a. Define a class called `Q1`, containing a class/static method called **`diagonals`**. This method should return a **`String`** containing a pattern and take 2 **`int`** and 2 **`char`** parameters. The first **`int`** is the height and width of the pattern, the second **`int`** is the size of the gap between the diagonals in the pattern, and the **`char`** parameters are the characters that should be used in the pattern.

The method should return a string representing some diagonals shown using the first char parameter with the rest of the shape filled in with the second char parameter. The primary diagonal in the shape should be from top left to bottom right and, there should always be a constant number of characters between the diagonals which is determined by the second int parameter.

A runtime exception should be thrown in any of the following situations (The class `IllegalArgumentException` would be a good one to use, but you can use another or define your own if you want):

- The char parameters are equal
- The size is not in the range 7-31 (inclusive)
- The size of the gap between diagonals must be at least 1

The following are examples of the string returned by the code when the method is called with the parameters shown. Note there is no new line character at the end of the pattern.

`Q1.diagonals(10, 3, '+', ' ');`

```
+   +   +
 +   +   +
  +   +
   +   +
+   +   +
 +   +   +
  +   +
   +   +
+   +   +
 +   +   +
```

`Q1.diagonals(12, 1, 'X', '!');`

```
X!X!X!X!X!X!
!X!X!X!X!X!X
X!X!X!X!X!X!
!X!X!X!X!X!X
X!X!X!X!X!X!
!X!X!X!X!X!X
X!X!X!X!X!X!
!X!X!X!X!X!X
X!X!X!X!X!X!
!X!X!X!X!X!X
X!X!X!X!X!X!
!X!X!X!X!X!X
```

(20%)

(Question Total 20%)

```java
public class Q1 {

    public static String diagonals(int size, int gap, char char1, char char2) {
        String pattern = "";
        if(char1 == char2){
            throw new IllegalArgumentException();
        }
        if (size < 7 || size > 31){
            throw new IllegalArgumentException();
        }
        if (gap < 1){
            throw new IllegalArgumentException();
        }
        for(int i = 0; i < size; i++){
            for (int j = 0; j < size; j++){
                if (i == j || (j-i) % (gap + 1) == 0){
                    pattern += char1;
                }else {
                    pattern += char2;
                }

                if(j == size - 1 && i != size-1){
                    pattern += "\n";
                }
            }
        }
        return pattern;
    }

    public static void main(String[] args) {
        System.out.println(Q1.diagonals(10, 3, '+', ' '));
        System.out.println(Q1.diagonals(12, 1, 'X', '!'));
    }
}
```

Question 2: Modelling and Processing Files and Objects
The world cup is currently under way, but the organisers have been very forgetful. They need a system to help them keep track of the progress of the tournament and determine which teams have won in the groups and which games should be played next.

a. Define a class named FootballTeam which implements the Team interface provided. This class will be used to keep track of the progress of teams in the group stages of the tournament. This means we need to be able to remember or calculate all of the relevant statistics. The methods in the FootballTeam class should be implemented based on the following descriptions:
•public String getName() - Returns the name of the team
•public int getWins() - Returns the number of games the team has won
•public int getLosses() - Returns the number of games the team has lost
•public int getDraws() - Returns the number of games the team has drawn
•public int getPoints() - Returns the number of points the team has scored (3 for a win, 1 for a draw and 0 for a loss)
•public int getGoalsFor() - Returns the number of goals that this team has scored in all of their games
•public int getGoalsAgainst() - Returns the number of goals that other teams have scored when playing this team
•public int addGame(Game g) - Adds the result of a game that was played by this team.1 In order to function in the testing system, you must also add a constructor with the following signature:
public FootballTeam(String name).

```java
public class FootballTeam implements Comparable<FootballTeam>, Team {
    private final String name;
    private int wins;
    private int draws;
    private int losses;
    private int goalsFor;
    private int goalsAgainst;
    private int points;

    public FootballTeam(String name) {
        this.name = name;
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public void addGame(Game game) {
        Result result = game.getResult(this);
        if (result == Result.WIN) {
            wins++;
            points += 3;
        } else if (result == Result.DRAW) {
            draws++;
            points++;
        } else {
            losses++;
        }
        goalsFor += game.getGoalsFor(this);
        goalsAgainst += game.getGoalsAgainst(this);
    }

    @Override
    public int getWins() {
        return wins;
    }

    @Override
    public int getLosses() {
        return losses;
    }

    @Override
    public int getDraws() {
        return draws;
    }

    @Override
    public int getPoints() {
        return points;
    }

    public int compareTo(FootballTeam other) {
        if (this.getPoints() > other.getPoints()) {
            return -1;
        } else if (this.getPoints() < other.getPoints()) {
            return +1;
        } else {
            if (this.getGoalsFor() - this.getGoalsAgainst() > other.getGoalsFor() - other.getGoalsAgainst()) {
                return -1;
            } else if (this.getGoalsFor() - this.getGoalsAgainst() < other.getGoalsFor() - other.getGoalsAgainst()) {
                return +1;
            } else {
                if (this.getGoalsFor() > other.getGoalsFor()) {
                    return -1;
                } else if (this.getGoalsFor() < other.getGoalsFor()) {
                    return +1;
                } else {
                    return this.name.compareTo(other.name);
                }
            }
        }
    }

    @Override
    public int getGoalsFor() {
        return goalsFor;
    }

    @Override
    public int getGoalsAgainst() {
        return goalsAgainst;
    }
}

```

语法

```
public int compareTo( NumberSubClass referenceName )
```

参数

referenceName -- 可以是一个 Byte, Double, Integer, Float, Long 或 Short 类型的参数。

返回值

- 如果指定的数与参数相等返回 0。
- 如果指定的数小于参数返回 -1。
- 如果指定的数大于参数返回 1。

b. Define a class named FootballGame which implements the Game interface provided. This class will be used to remember the statistics of a single game in the tournament between two teams. The methods in the FootballGame class should be implemented based on the following descriptions:
•public Result getResult(Team team) - Returns the result of the game (Result.WIN, Result.LOSS, or result.DRAW) for the team passed as a parameter
•public int getGoalsFor(Team team) - Returns the number of goals that were scored by the team passed as a parameter
•public int getGoalsAgainst(Team team) - Returns the number of goals that were against by the team passed as a parameter
For all of these methods, if the team parameter that is passed does not match either of the teams in the game then an IllegalArgumentException should be thrown.

```java
public class FootballGame implements Game{
    private Team teamA;
    private Team teamB;
    private int scoreA;
    private int scoreB;

    public FootballGame(Team teamA, int scoreA, Team teamB, int scoreB){
        this.teamA = teamA;
        this.teamB = teamB;
        this.scoreA = scoreA;
        this.scoreB = scoreB;
    }

    @Override
    public Result getResult(Team a) {
        if (!a.equals(teamA) && !a.equals(teamB)) {
            throw new IllegalArgumentException();
        }

        if (a.equals(teamA)) {
            if (scoreA > scoreB) {
                return Result.WIN;
            } else if (scoreA < scoreB) {
                return Result.LOSS;
            } else {
                return Result.DRAW;
            }
        } else if (a.equals(teamB)) {
            if (scoreA > scoreB) {
                return Result.LOSS;
            } else if (scoreA < scoreB) {
                return Result.WIN;
            } else {
                return Result.DRAW;
            }
        }
        return null;
    }

    @Override
    public int getGoalsFor(Team a) {
        if (a.equals(teamA)) {
            return scoreA;
        } else   {
            return scoreB;
        }
    }

    @Override
    public int getGoalsAgainst(Team a) {
        if (a.equals(teamA)) {
            return scoreB;
        } else {
            return scoreA;
        }
    }
}
```

c. Define a class named WorldCupGroup which implements the Group interface provided.
This class will be used to determine which teams can progress to the next round of the tournament as well as producing the table of results. The methods in the WorldCupGroup class should be implemented based on the following descriptions:
•public Team getWinner() - Returns the team object of the team that came first in the group
•public Team getRunnerUp() - Returns the team object of the team that came second in the group
•public String getTable() - Returns a string containing the current table for the group.
The getTable method should have output in the following format. The group name should be on the first line, the column headers should be on the second line, and the teams and their statistics should be on the remaining lines. An example of the output is given here:

```
Group A:
Team                         W    D    L    F    A    P
Netherlands                  2    1    0    5    1    7
Senegal                      2    0    1    5    4    6
Ecuador                      1    1    1    4    3    4
Qatar                        0    0    3    1    7    0
```

In this the following abbreviations are made:
•W - For the number of games the team has won
•D - For the number of games the team has drawn
•L - For the number of games the team has lost
•F - For the number of goals the team has scored
•A - For the number of goals that have been scored against this team
•P - For the number of points the team has scored
The teams should be primarily ordered from largest to smallest by points. If teams are equal on points, then they should be secondarily ordered by goal difference (F - A). Finally, if all of these are equal then they should be orders based on their names. All of the methods in this class rely on the ability to correctly order the teams in this way. Consider what approach you will use to sort the objects into the correct order.
In order to function in the testing system, you must also add a constructor with the following signature:
public WorldCupGroup(String groupName, Team a, Team b, Team c, Team d)

```java
public class WorldCupGroup implements Group{
    private String groupName;
    private Team a;
    private Team b;
    private Team c;
    private Team d;


    public WorldCupGroup(String groupName, Team a, Team b, Team c, Team d){
        this.groupName = groupName;
        this.a = a;
        this.b = b;
        this.c = c;
        this.d = d;
    }

    @Override
    public String getName() {
        return groupName;
    }

    @Override
    public String getTable() {
        Team[] teams = new Team[] {a, b, c, d};
        Team winner = getWinner();
        Team runnerUp = getRunnerUp();
        String table = "";
        table += ("Group " + getName() + ":\n");
        table += String.format("%-20s %3s %3s %3s %3s %3s %3s \n", "Team", "W", "D", "L", "A", "P");
        table += formatTableRow(winner);
        table += formatTableRow(runnerUp);

        Team third = null;
        for (Team i : teams) {
            if (i != winner && i != runnerUp) {
                if (third == null || (i.getPoints() > third.getPoints()) || (i.getPoints() == third.getPoints() && (i.getGoalsFor() - i.getGoalsAgainst()) > (third.getGoalsFor() - third.getGoalsAgainst())) || (i.getPoints() == winner.getPoints() && i.getGoalsFor() - i.getGoalsAgainst() == winner.getGoalsFor() - winner.getGoalsAgainst() && i.getName().compareTo(winner.getName()) < 0)) {
                    third = i;
                    table += formatTableRow(third);
                } else {
                    table += formatTableRow(i);
                }
            }
        }
        return table;
    }

    private String formatTableRow(Team team) {
        return String.format("%-20s %3d %3d %3d %3d %3d %3d\n",
                team.getName(), team.getWins(), team.getDraws(), team.getLosses(),
                team.getGoalsFor(), team.getGoalsAgainst(), team.getPoints());
    }


    @Override
    public Team getWinner() {
        Team[] teams = new Team[] {a,b,c,d};
        Team winner = a;
        for (Team i :teams){
            if (i.getPoints() > winner.getPoints() ||
                    (i.getPoints() == winner.getPoints() && i.getGoalsFor() - i.getGoalsAgainst() > winner.getGoalsFor() - winner.getGoalsAgainst()) ||
                    (i.getPoints() == winner.getPoints() && i.getGoalsFor() - i.getGoalsAgainst() == winner.getGoalsFor() - winner.getGoalsAgainst() && i.getName().compareTo(winner.getName()) < 0)){
                winner = i;
            }
        }
        return winner;
    }

    @Override
    public Team getRunnerUp() {
        Team[] teams = new Team[] {a,b,c,d};
        Team winner = getWinner();
        Team runnerup = null;
        for (Team i : teams){
            if(i != winner) {
                if (runnerup == null ||
                        (i.getPoints() == runnerup.getPoints() && i.getGoalsFor() - i.getGoalsAgainst() > runnerup.getGoalsFor() - runnerup.getGoalsAgainst()) ||
                        i.getPoints() > runnerup.getPoints()){
                    runnerup = i;
                }
            }
        }
        return runnerup;
    }
}
```

d. Define an class named `KnockoutFixture`, which implements the `Fixture` interface provided. This class represents a game that has yet to take place. Override the `toString` method in the class such that when called it returns a String in the following format `"Team Name vs Team Name"`.

In order to function in the testing system, you must also add a constructor with the following signature:
`public KnockoutFixture(Team teamA, Team teamB)`.

(5%)

```java
public class KnockoutFixture implements Fixture{
    private Team teamA;
    private Team teamB;

    public KnockoutFixture(Team teamA, Team teamB){
        this.teamA = teamA;
        this.teamB = teamB;
    }
    @Override
    public String toString(){
        String info = teamA.getName()+" vs "+teamB.getName();
        return info;
    }
}
```

e. Define a class named `Q2`. In this class you should define the following method:

- `public static void readGroups(String filename, Map<String, Team> teams, Map<String, Group> groups)`

This method should read the contents of the file name passed as a parameter. Each line of the file contains the information about one group in the world cup. There is the name of the group followed by the name of the four teams all separated by commas.

f. Continuing to work in the class named `Q2`. You should define the following method:

- `public static void readGames(String filename, Map<String, Team> teams, List<Game> games)`

This method should read the contents of the file name passed as a parameter. Each line of the file contains the information about one game in the world cup. There is the name of the one of the teams followed by their score in the game, followed by the name of the other team and their score in the game. All of these values are separated by commas.

g. Continuing to work in the class named `Q2`. You should define the following method:

- `public static void calculateKnockouts(Map<String, Group> groups, List<Fixture> fixtures)`

This method should use the map containing the groups to determine the fixtures for the next part of the tournament. Games are played by the top two teams in each group and are paired between groups A and B, C and D, E and F, and G and H. So The winner of group A will play the runner up of group B and the winner of group B will play the runner up of group A and so on.

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;
import java.util.stream.Collectors;


public class Q2 {
    public static void readGroups(String filename, Map<String, Team> teams, Map<String, Group> groups) {
        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
```

```java
            String line;
            while ((line = reader.readLine()) != null) {
                String[] data = line.split(",");

                String groupName = data[0];

                for (int i = 1; i < data.length; i++) {

                    String teamName = data[i];
                    Team team = new FootballTeam(teamName);
                    teams.put(teamName, team);
                }

                Group group = new WorldCupGroup(groupName, teams.get(data[1]), teams.get(data[2]), teams.get(data[3]), teams.get(data[4]));
                groups.put(groupName, group);
            }


        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        } catch (IOException e){
            e.printStackTrace();
            throw new RuntimeException(e);
        }
    }


public static void readGames(String filename, Map<String, Team> teams, List<Game> games){
    try (BufferedReader reader = new BufferedReader(new FileReader(filename))){
        String line;
        while ((line = reader.readLine()) != null){
            String[] data = line.split(",");
            String teamAName = data[0];
            String teamBName = data[2];

            Team teamA = teams.get(teamAName);
            Team teamB = teams.get(teamBName);

            int scoreA = Integer.parseInt(data[1]);
            int scoreB = Integer.parseInt(data[3]);

            Game game = new FootballGame(teamA,scoreA,teamB,scoreB);

            teamA.addGame(game);
            teamB.addGame(game);

            games.add(game);
        }
    } catch (FileNotFoundException e) {
        throw new RuntimeException(e);
    } catch (IOException e){
        e.printStackTrace();
        throw new RuntimeException(e);
    }
}


    /*public static void calculateKnockouts(Map<String,Group> groups, List<Fixture> fixtures){
        List<Team> winners = new ArrayList<>();
        List<Team> runnerUps = new ArrayList<>();

        for (Group group : groups.values()){
            winners.add(group.getWinner());
            runnerUps.add(group.getRunnerUp());
        }

        for (int i = 0; i < winners.size(); i++) {
            Fixture KnockoutFixture = null;
            if (i % 2 == 0) {
                Team teamA = winners.get(i);
                Team teamB = runnerUps.get(i + 1);
                KnockoutFixture = new KnockoutFixture(teamA, teamB);
            } else {
                Team teamA = winners.get(i);
                Team teamB = runnerUps.get(i - 1);
                KnockoutFixture = new KnockoutFixture(teamA, teamB);
            }
            fixtures.add(KnockoutFixture);

        }
    }*/

    public static void calculateKnockouts(Map<String, Group> groups, List<Fixture> fixtures){
        fixtures.add(new KnockoutFixture(groups.get("A").getWinner(), groups.get("B").getRunnerUp()));
        fixtures.add(new KnockoutFixture(groups.get("B").getWinner(), groups.get("A").getRunnerUp()));

        fixtures.add(new KnockoutFixture(groups.get("C").getWinner(), groups.get("D").getRunnerUp()));
        fixtures.add(new KnockoutFixture(groups.get("D").getWinner(), groups.get("C").getRunnerUp()));

        fixtures.add(new KnockoutFixture(groups.get("E").getWinner(), groups.get("F").getRunnerUp()));
        fixtures.add(new KnockoutFixture(groups.get("F").getWinner(), groups.get("E").getRunnerUp()));

        fixtures.add(new KnockoutFixture(groups.get("G").getWinner(), groups.get("H").getRunnerUp()));
        fixtures.add(new KnockoutFixture(groups.get("H").getWinner(), groups.get("G").getRunnerUp()));

    }

    public static void main(String[] args) {
        Map<String, Team> teams2 = new TreeMap<>();
        Map<String, Group> groups2 = new TreeMap<>();
        List<Game> games2 = new ArrayList<>();
        List<Fixture> fixtures2 = new ArrayList<>();

        Q2.readGroups("groups.txt", teams2, groups2);
        Q2.readGames("games.txt", teams2, games2);
        //System.out.println("Group num "+ groups2.size());
        //System.out.println("Team num " + teams2.size());
        Q2.calculateKnockouts(groups2, fixtures2);

        //System.out.println(groups2.get("A").getWinner().getName());
        //System.out.println(groups2.get("A").getWinner().getGoalsFor());
        //System.out.println(groups2.get("A").getRunnerUp().getName());
        //System.out.println(groups2.get("B").getWinner().getName());
        //System.out.println(groups2.get("C").getWinner().getName());
        //System.out.println(groups2.get("D").getWinner().getName());
        //System.out.println(groups2.get("A").getTable());

        //System.out.println();


        List<String> games1 =
fixtures2.stream().map(Fixture::toString).collect(Collectors.toList());
        Collections.sort(games1);
        for (String s : games1) {
            System.out.println(s);
        }
    }

}
```