3. WRITTEN RESPONSES

3 a.

3.a.i.

Being a piano player, I often spend hours listening to hundreds of piano pieces when searching for the next piece I want to play. The program's purpose is to help a piano learner quickly find an appropriate piano piece based on its level, speed, and/or character.

3.a.ii.

First, the user inputs a piano piece's *level*, *speed*, and *character*, then the number of desired search results. The program searches the music list based on the user's input and prints the search result as output.

In another search, the user only inputs the level and the number of desired search results. The program has a different output from the first search because there are no restrictions on *speed* and *character*.

3.a.iii.

Input: The user enters the *level*, *character*, and *speed* of a piano piece, then the number of desired search results.
Output: Printing a list of piano pieces based on user input

3 b.

3.b.i.

```
/**
 * Generate a list of PianoPiece objects from a CSV file with all piano pieces
 *
 * @return pianoPieceList, a list of all PianoPiece
 */
private List<PianoPiece> createPianoPieceList() {
    List<PianoPiece> pianoPieceList = new ArrayList<>();
    InputStreamReader musicListReader = new InputStreamReader(
            getClass().getClassLoader().getResourceAsStream("music-list.csv"));
    try (BufferedReader br = new BufferedReader(musicListReader)) {
        for (String line = br.readLine(); line != null; line = br.readLine()) {
            String[] pieceCharacteristics = line.split("\\|");
            PianoPiece pp = new PianoPiece(pieceCharacteristics[0], pieceCharacteristics[2],
                    pieceCharacteristics[5], pieceCharacteristics[6], pieceCharacteristics[9],
                    pieceCharacteristics[8]);
            pianoPieceList.add(pp);
        }
    } catch (IOException ioe) {
        System.out.printf("IOException occurred. Error message: %s", ioe.getMessage());
    }
    return pianoPieceList;
}
```

3.b.ii.

```
/**
 * Find all PianoPiece that fulfill a search criteria. Return them in a list.
 *
 * @param searchOptions, a SearchOptions object; pianoPieceList, a list of all
 *                       PianoPiece
 * @return searchResultList, a list of PianoPiece that fulfill the criteria
 */
private List<PianoPiece> searchMusic(SearchOptions searchOptions, List<PianoPiece> pianoPieceList) {

    List<PianoPiece> searchResultList = new ArrayList<>();
    // no need to check level, input validation guarantees level is never empty
    boolean searchWithLevelOnly = searchOptions.getMusicCharacter().equals("")
            && searchOptions.getSpeed().equals("");
    boolean searchWithAllOptions = !searchOptions.getMusicCharacter().equals("")
            && !searchOptions.getSpeed().equals("");

    for (PianoPiece pianoPiece : pianoPieceList) {
        if (searchOptions.getNumResults() != searchResultList.size()) {
            // level is never empty, due to input validation in captureSearchOptions method
            // only need to check level once instead of checking inside each if
            if (pianoPiece.getLevel().equalsIgnoreCase(searchOptions.getLevel())) {
                if (searchWithLevelOnly) {
                    // user only entered level
                    searchResultList.add(pianoPiece);
                } else if (searchWithAllOptions
                        && pianoPiece.getMusicCharacter().equalsIgnoreCase(searchOptions.getMusicCharacter())
                        && pianoPiece.getSpeed().equalsIgnoreCase(searchOptions.getSpeed())) {
                    // user entered level, musicCharacter, and speed
                    searchResultList.add(pianoPiece);
                } else if (!searchWithAllOptions
                        && (pianoPiece.getMusicCharacter().equalsIgnoreCase(searchOptions.getMusicCharacter())
                                || pianoPiece.getSpeed().equalsIgnoreCase(searchOptions.getSpeed()))) {
                    // user entered level and either musicCharacter or speed
                    searchResultList.add(pianoPiece);
                }
            }
        } else {
            break; // stop looping because all pieces were found
        }
    }
    return searchResultList;
}
```

3.b.iii.

pianoPieceList

3.b.iv.

pianoPieceList contains all 139 *PianoPiece* objects from the manually created music-list.csv file.

3.b.v.

If I did not use the list, I would need to create 139 variables, one for each *PianoPiece* object. I would also need to repeat, 139 times, the code that compares each *PianoPiece* object with the *SearchOptions* object.

The high number of variables would make the program code cluttered and difficult to read. This code would also be non-scalable since if I added a *PianoPiece* object to the pianoPieceList, I would need to create another variable and write another code block comparing the new *PianoPiece* to the *SearchOptions* object.

3 c.

3.c.i.

```java
/**
 * Find all PianoPiece that fulfill a search criteria. Return them in a list.
 *
 * @param searchOptions, a SearchOptions object; pianoPieceList, a list of all
 *                       PianoPiece
 * @return searchResultList, a list of PianoPiece that fulfill the criteria
 */
private List<PianoPiece> searchMusic(SearchOptions searchOptions, List<PianoPiece> pianoPieceList) {

    List<PianoPiece> searchResultList = new ArrayList<>();
    // no need to check level, input validation guarantees level is never empty
    boolean searchWithLevelOnly = searchOptions.getMusicCharacter().equals("")
            && searchOptions.getSpeed().equals("");
    boolean searchWithAllOptions = !searchOptions.getMusicCharacter().equals("")
            && !searchOptions.getSpeed().equals("");

    for (PianoPiece pianoPiece : pianoPieceList) {
        if (searchOptions.getNumResults() != searchResultList.size()) {
            // level is never empty, due to input validation in captureSearchOptions method
            // only need to check level once instead of checking inside each if
            if (pianoPiece.getLevel().equalsIgnoreCase(searchOptions.getLevel())) {
                if (searchWithLevelOnly) {
                    // user only entered level
                    searchResultList.add(pianoPiece);
                } else if (searchWithAllOptions
                        && pianoPiece.getMusicCharacter().equalsIgnoreCase(searchOptions.getMusicCharacter())
                        && pianoPiece.getSpeed().equalsIgnoreCase(searchOptions.getSpeed())) {
                    // user entered level, musicCharacter, and speed
                    searchResultList.add(pianoPiece);
                } else if (!searchWithAllOptions
                        && (pianoPiece.getMusicCharacter().equalsIgnoreCase(searchOptions.getMusicCharacter())
                                || pianoPiece.getSpeed().equalsIgnoreCase(searchOptions.getSpeed()))) {
                    // user entered level and either musicCharacter or speed
                    searchResultList.add(pianoPiece);
                }
            }
        } else {
            break; // stop looping because all pieces were found
        }
    }
    return searchResultList;
}
```

3.c.ii.

```java
if (action.equalsIgnoreCase("s")) {
    SearchOptions searchOptions = app.captureSearchOptions(br);
    List<PianoPiece> searchResultList = app.searchMusic(searchOptions, pianoPieceList);
    app.printSearchResult(searchResultList);
} else {
    System.out.println("Invalid choice. Try again.");
}
```

3.c.iii.

searchMusic creates searchResultList (which contains a user-specified number of results) by looping through pianoPieceList and comparing *level*, *musicCharacter*, and *speed* between each object and user input (searchOptions).

searchMusic contributes to the program's overall functionality by returning the list of *PianoPiece* based on user input.

3.c.iv.

searchMusic has two input parameters: pianoPieceList and searchOptions (input validation ensures *level* is never empty). It outputs searchResultList (a list of *PianoPiece*).

First, create an empty list (searchResultList) and two boolean variables (*searchWithLevelOnly*, *searchWithAllOptions*) to identify if all options or only *level* is being used for searching. Then, loop through pianoPieceList until the length of searchResultList is equal to *numResults* in searchOptions, or the entire pianoPieceList is traversed.

In each iteration, do the following if the level of the current pianoPiece is equal to the level in searchOptions:

- if *searchWithLevelOnly* is true, add the current pianoPiece to searchResultList.
- else, if *searchWithAllOptions* is true and the values in searchOptions and pianoPiece are equal, add the current pianoPiece to searchResultList.
- else, if the *level* and either *musicCharacter* or *speed* in searchOptions has a value, and they are equal to those in the current pianoPiece, add the current pianoPiece to searchResultList.

Return searchResultList upon exiting the loop.

3 d.

3.d.i.

First call:

Pass pianoPieceList and a *SearchOptions* object with *level* set to "5", *speed* set to "fast", *character* set to "happy", and *numResults* set to 4.

Second call:

Pass pianoPieceList and a *SearchOptions* object with *level* set to "5", *numResults* set to 4, and others set to empty.

3 d.ii.

Condition(s) tested by first call:

Tests if:

1. Length of searchResultList is equal to the numResults in searchOptions
2. *level* in the current pianoPiece and searchOptions are equal
3. *searchWithLevelOnly* is true
4. *searchWithAllOptions* is true, and *musicCharacter* and *speed* in the current pianoPiece and searchOptions are equal

Condition(s) tested by second call:

Tests if:

1. Length of searchResultList is equal to the numResults in searchOptions
2. *level* in the current pianoPiece and searchOptions are equal
3. *searchWithLevelOnly* is true

3.d.iii.

Results of the first call:

4 *PianoPiece* objects with their *level* set to "5", *speed* set to "fast", and *musicCharacter* set to "happy" are added to searchResultList. Then, searchResultList is returned.

Results of the second call:

The first 4 *PianoPiece* objects in pianoPieceList with a *level* of "5", regardless of *speed* and *musicCharacter*, are added to searchResultList. Then, searchResultList is returned.