Advanced Java

Project 4

In this project, you need to add more functionalities into your GUI chat system (for Project 3).

- Add the following commands:
  - /task tid TaskType Args
    - Create a Task object locally.
    - Example: /task t1 Pi 20
      - The class Pi must exist in the client side.
      - The class Pi must extend Task.
      - "20" is the initialization argument.
      - t1 is the series IDs of tasks. If tid is used, report an error.
      - Use reflection to fetch class Pi
  - /rexe tid [user]
    - For a client, ask the server to execute the task with ID tid via RMI.
    - If "user" is specified, the server asks the client named "user" to execute the tasks via RMI.
    - If "user" is not specified or the "user" does not exist, the server executes the tasks.
    - Example: /rexe t1 user
      - In this command /rexe, the client sends the task t1 to the host of user for execution remotely.
    - Note: task can be executed repeatedly.
  - /showtask
    - Show all the tasks for user

      E.g. Type

      /task t1 Pi 20

      → Display on the "Chat Region" panel:

      Task ID: t1, Task Type: Pi

      Type

      /task t2 GridifyStrings a b cc hello

      → Display on the "Chat Region" panel:

      Task ID: t2, Task Type: GridifyStrings

      Type

      /showtask

      → Display on the "Chat Region" panel:

      Task ID: t1, Task Type: Pi

Task ID: t2, Task Type: GridifyStrings

- Interface Task: The class of TaskType must extend Interface Task

```
interface Task extends Serializable {
    Object execute();
    void init(String init_str); // E.g., /task t1 Pi 20 → call init("20").
}
```

- Interface Compute: server and each client must create a RMI Remote object (which registers with rmiregistry), whose class implements Remote as follows.

```
interface Compute extends Remote {
    Object executeTask(Task t, String target) throws RemoteException;
}
```

- Naming for rmiregistry.
  - Each client uses his name for rebind().
  - Server uses the name "@SERVER".
- All classes defined below are in package **task**.

- Support Gridify:
  - A task is also a Gridify task with the following properties.
    - A Gridify task is the same as a task, except for the following situation.
      - In the command, "/rexe tid [user]", "user" is empty
      - In this case, the server will map the task into several tasks and then remotely call one client to work on each via RMI. Then, the returning results are reduced to one result which is sent back to the caller.
      - Hint: when remotely calling clients, use threads to do it in parallel.
  - A Gridify should be annotated as follows:

```
class GridifyStrings implements Task {
    String[] strings;

    // Return a vector of tasks.
    Vector<GridifyStrings> stringMapper(int num) {}
    Integer stringReducer(Vector<Integer> mapped_results) {}

    @Gridify(mapper="stringMapper", reducer ="stringReducer")
    Integer execute() { … }
}
```

e.g., /task g1 GridifyStrings a b c d e f g h i j k l m n o p q r s t u v w x y z

```
class GridifyPrime implements Task {
    long prime; // 379
```

```
    long min, max; // 2, 378
    long check_prime(long prime, long min, long max);

    // Return a vector of tasks.
    Vector<GridifyPrime> primeMapper(int num) {}
    // Collect a vector of mapped results and reduce them to one result.
    Long primeReducer(Vector<Long> mapped_results) {}

    @Gridify(mapper = "primeMapper", reducer = "primeReducer")
    Long execute() { … }
}
// Annotation definitions:
@Retention(RetentionPolicy.RUNTIME)
public @interface Gridify {
     String mapper() default "mapper" ;
     String reducer() default "reducer" ;
}
```

- mapper: E.g., Vector<GridifyPrime> primeMapper(int num)
  - Input:
    - num: The number of clients. So, the mapper knows how to split jobs.
  - Output:
    - Vector<GridifyPrime>: Vector of Tasks, which has the type GridifyPrime. So, the server can send out one task to each client.
- reducer: E.g., Long primeReducer(Vector<Long> mapped_results)
  - Input:
    - Vector<Long>: Vector of Task Result, which has the type Long.
  - Output:
    - Long: Return the final result, which has the type Long and must be the same as the type in Vector.
- execute: E.g., Long execute()
  - Input: (none)
  - Output:
    - Long: Return the final result, which has the type Long and must be the same as the type in reducer.
- Support a class DisplayResultForTA in clients as follows.
  - In order to let TA see the gridify result (TA can also use this to display TA's Gridify Task, like GridifyStrings), need to support the following function to show the result

execute() in WhiteBoard.

Class DisplayResultForTA {

public static String getUsername ( ) ; // get user name of client

public static void displayUsingWidget(String widget, int x, int y, String args);

// post a widget of type “widget” with argument=”args” at location x and y.

}

- Need to implement the following Tasks
  - Pi
    - Pi()
    - void init(String digits);
    - BigDicimal pi.execute() ;
    - E.g.

      Pi task = new Pi();

      task.init(“7”);

      task.execute() ; //3.1415927
  - GridifyPrime
    - GridifyPrime ()
    - void init(String arg)
      - arg = “value min-range max-range”
    - Long GridifyPrime.execute() ;
      - Return 1 if value is a prime
      - Return one of the divisors of value otherwise

        for(int divisor = min ; divisor <max ; divisor ++){

        if(value% divisor ==0) return divisor ;

        }

        return 1;
    - E.g.

      GridifyPrime task = new GridifyPrime () ;

      task.init(“121 1 121”);

      task.execute() ;// return 11

      GridifyPrime task2 = new GridifyPrime () ;

      Task2.init(“127 1 127”);

      task2.execute() ;// return 1, since 127 is a prime.
    - Support mapper and reducer method for Gridify annotation.
      - primeMapper
        - Split the range of divisor into pieces.
        - E.g.

          Prime task = new GridifyPrime () ; task.init(“121 1 121”);

--after mapping to two subTasks
Prime subTask1 = new GridifyPrime () ; task.init(“121 1 60”);
Prime subTask2 = new GridifyPrime () ; task.init(“121 61 121”);

- primteReducer
  - Collect all the subResult generated by Mapper, and merge into original result.

- Additional requirements:
  - Use multi-threading to handle different RMIs.
  - Show the result as String of remote execution in ChatRegion..
  - TA will give additional task class for testing when Demo, be sure all the reflection work well.

Good luck!

Due date: 6/12 (Upload your source code by 5:00PM on 6/12)
Demo date: TBA