Design Patterns

1 – Observer Pattern

Path: net.sf.freecol/client/gui/CanvasMouseListener.java

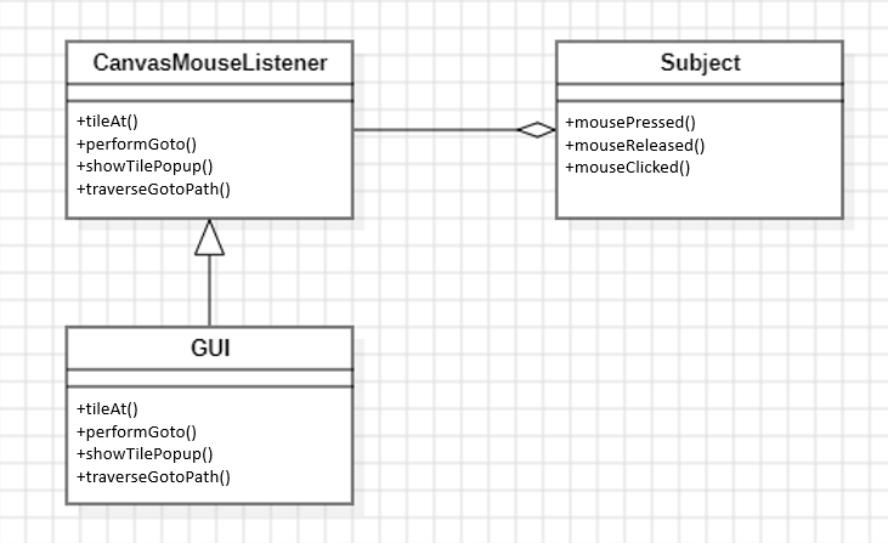
A classe CanvasMouseListener atua como um observador de eventos do 'mouse', respondendo dinamicamente às interações do usuário na área do Canvas. A relação um-para-muitos entre o componente Canvas e o CanvasMouseListener reflete o design pattern correspondente, permitindo que outros componentes sejam notificados e atualizados automaticamente quando ocorrem eventos de mouse durante o jogo.

```java
/**
 * Listens to mouse buttons being pressed at the level of the Canvas.
 */
Mike Pope +6
public final class CanvasMouseListener extends FreeColClientHolder
    implements MouseListener {

    private static final Logger logger = Logger.getLogger(CanvasMouseListener.class.getName());


    /**
     * Create a new mouse listener.
     *
     * @param freeColClient The enclosing {@code FreeColClient}.
     */
    Mike Pope +1
    public CanvasMouseListener(FreeColClient freeColClient) { super(freeColClient); }
```

Diagrama:



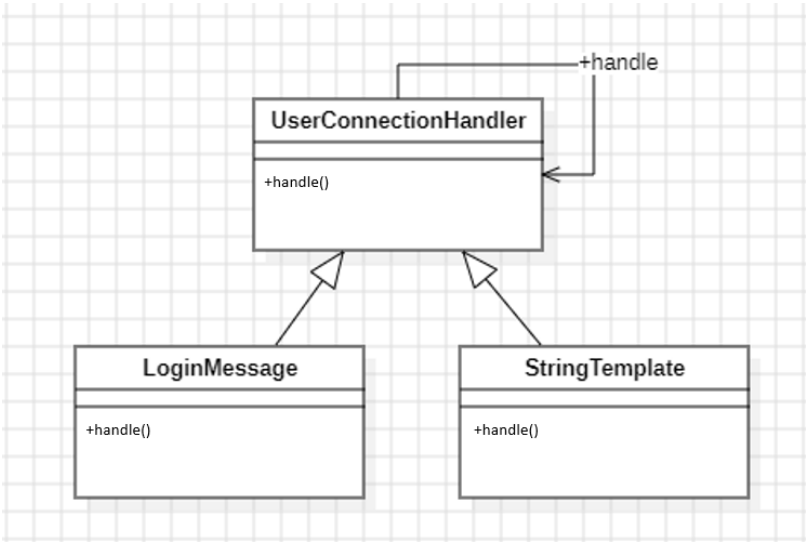
2 – Chain of Responsibility Pattern

Path: net.sf.freecol/server/control/UserConnectionHandler.java

Este design pattern gere mensagens para novas conexões, utilizando o método “handle” para lidar com diferentes tipos de mensagens (como LoginMessage). Isso proporciona flexibilidade e expansibilidade no processamento de mensagens do servidor.

```java
/**
 * {@inheritDoc}
 */
Mike Pope
public Message handle(Connection connection, Message message)
    throws FreeColException {
    final FreeColServer freeColServer = getFreeColServer();
    ChangeSet cs = null;
    switch (message.getType()) {
    case DisconnectMessage.TAG:
        break;
    case LoginMessage.TAG:
        cs = ((LoginMessage)message).loginHandler(freeColServer, connection);
        break;
    default:
        cs = ChangeSet.clientError((Player)null,
            StringTemplate.template( value: "server.couldNotLogin"));
        break;
    }
    return (cs == null) ? null
        : cs.build(freeColServer.getPlayer(connection));
}
```

Diagrama:



3 – Command Pattern

Path: net.sf.freecol/client/gui/action/ExecuteGotoOrdersAction.java

O design pattern Command guarda um comando como um objeto, para poder ser usado em várias classes. Ele oferece flexibilidade ao parametrizar clientes com diferentes comandos. Desta forma, deixa o código mais flexível e reutilizável.

```java
/**
 * An action for executing goto orders immediately.
 */
public class ExecuteGotoOrdersAction extends MapboardAction {

    public static final String id = "executeGotoOrdersAction";


    /**
     * Creates a new {@code ExecuteGotoOrdersAction}.
     *
     * @param freeColClient The {@code FreeColClient} for the game.
     */
    public ExecuteGotoOrdersAction(FreeColClient freeColClient) {
        super(freeColClient, id);
    }


    // Interface ActionListener

    /**
     * {@inheritDoc}
     */
    @Override
    public void actionPerformed(ActionEvent ae) {
        igc().executeGotoOrders();
    }
}
```

Diagrama: