Dodatkowe wskazówki do realizacji zadania nr 1

Zastosowanie wzorca projektowego Factory – stworzyć rejestr komend (uwaga: założenie, że stosujemy wzorzec projektowy *Strategy* tj. wszystkie komendy rozszerzają klasę abstrakcyjną *Command*, która posiada metodę abstrackyjną *excecute*):

```
public class CommandRegistry {
    private final Map<String, Class<? extends Command>> commandMap = new HashMap<>();

    // Register commands in the factory
    public void registerCommand(String name, Class<? extends Command> commandClass) {
        commandMap.put(name, commandClass);
    }

    // Create a command instance based on the string
    public Command createCommand(String commandName) {
        Class<? extends Command> commandClass = commandMap.get(commandName);

        if (commandClass == null) {
            throw new IllegalArgumentException("Unknown command: " + commandName);
        }

        try {
            return commandClass.getDeclaredConstructor().newInstance();
        } catch (Exception e) {
            throw new RuntimeException("Failed to create command instance", e);
        }
    }
}
```

W metodzie main:

```
CommandRegistry commandFactory = new CommandRegistry();
commandFactory.registerCommand("checkin", CheckinCommand.class);
commandFactory.registerCommand("checkout", CheckoutCommand.class);
commandFactory.registerCommand("view", ViewCommand.class);
commandFactory.registerCommand("list", ListCommand.class);
commandFactory.registerCommand("save", SaveCommand.class);
commandFactory.registerCommand("exit", ExitCommand.class);


while (true) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Please enter the command - valid commands are: view, list, checkin,
checkout, save, exit: ");
    System.out.println();
    String cmd = scanner.nextLine();
    Command command = commandFactory.createCommand(cmd);
    command.setHotel(hotel);

    if (command == null) {
        System.err.println("No such command, please try again...");
        continue;
    }
    command.execute();

}
```

Zalety:

1. **Extensibility**: dodanie nowej komendy wymaga utworzenia nowej klasy dziedziczącej po Command (alternatywa to zdefiniowanie interfejsu Command) oraz zarejestrowania jej w fabrce w metodzie main.
2. **Open/Closed Principle**: system jest otwarty na rozszerzenia i zamknięty na zmiany (nie są wymagane żadne zmiany w istniejącym kodzie podczas dodawania nowych funkcjonalności do programu).
3. **Dynamic Loading**: teoretycznie można nawet ładować komendy z zewnętrznych źródeł.

Dodatkowe rozszerzenie – auto rejestracja komend

Żeby uniknąć rejestrowania komend w metodzie *main* można skanować pakiet z komendami z wykorzystaniem mechanizmu refleksji (jest do tego biblioteka *Reflections*, która to ułatwia, ale można poprzez *classpath scanning*).