```java
package fi.paatti.research.paattiapplication.views.eventview;


import com.vaadin.data.Item;
import com.vaadin.data.util.sqlcontainer.SQLContainer;
import com.vaadin.data.util.sqlcontainer.query.QueryDelegate;
import com.vaadin.data.util.sqlcontainer.query.QueryDelegate.RowIdChangeEvent;
import com.vaadin.event.MouseEvents;
import com.vaadin.event.MouseEvents.ClickEvent;
import com.vaadin.terminal.StreamResource;
import com.vaadin.ui.Button;
import com.vaadin.ui.Embedded;
import com.vaadin.ui.VerticalLayout;
import fi.paatti.containers.TaskContainer;
import fi.paatti.containers.ChoiceContainer;
import fi.paatti.paattidatabaseutils.dbservice.PaattiResearchDBService;
import fi.paatti.paattidatabaseutils.dbservice.querydelegates.*;
import fi.paatti.paattidatabaseutils.names.PaattiColumnNames;
import fi.paatti.paattidatabaseutils.names.PaattiTableNames;
import fi.paatti.research.paattiapplication.ElementNames;
import fi.paatti.research.paattiapplication.PaattiResearchApplication;
import fi.paatti.research.paattiapplication.views.ApplicationView;
import fi.paatti.research.paattiapplication.views.ApplicationViewTab;
import fi.paatti.research.paattiapplication.views.eventview.tools.ContextMenu;
import fi.paatti.research.paattiapplication.views.eventview.tools.ContextMenu.MenuItem;
import fi.paatti.research.paattiapplication.views.eventview.tools.NodeFactory;
import fi.paatti.research.paattiapplication.views.eventview.tools.NodeFactory.MultiPathNode;
import fi.paatti.research.paattiapplication.views.eventview.tools.NodeFactory.MultiPathNode.NodePath;
import fi.paatti.research.paattiapplication.views.eventview.tools.NodeFactory.NODE_TYPE;
import fi.paatti.research.paattiapplication.views.eventview.tools.NodeFactory.RootNode;
import fi.paatti.research.paattiapplication.views.eventview.tools.NodeFactory.TreeNode;
import fi.paatti.research.paattiapplication.views.eventview.tools.SettingsView;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.logging.Level;
import javax.imageio.ImageIO;


/**
 * EventTool contains the tool(s) that are used to create an event tree.
 *
 * @author Tapio Keränen, t.tapio.keranen@student.jyu.fi
 */
public class EventTool extends ApplicationViewTab {
  private static final long serialVersionUID = 1L;
  // The size of the embedded canvas.
  private static final int CANVAS_WIDTH = 800, CANVAS_HEIGHT = 600;
  // The handle to the application needed by the drawing process.
  private PaattiResearchApplication application;
  // The connector that handles the drawing and the creation of the image file.
  private ConnectorSource connectors;
  // The resource that connects the canvas with the image.
  private StreamResource connectorResource;
  // The embedded canvas that displays the event tree.
  private Embedded canvas;
  // The factory that provides the functionality for creating new nodes.
  private NodeFactory nodeFactory;
  // The list that contains all the tree’s nodes.
  private ArrayList<TreeNode> nodes;
  // The layout that contains the currently selected node’s settings.
  private SettingsView settingsView;
  // The RMB activated context menu that displays various actions related to
  // what (node or empty area) was clicked.
  private ContextMenu contextMenu;

  /**
   * The enum that contains the node modes used by the event tool. The mode is
   * checked during a mouse click event that the MouseClickListener added to
   * the canvas registered. Take a look at the TreeViewListener subclass below
   * to see how these modes are being used.
   *
   * @author Tapio Keränen, t.tapio.keranen@student.jyu.fi
   */
  public enum NODE_MODES {
    SELECT_NODE, ADD_CHILD, ADD_NODE, MOVE_NODE, REMOVE_NODE
  }
  // The current node mode.
  private NODE_MODES nodeMode;

  // The row ids used when writing the event to the database.
  private Object eventRowId, taskRowId, choiceRowId;
// The message displayed to the user when trying to save a malformed event.
private String saveWarningMessage;

/**
 * EventTool constructor.
 *
 * @param view a parent view.
 * @param app a handle to the main application.
 */
  public EventTool(ApplicationView view, PaattiResearchApplication app) {
    super(view);

    application = app;
  contextMenu = new ContextMenu();
  connectors = new ConnectorSource();
  connectorResource = new StreamResource(connectors, connectors.makeImageFilename(), application);
  nodeFactory = new NodeFactory();
  nodes = new ArrayList<TreeNode>();
  nodeMode = NODE_MODES.SELECT_NODE;

  createLayout();
  }

  /**
  * {@inheritDoc}
  */
  @Override
  protected void createLayout() {
  canvas = new Embedded(null, connectorResource);
  canvas.addListener(new TreeViewListener());
  canvas.setWidth(CANVAS_WIDTH, UNITS_PIXELS);
  canvas.setHeight(CANVAS_HEIGHT, UNITS_PIXELS);

  settingsView = new SettingsView(this);

   VerticalLayout layout = new VerticalLayout();

   layout.addComponent(canvas);
   layout.addComponent(settingsView);

   setCompositionRoot(layout);
  }

  /**
  * {@inheritDoc}
  *
   * @param event db event item
   */
  @Override
  protected void fillTab(Item event) {
    setEventTree(event);
  }

  /**
   * {@inheritDoc}
   * <p/>
   * Clearing a tab resets the current event tree. Once the tree has been
   * reset, a default root node is added to the tree to make sure an event
   * always starts with a root node.
   */
  @Override
  protected void clearTab() {
    nodeFactory.reset();
    nodes.clear();
    nodeMode = NODE_MODES.SELECT_NODE;
    settingsView.setViewContent(null);

    nodes.add(nodeFactory.createNode(NODE_TYPE.ROOT, CANVAS_WIDTH / 2, 50));

    repaint();
  }

  /**
   * Saves the contents of the current event tree.
   *
   * @param title the event’s title
   * @param description the event’s description
   * @param eventId the event’s schedule id
   * @param eventTypeId the event’s type id
   * @param estimatedTime the event’s estimated time
   * @return the event’s row id
   */
  public Object saveEventTree(Object eventId, String title, String description, Object eventTypeId, Integer estimatedTime) {
    PaattiResearchDBService db = application.getDBConnection();
    SQLContainer eventContainer;
    TaskContainer taskContainer;
    ChoiceContainer choiceContainer;

    HashMap<TreeNode, Object> taskRowIDs = new HashMap<TreeNode, Object>();
    ArrayList<Object> choiceRowIDs = new ArrayList<Object>();

    try {
```

```java
      eventContainer = db.getSQLContainerFromDBTable(PaattiTableNames.EVENT, PaattiColumnNames.EVENT_eventID, false);
      eventContainer.addListener(new QueryDelegate.RowIdChangeListener() {
        public void rowIdChange(RowIdChangeEvent event) {
          eventRowId = event.getNewRowId().getId()[0];
          PaattiResearchApplication.logger.log(Level.FINER, "Updating eventRowId to {0}", eventRowId);
        }
      });
      saveEvent(eventContainer, eventId, title, description, eventTypeId, estimatedTime);

      taskContainer = db.getTaskContainerWithListener(eventRowId);
      taskContainer.addNewRowIdChangeListener(new NewRowIDListener() {
        public void rowIdChanged(NewRowIdEvent event) {
          taskRowId = event.getNewRowId();
          PaattiResearchApplication.logger.log(Level.FINER, "Updating taskRowId to {0}", taskRowId);
        }
      });
      saveTasks(taskContainer, taskRowIDs);

      choiceContainer = db.getChoiceContainerWithListener(eventRowId);
      choiceContainer.addNewRowIdChangeListener(new NewRowIDListener() {
        public void rowIdChanged(NewRowIdEvent event) {
          choiceRowId = event.getNewRowId();
          PaattiResearchApplication.logger.log(Level.FINER, "Updating choiceRowId to {0}", choiceRowId);
        }
      });
      saveChoices(choiceContainer, taskRowIDs, choiceRowIDs);

      removeUnusedChoices(choiceContainer, choiceRowIDs);
      removeUnusedTasks(taskContainer, taskRowIDs);
    } catch (Exception ex) {
      PaattiResearchApplication.logger.log(Level.SEVERE, "saveEventTree", ex);
    }

    repaint();

    return eventRowId;
  }

  /**
   * Writes event properties to the container given as a parameter, creating
   * new items of no previous matching item was found, otherwise using the old
   * ones for updating the values.
   *
   * @param eventContainer an SQL container with the event information
   * @param eventId the event’s id (null if new event)
   * @param title the event’s title
   * @param description the event’s description
   * @param eventTypeId the event’s type id (matching table EVENTTYPE)
   * @throws Exception if the commit fails
   */
  private void saveEvent(SQLContainer eventContainer, Object eventId, String title, String description, Object eventTypeId, ↙
Integer estimatedTime) throws Exception {
    PaattiResearchApplication.logger.log(Level.FINER, "Saving event...");

    Item eventItem = getEventItemFromContainer(eventId, eventContainer);

    eventItem.getItemProperty(PaattiColumnNames.EVENT_title).setValue(title);
    eventItem.getItemProperty(PaattiColumnNames.EVENT_description).setValue(description);
    eventItem.getItemProperty(PaattiColumnNames.EVENT_estimatedTime).setValue(estimatedTime);
    eventItem.getItemProperty(PaattiColumnNames.EVENT_status).setValue(-1);
    eventItem.getItemProperty(PaattiColumnNames.EVENT_SCHEDULE_scheduleID).setValue(-1);
    eventItem.getItemProperty(PaattiColumnNames.EVENT_EVENTTYPE_eventTypeID).setValue(eventTypeId);
    eventItem.getItemProperty(PaattiColumnNames.EVENT_EVENTTIME_eventTimeID).setValue(-1);
    eventItem.getItemProperty(PaattiColumnNames.EVENT_origEventID).setValue(-1);
    eventItem.getItemProperty(PaattiColumnNames.EVENT_rowStatus).setValue(PaattiQueryDelegate.ROWSTATUS_ACTIVE);

    eventRowId = eventItem.getItemProperty(PaattiColumnNames.EVENT_eventID).getValue();
    eventContainer.commit();

    PaattiResearchApplication.logger.log(Level.FINER, "Saved event.");
  }

  /**
   * Writes task properties to the container given as a parameter, creating
   * new items if no previous matching item was found, otherwise using the old
   * ones for updating the values.
   *
   * @param taskContainer an SQL container with the task information
   * @param taskRowIDs a hashmap containing the event’s tasks
   * @throws Exception if the commit fails
   */
  private void saveTasks(TaskContainer taskContainer, HashMap<TreeNode, Object> taskRowIDs) throws Exception {
    PaattiResearchApplication.logger.log(Level.FINER, "Saving tasks...");

    for (TreeNode node : nodes) {
      Item taskItem = getTaskItemFromContainer(node.getRealId(), taskContainer);

      taskItem.getItemProperty(PaattiColumnNames.TASK_description).setValue(node.getDescription());
      taskItem.getItemProperty(PaattiColumnNames.TASK_content).setValue(node.getContent());
      taskItem.getItemProperty(PaattiColumnNames.TASK_url).setValue(null);
      taskItem.getItemProperty(PaattiColumnNames.TASK_sequence).setValue(node.getId());
      taskItem.getItemProperty(PaattiColumnNames.TASK_EVENT_eventID).setValue(eventRowId);
      taskItem.getItemProperty(PaattiColumnNames.TASK_TASKTYPE_taskTypeID).setValue(node.getType());
      taskItem.getItemProperty(PaattiColumnNames.TASK_posX).setValue(node.getX());
      taskItem.getItemProperty(PaattiColumnNames.TASK_posY).setValue(node.getY());
      taskItem.getItemProperty(PaattiColumnNames.TASK_rowStatus).setValue(PaattiQueryDelegate.ROWSTATUS_ACTIVE);

      taskRowId = taskItem.getItemProperty(PaattiColumnNames.TASK_taskID).getValue();
      taskContainer.commit();
      taskRowIDs.put(node, String.valueOf(taskRowId));
      PaattiResearchApplication.logger.log(Level.FINER, "Putting (node, taskRowId) ({0}, {1}) to hashmap",
          new Object[] { node, taskRowId});
    }
    PaattiResearchApplication.logger.log(Level.FINER, "Saved tasks.");
  }

  /**
   * Writes choice properties to the container given as a parameter, creating
   * new items if no previous matching item was found, otherwise using the old
   * ones for updating the values.
   *
   * @param choiceContainer an SQL container with the choice information
   * @param taskRowIDs a hashmap containing the event’s tasks
   * @param choiceRowIDs an arraylist containing the event’s choices
   * @throws Exception if the commit fails
   */
  private void saveChoices(ChoiceContainer choiceContainer, HashMap<TreeNode, Object> taskRowIDs, ArrayList<Object> choiceRowIDs) 
throws Exception {
    PaattiResearchApplication.logger.log(Level.FINER, "Saving choices...");

    Item choiceItem;

    for (TreeNode node : nodes) {
      if (node instanceof MultiPathNode) {
        MultiPathNode node_ = (MultiPathNode) node;
        LinkedList<NodePath> nodePaths = node_.getNodePaths();

        int sequence = 1;

        for (NodePath path : nodePaths) {
          choiceItem = getChoiceItemFromContainer(node.getRealId(), sequence, choiceContainer);
          choiceItem.getItemProperty(PaattiColumnNames.CHOICE_TASK_contains_taskID).setValue(taskRowIDs.get(node));
          choiceItem.getItemProperty(PaattiColumnNames.CHOICE_TASK_leadsto_taskID).setValue(
              taskRowIDs.get(path.getDestinationNode()));
          choiceItem.getItemProperty(PaattiColumnNames.CHOICE_value).setValue(path.getValue());
          choiceItem.getItemProperty(PaattiColumnNames.CHOICE_description).setValue(path.getDescription());
          choiceItem.getItemProperty(PaattiColumnNames.CHOICE_sequence).setValue(sequence++);
          choiceItem.getItemProperty(PaattiColumnNames.CHOICE_rowStatus).setValue(PaattiQueryDelegate.ROWSTATUS_ACTIVE);
          choiceRowId = choiceItem.getItemProperty(PaattiColumnNames.CHOICE_choiceID).getValue();
          choiceContainer.commit();
          choiceRowIDs.add(String.valueOf(choiceRowId));
          PaattiResearchApplication.logger.log(Level.FINER, "Putting multi (node, choiceId) ({0}, {1}) to list",
              new Object[] { node.getRealId(), choiceRowId});
        }
      } else {
        choiceItem = getChoiceItemFromContainer(node.getRealId(), "1", choiceContainer);
        choiceItem.getItemProperty(PaattiColumnNames.CHOICE_TASK_contains_taskID).setValue(taskRowIDs.get(node));
        if (node.getChildren().size() > 0) {
          choiceItem.getItemProperty(PaattiColumnNames.CHOICE_TASK_leadsto_taskID).setValue(
              taskRowIDs.get(node.getChildren().getFirst()));
        } else {
          choiceItem.getItemProperty(PaattiColumnNames.CHOICE_TASK_leadsto_taskID).setValue(-1);
        }
        choiceItem.getItemProperty(PaattiColumnNames.CHOICE_value).setValue(0);
        choiceItem.getItemProperty(PaattiColumnNames.CHOICE_sequence).setValue(1);
        choiceItem.getItemProperty(PaattiColumnNames.CHOICE_rowStatus).setValue(PaattiQueryDelegate.ROWSTATUS_ACTIVE);

        choiceRowId = choiceItem.getItemProperty(PaattiColumnNames.CHOICE_choiceID).getValue();
        choiceContainer.commit();
        choiceRowIDs.add(String.valueOf(choiceRowId));
        PaattiResearchApplication.logger.log(Level.FINER, "Putting (node, choiceId) ({0}, {1}) to list",
            new Object[] { node.getRealId(), choiceRowId});
      }
    }
    PaattiResearchApplication.logger.log(Level.FINER, "Saved choices.");
  }

  /**
   * Removes those tasks from the database that aren’t part of the currently
   * modified event anymore.
   *
   * @param taskContainer an SQL container with the task information
   * @param taskRowIDs a hashmap containing the event’s tasks
   * @throws Exception if commit fails
   */
  private void removeUnusedTasks(TaskContainer taskContainer, HashMap<TreeNode, Object> taskRowIDs) throws Exception {
    PaattiResearchApplication.logger.log(Level.FINER, "Removing unused tasks... taskRowIDs values: {0}",
        Arrays.toString(taskRowIDs.values().toArray()));

    Object taskId = taskContainer.firstItemId();

    while (taskId != null) {
      Item taskItem = taskContainer.getItem(taskId);
```

```java
      PaattiResearchApplication.logger.log(Level.FINER, "Id {0} in values?",
          taskItem.getItemProperty(PaattiColumnNames.TASK_taskID).getValue());
      if (!taskRowIDs.containsValue(taskItem.getItemProperty(PaattiColumnNames.TASK_taskID).toString())) {
        PaattiResearchApplication.logger.log(Level.FINER, "No --> setting taskItem {0} to DELETED",
            taskItem.getItemProperty(PaattiColumnNames.TASK_taskID));
        taskItem.getItemProperty(PaattiColumnNames.TASK_rowStatus).setValue(PaattiQueryDelegate.ROWSTATUS_DELETED);
      } else {
        PaattiResearchApplication.logger.log(Level.FINER, "Yes, all ok");
      }
      taskId = taskContainer.nextItemId(taskId);
    }
    taskContainer.commit();

    PaattiResearchApplication.logger.log(Level.FINER, "Removed unused tasks.");
  }

  /**
   * Removes those choices from the database that aren’t part of the currently
   * modified event anymore.
   *
   * @param choiceContainer the SQL container with the choice information.
   * @param choiceRowIDs a hashmap containing the event’s choices.
   * @throws Exception if commit fails.
   */
  private void removeUnusedChoices(SQLContainer choiceContainer, ArrayList<Object> choiceRowIDs) throws Exception {
    PaattiResearchApplication.logger.log(Level.FINER, "Removing unused choices... choiceKeys values: {0}",
        Arrays.toString(choiceRowIDs.toArray()));

    Object choiceId = choiceContainer.firstItemId();

    while (choiceId != null) {
      Item choiceItem = choiceContainer.getItem(choiceId);

      PaattiResearchApplication.logger.log(Level.FINER, "Id {0} in values?",
          choiceItem.getItemProperty(PaattiColumnNames.CHOICE_choiceID).getValue());
      if (!choiceRowIDs.contains(choiceItem.getItemProperty(PaattiColumnNames.CHOICE_choiceID).toString())) {
        PaattiResearchApplication.logger.log(Level.FINER, "No --> setting choiceItem {0} to DELETED",
            choiceItem.getItemProperty(PaattiColumnNames.CHOICE_choiceID));
        choiceItem.getItemProperty(PaattiColumnNames.CHOICE_rowStatus).setValue(PaattiQueryDelegate.ROWSTATUS_DELETED);
      } else {
        PaattiResearchApplication.logger.log(Level.FINER, "Yes, all ok");
      }
      choiceId = choiceContainer.nextItemId(choiceId);
    }
    choiceContainer.commit();
    PaattiResearchApplication.logger.log(Level.FINER, "Removed unused choices");
  }

  /**
   * Returns the event item that matches the given id. If the container does
   * not have an item with the given id, a new SQLContainer item is created
   * and returned instead.
   *
   * @param eventId the id of the event item to look for.
   * @param eventContainer the SQL container with the event information.
   * @return the old item if found; otherwise a new SQLContainer item.
   */
  protected static Item getEventItemFromContainer(Object eventId, SQLContainer eventContainer) {
    PaattiResearchApplication.logger.log(Level.FINER, "eventId: {0}", eventId);

    // Creating a new event, so return a blank item
    if (eventId == null) {
      PaattiResearchApplication.logger.log(Level.FINER, "Creating a new event item");
      return eventContainer.getItem(eventContainer.addItem());
    }

    Object containerEventId = eventContainer.firstItemId();

    while (containerEventId != null) {
      Item eventItem = eventContainer.getItem(containerEventId);

      if (eventItem.getItemProperty(PaattiColumnNames.EVENT_eventID).toString().equals(eventId.toString())) {
        if (eventItem.getItemProperty(PaattiColumnNames.EVENT_SCHEDULE_scheduleID).toString().equals("-1")) {
          PaattiResearchApplication.logger.log(Level.FINER, "Not scheduled, returning an old event item");
          return eventItem;
        } else {
          PaattiResearchApplication.logger.log(Level.FINER, "Already scheduled, returning a new event item");
          return eventContainer.getItem(eventContainer.addItem());
        }
      }
      containerEventId = eventContainer.nextItemId(containerEventId);
    }
    PaattiResearchApplication.logger.log(Level.FINER, "Creating a new event item");
    return eventContainer.getItem(eventContainer.addItem());
  }

  /**
   * Returns the task item that matches the given id. If the container does
   * not have an item with the given id, a new SQLContainer item is created
   * and returned instead.
   *
   * @param taskId the id of the task item to look for.
   * @param taskContainer the SQL container with the task information.
   * @return the old item if found; otherwise a new SQLContainer item.
   */
  protected static Item getTaskItemFromContainer(Object taskId, SQLContainer taskContainer) {
    PaattiResearchApplication.logger.log(Level.FINER, "taskId: {0}, taskContainer size: {1}",
        new Object[] { taskId, taskContainer.size()});

    Object containerTaskId = taskContainer.firstItemId();

    while (containerTaskId != null) {
      Item containerTaskItem = taskContainer.getItem(containerTaskId);

      if (containerTaskItem.getItemProperty(PaattiColumnNames.TASK_taskID).toString().equals(taskId.toString())) {
        PaattiResearchApplication.logger.log(Level.FINER, "Returning an old task item");
        return containerTaskItem;
      }
      containerTaskId = taskContainer.nextItemId(containerTaskId);
    }
    PaattiResearchApplication.logger.log(Level.FINER, "Creating a new task item");
    return taskContainer.getItem(taskContainer.addItem());
  }

  /**
   * Returns the choice item that matches given parameters containsTaskId and
   * sequence.
   *
   * @param containsTaskId the id of the choice item to look for.
   * @param sequence the sequence if the choice item to look for.
   * @param choiceContainer an SQL container with the choice information.
   * @return old item if found; otherwise a new item.
   */
  protected static Item getChoiceItemFromContainer(Object containsTaskId, Object sequence, SQLContainer choiceContainer) {
    PaattiResearchApplication.logger.log(Level.FINER, "containsTaskId: {0}, choiceContainer size: {1}",
        new Object[] { containsTaskId, choiceContainer.size()});

    Object containerChoiceItem = choiceContainer.firstItemId();

    while (containerChoiceItem != null) {
      Item choiceItem = choiceContainer.getItem(containerChoiceItem);

      if (choiceItem.getItemProperty(PaattiColumnNames.CHOICE_TASK_contains_taskID).toString().equals(
          containsTaskId.toString())
              && choiceItem.getItemProperty(PaattiColumnNames.CHOICE_sequence).toString().equals(sequence.toString())) {
        PaattiResearchApplication.logger.log(Level.FINER, "Returning an old choice item");
        return choiceItem;
      }
      containerChoiceItem = choiceContainer.nextItemId(containerChoiceItem);
    }
    PaattiResearchApplication.logger.log(Level.FINER, "Creating a new choice item");
    return choiceContainer.getItem(choiceContainer.addItem());
  }

  /**
   * Initializes the event tree with the values gotten from the database by
   * using the event that was given as a parameter.
   *
   * @param event an SQL event item
   */
  private void setEventTree(Item event) {
    clearTab();

    PaattiResearchDBService db = view.getDBConnection();
    SQLContainer taskContainer = db.getTaskContainer(event.getItemProperty(PaattiColumnNames.EVENT_eventID));
    SQLContainer choiceContainer = db.getChoiceContainer(event.getItemProperty(PaattiColumnNames.EVENT_eventID));

    PaattiResearchApplication.logger.log(Level.FINER, "Setting event, event item: {0}", event);
    try {
      setTasks(taskContainer);
      setChoices(choiceContainer);
    } catch (Exception ex) {
      PaattiResearchApplication.logger.log(Level.SEVERE, "setEventTree", ex);
    }
    PaattiResearchApplication.logger.log(Level.FINER, "Event set");
    repaint();
  }

  /**
   * Adds tasks to the event tree by looping through the SQL container, setting
   * the tasks’ values to the ones contained by the database items.
   *
   * @param taskContainer an SQL container with the task information
   * @throws Exception if container size is zero
   */
  private void setTasks(SQLContainer taskContainer) throws Exception {
    if (taskContainer.size() == 0) {
      throw new Exception("Trying to load a tree with taskContainer.size() == 0");
    }

    TreeNode task;
    NODE_TYPE nodeType;
    // The first task is always the root node, so we'll just add the first
    // node's information to that one.
    setNodeProperties(nodes.get(0), taskContainer.getItem(taskContainer.firstItemId()));
    PaattiResearchApplication.logger.log(Level.FINER, "taskItem: {0}",
        taskContainer.getItem(taskContainer.firstItemId()));

    for (int i = 1; i < taskContainer.size(); i++) {
      Item taskItem = taskContainer.getItem(taskContainer.getIdByIndex(i));

      PaattiResearchApplication.logger.log(Level.FINER, "taskItem: {0}", taskItem);

      switch ((Integer) taskItem.getItemProperty(PaattiColumnNames.TASK_TASKTYPE_taskTypeID).getValue()) {
      case 2:
        nodeType = NODE_TYPE.RADIO;
        break;

      default:
        nodeType = NODE_TYPE.DEFAULT;
        break;
      }
      task = nodeFactory.createNode(nodeType, 0, 0);
      setNodeProperties(task, taskItem);

      nodes.add(task);
    }
  }

  /**
   * Adds choices to tasks by looping through the SQL container, using the
   * nodes' real ids to connect the parent and the child.
   *
   * @param choiceContainer an SQL container with the choice information
   * @throws Exception if container size is zero
   */
  private void setChoices(SQLContainer choiceContainer) throws Exception {
    if (choiceContainer.size() == 0) {
      throw new Exception("Trying to load a tree with choiceContainer.size() == 0");
    }

    for (int i = 0; i < choiceContainer.size(); i++) {
      Item choiceItem = choiceContainer.getItem(choiceContainer.getIdByIndex(i));

      TreeNode parentTask = getNodeByRealId(
          choiceItem.getItemProperty(PaattiColumnNames.CHOICE_TASK_contains_taskID).getValue());
      TreeNode childTask = getNodeByRealId(
          choiceItem.getItemProperty(PaattiColumnNames.CHOICE_TASK_leadsto_taskID).getValue());

      parentTask.addChild(childTask);

      if (parentTask instanceof MultiPathNode) {
        MultiPathNode multiPathTask = (MultiPathNode) parentTask;

        multiPathTask.createNodePath(choiceItem.getItemProperty(PaattiColumnNames.CHOICE_description).toString(),
            (Integer) choiceItem.getItemProperty(PaattiColumnNames.CHOICE_value).getValue(), childTask);
      }
    }
  }

  /**
   * A helper method for setting task item properties during tree loading.
   *
   * @param task a TreeNode to be modified
   * @param taskItem an Item that contains the values
   */
  private void setNodeProperties(TreeNode task, Item taskItem) {
    task.setRealId((Integer) taskItem.getItemProperty(PaattiColumnNames.TASK_taskID).getValue());
    task.setDescription(taskItem.getItemProperty(PaattiColumnNames.TASK_description).toString());
    task.setContent(taskItem.getItemProperty(PaattiColumnNames.TASK_content).toString());
    task.setX((Integer) taskItem.getItemProperty(PaattiColumnNames.TASK_posX).getValue());
    task.setY((Integer) taskItem.getItemProperty(PaattiColumnNames.TASK_posY).getValue());
  }

  /**
   * Returns the node with the matching real id (db row id).
   *
   * @param value the id of the node to look for.
   * @return the matching node if found; otherwise null.
   */
  protected TreeNode getNodeByRealId(Object value) {
    if (value == null) {
      return null;
    }

    int id = (Integer) value;

    for (int i = 0; i < nodes.size(); i++) {
      TreeNode node = nodes.get(i);

      if (node.getRealId() == id) {
        return node;
      }
```

```java
    }
    return null;
  }

  /**
   * Checks the "tree" for incorrect values and malformed paths. A warning
   * message is set if
   * <p>
   * 1. the tree is empty (containing just the root node),<br/>
   * 2. a MultiPathNode has no NodePaths added to it,<br/>
   * 3. one of the NodePath values (desc, value, destination) is invalid, or<br/>
   * 4. a node has no parent (excluding the root node).
   *
   * @return True if any defects are found, otherwise false.
   */
  protected boolean isUnfinished() {
    if (nodes.size() < 2) {
      saveWarningMessage = ElementNames.EVENT_VIEW_TOOL_WARNING_EMPTY_TREE;
      return true;
    }

    for (TreeNode node : nodes) {
      if (node instanceof MultiPathNode) {
        MultiPathNode multiNode = (MultiPathNode) node;
        LinkedList<NodePath> paths = multiNode.getNodePaths();

        if (paths.size() == 0) {
          saveWarningMessage = ElementNames.EVENT_VIEW_TOOL_WARNING_INVALID_PATH_COUNT;
          return true;
        }

        for (NodePath path : multiNode.getNodePaths()) {
          if (path.getDescription().length() == 0 || path.getDestinationNode() == null || path.getValue() < 0) {
            saveWarningMessage = ElementNames.EVENT_VIEW_TOOL_WARNING_INVALID_PATH_VALUES;
            return true;
          }
        }
      }

      if (node.getParents().size() == 0 && !(node instanceof RootNode)) {
        saveWarningMessage = ElementNames.EVENT_VIEW_TOOL_WARNING_NO_PARENT;
        return true;
      }
    }
    return false;
  }
  /**
   * Returns the warning message set when checking the tree for malformed
   * paths in the method isUnfinished().
   *
   * @return The warning (error) message.
   */
  protected String getWarningMessage() {
    return saveWarningMessage;
  }

  /**
   * Requests for the canvas to repaint itself and for a new image file to be
   * created.
   */
  public void repaint() {
    canvas.requestRepaint();
    connectorResource.setFilename(connectors.makeImageFilename());
  }

  /**
   * TreeViewListener handles the mouse events that are registered on the
   * canvas.
   */
  private class TreeViewListener implements MouseEvents.ClickListener {
    // Location of the previous mouse click (one click delay functionality).
    private int mouseClickX, mouseClickY;
    private TreeNode currentNode, previousNode;

    /**
     * TreeViewListener constructor.
     */
    private TreeViewListener() {}

    /**
     * Handles the mouse logic, telling the tool to create new nodes/move
     * old nodes/etc where the click occurred.
     *
     * @param event click event
     */
    public void click(ClickEvent event) {
      int x = event.getRelativeX();
      int y = event.getRelativeY();

      // Select the node that was clicked on (if any).
      currentNode = null;
      for (TreeNode node : nodes) {
        if (node.contains(x, y)) {
          currentNode = node;
          break;
        }
      }

      // Display the context menu if the registered button was RMB.
      if (event.getButton() == ClickEvent.BUTTON_RIGHT) {
        mouseClickX = x;
        mouseClickY = y;
        contextMenu.show(x, y, currentNode);

        if (currentNode != null) {
          settingsView.setViewContent(currentNode);
        }
        previousNode = currentNode;

      } else {
        // Check clicks on context menu if visible, ignoring everything else.
        if (contextMenu.isVisible()) {
          NODE_TYPE nodeType = null;

          MenuItem menuItem = contextMenu.getMenu(x, y);

          if (menuItem != null) {
            nodeMode = menuItem.getNodeMode();
            nodeType = menuItem.getNodeType();
          }

          // Actions that are supposed to happen immediately after
          // clicking a menu item:
          switch (nodeMode) {
          case ADD_NODE:
            TreeNode node = nodeFactory.createNode(nodeType, mouseClickX, mouseClickY);

            nodes.add(node);
            settingsView.setViewContent(node);
            previousNode = node;
            nodeMode = NODE_MODES.SELECT_NODE;
            break;

          case REMOVE_NODE:
            if (previousNode instanceof RootNode) {
              break;
            }
```

```java
            previousNode.detach();
            nodes.remove(previousNode);
            settingsView.setViewContent(null);
            nodeMode = NODE_MODES.SELECT_NODE;
            break;
          }
        } else {
          // Actions that are supposed to happen with one click delay
          // after selecting a menu item, e.g. selecting a node to add
          // as a child, or selecting a new location for a node:
          switch (nodeMode) {
          case MOVE_NODE:
            previousNode.moveTo(x, y);
            break;

          case ADD_CHILD:
            previousNode.addChild(currentNode);
            settingsView.setViewContent(previousNode);
            break;

          case SELECT_NODE:
            previousNode = currentNode;
            settingsView.setViewContent(currentNode);
            break;
          }
          // Reset select mode back to selection mode.
          nodeMode = NODE_MODES.SELECT_NODE;
        }
      }
      repaint();
    }
  }


  /**
   * ConnectorSource provides the image graphics context for the event tool to
   * paint on. The finished image is named using a timestamp as an identifier
   * and returned as a bytestream to the one requesting the image.
   */
  private class ConnectorSource implements StreamResource.StreamSource {
    private ByteArrayOutputStream imagebuffer = null;

    /**
     * Creates an image for the event tool to paint the tree on, and returns
     * the finished image as a bytestream.
     *
     * @return image as a bytestream
     */
    public InputStream getStream() {
      BufferedImage image = new BufferedImage(CANVAS_WIDTH, CANVAS_HEIGHT, BufferedImage.TYPE_INT_RGB);

      Graphics g = image.getGraphics();

      g.setColor(Color.WHITE);
      g.fillRect(0, 0, CANVAS_WIDTH, CANVAS_HEIGHT);
      g.setColor(Color.BLACK);
      g.drawRect(0, 0, CANVAS_WIDTH - 1, CANVAS_HEIGHT - 1);

      for (TreeNode node : nodes) {
        node.paint(g);
      }

      contextMenu.paint(g);

      try {
        imagebuffer = new ByteArrayOutputStream();
        ImageIO.write(image, "png", imagebuffer);
        return new ByteArrayInputStream(imagebuffer.toByteArray());
      } catch (IOException e) {
        PaattiResearchApplication.logger.log(Level.SEVERE, "getStream", e);
        return null;
      }
    }

    /**
     * Creates a name for the image file, using the current date’s timestamp
     * as an identifier.
     *
     * @return image file name
     */
    public String makeImageFilename() {
      SimpleDateFormat df = new SimpleDateFormat("yyyyMMddHHmmssSSS");
      String timestamp = df.format(new Date());

      return "eventtree-" + timestamp + ".png";
    }
  }

  /**
   * {@inheritDoc}
   * <p/>
   * Unused method inherited from ApplicationViewTab.
     *
     * @param event click event.
     */
    public void buttonClick(Button.ClickEvent event) {}
  }
```