***public interface GitRepository extends Serializable{,***

***String GIT\_FILE = ".mygit";***

***String commit(String commitMessage);***

***List<FileState> info();***

***String createBranch(String branchName);***

***String renameBranch(String branchName, String newName);***

***String deleteBranch(String branchName);***

***List<CommitMessage> log();***

***List<String> branches(); //list of branch names***

***List<Path> commitContent(String commitName);***

***String switchTo(String name); //name is either a commit name or a branch name***

***String getHead(); //return null if head refers commit with no branch***

***void save(); //saving to .mygit serialization to file (Object Stream)***

***String addIgnoredFileNameExp(String regex);***

}

1. Write an implementation of the above interface and static method init
   1. ***public static GitRepositoryImpl init()***  The file ***.mygit*** in the working (current) directory contains a snapshot of an object of the class GitRepositoryImpl. In the case of the file existence the method should return an object restored from the file, otherwise it returns the empty object
   2. ***public String commit(String commitMessage)*** adds commit. If reference to HEAD (consider the proper field as either name of commit or name of branch) contains name of commit it should return message saying about impossibility to do commit with no branch; if head equals null it should create first branch “master”. If working directory doesn’t have files in the either UNTRACKED or MODIFIED state it should return message saying about nothing to commit. Other requirements should follow your understanding of “commit”.
   3. ***public List<FileState> info()***  returns list of objects of the class FileState It creates object of the class FileState for each file in the working directory except those that have names matching a regular expression of the ignored files. Think of the structure, like Path and enum Status
   4. ***public String createBranch(String branchName)*** creates branch on the current commit and sets HEAD with that name. Returns a proper message saying about a result of creating
   5. ***public String renameBranch(String branchName, String newName)*** renames the specified branch with new name. In the case HEAD refers to being renamed branch the HEAD will be updated as well. Return a proper message
   6. ***public String deleteBranch(String branchName)*** deletes existing branch. Current branch (referred by the HEAD) cannot be deleted. Returns a proper message
   7. ***public List<CommitMessage> log()***  returns list of objects containing a pair of commit name and commit message from the HEAD to a first commit
   8. ***public List<String> branches()*** returns list of branch names. For current branch the asterisk follows the name
   9. ***public List<Path> commitContent(String commitName)*** Returns list of the Path objects of files included in a specified commit
   10. ***public String getHead()*** *Returns the name specified in the field “head” or null*
   11. ***public void save()*** Saves “this” to file “.mygit” of the working directory
   12. ***public String addIgnoredFileNameExp(String regex)*** adds regular expression for being ignored file names. If regex is incorrect returns a proper message. How to check regex correctness (“vasya”.matches(regex) throws exception in the case of incorrect regex
   13. Write method ***switchTo*** that takes either commit name or branch name and returns message saying about a result of the switching.
   14. Working directory contains the files in the state of either UNTRACKED or MODIFIED
       1. The method should return message like “switchTo may be done only after commit”
   15. The method takes a name equaled to the head
       1. The method should return a message like “switching to the current commit doesn’t make a sense
   16. Neither 1.1 nor 1.2
       1. Working directory should have exactly files with their content according to the commit or branch
       2. Head should contain the being switched either commit or branch
2. Write console application with menu
   1. Menu should contain the items matching the GitRepository interface
   2. Item exit should call the method save (no item for save)
3. Create runnable jar of the application (at applications start the static method init should be ran)
4. Move to some working directory and run the application jar
5. Run sanity test with the menu items
   1. Create three files and perform different commits with updated files and using such commands as “status” and “log” make sure that the application works properly
   2. Create branches and additional commits to different branches. Using the command “branches” and “log” make sure the application works properly
   3. Perform switchTo and by introspecting the file contents make sure that the application works properly
   4. Exit from application
   5. Start again and make sure that the previous state has been restored
6. **Assumptions:**
   1. Working directory doesn’t contain nested directories
   2. The user may only either create or update text files (neither remove nor rename)