Midterm Answer Key
CPE 203 – Winter 2019
Instructor: Foote

Errata:

```
Page 5, 1c:  "not" should be "now"...
    "Fill in the implementation of the following method, which
     is now part of HockeyLeague."

Page 9 / 3. Continued
    crayonPencil should be blackCrayonPencil
```

1. Creating Classes and Methods

Assume the following partial class/interface definitions for HockeyLeague, Team and Fan. Read all the code before you begin

a) Fill in all the methods marked **TODO** .

```
public class HockeyLeague {
    private final String name;
    private final Map<String, Team> teamsByName;
    private final List<Fan> fans;

    public HockeyLeague(String name) {
            teamsByName = new HashMap<String, Team>();
A:          this.name = name;
B:          this.fans = new ArrayList<Fan>();
    }

    public void addTeam(Team t){
C:      teamByName.put(t.getName(), t);
    }

    public Team getTeam(String name){
D:      return teamsByName.get(name);
    }

    public void addFan(Fan fan){
E:      fans.add(fan);
    }
```

```
   /*
    * This method of HockeyLeague returns a list of the team names
    * sorted in a case-sensitive manner.
    */
    public List<String> teamNamesSorted() {
        List<String> result
            = new ArrayList<String>(teamsByName.keySet());
        result.sort(      /* TODO  Fill this in */
            (String s1, String s2) -> {
                return s1.compareTo(s2);
             }
           );
       return result;
    }
A:  Uses compareTo()
B:  Correctly makes a class that implements Comparator<String>

   /* TODO
    * This method of HockeyLeague returns a string containing the name
    * and total number of teams, in the following format:
    *      “HockeyLeague Name (31 teams)”
    */
    public String toString() {
        return name + “(“ + teamsByName.size() + “ teams”;
    }
C:  Includes name
D:  includes teamsByName.size()



public class Team {
    private final String name;
    private int totalPlayerSalaries;  // in dollars

    public Team(Sring name, int totalPlayerSalaries) {
E:      this.name = name;
F:      this.totalPlayerSalaries = totalPlayerSalaries;
    }

    public String getName() {
G:      return name;
    }

    public int getTotalPlayerSalaries() {
H:      return totalPlayerSalaries;
    }

    public void addToTotalPlayerSalaries(final int amount) {
I:      totalPlayerSalaries += amount;
    }

}
```

(1 continued) Answer the following questions using the above classes.

b. Fill in the method below so that it creates and returns a `HockeyLeague` object with two teams and three fans, one for the first team and two for the second. Use any names and other values you like. This method is part of a class that is not `HockeyLeague`, `Team` or `Fan`.

```
public HockeyLeague createTestHockeyLeague() {
A:   HockeyLeague league = new HockeyLeague();
B:   Team t1 = new Team(“Mid Ice Crisis”, 1000);
     Team t2 = new Team(“Honey Badgers”, 2000);
C:   league.addTeam(t1);
     league.addTeam(t2);
D:   Fan f1 = new Fan(t1, 40);
     Fan f2 = new Fan(t2, 41);
     Fan f3 = new Fan(t2, 42);
E:   league.addFan(f1);
     league.addFan(f2);
     league.addFan(f3);
F:   return league;
J:   Uses local variables for teams (and not the less efficient and
     error-prone new Fan(league.getTeam(“...”)));
 }
```

c. Add a method to the `HockeyLeague` class to increase the total salary of every team in the league by a fixed amount. Fill in the implementation of the following method, which is not part of `HockeyLeague`.

```
    public void increaseTeamSalaries(final int amount)
    {
        for (Team t : teamsByName.values()) {
            t.addToTotalPlayerSalaries(amount);
        }
    }
G:  uses teamsByName.values(), or other valid way of getting teams
H:  iterates
I:  calls team.addToPlayerSalaries(fee)
```

2. Object.equals() and Object.hashCode()

Complete the following class. Implement equals and hashCode so that this class can be used as a key in a hash table, and make any other needed changes or additions to the class definition. You may assume that arguments to the constructor are never null.

```
public   final       class              PhoneNumber {

    public final String countryCode;
    public final String areaCode;
    public final String localNumber;

    public PhoneNumber(String countryCode, String areaCode,
                       String localNumber) {
        // implementation not shown, but it is correct and reasonable
    }

    public boolean equals(Object other) {
        // TODO:  Fill in code here
        if (other instanceof PhoneNumber) {
            PhoneNumber op = (PhoneNumber) other;
            return countryCode.equals(op.countryCode) &&
                     areaCode.equals(op.areaCode) &&
                     localNumber.equals(op.otherNumber);
        } else {
            return false;
        }
    }

    /**
     * Returns a value consistent with the definition of equals().
     */
    public int hashCode() {
        return Objects.hash(countryCode, areaCode, localNumber);
    }
```

A - class is final

B - equals checks null

C - equals uses String.equals on components

D - equals checks countryCode somehow, even if incorrectly

E - equals checks areaCode somehow, even if incorrectly

F - equals checks localNumber somehow, even if incorrectly

G - equals checks instanceof

H - equals does downcast

I - hashCode uses countryCode correctly

J - hashCode uses areaCode correctly

K - hashCode uses localNumber correctly

L - hashCode combines the three values appropriately

Z - Other bug

3. Continued

For each code fragment below, write **A** if the fragment will always compile and run, **M** if the fragment will compile but might fail at runtime, and **F** if the fragment will fail to compile. You may assume that the declared methods will not fail when called. Each code fragment is independent; an assignment statement in one does not affect the following fragments.

Code Fragment	A/M/F
`writingInstrument.write();`	A
`pen.write();`	A
`crayonPencil.write();`	A
`((Pencil) crayonPencil).sharpen();`	A
`((Crayon) crayonPencil).sharpen();`	F
`((Crayon) crayonPencil).write()`	A
`((CrayonPencil) writingInstrument).sharpen();`	M
`((FountainPen) crayon).leak();`	M
`writingInstrument = redCrayon;`	A
`writingInstrument = crayonPencil;`	A
`crayon = crayonPencil;`	A
`fountainPen = crayon;`	F
`redCrayon = writingInstrument;`	F
`pen = writingInstrument;`	F
`((WritingInstrument) redCrayon).peel()`	F
`((RedCrayon) pen).peel()`	M