Milliways

Milliways, also known as *The Restaurant at the End of the Universe*, is a fictional eating place described by Douglas Adams in his *The Hitchhiker's Guide to the Galaxy*. In its vast halls, the restaurant accommodates different races, with different, possibly incompatible, requirements.

All classes must reside in the package named `it.polito.oop.milliways`.

The class named `MainClass` in package `main` contains examples using the main methods.

Students are free to access the JDK documentation.

R1: Races

The facade class for the system is `Restaurant` through which all objects are created.
The class `Race` represents a race in the multiverse. Each race has a unique name, specified as argument to the factory method `defineRace()` in class *`Restaurant`*; if the same name is specified when creating two objects the exception `MillywaysException` is thrown. The method returns an object of class `Race`. The method `getName` returns the name of the race.

A *race* has certain *requirements*. A requirements is a `String`, such as *`"Class M atmosphere (nitrogen-oxygen)"`*, or *`"Low gravity (less than 2g)"`*. The requirements for a given race are specified using the method `addRequirement`; if the same requirement is added more than once, the method throws the exception `MilliwaysException`. The method `getRequirements` returns the list of all requirements for the race, sorted alphabetically.

R2: Party

The class `Party` represents a group of table companions. A new party can be created by means of the method `createParty()` in class *`Restaurant`*.

A party is composed of companions from different races. The method `addCompanions()` adds to the party a given number of companions of a certain *race*.

The method `getNum()` with no arguments returns the total number of companions in the group. When a `race` is specified it returns the number of companions in the group of that specific race.

The method `getRequirements()` return a list of all the requirements for the party, that is, the union of requirements of all the races composing the party, sorted in alphabetically and without repetitions.

R3: Hall

The class `Hall` represents a hall at Milliways. Halls have infinity capacity and a certain number of *facilities*. Each hall has a unique numeric `id`, specified when an object is created using method `defineHall()` in class *`Restaurant`*. If the same id is specified when creating two objects the exception *`MilliwaysException`* is thrown; the method `getId()` returns the `id` of a hall.

The method `addFacility()` add a facility to the hall. A facility is represented by a *`String`*, for instance *`"Class X atmosphere (methane)"`*; if the same facility is added more than once, the method throws the exception *`MilliwaysException`*. The method `getFacilities()` returns the list of all facilities available in the hall, sorted alphabetically.

The method `isSuitable` check whether a party may be accommodated in the hall, that is, all its requirements are satisfied by the hall's facilities. A facility meets a requirement when the strings describing them are equal.

R4: Restaurant

The class `Restaurant` represent the whole Milliways, and it is used to record the parties.

The method `getHallList()` returns the list of all halls, in the order they have been inserted.

The method `seat()` called with two arguments, accommodates the given party in the given hall; if the hall was not *suitable* for the party, the exception *`MilliwaysException`* is thrown. When *`seat`* is invoked with only the *party* as argument, it scans all the halls in the restaurant and accommodates the party in the first suitable one. Halls are checked in order they have been added, and if not suitable hall is found, the exception *`MilliwaysException`* is thrown. Both methods returns the *hall* where the party has been successfully seated.

R5: Stats

Class *`Restaurant`* provides a few methods to compute stats on the Milliways.

The method `statComposition()` summarizes compositions of the parties, that is, the beings that ate at the restaurant, according to their race. The method returns a *`Map`* with the *race* as key, and the total number of table companions recorded so far in parties.

The method `statFacility()` reports all facilities available in the restaurant, sorted by the number of halls where they are available and then alphabetically.

The method `statHalls()` returns a *`Map`* having as key a number of facilities and as value the list of halls providing that number of facilities. Keys and halls are in ascending order.