排行榜对象
 *
 * @param teams 参与排行的所有队伍, not null
 */
public Ranking(Set<String> teams) {
    this.teams = teams;
    for (String t : teams) {
        scores.put(t, 0); //将各支队伍的总积分初始化为 0
        played.put(t, 0); //将各支队伍的已比赛场次初始化为 0
    }
}

/**
 * 将一场已完成的比赛加入到当前排行榜中, 更新两支球队的总积分和比赛场次
 *
 * @param g 一场已经完成的比赛, 该比赛的双方队伍均应已被加入当前排行榜对象
 * @throws GameExistException 如果该比赛已经被加入到当前排行榜中
 */
public void addGame(Game g) throws GameExistException {
    // 判断 g 是否已经被加入到当前排行榜中, 若是, 抛出异常, 算法结束
    if (games.contains(g))
        throw new GameExistException("该比赛已被加入, 不能重复加入");
    // 若否, 将 g 加入进来
    games.add(g);

    // 获取参加比赛的两支队伍
    String ta = g.getTeamA();
    String tb = g.getTeamB();
    // 将两支队伍的参赛场次分别+1
    played.put(ta, played.get(ta) + 1);
    played.put(tb, played.get(tb) + 1);

    // 获取两支队伍在比赛中的得分, 记录在 sa 和 sb 中。根据"赢、平、输"三种情况,
    // 分别给相应的队伍 3 分 (赢)、0 分 (平)、1 分 (输)
    int sa, sb;
    if (g.getScoreA() > g.getScoreB()) {
        sa = 3; sb = 0;
    } else if (g.getScoreA() == g.getScoreB()) {
        sa = 1; sb = 1;
    } else {
        sa = 0; sb = 3;
    }

    // 更新排行榜中两支球队的总积分
```

```

        scores.put(ta, scores.get(ta) + sa);
        scores.put(tb, scores.get(tb) + sb);
    }

    /**
     * 根据 scores 中各队伍的总积分进行从高到低的排序，生成一个字符串描述当前排行榜
     * 每一行包含四个部分："名次 队伍名称 总积分 比赛场次"，例如"1 HIT 10 3"
     */
    public String toString() {
        // 根据 scores 中包含的数据，按总积分进行排序，结果存储于 entries 中
        List<Map.Entry<String, Integer>> entries = new ArrayList<>(scores.entrySet());
        // 以下为对 entries 中元素进行排序的代码，省略
        ...
        // 以下为遍历 entries 生成输出字符串的代码，省略
        String str = ...
        return str;
    }
}

```

#### 客户端代码示例

```

1. public static void main(String[] args) {
2.     Set<String> teams = new HashSet<>();
3.     teams.add("HIT");
4.     teams.add("THU");
5.     teams.add("SJTU");

6.     Ranking r = new Ranking(teams);
7.     r.addGame(new Game("HIT", "THU", LocalDateTime.of(2023, 1, 1, 15, 0, 0), 1, 2));
8.     r.addGame(new Game("HIT", "SJTU", LocalDateTime.of(2023, 1, 5, 15, 0, 0), 3, 3));
9.     r.addGame(new Game("THU", "SJTU", LocalDateTime.of(2023, 1, 10, 15, 0, 0), 2, 0));

10.    System.out.print(r.toString());
11. }

```

注 1：以下各题目相互独立，做某个题目时无需考虑其他题目所要求的修改或扩展。

注 2：请从以下第三至第九题中选择 45 分的题目作答。教师按从前往后的次序对前 45 分的已答题目进行评分。只要答题区有答案，即认为该题已选，请务必谨慎！

**三 (10 分)** 目前 Game 的构造方法有五个参数，客户端构造 Game 对象时需要写 5 个参数，比较复杂。例如上面客户端代码第 7 行 addGame() 的参数中构造了一个 Game 对象，表示两支队伍 HIT 和 THU 在 2023 年 1 月 1 日 15:00 举行了比赛，比分为 1:2。

如果要简化一下该构造方法的使用，压缩为两个参数：第 1、2、4、5 个参数合并为 String info，而 date 参数不变，从而客户端可简化为：

```
new Game("HIT(1) vs THU(2)", LocalDateTime.of(2023, 1, 1, 15, 0, 0));
```



授课教师

姓名

学号

院系

密

封

线

请写出在 `Game` 类的构造函数中匹配该 `info` 参数语法的正则表达式，从而可以从中抽取出两支队伍的名称和得分。注：队伍名称由大写字母构成，比分是 `0` 或正整数。

**四 (10 分)** 请根据 `Ranking` 类的 AF 及其各个方法的 spec，写出其 RI。

**五 (10 分)** 观察 `Ranking` 类的 `addGame()` 方法。在当前代码中，该方法的 `pre-condition` 是被如何处理的？如果客户端代码调用 `addGame()` 时违反了该 `pre-condition`，运行时会产生什么效果？如何修改代码以增强其健壮性？

**六 (10 分)** 仍然是 `addGame()` 方法，其中有这样一行：

```
if (games.contains(g)) throw new GameExistException();
```

这句话是为了判定比赛 `g` 是否之前已经被加入到当前排行榜中了。但是，因为当前 ADT 设计中缺失某些内容，该代码并不能达到预期效果。为了让该行代码起作用，需要对那个（些）类做什么修改或扩展？简要写出你的思路。

---

**七 (10 分)** 仍然是 `addGame()` 方法，其中有一段 `if-else` 代码，这是基于当前足球比赛的规则来统计积分，即“胜、平、负”分别计 3、1、0 分。但这个规则对子类型 `BasketBallGame` 不适用（篮球比赛中胜者队伍得 1 分、负者队伍得 0 分）；即使对足球比赛而言，未来很有可能会更换积分规则，例如“胜、平、负”分别计 4、2、0 分。如何对代码进行改造，使之能够灵活适应上述情况？简要描述你的设计思路。

**八 (5 分)** 目前客户端使用 `Game g = new Game(...)` 或 `Game bg = new BasketballGame(...)` 的方式创建不同类型的 `Game` 对象。如何修改 ADT 的当前设计，可以使客户端在不需要了解 `Game` 及其子类型的情况下创建各种类型的 `Game` 对象？给出设计思路描述，并给出在你的设计思路下客户端创建不同类型 `Game` 对象的代码示例。

**九 (5 分)** 类 `BasketballGame` 相比于其父类 `Game`，RI 增加了 `sa != sb`。那么，`BasketBallGame` 和 `Game` 之间是否符合 LSP？请说明理由。另外，如何改造 `Game` 和 `BasketballGame` 的构造方法，以确保它们的 RI 不被违反？