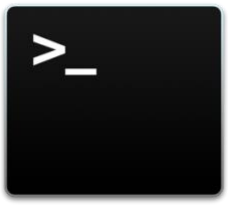
Project: Mini Shell

Java



Study Course

B206 Operating Systems

By

Chehab Hany Mohamed Elsayed Elsayed Elmenoufi

Student Number: GH1034223

Under the Guidance of

Prof. Mazhar Hameed

GitHub URL: https://github.com/Chippo90/Operating-Systems/tree/main

Video Recording URL: https://youtu.be/tNRB-2LiddU

Gisma University of Applied Sciences

Berlin, Germany

June 2025

Table of Contents

1. Introduction

In this project, I have developed a simplified shell in Java, named MiniShell, as part of the B206 Operating Systems module. The shell shows basic functions of a terminal, focusing on only three commands: cd, ls, and exit.

This implementation shows operating system concepts like process control and file system navigation. ('B206-0-Java-Introduction.pdf', no date)

2. System Design

The MiniShell is structured around a loop that reads user input, parses the command, and delegates execution to the handler. (Silberschatz, Galvin and Gagne, no date)

The system is having the following components:

2.1 Shell Loop

- Ask the user for input.
- Maintain the directory.
- Route the input to the command parser.

2.1 Command Parser

- Split the input string into command and comments.
- Identify the commands (cd, ls, exit).
- Move unsupported commands to error handler.

2.2 Command Handlers

- CD Handler: Change the directory.
- LS Handler: List files in the directory.
- EXIT Handler: Exit the program.
- Unsupported Commands: Display an error message.

2.3 File System Interaction

- Use Java's File class to act with the file system.
- Make sure the right listing of directories.

3. Implementation

The MiniShell is implemented in Java using standard libraries, ensuring to be simple and compatible across sevral operating systems. (*Intro to Java Programming - Course for Absolute Beginners*, 2019)

Below are the components used in the implementation:

3.1 Imports

```
import java.io.File;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
```

- **import java.io.File;**: Allow working with file and directory.
- **import java.util.HashMap**: Provide a map to store commands.
- **import java.util.Map;**: Define the interface used by HashMap.
- **import java.util.Scanner;**: Enable reading user input. (*Java Packages*, no date)

3.2 Classes

```
public class MiniShell {
    private File currentDirectory;  4 usages
    private Map<String, Object> commands;  5 usages
```

- **public class MiniShell**: Declare the main class for the shell.
- **private File currentDirectory;**: Store the directory.
- **private Map<String, Object> commands;**: Store command names.(*Java Classes & Objects - YouTube*, no date)

3.3 Registering Commands

```
public MiniShell() {  1 usage
    currentDirectory = new File(System.getProperty("user.dir"));

    commands = new HashMap<>();
    commands.put("cd", new CdCommand());
    commands.put("ls", new LsCommand());
    commands.put("exit", new ExitCommand());
}
```

- **MiniShell**: Start the shell.
- **System.getProperty("user.dir")**: Call the current directory.
- **commands.put**: Register the cd, ls, and exit commands.

3.4 Main Loop

```
public void run() {  1 usage
    Scanner scanner = new Scanner(System.in);
    boolean running = true;
```

- **run**: Start the shell.
- **Scanner scanner = new Scanner(System.in);**: Create a scanner to read input.
- **boolean running = true;**: Alert to control the loop.

```
while (running) {
    System.out.print(currentDirectory.getAbsolutePath() + " $ ");
    String input = scanner.nextLine().trim();
    String[] parts = input.split( regex: "\\s+");
    if (parts.length == 0 || parts[0].isEmpty()) continue;
```

- **while (running)**: Repeat until exit.
- **System.out.print**: Display the directory.

- **scanner.nextLine().trim**: Read and the input.
- **input.split**: Split input by spaces into command and comments.
- **if (...) continue;**: Skip empty input.

```
String cmd = parts[0];
Object command = commands.get(cmd);
```

- **cmd**: Extract the command.
- **commands.get**: Get the command.

```
if (command != null) {
    switch (cmd) {
        case "cd":
            ((CdCommand) command).run(parts, shell: this);
            break;
        case "ls":
            ((LsCommand) command).run( shell: this);
            break;
        case "exit":
            running = ((ExitCommand) command).run();
            break;
    }
} else {
    System.out.println("Unknown command: " + cmd);
}
```

- **switch(cmd)**: Execute the matching command.
- **CdCommand**: Run the cd command.
- **LsCommand**: Run the ls command.
- **ExitCommand**: Run the exit command.
- **else**: Print error.

```
    scanner.close();
}
```

- **scanner.close**: Close the scanner after shell exits.

```
public File getCurrentDirectory() {  2 usages
    return currentDirectory;
}

public void setCurrentDirectory(File dir) {  1 usage
    currentDirectory = dir;
}
```

- **getCurrentDirectory**: Return the directory.
- **setCurrentDirectory(File dir)**: Update the directory.

```
public static void main(String[] args) {
    new MiniShell().run();
}
```

- **main(String**: Enter the application.
- **new MiniShell().run**: Start the shell.

3.5 CD Command

```
import java.io.File;

public class CdCommand {  2 usages
    public void run(String[] args, MiniShell shell) {  1 usage
        if (args.length < 2) {
            System.out.println("Usage: cd <directory>");
            return;
        }

        File newDir = new File(shell.getCurrentDirectory(), args[1]);
        if (newDir.exists() && newDir.isDirectory()) {
            shell.setCurrentDirectory(newDir);
        } else {
            System.out.println("Directory not found: " + args[1]);
        }
    }
}
```

- **import java.io.File:** Allow working with file and directory.
- **public class cdcommand:** Declare the CdCommand class.
- **public void run:** Define the run method.
- **System.out.println:** Check if a directory name was provided.
- **File newDir = new File:** Create a new File.
- **if newDir.exists:** Check if the directory exists.
- **System.out.println**: Print an error if the directory doesn't exist.

3.6 LS Command

```
import java.io.File;

public class LsCommand {  2 usages
    public void run(MiniShell shell) {  1 usage
        File[] files = shell.getCurrentDirectory().listFiles();
        if (files != null) {
            for (File f : files) {
                System.out.println((f.isDirectory() ? "[FOLDER] " : "      ") + f.getName());
            }
        }
    }
}
```

- **import java.io.File:** Allow working with file and directory.
- **public class LsCommand:** Declare the LsCommand class.
- **public void run**: Define the run method.
- **shell.getCurrentDirectory:** Retrieve all files and directories.
- **if (files != null):** Check if the directory can be accessed.
- **System.out.println:** Move through each file or folder.
- **Adds a** [FOLDER] prefix if the item is a directory, for differentiation.

3.7 Exit Command

- **public class ExitCommand**: Declare the ExitCommand class.
- **public boolean run():** Define the run method.
- **System.out.println:** Print a message informing that the shell is closing.
- **return false:** Return false to stop the main loop in the MiniShell class.

4. Challenges and Solutions

4.1 Multi Operating Systems Compatibility

- Challenge: Ensuring the shell works on different operating systems.
- Solution: Relied on Java File class, which maintain OS-specific file system details.

4.2 Unsupported Commands

- Challenge: Preventing the shell from crashing on unknown input.
- Solution: Implemented a case in the command parser to report unsupported commands.

4.3 Keeping Track

- Challenge: Keeping track of the current directory across commands.
- Solution: Used (currentDirectory) that is updated by the CD Command.

5. Results

The MiniShell was tested on Windows and MacOS and successfully executed the commands:

- CD: Changed directories.
- LS: Displayed directory contents.
- EXIT: Terminated the shell.

6. Conclusion and Future Work

This project provided hands on experience with operating system concepts. The MiniShell demonstrates how a shell can be built using Java libraries while maintaining simplicity.

Future Work:

- Implement User Interface.
- Implement command history and auto completion.
- Add new commands like mkdir, tree, etc.

7. References

‘B206-0-Java-Introduction.pdf’ (no date).

Intro to Java Programming - Course for Absolute Beginners (2019). Available at: https://www.youtube.com/watch?v=GoXwIVyNvX0 (Accessed: 24 June 2025).

Java Classes & Objects - YouTube (no date). Available at: https://www.youtube.com/watch?v=IUqKuGNasdM (Accessed: 24 June 2025).

Java Packages (no date). Available at: https://www.w3schools.com/java/java_packages.asp (Accessed: 24 June 2025).

Silberschatz, A., Galvin, P.B. and Gagne, G. (no date) ‘Operating System Concepts’.