```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Stack;

public class CommandProcessor implements ICommandProcessor {

	protected Stack<ICommand> commandStack = new Stack<>();
	private int commandsLimit;
	protected String fileName;
	protected Stack<ICommand> transactionStack = new Stack<>();
	protected boolean transactionProgress = false;
	protected boolean transactionSuccessful = false;
	private File file = new File("commands.txt");
	private FileWriter fileWriter = null;

	public CommandProcessor(int commandsLimit) {
		this.commandsLimit = commandsLimit;
	}

	@Override
	public boolean commit(ICommand command) {
		/**
		 * Snapshot is taken when command stack size is more
than commands Limit
		 * and no transaction is in progress
		 **/
		if (commandStack.size() + 1 > this.commandsLimit && !
this.transactionProgress) {
			Database.getDatabase().snapShot();
			try {
				fileWriter = new FileWriter(file, false);
				fileWriter.write("");
				fileWriter.flush();
				fileWriter.close();
			} catch (IOException e) {
				e.printStackTrace();
			}
			clear();
		}
		commandStack.push(command);
		return command.execute();
	}

	@Override
	public boolean commit(ICommand command, Stack<ICommand>
transactionDataStack) {
		commandStack.push(command);
		boolean executionResult = command.execute();
		if (transactionProgress) {
			transactionDataStack.push(command);
			if (this.transactionSuccessful) {
```

```
                        this.transactionSuccessful =
executionResult;
                }
            }
            return executionResult;
      }

      public int getCommandsLimit() {
            return commandsLimit;
      }

      public void setCommandsLimit(int commandsLimit) {
            this.commandsLimit = commandsLimit;
      }

      public ICommand currentCommand() {
            if (commandStack.isEmpty()) {
                  return null;
            }
            return commandStack.peek();
      }

      public boolean canUndo() {
            if (commandStack.isEmpty()) {
                  return false;
            }
            return currentCommand().canUndo();
      }

      public boolean canRedo() {
            if (commandStack.isEmpty()) {
                  return false;
            }
            return currentCommand().canRedo();
      }

      public boolean redo() {
            if (commandStack.isEmpty() || !
currentCommand().canRedo()) {
                  return false;
            }
            return commandStack.pop().execute();
      }

      public boolean undo() {
            if (commandStack.isEmpty() || !
currentCommand().canUndo()) {
                  return false;
            }
            return commandStack.pop().undo();
      }
```

```
	public void clear() {
		if (commandStack.isEmpty()) {
			return;
		}
		commandStack.clear();
	}

	@Override
	public void startTransaction() {
		this.transactionProgress = true;
		this.transactionSuccessful = true;
	}

	@Override
	public boolean isTransactionSuccessful() {
		return this.transactionSuccessful;
	}

	@Override
	public void endTransaction(Stack<ICommand> transactionStack)
{
		this.transactionProgress = false;
		transactionStack.clear();
	}

	@Override
	public void rollbackTransaction(Stack<ICommand>
transactionStack) {
		while (!transactionStack.empty()) {
			transactionStack.pop().undo();
		}
		this.transactionProgress = false;
	}

}
```