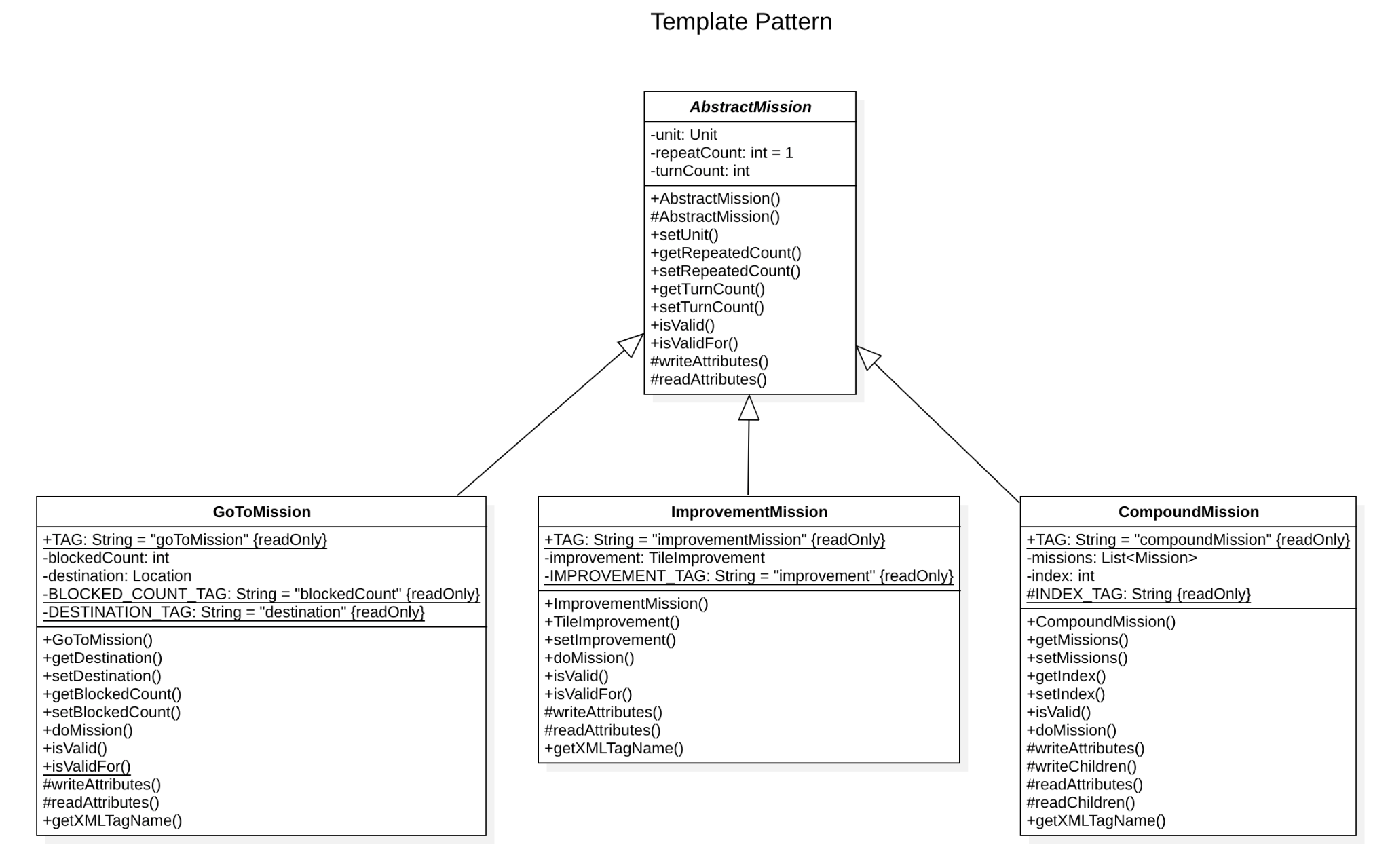
Design Patterns

# Template Pattern



In the class net.sf.freecol.common.model.mission.AbstractMission and in its subclasses net.sf.freecol.common.model.mission.GoToMission, net.sf.freecol.common.model.mission.ImprovementMission and net.sf.freecol.common.model.mission.CompoundMission we can identify an instance of the Design Pattern Template Pattern. The abstract class represents a general mission, providing methods prevalent for all missions upon which the subclasses build through their specific methods.

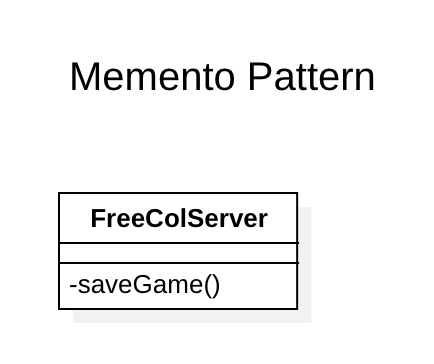
net.sf.freecol.common.model.mission.AbstractMission

| /\*\* \* The AbstractMission provides basic methods for building Missions. \*/ public abstract class AbstractMission extends FreeColGameObject implements Mission {  /\*\* \* Returns true if the Unit this mission was assigned to is \* neither null nor has been disposed, and the repeat count of the \* mission is greater than zero. \* \* **@return** a {**@code** boolean} value \*/ @Override public boolean isValid() {  return repeatCount > 0  && unit != null && !unit.isDisposed(); }  /\*\* \* {**@inheritDoc**} \*/ @Override protected void writeAttributes(FreeColXMLWriter xw) throws XMLStreamException {  super.writeAttributes(xw);   xw.writeAttribute(UNIT\_TAG, unit);   xw.writeAttribute(TURN\_COUNT\_TAG, turnCount);   xw.writeAttribute(REPEAT\_COUNT\_TAG, repeatCount); }  /\*\* \* {**@inheritDoc**} \*/ @Override protected void readAttributes(FreeColXMLReader xr) throws XMLStreamException {  super.readAttributes(xr);   unit = xr.makeFreeColObject(getGame(), UNIT\_TAG, Unit.class, true);   turnCount = xr.getAttribute(TURN\_COUNT\_TAG, 0);   repeatCount = xr.getAttribute(REPEAT\_COUNT\_TAG, 1); } |
| --- |

net.sf.freecol.common.model.mission.GoToMission

| /\*\* \* The GoToMission causes a Unit to move towards its destination. \*/ public class GoToMission extends AbstractMission {  /\*\* \* Returns true if the mission is still valid. \* \* @return a {@code boolean} value \*/ @Override public boolean isValid() {  // **FIXME:** check for disposed destinations  return destination != null && destination.canAdd(getUnit())  && super.isValid(); } /\*\* \* {@inheritDoc} \*/ @Override protected void writeAttributes(FreeColXMLWriter xw) throws XMLStreamException {  super.writeAttributes(xw);   xw.writeAttribute(DESTINATION\_TAG, destination);   xw.writeAttribute(BLOCKED\_COUNT\_TAG, blockedCount); }  /\*\* \* {@inheritDoc} \*/ @Override protected void readAttributes(FreeColXMLReader xr) throws XMLStreamException {  super.readAttributes(xr);   destination = xr.getLocationAttribute(getGame(), DESTINATION\_TAG,  false);   blockedCount = xr.getAttribute(BLOCKED\_COUNT\_TAG, 0); } |
| --- |

# Memento Pattern

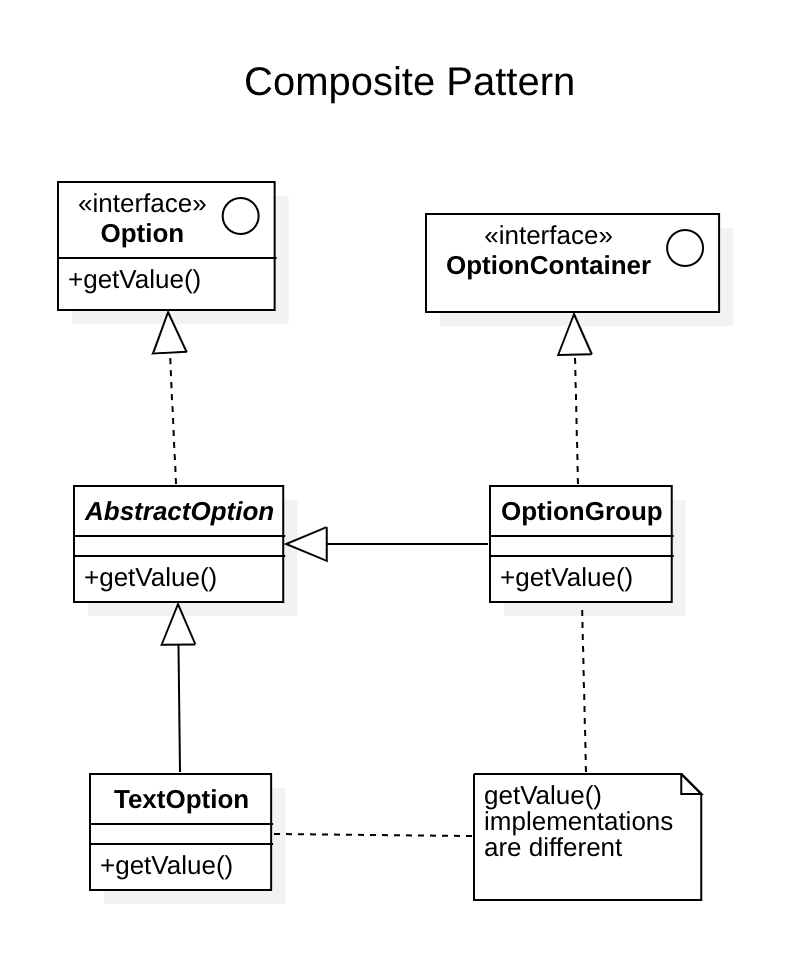


In the class net.sf.freecol.server.FreeColServer we find an example of the Design Pattern Memento Pattern. The private method saveGame() from the class FreeColServer proceeds to save a game, storing a state in which the necessary data is kept so that this instance of the game may be returned to if so wished.

net.sf.freecol.server.FreeColServer

| /\*\* \* Saves a game. \* \* @param file The file where the data will be written. \* @param owner An optional name to use as the owner of the game. \* @param options Optional client options to save in the game. \* @param active An optional active {@code Unit}. \* @param image A thumbnail {@code Image} value to save in the game. \* @exception IOException If a problem was encountered while trying \* to open, write or close the file. \*/ private void saveGame(File file, String owner, OptionGroup options,  Unit active, BufferedImage image) throws IOException {  // Try to GC now before launching into the save, as a failure  // here can lead to a corrupt saved game file (BR#3146).  // Alas, gc is only advisory, but it is all we have got.  garbageCollect();  try (JarOutputStream fos = new JarOutputStream(Files  .newOutputStream(file.toPath()))) {  if (image != null) {  fos.putNextEntry(new JarEntry(FreeColSavegameFile.THUMBNAIL\_FILE));  ImageIO.write(image, "png", fos);  fos.closeEntry();  }   if (options != null) {  fos.putNextEntry(new JarEntry(FreeColSavegameFile.CLIENT\_OPTIONS));  options.save(fos, null, true);  fos.closeEntry();  }   Properties properties = new Properties();  properties.setProperty("map.width",  Integer.toString(this.serverGame.getMap().getWidth()));  properties.setProperty("map.height",  Integer.toString(this.serverGame.getMap().getHeight()));  fos.putNextEntry(new JarEntry(FreeColSavegameFile.SAVEGAME\_PROPERTIES));  properties.store(fos, null);  fos.closeEntry();   // save the actual game data  fos.putNextEntry(new JarEntry(FreeColSavegameFile.SAVEGAME\_FILE));  try (  // throws IOException   FreeColXMLWriter xw = new FreeColXMLWriter(fos,  FreeColXMLWriter.WriteScope.toSave(), false)) {  xw.writeStartDocument("UTF-8", "1.0");   xw.writeComment(FreeCol.getConfiguration().toString());  xw.writeCharacters("\n");   xw.writeStartElement(SAVED\_GAME\_TAG);   // Add the attributes:  xw.writeAttribute(OWNER\_TAG,  (owner != null) ? owner : FreeCol.getName());   xw.writeAttribute(PUBLIC\_SERVER\_TAG, this.getPublicServer());   xw.writeAttribute(SINGLE\_PLAYER\_TAG, this.singlePlayer);   xw.writeAttribute(FreeColSavegameFile.VERSION\_TAG,  SAVEGAME\_VERSION);   xw.writeAttribute(RANDOM\_STATE\_TAG,  getRandomState(this.random));   xw.writeAttribute(DEBUG\_TAG, FreeColDebugger.getDebugModes());   if (active != null) {  this.serverGame.setInitialActiveUnitId(active.getId());  }   this.serverGame.toXML(xw); // Add the game   if (this.aiMain != null) { // Add the AIObjects  this.aiMain.toXML(xw);  }   xw.writeEndElement();  xw.writeEndDocument();  }  fos.closeEntry();  } catch (XMLStreamException e) {  throw new IOException("Failed to save (XML): " + file.getName(), e);  } } |
| --- |

# Composite Pattern



In the interface net.sf.freecol.comon.option.Option and class net.sf.freecol.common.option.OptionGroup we can identify an instance of the Design Pattern Composite Pattern. The class OptionGroup extends the abstract class AbstractOption, which in turn implements the interface Option, therefore, the class OptionGroup also implements this interface, granting it the method getValue(). This method will produce different results in the circumstance where we’re only dealing with a single Option and when we’re handling an OptionGroup, despite both implementing the same interface. From the perspective of the program, there isn't a way of discerning whether we’re handling an Option or an OptionGroup, consequently, we can label this as a Composite Pattern.

net.sf.freecol.comon.option.Option

| /\*\* \* Gets the value of this option. \* \* @return The value of this {@code Option}. \*/ public T getValue(); |
| --- |

net.sf.freecol.common.option.OptionGroup

| /\*\* \* {**@inheritDoc**} \*/ @Override public OptionGroup getValue() {  return this; } |
| --- |

Beatriz de Caiola Machado, 62551