Task submission instruction and requirements

Task submission instruction

1. Register on GitHub (if not registered yet)
2. Create a new **private** repository with name firstName_secondName (i.e. ivan_ivaniv)
3. On your local machine create a **Maven** project.
4. Init git repo in your root project directory: '*git init*'
5. Add remote origin: '*git remote add origin https://github.com/xyz/ivan_ivaniv.git*'
6. Make changes in your project
7. Check that your project is compiled (use maven commands for that)
8. Commit and push changed files
9. Go to repository **Settings -> Manage access -> Invite a collaborator**
   and invite **imentor-review** user
10. Send an email with:

- First name
- Last name
- Email address that was used when registered to the program
- Link to the GitHub repository

Test project requirements

1. Project should be built with Java 1.8+ or scala 2.12
   Hints: use ***maven.compiler.source*** and ***maven.compiler.target*** properties to set version
2. Code should be properly formatted according to the Java code conventions
3. Any other libraries and frameworks could be used
4. `*mvn test*` — should run all tests, if tests are presented in the project
   Hint: do not forget about ***maven-surefire-plugin*** to run the tests
5. `*mvn exec:java*` — should execute project's Main method, if any exists
6. The project execution should be in two commands: `*git clone ...*`, `*mvn ...*`
7. Any configurations and resources should be bundled in the git repository or maven `pom.xml`

Task

Having *JoinOperation* interface, provide three implementations of this interface:

Java Task

```
public interface JoinOperation<D1, D2, R> {

    Collection<R> join(Collection<D1> leftCollection, Collection<D2> rightCollection);

}
```

1. InnerJoinOperation
2. LeftJoinOperation
3. RightJoinOperation

D1 - is a **generic** type of the elements in left collection

D2 - is a **generic** type of the elements in right collection

R - is a **generic** type of the elements in resulting collection

For the InnerJoinOperation, LeftJoinOperation and RightJoinOperation create two classes that hold the data:

`DataRow<K extends Comparable<K>, V>, where K is a generic type of the key, V is a generic type of the value.`

`JoinedDataRow<K extends Comparable<K>, V1, V2>, where K is a generic type of the key, V1 and V2 are generic types of the values`

For our task, DataRow<K, V> is your D1/D2 and JoinedDataRow<K, V1, V2> is R in the implementation classes. **IMPORTANT**: **Do not change JoinOperation interface!**

So, the signature in the InnerJoinOperation, LeftJoinOperation, and RightJoinOperation should look like next:

Collection<JoinedDataRow<K, V1, V2>> join(Collection<DataRow<K, V1>> leftCollection, Collection<DataRow<K, V2>> rightCollection);

Join should be performed by key: **K**

From an algorithm point of view:

1. For simplicity, we can say there are no duplicated keys in each separate collection.
2. The collections are always sorted.

Create JUnit tests for all implementations.

Junit test should be executed by ***mvn test*** command.

Example:

Having

```
leftCollection = [DataRow(0, "Ukraine"), DataRow(1, "Germany"), DataRow(2, "France")]

rightCollection = [DataRow(0, "Kyiv"), DataRow(1, "Berlin"), DataRow(3, "Budapest")]
```

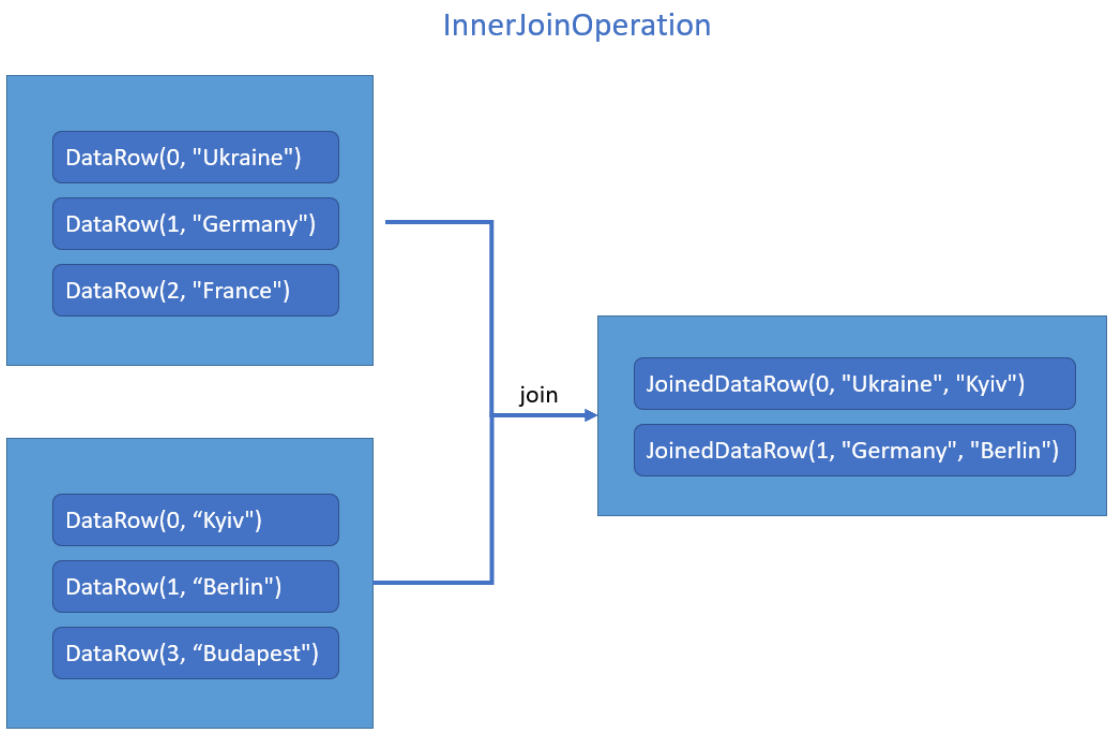
InnerJoinOperation.join gives: [JoinedDataRow(0, "Ukraine", "Kyiv"), JoinedDataRow(1, "Germany", "Berlin")]

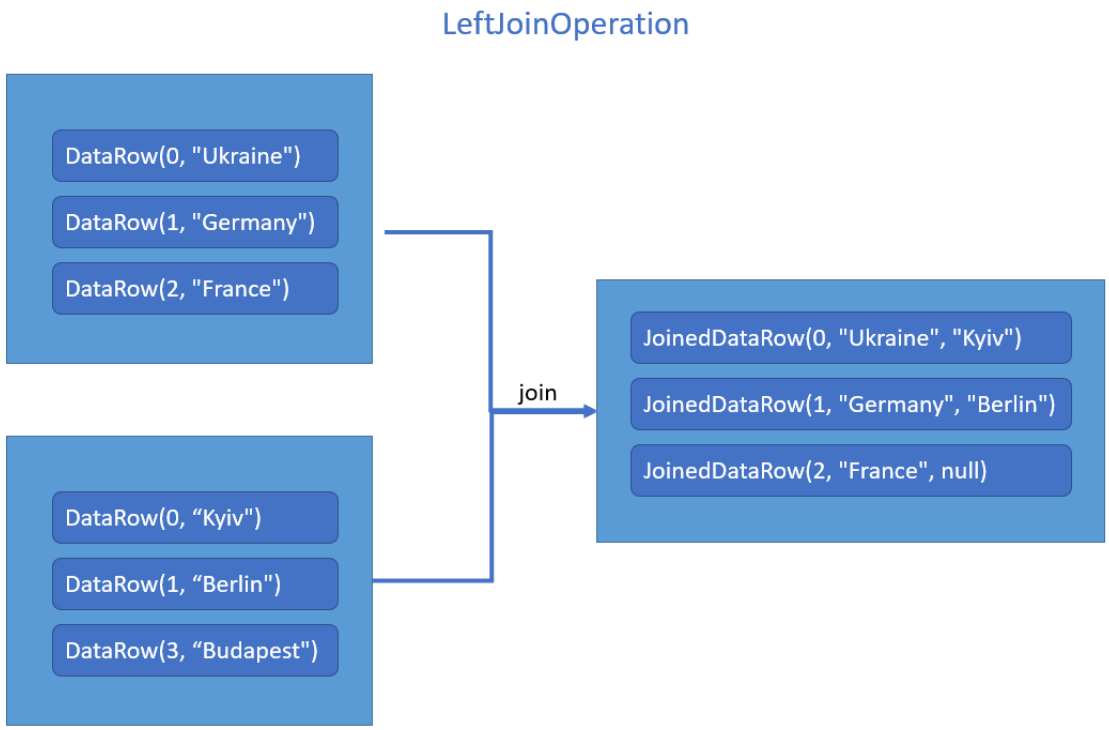
LeftJoinOperation.join gives: [JoinedDataRow(0, "Ukraine", "Kyiv"), JoinedDataRow(1, "Germany", "Berlin"), JoinedDataRow(2, "France", null)]

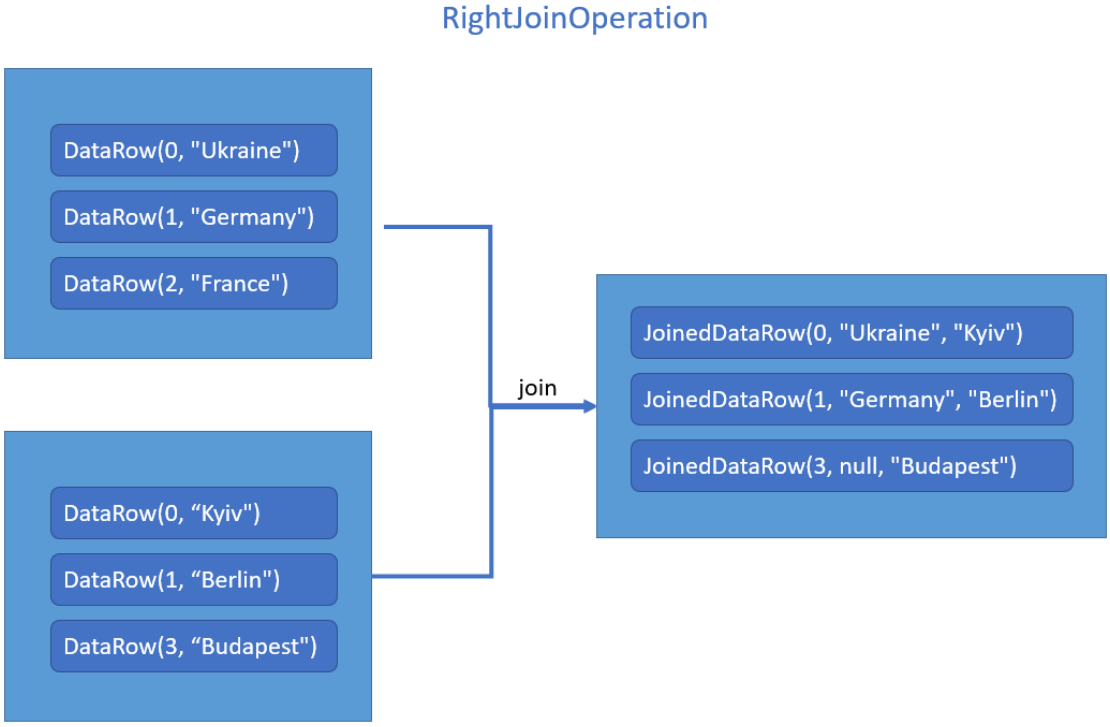
RightJoinOperation.join gives: [JoinedDataRow(0, "Ukraine", "Kyiv"), JoinedDataRow(1, "Germany", "Berlin"), JoinedDataRow(3, null, "Budapest")]

Pay attention on the order of the K, V1, V2 in the resulting set.

Graphical representation:







Additional hints:

1. You need simple understanding of Java Generics to complete this task.
2. You might notice that join operation is similar to SQL join operations. Yes, that's true, recall SQL it might help you to resolve the task!
3. Do not forget to test corner cases in your JUnit tests.
4. If you are stuck just make an assumption and implement the task based on it. Later we can discuss this on interview.