|  |  |  |
| --- | --- | --- |
| **CS 1302 – Sample Test 5** | **Name:** | **KEY** |
| No electronics, notes, smart watches, *etc.* allowed. |  | Print: FirstName LastName |

1. (55 points) Write the entire *Game* class (see the handout provided)

**public** **class** Game {

 Map<Integer,Player> players = **new** HashMap<>();

 **public** **boolean** addPlayer(Player p) {

 **int** id = p.getId();

 **if**(players.containsKey(id)) {

 **return** **false**;

 }

 players.put(id,p);

 **return** **true**;

 }

 // Not asked for

 **public** **int** getNumPlayers() {

 **return** players.size();

 }

 **public** Player getPlayer(**int** id) {

 **if**(players.containsKey(id)) {

 **return** players.get(id);

 }

 **return** **null**;

 }

 **public** Player removePlayer(**int** id) {

 // Since assume player exists, no check for key

 **return** players.remove(id);

 }

 **public** Player lowestLevelPlayer() {

 Player lowLevelPlayer = **null**;

 **double** lowLevel = Double.***MAX\_VALUE***;

 **for**(Player p : players.values()) {

 **double** level = p.getLevel();

 **if**(level < lowLevel) {

 lowLevel = level;

 lowLevelPlayer = p;

 }

 }

 **return** lowLevelPlayer;

 }

 **public** ArrayList<Player> playersBelowList(**double** levelThreshold) {

 ArrayList<Player> playersBelow = **new** ArrayList<>();

 **for**(Player p : players.values()) {

 **double** level = p.getLevel();

 **if**(level < levelThreshold) {

 playersBelow.add(p);

 }

 }

 **return** playersBelow;

 }

 **public** Map<Integer, Player> playersBelowMap(**double** levelThreshold) {

 Map<Integer, Player> playersBelow = **new** HashMap<>();

 **for**(Player p : players.values()) {

 **double** level = p.getLevel();

 **if**(level < levelThreshold) {

 playersBelow.put(p.getId(),p);

 }

 }

 **return** playersBelow;

 }

 **public** List<Integer> idsBetween(**int** from, **int** to) {

 List<Integer> ids = **new** ArrayList<>();

 **for**(**int** id : players.keySet()) {

 **if**(id>=from && id<=to) {

 ids.add(id);

 }

 }

 **return** ids;

 }

 // Not asked for

 @Override

 **public** String toString() {

 String msg = "Players:\n";

 **for**(Player p : players.values()) {

 msg += p + "\n";

 }

 **return** msg;

 }

}

1. (10 points) Pick a problem from: (a) Ch 9, Sec 2.1, Exercises 1-4. (b) Ch 9, Sec 7.1, Exercises 5, 6, 9

**Solution: See code download for Ch 9.**

1. (10 points) Pick another problem from: (a) Ch 9, Sec 2.1, Exercises 1-4. (b) Ch 9, Sec 7.1, Exercises 5, 6, 9

There will be 2 or 3 recursion problems chosen from different types:

**Solution: See code download for Ch 10.**

1. (10 points) Pick a problem from: (a) Ch 10, examples in Sections 4.1-4.5, (b) Ch 10, Sec 4.6, Exercises 3-5.
2. (10 points) Pick a problem from Ch 10, examples in Sections 5.2-5.3
3. (10 points) Pick a problem from: (a) Ch 10, examples in Sections 6.1-6.3, or (b) Ch 10, Sec 6.4, Exercise 11
4. (10 points) Pick a problem from: (a) Ch 10, examples in Sections 7.1-7.3, or (b) Ch 10, Sec 7.4, Exercises 12-13
5. (10 points total) Given the skeleton of a binary search method and recursive helper method, that searches for an integer in an array (or ArrayList) of integers, fill in the blanks of the missing code. **The best way to study for this is to be able to write the code for the example in Ch 10, Sec 8.5.**

**Solution: See Ch 10, Sec 8.5**

1. (10 points total) Given the skeleton of a binary search method and recursive helper method, that searches for an *Employee* in an array (or ArrayList) of *Employee* objects, given that the *Employee* class implements the *Comparable* interface, fill in the blanks of the missing code. **The best way to study for this is to study the changes we made to example in Ch 10, Sec 8.5, when I presented a similar problem in class. Hint: the main thing you need to change is:** key < vals[mid] and key > vals[mid].

**Solution: Class Notes for April 4, 2024**