GoF Design Patterns

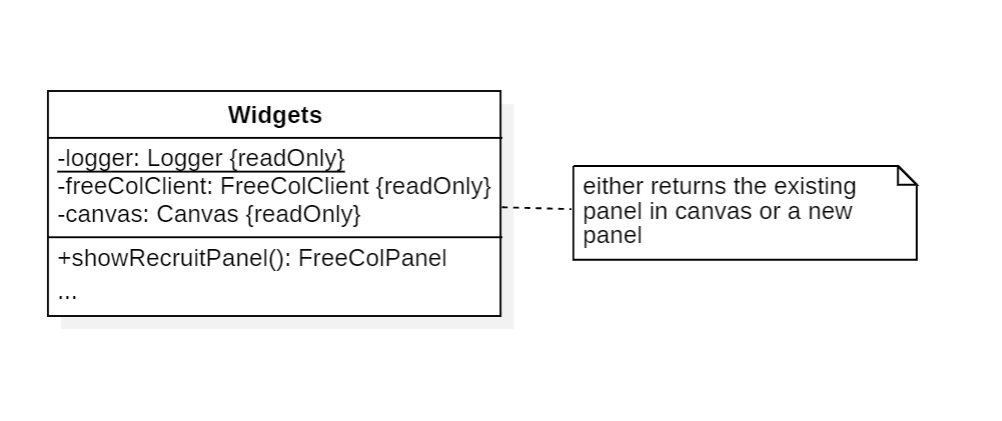
Singleton

| public FreeColPanel showPurchasePanel() {  PurchasePanel panel  = this.canvas.getExistingFreeColPanel(PurchasePanel.class);  if (panel == null) {  panel = new PurchasePanel(this.freeColClient);  panel.update();  this.canvas.showFreeColPanel(panel, PopupPosition.CENTERED, false);  }  return panel; } |
| --- |

This GoF design pattern is found implemented in the method net.sf.freecol.client.gui.Widgets.showPurchasePanel.

As we can see, this method restricts the initialization of the PurchasePanel class, ensuring that only one instance of the class is created. It starts by searching for an instance in the canvas, and only creates the object if it isn’t found.

The Class diagram for this pattern is as follows:



Template

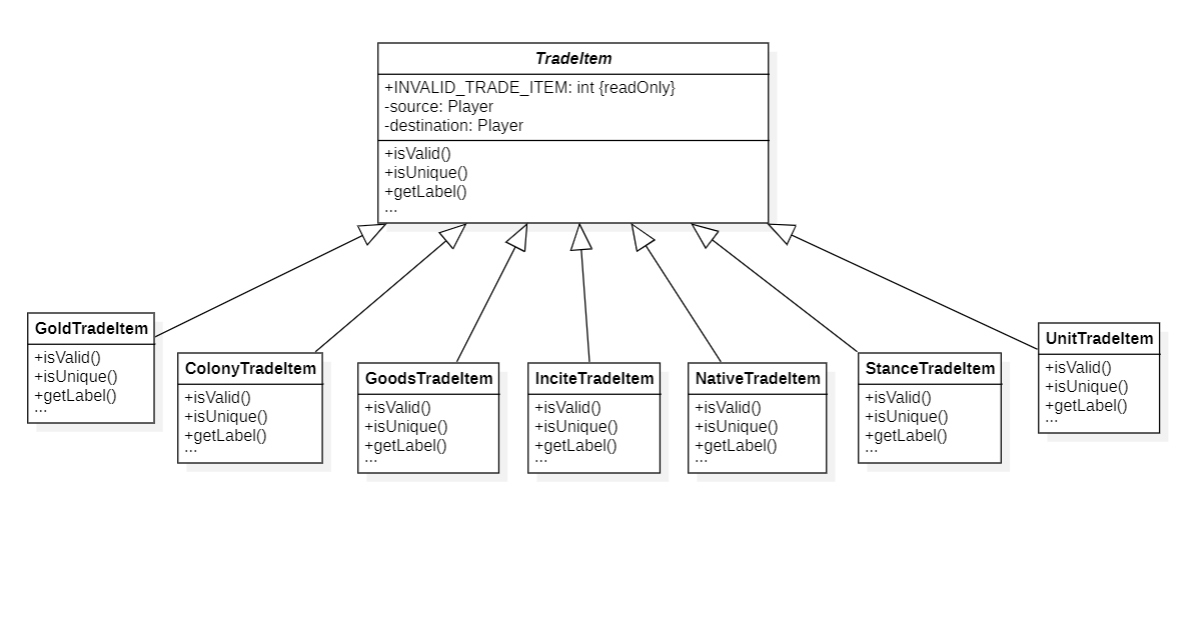
| public abstract class TradeItem extends FreeColGameObject {  ...   /\*\*  \* Is this trade item valid? That is, is the request well formed.  \*  \* **@return** True if the item is valid.  \*/  public abstract boolean isValid();   /\*\*  \* Is this trade item unique?  \* This is true for the StanceTradeItem and the GoldTradeItem,  \* and false for all others.  \*  \* **@return** True if the item is unique.  \*/  public abstract boolean isUnique();   /\*\*  \* Get a label for this item.  \*  \* **@return** A {**@code** StringTemplate} describing this item.  \*/  public abstract StringTemplate getLabel();   /\*\*  \* Get the colony to trade.  \*  \* **@param** game A {**@code** Game} to look for the colony in.  \* **@return** The {**@code** Colony} to trade.  \*/  public Colony getColony(Game game) { return null; }   ...   /\*\*  \* Get the gold to trade.  \*  \* **@return** The gold to trade.  \*/  public int getGold() { return 0; }  ...  } |
| --- |

| public class ColonyTradeItem extends TradeItem {  ...   /\*\*  \* {**@inheritDoc**}  \*/  @Override  public boolean isUnique() {  return false;  }   /\*\*  \* {**@inheritDoc**}  \*/  @Override  public StringTemplate getLabel() {  return StringTemplate.template(Messages.descriptionKey("model.tradeItem.colony"))  .addName("%colony%", colonyName);  }   /\*\*  \* {**@inheritDoc**}  \*/  @Override  public Colony getColony(Game game) {  return game.getFreeColGameObject(colonyId, Colony.class);  }  ...  } |
| --- |

This GoF design pattern is found in the class net.sf.freecol.common.model.TradeItem and its subclasses.

This class acts as a general trade item class, where some of the more generic trade item methods are implemented and some of the specific methods are left to be implemented in the subclasses.

The Class diagram for this pattern is as follows:



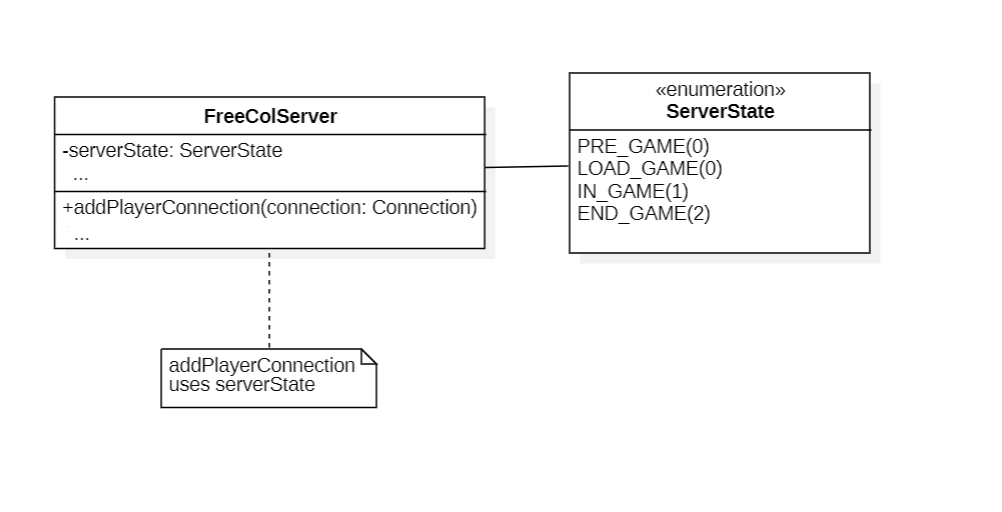
State

| public final class FreeColServer {   ...   /\*\* The current state of the game. \*/  private ServerState serverState = ServerState.PRE\_GAME;   ...   /\*\*  \* Add player connection. That is, a user connection is now  \* transitioning to a player connection.  \*  \* **@param** connection The new {**@code** Connection}.  \*/  public void addPlayerConnection(Connection connection) {  switch (this.serverState) {  case PRE\_GAME: case LOAD\_GAME: case IN\_GAME:  connection.setMessageHandler(this.inputHandler);  break;  case END\_GAME: default:  return;  }   getServer().addConnection(connection);  updateMetaServer();  }  ...  } |
| --- |

This GoF design pattern is found in the class net.sf.freecol.server.FreeColServer.

This class has a private attribute called serverState which stores an enumeration indicating the state of the server. We can see that it has some methods that change their behavior according to the state of the class.

The Class diagram for this pattern is as follows:



Tiago Sousa, 63324