```
package database;

public class MongoDB_CONFIG {

    public static String DATABASE_URL = "YOUR_URL_HERE";

}
```

```
package database;

import categories.*;
import categoryrules.DateRule;
import categoryrules.IntegerRule;
import categoryrules.StringRule;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import org.bson.Document;
import principal_resource_attributes.DateAttribute;
import principal_resource_attributes.IntegerAttribute;
import principal_resource_attributes.StringAttribute;

import java.util.*;


public class MongoMain {
    private static final String DB_NAME = "MongoDB";
    private MongoClient mongoClient;
    private MongoDatabase database;

    public MongoMain(MongoClient mongoClient) {
        this.mongoClient = mongoClient;
        database = mongoClient.getDatabase(DB_NAME);
    }

    public void createCollectionIfNotExists(String collectionName) {
        boolean collectionExists = database.listCollectionNames()
                .into(new ArrayList<>())
                .stream()
                .anyMatch(name -> name.equalsIgnoreCase(collectionName));

        if (!collectionExists) {
            database.createCollection(collectionName);
        }
    }

    public boolean databaseEmpty() {
        String[] collectionNames = {"principalCategories", "principals", "actions", "
resources", "undoClass"};
        boolean isEmpty = true;
        for (String s : collectionNames) {
            boolean collectionExists = database.listCollectionNames()
                    .into(new ArrayList<>())
                    .stream()
                    .anyMatch(name -> name.equalsIgnoreCase(s));
            if (collectionExists && database.getCollection(s).countDocuments() > 0) {
                isEmpty = false;
```

```
            }
        }
        return isEmpty;
    }

    public void saveUndoClass(UndoClass undoClass){
        MongoCollection<Document> undoClassCollection = database.getCollection("
undoClass");
        List<Document> docs = new ArrayList<>();
        for(List<Object> currEntry : undoClass.getActionTracker()){
            UndoClass.UNDO_TYPE actionType = (UndoClass.UNDO_TYPE) currEntry.
get(0);
            Document type = new Document("type", actionType.name());

            if (actionType == UndoClass.UNDO_TYPE.UPDATE_PRINCIPAL) {
                Principal oldPrincipal = (Principal) currEntry.get(1);
                Principal newPrincipal = (Principal) currEntry.get(2);
                type.put("oldPrincipal", convertSinglePrincipalToDocument(oldPrincipal));
                type.put("newPrincipal", convertSinglePrincipalToDocument(newPrincipal));

            } else if (actionType == UndoClass.UNDO_TYPE.CREATE_PRINCIPAL) {
                Principal principal = (Principal) currEntry.get(1);
                type.put("principal", convertSinglePrincipalToDocument(principal));

            } else if (actionType == UndoClass.UNDO_TYPE.REMOVE_PRINCIPAL) {
                Principal principal = (Principal) currEntry.get(1);
                type.put("principal", convertSinglePrincipalToDocument(principal));

            } else if (actionType == UndoClass.UNDO_TYPE.ADD_RESOURCE) {
                Resource resource = (Resource) currEntry.get(1);
                type.put("resource", convertSingleResourceToDocument(resource));

            } else if (actionType == UndoClass.UNDO_TYPE.REMOVE_RESOURCE) {
                Resource toAdd = (Resource) currEntry.get(1);
                type.put("resource", convertSingleResourceToDocument(toAdd));
                Map<ResourceAction, List<PrincipalCategory>> assignedPerms =
(Map<ResourceAction, List<PrincipalCategory>>) currEntry.get(2);
                List<Document> docList = new ArrayList<>();
                for(ResourceAction a : assignedPerms.keySet()){
                    List<PrincipalCategory> curr = assignedPerms.get(a);
                    docList.add(convertSingleActionToDocument(a).append("categories",
convertJuniorCategoriesToDocuments(curr)));
                }
                type.put("actionMapping", docList);
            } else if (actionType == UndoClass.UNDO_TYPE.ADD_ACTION) {
                ResourceAction a = (ResourceAction) currEntry.get(1);
                type.put("action", convertSingleActionToDocument(a));

            } else if (actionType == UndoClass.UNDO_TYPE.REMOVE_ACTION) {
```

```
                ResourceAction a = (ResourceAction) currEntry.get(1);
                List<PrincipalCategory> categories = (List<PrincipalCategory>) currEntry.
get(2);
                type.put("action", convertSingleActionToDocument(a));
                type.put("categories", convertJuniorCategoriesToDocuments(categories));

            } else if (actionType == UndoClass.UNDO_TYPE.UPDATE_CATEGORY) {
                PrincipalCategory oldCategory = (PrincipalCategory) currEntry.get(1);
                PrincipalCategory newCategory = (PrincipalCategory) currEntry.get(2);
                type.put("oldCategory",
convertSinglePrincipalCategoryToDocument(oldCategory));
                type.put("newCategory",
convertSinglePrincipalCategoryToDocument(newCategory));

            } else if (actionType == UndoClass.UNDO_TYPE.CREATE_CATEGORY) {
                PrincipalCategory category = (PrincipalCategory) currEntry.get(1);
                type.put("category", convertSinglePrincipalCategoryToDocument(category));

            } else if (actionType == UndoClass.UNDO_TYPE.REMOVE_CATEGORY) {
                PrincipalCategory oldCategory = (PrincipalCategory) currEntry.get(1);
                List<PrincipalCategory> oldSeniorCategories = (List<PrincipalCategory>)
currEntry.get(2);
                type.put("category",
convertSinglePrincipalCategoryToDocument(oldCategory));
                type.put("seniorCategories",
convertJuniorCategoriesToDocuments(oldSeniorCategories));

            } else if (actionType == UndoClass.UNDO_TYPE.UPDATE_PERMISSIONS) {
                PrincipalCategory curr = (PrincipalCategory) currEntry.get(1);
                List<ResourceAction> oldActions = (List<ResourceAction>) currEntry.get(2);
                type.put("category", convertSinglePrincipalCategoryToDocument(curr));
                type.put("actions", convertActionsToDocuments(oldActions));

            } else if (actionType == UndoClass.UNDO_TYPE.UPDATE_HIERARCHY) {
                PrincipalCategory curr = (PrincipalCategory) currEntry.get(1);
                List<PrincipalCategory> oldJrCategories = (List<PrincipalCategory>)
currEntry.get(2);
                type.put("category", convertSinglePrincipalCategoryToDocument(curr));
                type.put("juniorCategories",
convertJuniorCategoriesToDocuments(oldJrCategories));

            }
            docs.add(type);
        }
        if(!docs.isEmpty()) {
            undoClassCollection.insertMany(docs);
        }
    }
```

```
   public UndoClass getUndoClass(){
      MongoCollection<Document> undoClassCollection = database.getCollection("
undoClass");
      UndoClass undoClass = new UndoClass();
      for(Document doc : undoClassCollection.find()){
         UndoClass.UNDO_TYPE actionType = UndoClass.UNDO_TYPE.valueOf(doc.
getString("type"));
         if (actionType == UndoClass.UNDO_TYPE.UPDATE_PRINCIPAL) {
            Principal oldPrincipal = convertDocumentToSinglePrincipal((Document) doc.
get("oldPrincipal"));
            Principal newPrincipal = convertDocumentToSinglePrincipal((Document) doc.
get("newPrincipal"));
            undoClass.addUpdatePrincipal(oldPrincipal, newPrincipal);

         } else if (actionType == UndoClass.UNDO_TYPE.CREATE_PRINCIPAL) {
            Principal principal = convertDocumentToSinglePrincipal((Document) doc.get("
principal"));
            undoClass.addCreatePrincipal(principal);

         } else if (actionType == UndoClass.UNDO_TYPE.REMOVE_PRINCIPAL) {
            Principal principal = convertDocumentToSinglePrincipal((Document) doc.get("
principal"));
            undoClass.addRemovePrincipal(principal);

         } else if (actionType == UndoClass.UNDO_TYPE.ADD_RESOURCE) {
            Document resourceDoc = (Document) doc.get("resource");
            String resourceName =  (String) resourceDoc.get("name");
            undoClass.addAddResource(new Resource(resourceName));

         } else if (actionType == UndoClass.UNDO_TYPE.REMOVE_RESOURCE) {
            Map<ResourceAction, List<PrincipalCategory>> assignedPerms = new
HashMap<>();
            Document resourceDoc = (Document) doc.get("resource");
            Resource resource = new Resource(resourceDoc.getString("name"));
            List<Document> actionMapping = (List<Document>) doc.get("actionMapping"
);
            for(Document d : actionMapping){
               ResourceAction fromDb = new ResourceAction(d.getString("name"),
resource);
               List<PrincipalCategory> tempList = new ArrayList<>();
               List<Document> associatedCategories = (List<Document>) d.get("
categories");
               for(Document categoryDoc : associatedCategories){
                  tempList.add(new PrincipalCategory(categoryDoc.getString("name")));
               }
               assignedPerms.put(fromDb, tempList);
            }
            undoClass.addRemoveResource(resource, assignedPerms);
```

```
        } else if (actionType == UndoClass.UNDO_TYPE.ADD_ACTION) {
            Document actionDoc = (Document) doc.get("action");
            Document resourceDoc = (Document) actionDoc.get("resource");
            ResourceAction a = new ResourceAction(actionDoc.getString("name"), new
Resource(resourceDoc.getString("name")));
            undoClass.addAddAction(a);

        } else if (actionType == UndoClass.UNDO_TYPE.REMOVE_ACTION) {
            Document actionDoc = (Document) doc.get("action");
            Document resourceDoc = (Document) actionDoc.get("resource");
            List<Document> categoryDocs = (List<Document>) doc.get("categories");
            List<PrincipalCategory> associatedCategories = new ArrayList<>();
            for(Document d : categoryDocs){
                associatedCategories.add(new PrincipalCategory(d.getString("name")));
            }
            ResourceAction action = new ResourceAction(actionDoc.getString("name"),
new Resource(resourceDoc.getString("name")));
            undoClass.addRemoveAction(action, associatedCategories);

        } else if (actionType == UndoClass.UNDO_TYPE.UPDATE_CATEGORY) {
            Document newCategoryDoc = (Document) doc.get("newCategory");
            Document oldCategoryDoc = (Document) doc.get("oldCategory");
            PrincipalCategory newCategory =
convertDocumentToSinglePrincipalCategory(newCategoryDoc);
            PrincipalCategory oldCategory =
convertDocumentToSinglePrincipalCategory(oldCategoryDoc);
            undoClass.addUpdateCategory(oldCategory, newCategory);

        } else if (actionType == UndoClass.UNDO_TYPE.CREATE_CATEGORY) {
            Document categoryDoc = (Document) doc.get("category");
            PrincipalCategory principalCategory =
convertDocumentToSinglePrincipalCategory(categoryDoc);
            undoClass.addCreateCategory(principalCategory);

        } else if (actionType == UndoClass.UNDO_TYPE.REMOVE_CATEGORY) {
            Document categoryDoc = (Document) doc.get("category");
            List<Document> seniorCategoryDocs = (List<Document>) doc.get("
seniorCategories");
            PrincipalCategory notFixedCategory =
convertDocumentToSinglePrincipalCategory(categoryDoc);
            List<PrincipalCategory> seniorCategoryList = new ArrayList<>();
            for(Document d : seniorCategoryDocs){
                seniorCategoryList.add(new PrincipalCategory(d.getString("name")));
            }
            undoClass.addRemoveCategory(notFixedCategory, seniorCategoryList);

        } else if (actionType == UndoClass.UNDO_TYPE.UPDATE_PERMISSIONS) {
            Document categoryDoc = (Document) doc.get("category");
            List<ResourceAction> actionList =
```

```
convertDocumentsToActions((List<Document>) doc.get("actions"));
                PrincipalCategory category =
convertDocumentToSinglePrincipalCategory(categoryDoc);
                undoClass.addUpdatePermissions(category, actionList);

            } else if (actionType == UndoClass.UNDO_TYPE.UPDATE_HIERARCHY) {
                Document categoryDoc = (Document) doc.get("category");
                PrincipalCategory category =
convertDocumentToSinglePrincipalCategory(categoryDoc);
                List<Document> jrCategoryDocs = (List<Document>) doc.get("
juniorCategories");
                List<PrincipalCategory> oldJrCategories = new ArrayList<>();
                for(Document d : jrCategoryDocs){
                    oldJrCategories.add(new PrincipalCategory(d.getString("name")));
                }
                undoClass.addUpdateHierarchy(category, oldJrCategories);

            }
        }
        return undoClass;
    }


    public void savePrincipalCategory(PrincipalCategory principalCategory) {
        MongoCollection<Document> principalCategoriesCollection = database.
getCollection("principalCategories");
        Document principalCategoryDoc = new Document("name", principalCategory.
getName());
        principalCategoryDoc.put("juniorCategories",
convertJuniorCategoriesToDocuments(principalCategory.getJuniorCategories()));
        principalCategoryDoc.put("principals",
convertPrincipalsToDocuments(principalCategory.getPrincipals()));
        principalCategoryDoc.put("actions",
convertActionsToDocuments(principalCategory.getActions()));
        principalCategoryDoc.put("stringRules",
convertStringRulesToDocuments(principalCategory.getStringRules()));
        principalCategoryDoc.put("integerRules",
convertIntegerRulesToDocuments(principalCategory.getIntegerRules()));
        principalCategoryDoc.put("dateRules",
convertDateRulesToDocuments(principalCategory.getDateRules()));

        principalCategoriesCollection.insertOne(principalCategoryDoc);
    }

    public List<Document>
convertJuniorCategoriesToDocuments(List<PrincipalCategory> juniorCategories) {
        List<Document> docs = new ArrayList<>();
        for (PrincipalCategory juniorCategory : juniorCategories) {
            Document doc = new Document("name", juniorCategory.getName());
```

```
            docs.add(doc);
        }
        return docs;
    }

    public List<Document> convertPrincipalsToDocuments(List<Principal> principals) {
        List<Document> docs = new ArrayList<>();
        for (Principal principal : principals) {
            Document doc = new Document("name", principal.getName());
            doc.put("stringAttributes", convertAttributesToDocuments(principal.
getStringAttributeList(), "StringAttribute"));
            doc.put("integerAttributes", convertAttributesToDocuments(principal.
getIntegerAttributeList(), "IntegerAttribute"));
            doc.put("dateAttributes", convertAttributesToDocuments(principal.
getDateAttributeList(), "DateAttribute"));
            docs.add(doc);
        }
        return docs;
    }

    public List<Document> convertActionsToDocuments(List<ResourceAction> actions) {
        List<Document> docs = new ArrayList<>();
        for (ResourceAction action : actions) {
            Document doc = new Document("name", action.getName());
            doc.put("resource", new Document("name", action.getResource().getName()));
            docs.add(doc);
        }
        return docs;
    }

    public List<Document> convertStringRulesToDocuments(List<StringRule>
stringRules) {
        List<Document> docs = new ArrayList<>();
        for (StringRule rule : stringRules) {
            Document doc = new Document("attribute", rule.getAttribute().getName());
            doc.put("attributeValue", rule.getAttribute().getValue());
            doc.put("requirements", rule.getRequirements());
            docs.add(doc);
        }
        return docs;
    }

    public List<Document> convertIntegerRulesToDocuments(List<IntegerRule>
integerRules) {
        List<Document> docs = new ArrayList<>();
        for (IntegerRule rule : integerRules) {
            Document doc = new Document("attribute", rule.getAttribute().getName());
            doc.put("attributeValue", rule.getAttribute().getValue());
            doc.put("lowerBound", rule.getLowerBound());
```

```
            doc.put("upperBound", rule.getUpperBound());
            docs.add(doc);
        }
        return docs;
    }

    public List<Document> convertDateRulesToDocuments(List<DateRule> dateRules) {
        List<Document> docs = new ArrayList<>();
        for (DateRule rule : dateRules) {
            Document doc = new Document("attribute", rule.getAttribute().getName());
            doc.put("attributeValue", rule.getAttribute().getValue());
            doc.put("lowerBound", rule.getLowerBound());
            doc.put("upperBound", rule.getUpperBound());
            docs.add(doc);
        }
        return docs;
    }



    public <T> List<Document> convertAttributesToDocuments(List<T> attributes, String
attributeType) {
        List<Document> docs = new ArrayList<>();
        for (T attribute : attributes) {
            Document doc = new Document();
            if (attributeType.equals("StringAttribute")) {
                StringAttribute stringAttribute = (StringAttribute) attribute;
                doc.put("name", stringAttribute.getName());
                doc.put("value", stringAttribute.getValue());
            } else if (attributeType.equals("IntegerAttribute")) {
                IntegerAttribute integerAttribute = (IntegerAttribute) attribute;
                doc.put("name", integerAttribute.getName());
                doc.put("value", integerAttribute.getValue());
            } else if (attributeType.equals("DateAttribute")) {
                DateAttribute dateAttribute = (DateAttribute) attribute;
                doc.put("name", dateAttribute.getName());
                doc.put("value", dateAttribute.getValue());
            }
            docs.add(doc);
        }
        return docs;
    }

    public PrincipalCategory convertDocumentToSinglePrincipalCategory(Document doc)
 {
        String name = doc.getString("name");
        List<PrincipalCategory> juniorCategories =
convertDocumentsToJuniorCategories((List<Document>) doc.get("juniorCategories"));
        List<Principal> principals = convertDocumentsToPrincipals((List<Document>) doc.
```

```
get("principals"));
        List<ResourceAction> actions = convertDocumentsToActions((List<Document>)
doc.get("actions"));
        List<StringRule> stringRules = convertDocumentsToStringRules((List<Document>)
 doc.get("stringRules"));
        List<IntegerRule> integerRules =
convertDocumentsToIntegerRules((List<Document>) doc.get("integerRules"));
        List<DateRule> dateRules = convertDocumentsToDateRules((List<Document>)
doc.get("dateRules"));

        PrincipalCategory principalCategory = new PrincipalCategory(name);
        principalCategory.setJuniorCategories(juniorCategories);
        principalCategory.setPrincipals(principals);
        principalCategory.setActions(actions);
        principalCategory.setStringRules(stringRules);
        principalCategory.setIntegerRules(integerRules);
        principalCategory.setDateRules(dateRules);

        return principalCategory;
    }

    public ResourceAction convertDocumentToSingleAction(Document doc) {
        String name = doc.getString("name");
        Document resourceDoc = (Document) doc.get("resource");
        String resourceName = resourceDoc.getString("name");
        Resource resource = new Resource(resourceName);
        ResourceAction action = new ResourceAction(name, resource);
        return action;
    }

    public Document convertSinglePrincipalCategoryToDocument(PrincipalCategory
principalCategory) {
        Document principalCategoryDoc = new Document("name", principalCategory.
getName());
        principalCategoryDoc.put("juniorCategories",
convertJuniorCategoriesToDocuments(principalCategory.getJuniorCategories()));
        principalCategoryDoc.put("principals",
convertPrincipalsToDocuments(principalCategory.getPrincipals()));
        principalCategoryDoc.put("actions",
convertActionsToDocuments(principalCategory.getActions()));
        principalCategoryDoc.put("stringRules",
convertStringRulesToDocuments(principalCategory.getStringRules()));
        principalCategoryDoc.put("integerRules",
convertIntegerRulesToDocuments(principalCategory.getIntegerRules()));
        principalCategoryDoc.put("dateRules",
convertDateRulesToDocuments(principalCategory.getDateRules()));

        return principalCategoryDoc;
    }
```

```
    public Document convertSingleActionToDocument(ResourceAction action){
        Document doc = new Document("name", action.getName());
        doc.put("resource", new Document("name", action.getResource().getName()));
        return doc;
    }

    public Document convertSingleResourceToDocument(Resource r){
        return new Document("name", r.getName());
    }

    public Document convertSinglePrincipalToDocument(Principal principal) {
        Document doc = new Document("name", principal.getName());
        doc.put("stringAttributes", convertAttributesToDocuments(principal.
getStringAttributeList(), "StringAttribute"));
        doc.put("integerAttributes", convertAttributesToDocuments(principal.
getIntegerAttributeList(), "IntegerAttribute"));
        doc.put("dateAttributes", convertAttributesToDocuments(principal.
getDateAttributeList(), "DateAttribute"));
        return doc;
    }

    public Principal convertDocumentToSinglePrincipal(Document doc){
        String name = doc.getString("name");
        List<StringAttribute> stringAttributes =
convertDocumentsToStringAttributes((List<Document>) doc.get("stringAttributes"));
        List<IntegerAttribute> integerAttributes =
convertDocumentsToIntegerAttributes((List<Document>) doc.get("integerAttributes"));
        List<DateAttribute> dateAttributes =
convertDocumentsToDateAttributes((List<Document>) doc.get("dateAttributes"));

        Principal principal = new Principal(name);
        principal.setStringAttributeList(stringAttributes);
        principal.setIntegerAttributeList(integerAttributes);
        principal.setDateAttributeList(dateAttributes);
        return principal;
    }

    public void saveAssignCategories(AssignCategories assignCategories, UndoClass
undoClass) {
        database.getCollection("principalCategories").drop();
        database.getCollection("principals").drop();
        database.getCollection("actions").drop();
        database.getCollection("resources").drop();
        database.getCollection("undoClass").drop();

        createCollectionIfNotExists("principalCategories");
        createCollectionIfNotExists("principals");
        createCollectionIfNotExists("actions");
```

```
        createCollectionIfNotExists("resources");
        createCollectionIfNotExists("undoClass");

        saveUndoClass(undoClass);


        for (Principal principal : assignCategories.getPrincipals()) {
            savePrincipal(principal);
        }
        for (PrincipalCategory principalCategory : assignCategories.
getPrincipalCategories()) {
            savePrincipalCategory(principalCategory);
        }
        for (ResourceAction action : assignCategories.getResourceActions()) {
            saveAction(action);
        }
        for (Resource resource : assignCategories.getResources()) {
            saveResource(resource);
        }
    }

    public void savePrincipal(Principal principal) {
        MongoCollection<Document> principalsCollection = database.getCollection("
principals");
        Document principalDoc = new Document("name", principal.getName());
        principalDoc.put("stringAttributes", convertAttributesToDocuments(principal.
getStringAttributeList(), "StringAttribute"));
        principalDoc.put("integerAttributes", convertAttributesToDocuments(principal.
getIntegerAttributeList(), "IntegerAttribute"));
        principalDoc.put("dateAttributes", convertAttributesToDocuments(principal.
getDateAttributeList(), "DateAttribute"));
        principalsCollection.insertOne(principalDoc);
    }

    public void saveAction(ResourceAction action) {
        MongoCollection<Document> actionsCollection = database.getCollection("actions"
);
        Document actionDoc = new Document("name", action.getName());
        actionDoc.put("resource", new Document("name", action.getResource().getName()
));
        actionsCollection.insertOne(actionDoc);
    }

    public void saveResource(Resource resource) {
        MongoCollection<Document> resourcesCollection = database.getCollection("
resources");
        Document resourceDoc = new Document("name", resource.getName());
        resourcesCollection.insertOne(resourceDoc);
    }
```

```
    public AssignCategories getAssignCategories() {
        List<Principal> principals = getPrincipals();
        List<PrincipalCategory> principalCategories = getPrincipalCategories();
        List<ResourceAction> resourceActions = getActions();
        List<Resource> resources = getResources();

        AssignCategories toReturn = new AssignCategories(principals,
principalCategories);
        toReturn.setResourceActions(resourceActions);
        toReturn.setResources(resources);
        return toReturn;
    }

    public List<Principal> getPrincipals() {
        MongoCollection<Document> principalsCollection = database.getCollection("
principals");
        List<Principal> principals = new ArrayList<>();

        for (Document doc : principalsCollection.find()) {
            String name = doc.getString("name");
            List<StringAttribute> stringAttributes =
convertDocumentsToStringAttributes((List<Document>) doc.get("stringAttributes"));
            List<IntegerAttribute> integerAttributes =
convertDocumentsToIntegerAttributes((List<Document>) doc.get("integerAttributes"));
            List<DateAttribute> dateAttributes =
convertDocumentsToDateAttributes((List<Document>) doc.get("dateAttributes"));

            Principal principal = new Principal(name);
            principal.setStringAttributeList(stringAttributes);
            principal.setIntegerAttributeList(integerAttributes);
            principal.setDateAttributeList(dateAttributes);
            principals.add(principal);
        }

        return principals;
    }


    public List<StringAttribute> convertDocumentsToStringAttributes(List<Document>
docs) {
        List<StringAttribute> attributes = new ArrayList<>();
        for (Document doc : docs) {
            String name = doc.getString("name");
            String value = doc.getString("value");
            attributes.add(new StringAttribute(name, value));
        }
        return attributes;
    }
```

```
    public List<PrincipalCategory> getPrincipalCategories() {
        MongoCollection<Document> principalCategoriesCollection = database.
getCollection("principalCategories");
        List<PrincipalCategory> principalCategories = new ArrayList<>();

        for (Document doc : principalCategoriesCollection.find()) {
            String name = doc.getString("name");
            List<PrincipalCategory> juniorCategories =
convertDocumentsToJuniorCategories((List<Document>) doc.get("juniorCategories"));
            List<Principal> principals = convertDocumentsToPrincipals((List<Document>)
doc.get("principals"));
            List<ResourceAction> actions = convertDocumentsToActions((List<Document>)
 doc.get("actions"));
            List<StringRule> stringRules =
convertDocumentsToStringRules((List<Document>) doc.get("stringRules"));
            List<IntegerRule> integerRules =
convertDocumentsToIntegerRules((List<Document>) doc.get("integerRules"));
            List<DateRule> dateRules = convertDocumentsToDateRules((List<Document>)
doc.get("dateRules"));

            PrincipalCategory principalCategory = new PrincipalCategory(name);
            principalCategory.setJuniorCategories(juniorCategories);
            principalCategory.setPrincipals(principals);
            principalCategory.setActions(actions);
            principalCategory.setStringRules(stringRules);
            principalCategory.setIntegerRules(integerRules);
            principalCategory.setDateRules(dateRules);
            principalCategories.add(principalCategory);
        }

        return principalCategories;
    }

    public List<ResourceAction> getActions() {
        MongoCollection<Document> actionsCollection = database.getCollection("actions"
);
        List<ResourceAction> actions = new ArrayList<>();

        for (Document doc : actionsCollection.find()) {
            String name = doc.getString("name");
            Document resourceDoc = (Document) doc.get("resource");
            String resourceName = resourceDoc.getString("name");
            Resource resource = new Resource(resourceName);

            ResourceAction action = new ResourceAction(name, resource);
            actions.add(action);
        }
```

```
        return actions;
    }

    public List<Resource> getResources() {
        MongoCollection<Document> resourcesCollection = database.getCollection("
resources");
        List<Resource> resources = new ArrayList<>();

        for (Document doc : resourcesCollection.find()) {
            String name = doc.getString("name");
            Resource resource = new Resource(name);
            resources.add(resource);
        }

        return resources;
    }

    public List<IntegerAttribute> convertDocumentsToIntegerAttributes(List<Document>
docs) {
        List<IntegerAttribute> attributes = new ArrayList<>();
        for (Document doc : docs) {
            String name = doc.getString("name");
            int value = doc.getInteger("value");
            attributes.add(new IntegerAttribute(name, value));
        }
        return attributes;
    }

    public List<DateAttribute> convertDocumentsToDateAttributes(List<Document>
docs) {
        List<DateAttribute> attributes = new ArrayList<>();
        for (Document doc : docs) {
            String name = doc.getString("name");
            Date value = doc.getDate("value");
            attributes.add(new DateAttribute(name, value));
        }
        return attributes;
    }

    public List<PrincipalCategory>
convertDocumentsToJuniorCategories(List<Document> docs) {
        List<PrincipalCategory> juniorCategories = new ArrayList<>();
        for (Document doc : docs) {
            String name = doc.getString("name");
            juniorCategories.add(new PrincipalCategory(name));
        }
        return juniorCategories;
    }
```

```
    public List<Principal> convertDocumentsToPrincipals(List<Document> docs) {
        List<Principal> principals = new ArrayList<>();
        for (Document doc : docs) {
            String name = doc.getString("name");
            List<StringAttribute> stringAttributes =
convertDocumentsToStringAttributes((List<Document>) doc.get("stringAttributes"));
            List<IntegerAttribute> integerAttributes =
convertDocumentsToIntegerAttributes((List<Document>) doc.get("integerAttributes"));
            List<DateAttribute> dateAttributes =
convertDocumentsToDateAttributes((List<Document>) doc.get("dateAttributes"));

            Principal principal = new Principal(name);
            principal.setStringAttributeList(stringAttributes);
            principal.setIntegerAttributeList(integerAttributes);
            principal.setDateAttributeList(dateAttributes);
            principals.add(principal);
        }
        return principals;
    }

    public List<ResourceAction> convertDocumentsToActions(List<Document> docs) {
        List<ResourceAction> actions = new ArrayList<>();
        for (Document doc : docs) {
            String name = doc.getString("name");
            Document resourceDoc = (Document) doc.get("resource");
            String resourceName = resourceDoc.getString("name");
            Resource resource = new Resource(resourceName);

            ResourceAction action = new ResourceAction(name, resource);
            actions.add(action);
        }
        return actions;
    }

    public List<StringRule> convertDocumentsToStringRules(List<Document> docs) {
        List<StringRule> stringRules = new ArrayList<>();
        for (Document doc : docs) {
            String attributeName = doc.getString("attribute");
            String attributeValue = doc.getString("attributeValue");
            StringAttribute attribute = new StringAttribute(attributeName, attributeValue);
            List<String> requirements = (List<String>) doc.get("requirements");

            StringRule rule = new StringRule(attribute, requirements);
            stringRules.add(rule);
        }
        return stringRules;
    }
```

```
    public List<IntegerRule> convertDocumentsToIntegerRules(List<Document> docs) {
        List<IntegerRule> integerRules = new ArrayList<>();
        for (Document doc : docs) {
            String attributeName = doc.getString("attribute");
            int attributeValue = doc.getInteger("attributeValue");
            IntegerAttribute attribute = new IntegerAttribute(attributeName, attributeValue);
            int lowerBound = doc.getInteger("lowerBound");
            int upperBound = doc.getInteger("upperBound");

            IntegerRule rule = new IntegerRule(attribute, lowerBound, upperBound);
            integerRules.add(rule);
        }
        return integerRules;
    }

    public List<DateRule> convertDocumentsToDateRules(List<Document> docs) {
        List<DateRule> dateRules = new ArrayList<>();
        for (Document doc : docs) {
            String attributeName = doc.getString("attribute");
            Date attributeValue = doc.getDate("attributeValue");
            DateAttribute attribute = new DateAttribute(attributeName, attributeValue);
            Date lowerBound = doc.getDate("lowerBound");
            Date upperBound = doc.getDate("upperBound");

            DateRule rule = new DateRule(attribute, lowerBound, upperBound);
            dateRules.add(rule);
        }
        return dateRules;
    }


}
```

```
import categories.*;
import com.mongodb.ConnectionString;
import com.mongodb.MongoClientSettings;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import database.MongoMain;
import database.UndoClass;
import de.flapdoodle.embed.mongo.*;
import de.flapdoodle.embed.mongo.config.MongodConfig;
import de.flapdoodle.embed.mongo.config.Net;
import de.flapdoodle.embed.mongo.distribution.Version;
import guipanels.HelperClass;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import javax.swing.*;
import java.io.IOException;
import java.net.ServerSocket;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Map;

import static org.junit.jupiter.api.Assertions.*;

public class MongoMainTest {
    MongoMain mongoMain;
    MongodExecutable mongodExecutable;
    MongodProcess mongodProcess;

    @BeforeEach
    public void setUp() throws IOException {
        String ip = "localhost";
        int port = getFreePort();

        MongodConfig config = MongodConfig.builder()
                .version(Version.Main.PRODUCTION)
                .net(new Net(ip, port, de.flapdoodle.embed.process.runtime.Network.
localhostIsIPv6()))
                .build();

        MongodStarter starter = MongodStarter.getDefaultInstance();
        mongodExecutable = starter.prepare(config);
//        mongodExecutable.start();
        //mongodExecutable.start();

        MongoClientSettings settings = MongoClientSettings.builder()
                .applyConnectionString(new ConnectionString("mongodb://" + ip + ":" + port))
```

```
                .build();

        MongoClient mongoClient = MongoClients.create(settings);
        mongoMain = new MongoMain(mongoClient);
        mongodProcess = mongodExecutable.start();

        mongodExecutable.start();

    }

    @AfterEach
    public void tearDown() {
        if (mongodExecutable != null) {
            mongodProcess.stop();
            mongodExecutable.stop();
        }
    }

    private PrincipalCategory createSamplePrincipalCategory(String name) {
        return new PrincipalCategory(name);
    }

    private ResourceAction createAction(String actionName, String resourceName) {
        Resource resource = new Resource(resourceName);
        return new ResourceAction(actionName, resource);
    }

    private Principal createSamplePrincipal(String name, List<PrincipalCategory>
categories) {
        Principal principal = new Principal(name);
        return principal;
    }

    @Test
    public void testGetUndoClass() {
        // Create sample data
        PrincipalCategory pc1 = createSamplePrincipalCategory("Category1");
        PrincipalCategory pc2 = createSamplePrincipalCategory("Category2");
        Principal principal = createSamplePrincipal("TestPrincipal", Arrays.asList(pc1, pc2)
);
        ResourceAction action = createAction("TestAction", "TestResource");

        // Create an UndoClass object and add some actions
        UndoClass undoClass = new UndoClass();
        undoClass.addCreatePrincipal(principal);
        undoClass.addAddAction(action);

        // Save the UndoClass object
        mongoMain.saveUndoClass(undoClass);
```

```
        // Retrieve the UndoClass object
        UndoClass retrievedUndoClass = mongoMain.getUndoClass();

        // Test if the retrieved object is not null and has the correct number of actions
        assertNotNull(retrievedUndoClass);
        assertEquals(undoClass.getActionTracker().size(), retrievedUndoClass.
getActionTracker().size());
    }

    @Test
    public void testGetAssignCategories() {
        // Create sample data
        PrincipalCategory pc1 = createSamplePrincipalCategory("Category1");
        PrincipalCategory pc2 = createSamplePrincipalCategory("Category2");
        Principal principal = createSamplePrincipal("TestPrincipal", Arrays.asList(pc1, pc2)
);

        // Create an AssignCategories object and add some data
        AssignCategories assignCategories = new AssignCategories(new ArrayList<>(),
new ArrayList<>());
        assignCategories.addPrincipal(principal);
        assignCategories.addResource(new Resource("TestResource"));
        mongoMain.saveAssignCategories(assignCategories, new UndoClass());

        // Retrieve the AssignCategories object
        AssignCategories retrievedAssignCategories = mongoMain.getAssignCategories();

        // Test if the retrieved object is not null and has the correct number of items
        assertNotNull(retrievedAssignCategories);
        assertEquals(assignCategories.getPrincipals().size(), retrievedAssignCategories.
getPrincipals().size());
        assertEquals(assignCategories.getResourceActions().size(),
retrievedAssignCategories.getResourceActions().size());
        assertEquals(assignCategories.getPrincipalCategories().size(),
retrievedAssignCategories.getPrincipalCategories().size());
    }

    private static int getFreePort() {
        try (ServerSocket serverSocket = new ServerSocket(0)) {
            return serverSocket.getLocalPort();
        } catch (IOException e) {
            throw new RuntimeException("Failed to find a free port", e);
        }
    }


    @Test
    public void testUndoRemovePrincipal() {
```

```
        AssignCategories assignCategories = new AssignCategories(new ArrayList<>(),
new ArrayList<>());
        int prevSize = assignCategories.getPrincipals().size();
        UndoClass undoClass = new UndoClass();
        undoClass.addRemovePrincipal(new Principal("John"));
        simulateUndo(assignCategories, undoClass);
        assertEquals(prevSize+1, assignCategories.getPrincipals().size());
    }

    @Test
    public void testUndoCreatePrincipal() {
        AssignCategories assignCategories = new AssignCategories(new ArrayList<>(),
new ArrayList<>());
        int prevSize = assignCategories.getPrincipals().size();
        Principal p = new Principal("John");

        UndoClass undoClass = new UndoClass();
        undoClass.addCreatePrincipal(p);
        simulateUndo(assignCategories, undoClass);
        assertEquals(prevSize, assignCategories.getPrincipals().size());
    }

    @Test
    public void testUndoCreateCategory() {
        AssignCategories assignCategories = new AssignCategories(new ArrayList<>(),
new ArrayList<>());
        int prevSize = assignCategories.getPrincipals().size();
        PrincipalCategory pc = new PrincipalCategory("Everyone");
        pc.addPrincipal(new Principal("John"));
        assignCategories.addPrincipalCategory(pc);
        UndoClass undoClass = new UndoClass();
        undoClass.addCreateCategory(pc);
        simulateUndo(assignCategories, undoClass);
        assertEquals(prevSize, assignCategories.getPrincipalCategories().size());
    }

    @Test
    public void testUndoRemoveCategory() {
        AssignCategories assignCategories = new AssignCategories(new ArrayList<>(),
new ArrayList<>());
        int prevSize = assignCategories.getPrincipals().size();
        UndoClass undoClass = new UndoClass();
        undoClass.addRemovePrincipal(new Principal("John"));
        simulateUndo(assignCategories, undoClass);
        assertEquals(prevSize+1, assignCategories.getPrincipals().size());
    }

    public void simulateUndo(AssignCategories assignCategories, UndoClass
undoClass){
```

```
        if (!undoClass.getActionTracker().isEmpty()) {
            List<Object> lastEntry = undoClass.getActionTracker().get(undoClass.
getActionTracker().size() - 1);
            try {
                UndoClass.UNDO_TYPE actionType = (UndoClass.UNDO_TYPE)
lastEntry.get(0);
                if (actionType == UndoClass.UNDO_TYPE.CREATE_PRINCIPAL) {
                    Principal principal = (Principal) lastEntry.get(1);
                    assignCategories.removePrincipal(HelperClass.
getPrincipalByName(assignCategories.getPrincipals(), principal.getName()));
                    assignCategories.evaluatePrincipalCategories();

                } else if (actionType == UndoClass.UNDO_TYPE.
REMOVE_PRINCIPAL) {
                    Principal oldPrincipal = (Principal) lastEntry.get(1);
                    assignCategories.addPrincipal(oldPrincipal);
                    assignCategories.evaluatePrincipalCategories();
                }
                    else if (actionType == UndoClass.UNDO_TYPE.
CREATE_CATEGORY) {
                    PrincipalCategory category = (PrincipalCategory) lastEntry.get(1);
                    PrincipalCategory fixedRef = HelperClass.
getCategoryByName(assignCategories.getPrincipalCategories(), category.getName());
                    assignCategories.removePrincipalCategory(fixedRef);
                    for (PrincipalCategory pc : assignCategories.getPrincipalCategories())
{
                        pc.getJuniorCategories().remove(fixedRef);
                    }

                } else if (actionType == UndoClass.UNDO_TYPE.
REMOVE_CATEGORY) {
                    PrincipalCategory oldCategory = (PrincipalCategory) lastEntry.get(1);
                    List<PrincipalCategory> oldSeniorCategories =
(List<PrincipalCategory>) lastEntry.get(2);
                    oldCategory.getPrincipals().clear();
                    List<ResourceAction> fixedActions = new ArrayList<>();
                    for (ResourceAction oldAction : oldCategory.getActions()) {
                        ResourceAction actionRef = HelperClass.
getActionByName(assignCategories.getResourceActions(), oldAction.getName(),
oldAction.getResource().getName());
                        fixedActions.add(actionRef);
                    }
                    oldCategory.setActions(fixedActions);
                    List<Principal> fixedPrincipals = new ArrayList<>();
                    for (Principal p : oldCategory.getPrincipals()) {
                        Principal principalRef = HelperClass.
getPrincipalByName(assignCategories.getPrincipals(), p.getName());
                        fixedPrincipals.add(principalRef);
                    }
```

```
                    oldCategory.setPrincipals(fixedPrincipals);
                    List<PrincipalCategory> fixedJrCategories = new ArrayList<>();
                    for (PrincipalCategory jr : oldCategory.getJuniorCategories()) {
                        PrincipalCategory jrRef = HelperClass.
getCategoryByName(assignCategories.getPrincipalCategories(), jr.getName());
                        fixedJrCategories.add(jrRef);
                    }
                    oldCategory.setJuniorCategories(fixedJrCategories);
                    for (PrincipalCategory oldSenior : oldSeniorCategories) {
                        PrincipalCategory oldRef = HelperClass.
getCategoryByName(assignCategories.getPrincipalCategories(), oldSenior.getName());
                        oldRef.addJuniorCategory(oldCategory);
                    }
                    assignCategories.addPrincipalCategory(oldCategory);
                    assignCategories.evaluatePrincipalCategories();

                }
            } catch(Exception err){
                err.printStackTrace();
            }
            undoClass.getActionTracker().remove(lastEntry);
        }
    }

}
```

```
package guipanels;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class NumericTextField extends JTextField {

    public NumericTextField() {

        addKeyListener(new KeyAdapter() {
            @Override
            public void keyTyped(KeyEvent e) {
                char c = e.getKeyChar();
                if (!Character.isDigit(c)) {
                    e.consume();
                }
            }
        });
        setBorder(BorderFactory.createLineBorder(Color.RED));
    }
}
```

```
package categories;

import principal_resource_attributes.DateAttribute;
import principal_resource_attributes.IntegerAttribute;
import principal_resource_attributes.StringAttribute;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class Principal {

    private List<StringAttribute> stringAttributeList;
    private List<IntegerAttribute> integerAttributeList;
    private List<DateAttribute> dateAttributeList;
    private String name;

    public Principal(String name){
        this.name = name;
        stringAttributeList = new ArrayList<>();
        integerAttributeList = new ArrayList<>();
        dateAttributeList = new ArrayList<>();
    }
    public Principal(Principal other) {
        this.name = other.name;
        this.stringAttributeList = new ArrayList<>();
        for (StringAttribute attribute : other.stringAttributeList) {
            this.stringAttributeList.add(new StringAttribute(attribute));
        }
        this.integerAttributeList = new ArrayList<>();
        for (IntegerAttribute attribute : other.integerAttributeList) {
            this.integerAttributeList.add(new IntegerAttribute(attribute));
        }
        this.dateAttributeList = new ArrayList<>();
        for (DateAttribute attribute : other.dateAttributeList) {
            this.dateAttributeList.add(new DateAttribute(attribute));
        }
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Principal principal = (Principal) o;
        return Objects.equals(name, principal.name) &&
                Objects.equals(stringAttributeList, principal.stringAttributeList) &&
                Objects.equals(integerAttributeList, principal.integerAttributeList) &&
                Objects.equals(dateAttributeList, principal.dateAttributeList);
    }
```

```
    @Override
    public int hashCode() {
        return Objects.hash(name, stringAttributeList, integerAttributeList,
dateAttributeList);
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public List<StringAttribute> getStringAttributeList() {
        return stringAttributeList;
    }

    public void setStringAttributeList(List<StringAttribute> stringAttributeList) {
        this.stringAttributeList = stringAttributeList;
    }

    public List<IntegerAttribute> getIntegerAttributeList() {
        return integerAttributeList;
    }

    public void setIntegerAttributeList(List<IntegerAttribute> integerAttributeList) {
        this.integerAttributeList = integerAttributeList;
    }

    public List<DateAttribute> getDateAttributeList() {
        return dateAttributeList;
    }

    public void setDateAttributeList(List<DateAttribute> dateAttributeList) {
        this.dateAttributeList = dateAttributeList;
    }

}
```

```
package categories;

import categoryrules.*;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class PrincipalCategory {
    private String name;
    private List<PrincipalCategory> juniorCategories;
    private List<Principal> principals;
    private List<ResourceAction> actions;
    private List<StringRule> stringRules;
    private List<IntegerRule> integerRules;
    private List<DateRule> dateRules;

    public PrincipalCategory(String name) {
        this.name = name;
        principals = new ArrayList<>();
        stringRules = new ArrayList<>();
        integerRules = new ArrayList<>();
        dateRules = new ArrayList<>();
        juniorCategories = new ArrayList<>();
        actions = new ArrayList<>();
    }
    public PrincipalCategory(PrincipalCategory other) {
        this.name = other.name;
        this.juniorCategories = new ArrayList<>();
        for (PrincipalCategory category : other.getJuniorCategories()) {
            this.juniorCategories.add(new PrincipalCategory(category));
        }
        this.principals = new ArrayList<>();
        for (Principal principal : other.getPrincipals()) {
            this.principals.add(new Principal(principal));
        }
        this.actions = new ArrayList<>();
        for (ResourceAction action : other.getActions()) {
            this.actions.add(new ResourceAction(action));
        }
        this.stringRules = new ArrayList<>();
        for (StringRule rule : other.getStringRules()) {
            this.stringRules.add(new StringRule(rule));
        }
        this.integerRules = new ArrayList<>();
        for (IntegerRule rule : other.getIntegerRules()) {
            this.integerRules.add(new IntegerRule(rule));
        }
        this.dateRules = new ArrayList<>();
```

```
        for (DateRule rule : other.getDateRules()) {
            this.dateRules.add(new DateRule(rule));
        }
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        PrincipalCategory that = (PrincipalCategory) o;
        return Objects.equals(name, that.name) &&
                Objects.equals(juniorCategories, that.juniorCategories) &&
                Objects.equals(principals, that.principals) &&
                Objects.equals(actions, that.actions) &&
                Objects.equals(stringRules, that.stringRules) &&
                Objects.equals(integerRules, that.integerRules) &&
                Objects.equals(dateRules, that.dateRules);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, juniorCategories, principals, actions, stringRules,
integerRules, dateRules);
    }

    public List<ResourceAction> getActions(){
        return actions;
    }

    public void setActions(List<ResourceAction> actions){
        this.actions = actions;
    }

    public void addAction(ResourceAction a){
        actions.add(a);
    }

    public void removeAction(ResourceAction a){
        actions.remove(a);
    }

    public List<PrincipalCategory> getJuniorCategories(){
        return juniorCategories;
    }

    public void setJuniorCategories(List<PrincipalCategory> juniorCategories){
         this.juniorCategories = juniorCategories;
    }
```

```
public void addJuniorCategory(PrincipalCategory p){
    juniorCategories.add(p);
}

public void removeJuniorCategory(PrincipalCategory p){
    juniorCategories.remove(p);
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public List<Principal> getPrincipals() {
    return principals;
}

public void setPrincipals(List<Principal> principals) {
    this.principals = principals;
}

public void addPrincipal(Principal p){
    principals.add(p);
}

public void removePrincipal(Principal p){
    principals.remove(p);
}

public List<StringRule> getStringRules() {
    return stringRules;
}

public void setStringRules(List<StringRule> stringRules) {
    this.stringRules = stringRules;
}

public List<IntegerRule> getIntegerRules() {
    return integerRules;
}

public void setIntegerRules(List<IntegerRule> integerRules) {
    this.integerRules = integerRules;
}

public List<DateRule> getDateRules() {
```

```
        return dateRules;
    }

    public void setDateRules(List<DateRule> dateRules) {
        this.dateRules = dateRules;
    }

}
```

```
package guipanels;

import categories.*;
import categories.ResourceAction;

import javax.swing.*;
import java.awt.*;
import java.util.HashMap;
import java.util.Map;


public class PrincipalRulePanel {
    private JPanel innerPanel;
    private Map<ResourceAction, JCheckBox> actionJCheckBoxMap;
    private AssignCategories assignCategories;
    public PrincipalRulePanel(AssignCategories assignCategories){
        innerPanel = new JPanel();
        actionJCheckBoxMap = new HashMap<>();
        this.assignCategories = assignCategories;
        init();
    }

    public void init(){
        innerPanel.setLayout(new GridLayout(6, 6));
        for(ResourceAction a: assignCategories.getResourceActions()){
            JPanel currGridCell = new JPanel(new BorderLayout());
            innerPanel.add(currGridCell);
            String temp1 = a.getResource().getName();
            String temp2 = a.getName();
            currGridCell.add(new JLabel("<html>Resource: " + temp1 + "<br>Action: " +
temp2 + "</html>"), BorderLayout.CENTER);
            JCheckBox isSelectedCheckBox = new JCheckBox();
            actionJCheckBoxMap.put(a, isSelectedCheckBox);
            currGridCell.add(isSelectedCheckBox, BorderLayout.NORTH);
            currGridCell.setBorder(BorderFactory.createLineBorder(Color.BLACK));
        }
    }
    public Map<ResourceAction, JCheckBox> getActionJCheckBoxMap(){
        return actionJCheckBoxMap;
    }

    public JPanel getPanel(){
        return innerPanel;
    }

}
```

```
package guipanels;

import java.awt.*;
import javax.swing.*;

public class RegistrationForm {

    private JPanel panel;
    private int NUM_OF_ROWS = 6;
    private int NUM_OF_COLUMNS = 2;
    private int[] columnWidths;
    private JPanel[][] panelArray;

    private JPanel[] leftPanels;
    private JPanel[][] rightPanels;


    public RegistrationForm() {
        columnWidths = new int[NUM_OF_COLUMNS];
        panelArray = new JPanel[NUM_OF_ROWS][NUM_OF_COLUMNS];
        panel = new JPanel(new GridBagLayout());
        leftPanels = new JPanel[NUM_OF_ROWS];
        rightPanels = new JPanel[NUM_OF_ROWS][3];
        init();
    }

    public void init(){
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.fill = GridBagConstraints.BOTH;
        gbc.weightx = 1.0;

        columnWidths[0] = 1;
        columnWidths[1] = 2;
        for (int row = 0; row < NUM_OF_ROWS; row++) {
            gbc.gridy = row;
            gbc.weighty = 1.0;
            for (int col = 0; col < NUM_OF_COLUMNS; col++) {
                JPanel currentPanel = new JPanel();
                panelArray[row][col] = currentPanel;
                gbc.gridx = col;
                gbc.weightx = columnWidths[col];
                panel.add(currentPanel, gbc);
            }
        }


        for(int j = 0; j< NUM_OF_ROWS; j++){
            String[] attributeTypes = {"String attribute", "Integer attribute", "Date attribute"}; //
 KEEP THIS HERE OTHERWISE GETSELECTEDINDEX WONT WORK!
```

```
            JPanel currLeftPanel = panelArray[j][0];
            JPanel currRightPanel = panelArray[j][1];
//              currLeftPanel.setBorder(BorderFactory.createLineBorder(Color.ORANGE));
            JComboBox<String> attributeTypeComboBox = new
JComboBox<>(attributeTypes);
            currLeftPanel.add(attributeTypeComboBox);
            JTextField currField = new JTextField(10);

            CardLayout cl = new CardLayout();
            currRightPanel.setLayout(cl);

            JPanel stringPanel = new JPanel();
            stringPanel.setLayout(new BorderLayout());
            stringPanel.setBackground(Color.WHITE);
//              stringPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            JTextField tempField = new JTextField();
            stringPanel.add(tempField, BorderLayout.CENTER);
            currRightPanel.add(stringPanel, attributeTypes[0]);
            rightPanels[j][0] = stringPanel;

            JPanel numberPanel = new JPanel();
            numberPanel.setBackground(Color.WHITE);
            numberPanel.setLayout(new BorderLayout());
//              numberPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            NumericTextField tempNumericField = new NumericTextField();
            numberPanel.add(tempNumericField, BorderLayout.CENTER);
            currRightPanel.add(numberPanel, attributeTypes[1]);
            rightPanels[j][1] = numberPanel;

            JPanel datePanel = new JPanel();
            datePanel.setBackground(Color.WHITE);
//              datePanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            currRightPanel.add(datePanel, attributeTypes[2]);
            JTextField dateField = new JTextField(30);
            dateField.setEditable(false);
            datePanel.add(dateField);
            JButton selectDateButton = new JButton("Select date");
            datePanel.add(selectDateButton);
            rightPanels[j][2] = datePanel;

            selectDateButton.addActionListener(e -> {
                if(!HelperClass.isFrameOpen) {
                    HelperClass.showCalendar(dateField);
                }
            });

            String selectedItem = String.valueOf(attributeTypeComboBox.getSelectedItem())
;
            if(selectedItem != null){
```

```
            cl.show(currRightPanel, selectedItem);
        }

        attributeTypeComboBox.addActionListener(e -> {
            // Get the selected item from the JComboBox and print it
            String selectedItem1 = String.valueOf(attributeTypeComboBox.
getSelectedItem());
            if (selectedItem1 != null) {
                cl.show(currRightPanel, selectedItem1);
            }
        });
        JCheckBox enabledCheckbox = new JCheckBox();
        if(j == 0){
            currField.setText("Name");
            currField.setEditable(false);
            currField.setEnabled(false);
            enabledCheckbox.setSelected(true);
            enabledCheckbox.setEnabled(false);
            attributeTypeComboBox.setSelectedIndex(0);
            attributeTypeComboBox.setEnabled(false);
        } else {
            currField.setText("Custom " + j);
            enabledCheckbox.setSelected(false);
        }
        currLeftPanel.add(currField);
        currLeftPanel.add(enabledCheckbox);

        leftPanels[j] = currLeftPanel;
        }

      }

   public int getNUM_OF_ROWS(){
      return NUM_OF_ROWS;
   }
   public int getNUM_OF_COLUMNS(){
      return NUM_OF_COLUMNS;
   }

   public JPanel[] getLeftPanels() {
      return leftPanels;
   }

   public JPanel[][] getRightPanels() {
      return rightPanels;
   }

   public JPanel[][] getPanelArray(){
      return panelArray;
```

```
    }

    public JPanel getPanel() {
        return panel;
    }

}
```

```
package categories;

import java.util.Objects;

public class Resource {
    private String name;

    public Resource(String name){
        this.name = name;
    }
    public Resource(Resource other) {
        this.name = other.name;
    }

    public String getName(){
        return name;
    }

    public void setName(String name){
        this.name = name;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Resource resource = (Resource) o;
        return Objects.equals(name, resource.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name);
    }

}
```

```
package categories;


import java.util.Objects;

public class ResourceAction {
    private String name;
    private Resource resource;
    public ResourceAction(String name, Resource resource){
        this.name = name;
        this.resource = resource;
    }
    public ResourceAction(ResourceAction other) {
        this.name = other.name;
        this.resource = new Resource(other.resource);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        ResourceAction action = (ResourceAction) o;
        return Objects.equals(name, action.name) &&
                Objects.equals(resource, action.resource);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, resource);
    }

    public Resource getResource() {
        return resource;
    }

    public void setResource(Resource resource) {
        this.resource = resource;
    }

    public String getName(){
        return name;
    }

    public void setName(String name){
        this.name = name;
    }

    @Override
    public String toString(){
```

```
        return "Resource: " + resource.getName() + " action: " + name;
    }

}
```

```
package guipanels;

import javax.swing.*;
import java.awt.*;

public class ResourceRulePanel {
    private JPanel innerPanel;
    private JTextField[] textFieldArray;
    private JTextField resourceNameField;

    public ResourceRulePanel(){
        innerPanel = new JPanel();
        textFieldArray = new JTextField[10];
        resourceNameField = new JTextField();
        init();
    }

    public void init(){
        innerPanel.setLayout(new GridBagLayout()); // Set the layout for innerPanel to
GridBagLayout
        GridBagConstraints innerPanelGBC = new GridBagConstraints(); // Create
GridBagConstraints object for innerPanel

        innerPanelGBC.fill = GridBagConstraints.BOTH;
        innerPanelGBC.gridx = 0;
        innerPanelGBC.gridy = 0;
        innerPanelGBC.weightx = 1.0;
        innerPanelGBC.weighty = 0.2; // Set JLabel to take up 40% of the height
        innerPanel.add(new JLabel("Enter the resource name"), innerPanelGBC);

        innerPanelGBC.weighty = 0.15;
        innerPanelGBC.gridy = 1;
        innerPanel.add(resourceNameField, innerPanelGBC);

        innerPanelGBC.weighty = 0.2;
        innerPanelGBC.gridy = 2;
        innerPanel.add(new JLabel("Enter the actions for this resource"), innerPanelGBC);

        JPanel nestedPanel = new JPanel();
        nestedPanel.setLayout(new GridLayout(5, 2));
        innerPanelGBC.gridy = 3;
        innerPanelGBC.weighty = 0.4; // Set nestedPanel to take up 60% of the height
        innerPanel.add(nestedPanel, innerPanelGBC);
        for(int i = 0; i<textFieldArray.length; i++){
            textFieldArray[i] = new JTextField();
            nestedPanel.add(textFieldArray[i]);
        }
    }
```

```
    public JTextField[] getTextFieldArray(){
        return textFieldArray;
    }

    public JTextField getResourceNameField(){
        return resourceNameField;
    }

    public JPanel getPanel(){
        return innerPanel;
    }
}
```

```
package database;

import categories.AssignCategories;
import com.mongodb.client.MongoClients;

import javax.swing.*;
import java.awt.*;

public class SaveAssignCategoriesWorker extends SwingWorker<Void, Void> {
    private final JFrame frame;
    private final AssignCategories assignCategories;
    private final JButton saveButton;
    private final UndoClass undoClass;

    public SaveAssignCategoriesWorker(JFrame frame, AssignCategories
assignCategories, UndoClass undoClass, JButton saveButton) {
        this.frame = frame;
        this.assignCategories = assignCategories;
        this.saveButton = saveButton;
        this.undoClass = undoClass;
    }

    @Override
    protected Void doInBackground() throws Exception {
        saveButton.setEnabled(false);
        MongoMain mongoMain = new MongoMain(MongoClients.
create(MongoDB_CONFIG.DATABASE_URL));
        mongoMain.saveAssignCategories(assignCategories, undoClass);
        return null;
    }

    @Override
    protected void done() {
        try {
            JOptionPane.showMessageDialog(frame, "Saved");
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            saveButton.setEnabled(true);
        }
    }
}
```

```
package guipanels;

import categories.AssignCategories;
import database.LoadDataWorker;
import database.LoadFileWorker;
import database.UndoClass;

import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;

public class StartPanel {

    CardLayout cl;
    JFrame frame;
    JPanel START_PANEL;

    public StartPanel() {
        cl = new CardLayout();
        frame = new JFrame();
        init();
    }

    public void init(){
        START_PANEL = new JPanel();
        START_PANEL.setLayout(cl);

        // Create a panel with two buttons
        JPanel buttonPanel = new JPanel(new BorderLayout());

        // Load File button
        JButton loadFileButton = new JButton("New policy");
        JButton tempButton = new JButton("Load policy");

        loadFileButton.addActionListener(e -> {
            int res = JOptionPane.showConfirmDialog(frame, "This will clear any previous
data in the database!");
            if (res == JOptionPane.YES_OPTION) {
                LoadFileWorker loadFileWorker = new LoadFileWorker(frame,
START_PANEL, cl, loadFileButton, tempButton, new AssignCategories(new
ArrayList<>(), new ArrayList<>()),
                        new UndoClass());
                loadFileWorker.execute();
            }
        });

        buttonPanel.add(loadFileButton, BorderLayout.WEST);
```

```
        tempButton.addActionListener(acl -> {
            LoadDataWorker loadDataWorker = new LoadDataWorker(frame,
START_PANEL, cl, tempButton, loadFileButton);
            loadDataWorker.execute();
        });

        buttonPanel.add(tempButton, BorderLayout.EAST);

        JLabel label = new JLabel("Policy manager", SwingConstants.CENTER);
        label.setFont(new Font("Calibri", Font.BOLD, 30));
        buttonPanel.add(label, BorderLayout.CENTER);

        START_PANEL.add(buttonPanel, "buttonPanel");

        // Show the first panel
        cl.show(START_PANEL, "buttonPanel");

        frame.add(START_PANEL);
        frame.setPreferredSize(new Dimension(1600, 900));
//        frame.setPreferredSize(new Dimension(1280, 720));
        frame.setResizable(true);
        frame.setVisible(true);
        frame.pack();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(StartPanel::new);
    }
}
```

```
package principal_resource_attributes;


import java.util.Objects;

public class StringAttribute {
    private String value;
    private String name;

    public StringAttribute(String name, String value) {
        this.name = name.toLowerCase();
        this.value = value.toLowerCase();
    }
    public StringAttribute(StringAttribute other) {
        this.name = other.name;
        this.value = other.value;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name.toLowerCase();
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value.toLowerCase();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        StringAttribute that = (StringAttribute) o;
        return Objects.equals(value, that.value) &&
                Objects.equals(name, that.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(value, name);
    }

    @Override
```

```
    public String toString() {
        return "Type: String attribute | " + name.toString() + " | Value: " + value;
    }
}
```

```
package categoryrules;

import principal_resource_attributes.StringAttribute;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class StringRule{

    private List<String> requirements;
    private StringAttribute attribute;

    public StringAttribute getAttribute() {
        return attribute;
    }

    public void setAttribute(StringAttribute attribute) {
        this.attribute = attribute;
    }

    public StringRule(StringRule other) {
        this.attribute = new StringAttribute(other.attribute);
        this.requirements = new ArrayList<>(other.requirements);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        StringRule that = (StringRule) o;
        return Objects.equals(requirements, that.requirements) &&
                Objects.equals(attribute, that.attribute);
    }

    @Override
    public int hashCode() {
        return Objects.hash(requirements, attribute);
    }

    public StringRule(StringAttribute attribute, List<String> requirements){
        this.attribute = attribute;
        this.requirements = requirements;
    }

    public List<String> getRequirements(){
        return requirements;
    }
```

```
    public void setRequirements(List<String> requirements){
        this.requirements = requirements;
    }

    public void addRequirement(String s){
        requirements.add(s);
    }

    public void removeRequirement(String s){
        requirements.remove(s);
    }

    @Override
    public String toString(){
        return attribute.getName() + " = " + requirements.toString();
    }
}
```

```
package database;

import categories.ResourceAction;
import categories.Principal;
import categories.PrincipalCategory;
import categories.Resource;

import java.util.*;

public class UndoClass {
    public enum UNDO_TYPE {
        UPDATE_PRINCIPAL,
        CREATE_PRINCIPAL,
        REMOVE_PRINCIPAL,
        ADD_RESOURCE,
        REMOVE_RESOURCE,
        ADD_ACTION,
        REMOVE_ACTION,
        UPDATE_CATEGORY,
        CREATE_CATEGORY,
        REMOVE_CATEGORY,
        UPDATE_PERMISSIONS,
        UPDATE_HIERARCHY
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        UndoClass that = (UndoClass) o;
        return Objects.equals(actionTracker, that.actionTracker);
    }

    @Override
    public int hashCode() {
        return Objects.hash(actionTracker);
    }


    List<List<Object>> actionTracker;
    public UndoClass(){
        actionTracker = new ArrayList<>();
    }
    public List<List<Object>> getActionTracker() {
        return actionTracker;
```

```
    }

    public void addUpdatePrincipal(Principal oldPrincipal, Principal newPrincipal){
        List<Object> info = new ArrayList<>();
        info.add(UNDO_TYPE.UPDATE_PRINCIPAL);
        info.add(oldPrincipal);
        info.add(newPrincipal);
        actionTracker.add(info);
    }
    public void addCreatePrincipal(Principal principal){
        List<Object> nameList = new ArrayList<>();
        nameList.add(UNDO_TYPE.CREATE_PRINCIPAL);
        nameList.add(principal);
        actionTracker.add(nameList);
    }
    public void addRemovePrincipal(Principal p){
        List<Object> info = new ArrayList<>();
        info.add(UNDO_TYPE.REMOVE_PRINCIPAL);
        info.add(p);
        actionTracker.add(info);
    }
    public void addAddResource(Resource resource){
        List<Object> nameList = new ArrayList<>();
        nameList.add(UNDO_TYPE.ADD_RESOURCE);
        nameList.add(resource);
        actionTracker.add(nameList);
    }
    public void addRemoveResource(Resource r, Map<ResourceAction,
List<PrincipalCategory>> assignedPerms){
        List<Object> nameList = new ArrayList<>();
        nameList.add(UNDO_TYPE.REMOVE_RESOURCE);
        nameList.add(r);
        nameList.add(assignedPerms);
        actionTracker.add(nameList);
    }
    public void addAddAction(ResourceAction a){
        List<Object> nameList = new ArrayList<>();
        nameList.add(UNDO_TYPE.ADD_ACTION);
        nameList.add(a);
        actionTracker.add(nameList);
    }
    public void addRemoveAction(ResourceAction a, List<PrincipalCategory>
principalCategories){
        List<Object> nameList = new ArrayList<>();
        nameList.add(UNDO_TYPE.REMOVE_ACTION);
        nameList.add(a);
        nameList.add(principalCategories);
        actionTracker.add(nameList);
    }
```

```
    public void addUpdateCategory(PrincipalCategory oldCategory, PrincipalCategory
newCategory){
        List<Object> infoList = new ArrayList<>();
        infoList.add(UNDO_TYPE.UPDATE_CATEGORY);
        infoList.add(oldCategory);
        infoList.add(newCategory);
        actionTracker.add(infoList);
    }
    public void addCreateCategory(PrincipalCategory category){
        List<Object> infoList = new ArrayList<>();
        infoList.add(UNDO_TYPE.CREATE_CATEGORY);
        infoList.add(category);
        actionTracker.add(infoList);
    }
    public void addRemoveCategory(PrincipalCategory oldCategory,
List<PrincipalCategory> oldSeniorCategories){
        List<Object> infoList = new ArrayList<>();
        infoList.add(UNDO_TYPE.REMOVE_CATEGORY);
        infoList.add(oldCategory);
        infoList.add(oldSeniorCategories);
        actionTracker.add(infoList);
    }
    public void addUpdatePermissions(PrincipalCategory category,
List<ResourceAction> oldActions){
        List<Object> infoList = new ArrayList<>();
        infoList.add(UNDO_TYPE.UPDATE_PERMISSIONS);
        infoList.add(category);
        infoList.add(oldActions);
        actionTracker.add(infoList);
    }
    public void addUpdateHierarchy(PrincipalCategory pc, List<PrincipalCategory>
oldJuniorCategoryNames){
        List<Object> infoList = new ArrayList<>();
        infoList.add(UNDO_TYPE.UPDATE_HIERARCHY);
        infoList.add(pc);
        infoList.add(oldJuniorCategoryNames);
        actionTracker.add(infoList);
    }
}
```

```
package categories;

import categoryrules.*;
import principal_resource_attributes.*;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class AssignCategories {

        List<Principal> principals;
        List<PrincipalCategory> principalCategories;
        List<ResourceAction> resourceActions;
        List<Resource> resources;

    public AssignCategories(List<Principal> principals, List<PrincipalCategory>
principalCategories) {
        this.principals = principals;
        this.principalCategories = principalCategories;
        resources = new ArrayList<>();
        resourceActions = new ArrayList<>();
        evaluatePrincipalCategories();
    }

    public List<ResourceAction> getResourceActions() {
        return resourceActions;
    }

    public List<Resource> getResources() {
        return resources;
    }

    public void setResources(List<Resource> resources) {
        this.resources = resources;
    }

    public void setResourceActions(List<ResourceAction> resourceActions) {
        this.resourceActions = resourceActions;
    }

    public void addResource(Resource r) {
        resources.add(r);
    }

    public void removeResource(Resource r) {
        resources.remove(r);
    }
```

```
    public void addPrincipalCategory(PrincipalCategory p) {
        principalCategories.add(p);
    }

    public void removePrincipalCategory(PrincipalCategory p) {
        principalCategories.remove(p);
    }

    public void addPrincipal(Principal p) {
        principals.add(p);
    }

    public void removePrincipal(Principal p) {
        principals.remove(p);
    }

    public void evaluatePrincipalCategories() {
        for (PrincipalCategory category : principalCategories) {
            category.getPrincipals().clear();
            for (Principal principal : principals) {
                boolean inCategory = true;
                for (StringRule rule : category.getStringRules()) {
                    boolean noMatch = true;
                    for (StringAttribute principalAttribute : principal.getStringAttributeList()) {
                        if (principalAttribute.getName().strip().equalsIgnoreCase(rule.
getAttribute().getName().strip())) {
                            noMatch = false;
                            if (!rule.getRequirements().contains(principalAttribute.getValue())) {
                                inCategory = false;
                            }
                        }
                    }
                    if (noMatch) {
                        inCategory = false;
                    }
                }
                for (IntegerRule rule : category.getIntegerRules()) {
                    boolean noMatch = true;
                    for (IntegerAttribute integerAttribute : principal.getIntegerAttributeList()) {
                        if (integerAttribute.getName().strip().equalsIgnoreCase(rule.getAttribute()
.getName().strip())) {
                            noMatch = false;
                            if (integerAttribute.getValue() < rule.getLowerBound() ||
integerAttribute.getValue() > rule.getUpperBound()) {
                                inCategory = false;
                            }
                        }
                    }
                    if (noMatch) {
```

```
                    inCategory = false;
                }
            }
            for (DateRule rule : category.getDateRules()) {
                boolean noMatch = true;
                for (DateAttribute dateAttribute : principal.getDateAttributeList()) {
                    if (dateAttribute.getName().strip().equalsIgnoreCase(rule.getAttribute().
getName().strip())) {
                        noMatch = false;
                        if (dateAttribute.getValue().before(rule.getLowerBound()) ||
dateAttribute.getValue().after(rule.getUpperBound())) {
                            inCategory = false;
                        }
                    }
                }
                if (noMatch) {
                    inCategory = false;
                }
            }
            if (inCategory) {
                category.addPrincipal(principal);
            } else {
                category.removePrincipal(principal);
            }
        }
    }
}

public List<Principal> getPrincipals() {
    return principals;
}

public void setPrincipals(List<Principal> principals) {
    this.principals = principals;
}

public List<PrincipalCategory> getPrincipalCategories() {
    return principalCategories;
}

public void setPrincipalCategories(List<PrincipalCategory> principalCategories) {
    this.principalCategories = principalCategories;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof AssignCategories)) return false;
    AssignCategories that = (AssignCategories) o;
```

```
        return Objects.equals(principals, that.principals) &&
                Objects.equals(principalCategories, that.principalCategories) &&
                Objects.equals(resourceActions, that.resourceActions) &&
                Objects.equals(resources, that.resources);
    }

    @Override
    public int hashCode() {
        return Objects.hash(principals, principalCategories, resourceActions, resources);
    }


}
```

```
package guipanels;

import principal_resource_attributes.*;

import javax.swing.*;
import java.awt.*;
import java.util.*;
import java.util.List;

public class CategoryRulePanel {
    List<StringAttribute> ALL_STRING_ATTRIBUTES;
    List<IntegerAttribute> ALL_INTEGER_ATTRIBUTES;
    List<DateAttribute> ALL_DATE_ATTRIBUTES;
    JPanel[] panelArray; // to store the rows of grid panel
    private JPanel panel;
    private JTextField categoryNameField;

    private Map<StringAttribute, JTextField> stringAttributeFieldMap;

    private Map<Integer, JCheckBox> isEnabledMap;

    private Map<IntegerAttribute, JPanel> integerRangeValueMap;

    private Map<DateAttribute, JPanel> dateBetweenValueMap;

    int combined_row_size;

    public CategoryRulePanel(List<StringAttribute> ALL_STRING_ATTRIBUTES,
List<IntegerAttribute> ALL_INTEGER_ATTRIBUTES,
                             List<DateAttribute> ALL_DATE_ATTRIBUTES){
        this.ALL_STRING_ATTRIBUTES = ALL_STRING_ATTRIBUTES;
        this.ALL_DATE_ATTRIBUTES = ALL_DATE_ATTRIBUTES;
        this.ALL_INTEGER_ATTRIBUTES = ALL_INTEGER_ATTRIBUTES;
        combined_row_size = ALL_STRING_ATTRIBUTES.size() +
ALL_DATE_ATTRIBUTES.size() + ALL_INTEGER_ATTRIBUTES.size();
        panelArray = new JPanel[combined_row_size];
        stringAttributeFieldMap = new HashMap<>();
        integerRangeValueMap = new HashMap<>();
        isEnabledMap = new HashMap<>();
        dateBetweenValueMap = new HashMap<>();
        panel = new JPanel();
        categoryNameField = new JTextField();
        init();
    }

    public Map<DateAttribute, JPanel> getDateBetweenValueMap() {
        return dateBetweenValueMap;
    }
```

```
    public Map<Integer, JCheckBox> getIsEnabledMap(){
        return isEnabledMap;
    }

    public Map<StringAttribute, JTextField> getStringAttributeJTextFieldMap(){
        return stringAttributeFieldMap;
    }

    public Map<IntegerAttribute, JPanel> getIntegerRangeValueMap() {
        return integerRangeValueMap;
    }

    public JTextField getCategoryNameField(){
        return categoryNameField;
    }

    public JPanel[] getPanelArray() {
        return panelArray;
    }

    public void init(){
        panel.setLayout(new GridBagLayout());
        GridBagConstraints mainGBC = new GridBagConstraints();
        mainGBC.fill = GridBagConstraints.BOTH;
        mainGBC.gridx = 0;
        mainGBC.gridy = 0;
        mainGBC.weightx = 1.0;
        mainGBC.weighty = 10.0;

        JPanel selectorPanel = new JPanel();
        selectorPanel.setName("Selector panel");

//        selectorPanel.setBorder(BorderFactory.createLineBorder(Color.GREEN, 3));
        panel.add(selectorPanel, mainGBC);
        selectorPanel.setLayout(new GridBagLayout());
        GridBagConstraints selectorGBC = new GridBagConstraints();

        selectorGBC.fill = GridBagConstraints.BOTH;
        selectorGBC.weighty = 1;

        selectorGBC.gridx = 1;
        selectorGBC.gridy = 0;
        JPanel categoryNamePanel = new JPanel();
        categoryNamePanel.setLayout(new BorderLayout());
        selectorGBC.weightx = 0.9;
        categoryNameField = new JTextField(50);
        categoryNamePanel.add(categoryNameField, BorderLayout.CENTER);
        selectorPanel.add(categoryNamePanel, selectorGBC);
//        categoryNamePanel.setBorder(BorderFactory.createLineBorder(Color.
```

```
MAGENTA));
        JLabel categoryLabel = new JLabel("<html>Category<br>Name</html>");
        categoryNamePanel.add(categoryLabel, BorderLayout.WEST);

        mainGBC.gridx = 0;
        mainGBC.gridy = 1;
        mainGBC.weightx = 1.0;
        mainGBC.weighty = 10.0;
        JPanel gridPanel = new JPanel();
        gridPanel.setName("Grid panel");
//        gridPanel.setPreferredSize(new Dimension(1200, 500));

        JScrollPane scrollPane = new JScrollPane(gridPanel);
        scrollPane.setPreferredSize(new Dimension(1200, 500));
        panel.add(scrollPane, mainGBC);

//        gridPanel.setLayout(new GridLayout(10, 0));
        gridPanel.setLayout(new GridBagLayout());
        GridBagConstraints tempGBC = new GridBagConstraints();
        tempGBC.fill = GridBagConstraints.HORIZONTAL;
        tempGBC.weightx = 1;
        tempGBC.weighty = 0.1;

        for(int i=0; i<combined_row_size; i++){
            JPanel currPanel = new JPanel();
            currPanel.setPreferredSize(new Dimension(1200, 60));
            currPanel.setLayout(new BorderLayout());
            JCheckBox checkBox = new JCheckBox();
            checkBox.setSelected(false);
            currPanel.add(checkBox, BorderLayout.WEST);
            panelArray[i] = currPanel;
            isEnabledMap.put(i, checkBox);
            if(i < ALL_STRING_ATTRIBUTES.size()){
                StringAttribute currAttribute = ALL_STRING_ATTRIBUTES.get(i);
                JPanel stringJPanel = createStringAttributePanel(currAttribute);
                currPanel.add(stringJPanel, BorderLayout.CENTER);
                for(Component tempC: stringJPanel.getComponents()){
                    if(tempC instanceof JTextField){
                        stringAttributeFieldMap.put(currAttribute, (JTextField) tempC);
                    }
                }
            } else {
                int fixedIndex = i - ALL_STRING_ATTRIBUTES.size();
                if(fixedIndex < ALL_INTEGER_ATTRIBUTES.size()){
                    IntegerAttribute currAttribute = ALL_INTEGER_ATTRIBUTES.
get(fixedIndex);
                    JPanel integerJPanel = createIntegerAttributePanel(currAttribute);
                    currPanel.add(integerJPanel, BorderLayout.CENTER);
                } else {
```

```
                int finalFixedIndex = i - ALL_STRING_ATTRIBUTES.size() -
ALL_INTEGER_ATTRIBUTES.size();
                if (finalFixedIndex < ALL_DATE_ATTRIBUTES.size()) {
                    DateAttribute currAttribute = ALL_DATE_ATTRIBUTES.
get(finalFixedIndex);
                    JPanel dateJPanel = createDateAttributePanel(currAttribute);
                    currPanel.add(dateJPanel, BorderLayout.CENTER);
                }
            }
        }
        tempGBC.gridx = 0;
        tempGBC.gridy = i;
        gridPanel.add(panelArray[i], tempGBC);
    }

  }

  public JPanel createStringAttributePanel(StringAttribute currAttr){
    JPanel tempPanel = new JPanel();
    tempPanel.setLayout(new BorderLayout());
    tempPanel.setName(currAttr.getName());

    Icon infoIcon = UIManager.getIcon("OptionPane.informationIcon");
    JLabel infoLabel = new JLabel(infoIcon);
    String infoText = "<html>" +
          "Enter a list of strings, separated by commas, with NO SPACES" +
          "<br>e.g. John,David,Mark,Simon,Christopher" +
          "</div></html>";
    infoLabel.setToolTipText(infoText);
    tempPanel.add(infoLabel, BorderLayout.LINE_START);

    JLabel nameLabel = new JLabel("Attribute name: " + currAttr.getName());
    tempPanel.add(nameLabel, BorderLayout.PAGE_START);

    JTextField strTextField = new JTextField();
    tempPanel.add(strTextField, BorderLayout.CENTER);
    // support entering lists e.g. john,david,mark
    return tempPanel;
  }

  public JPanel createIntegerAttributePanel(IntegerAttribute currAttr){
    JPanel tempPanel = new JPanel();
    tempPanel.setName(currAttr.getName());
    tempPanel.setLayout(new BorderLayout());

    JPanel leftPanel = new JPanel();
    leftPanel.setLayout(new BorderLayout());
    leftPanel.setName("Left panel");
    tempPanel.add(leftPanel, BorderLayout.LINE_START);
```

```
        Icon infoIcon = UIManager.getIcon("OptionPane.informationIcon");
        JLabel infoLabel = new JLabel(infoIcon);
        String infoText = "<html>" +
                "For the greater than, less than or equal to options, enter a number e.g. 123"
+
                "<br>For the range option, enter two numbers in each of the boxes" +
                "</div></html>";
        infoLabel.setToolTipText(infoText);
        leftPanel.add(infoLabel, BorderLayout.LINE_START);

        JLabel nameLabel = new JLabel("Attribute name: " + currAttr.getName());
        leftPanel.add(nameLabel, BorderLayout.PAGE_START);

        JPanel doubleNumericField = new JPanel();
        integerRangeValueMap.put(currAttr, doubleNumericField);
        doubleNumericField.setName("Double numeric field");
        doubleNumericField.setLayout(new BoxLayout(doubleNumericField, BoxLayout.
X_AXIS));
        doubleNumericField.add(Box.createHorizontalGlue());

        NumericTextField lowerBoundField = new NumericTextField();
        lowerBoundField.setName("Lower bound field");
        NumericTextField upperBoundField = new NumericTextField();
        upperBoundField.setName("Upper bound field");

        doubleNumericField.add(lowerBoundField);
        doubleNumericField.add(Box.createHorizontalStrut(10));
        doubleNumericField.add(upperBoundField);
        doubleNumericField.add(Box.createHorizontalGlue());

        tempPanel.add(doubleNumericField, BorderLayout.CENTER);

        return tempPanel;
    }

    public JPanel createDateAttributePanel(DateAttribute currAttr) {
        JPanel tempPanel = new JPanel();
        tempPanel.setName(currAttr.getName());
        tempPanel.setLayout(new BorderLayout());

        JPanel leftPanel = new JPanel();
        leftPanel.setName("Left panel");
        leftPanel.setLayout(new BorderLayout());
        tempPanel.add(leftPanel, BorderLayout.LINE_START);

        Icon infoIcon = UIManager.getIcon("OptionPane.informationIcon");
        JLabel infoLabel = new JLabel(infoIcon);
```

```
        String infoText = "";
        infoLabel.setToolTipText(infoText);
        leftPanel.add(infoLabel, BorderLayout.LINE_START);

        JLabel nameLabel = new JLabel("Attribute name: " + currAttr.getName());
        leftPanel.add(nameLabel, BorderLayout.PAGE_START);

        tempPanel.add(createBetweenPanel(currAttr), BorderLayout.CENTER);

        return tempPanel;
    }

    public JPanel createBetweenPanel(DateAttribute attr){
        JPanel betweenPanel = new JPanel();
        dateBetweenValueMap.put(attr, betweenPanel);
        betweenPanel.setName("Between panel");
        betweenPanel.setLayout(new BoxLayout(betweenPanel, BoxLayout.X_AXIS));

        JPanel leftBetweenPanel = new JPanel(new BorderLayout());
        leftBetweenPanel.setName("Left between panel");
        JPanel rightBetweenPanel = new JPanel(new BorderLayout());
        rightBetweenPanel.setName("Right between panel");
        JButton firstBetweenButton = new JButton("Select first date");
        JButton secondBetweenButton = new JButton("Select second date");
        JTextField firstBetweenTextField = new JTextField();
        JTextField secondBetweenTextField = new JTextField();
        firstBetweenTextField.setEditable(false);
        secondBetweenTextField.setEditable(false);

        firstBetweenButton.addActionListener(e -> {
            if(!HelperClass.isFrameOpen) {
                HelperClass.showCalendar(firstBetweenTextField);
            }
        });

        secondBetweenButton.addActionListener(e -> {
            if(!HelperClass.isFrameOpen) {
                HelperClass.showCalendar(secondBetweenTextField);
            }
        });

        leftBetweenPanel.add(firstBetweenTextField, BorderLayout.CENTER);
        leftBetweenPanel.add(firstBetweenButton, BorderLayout.LINE_END);

        rightBetweenPanel.add(secondBetweenTextField, BorderLayout.CENTER);
        rightBetweenPanel.add(secondBetweenButton, BorderLayout.LINE_END);

        betweenPanel.add(Box.createHorizontalGlue());
        betweenPanel.add(leftBetweenPanel);
```

```
        betweenPanel.add(Box.createHorizontalStrut(10));
        betweenPanel.add(rightBetweenPanel);
        betweenPanel.add(Box.createHorizontalGlue());
        return betweenPanel;
    }

    public JPanel getPanel(){
        return panel;
    }

}
```

```
package principal_resource_attributes;


import java.util.Date;
import java.util.Objects;

public class DateAttribute {
    private Date value;
    private String name;

    public DateAttribute(String name, Date value) {
        this.name = name.toLowerCase();
        this.value = value;
    }
    public DateAttribute(DateAttribute other) {
        this.value = new Date(other.getValue().getTime());
        this.name = other.getName();
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name.toLowerCase();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        DateAttribute that = (DateAttribute) o;
        return Objects.equals(value, that.value) &&
                Objects.equals(name, that.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(value, name);
    }

    public Date getValue() {
        return value;
    }

    public void setValue(Date value) {
        this.value = value;
    }
```

```
    @Override
    public String toString() {
        return "Type: Date attribute | " + name.toString() + " | Value: " + value;
    }
}
```

```
package categoryrules;

import principal_resource_attributes.DateAttribute;

import java.util.Date;
import java.util.Objects;

public class DateRule{
    private Date lowerBound;
    private Date upperBound;
    private DateAttribute attribute;

    public DateAttribute getAttribute() {
        return attribute;
    }

    public void setAttribute(DateAttribute attribute) {
        this.attribute = attribute;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        DateRule that = (DateRule) o;
        return Objects.equals(lowerBound, that.lowerBound) &&
                Objects.equals(upperBound, that.upperBound) &&
                Objects.equals(attribute, that.attribute);
    }

    @Override
    public int hashCode() {
        return Objects.hash(lowerBound, upperBound, attribute);
    }


    public DateRule(DateAttribute attribute, Date date1, Date date2){
        this.attribute = attribute;
        lowerBound = date1;
        upperBound = date2;
    }
    public DateRule(DateRule other) {
        this.lowerBound = new Date(other.lowerBound.getTime());
        this.upperBound = new Date(other.upperBound.getTime());
        this.attribute = new DateAttribute(other.attribute);
    }

    public Date getLowerBound() {
        return lowerBound;
```

```
    }

    public void setLowerBound(Date lowerBound) {
        this.lowerBound = lowerBound;
    }

    public Date getUpperBound() {
        return upperBound;
    }

    public void setUpperBound(Date upperBound) {
        this.upperBound = upperBound;
    }

    @Override
    public String toString(){
        return attribute.getName().toString() +  "[ " + lowerBound.toString() + " - " +
upperBound.toString() + " ]";
    }
}
```

```
package guipanels;
import categories.*;
import categories.ResourceAction;
import database.SaveAssignCategoriesWorker;
import database.UndoClass;
import org.graphstream.graph.*;
import org.graphstream.graph.implementations.SingleGraph;
import org.graphstream.ui.geom.Point2;
import org.graphstream.ui.geom.Point3;
import org.graphstream.ui.graphicGraph.GraphicElement;
import org.graphstream.ui.swing_viewer.DefaultView;
import org.graphstream.ui.swing_viewer.SwingViewer;
import org.graphstream.ui.swing_viewer.util.DefaultMouseManager;
import org.graphstream.ui.view.Viewer;
import org.graphstream.ui.view.camera.Camera;
import org.graphstream.ui.view.util.InteractiveElement;
import org.graphstream.ui.view.util.MouseManager;
import principal_resource_attributes.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseEvent;
import java.util.*;
import java.util.List;

public class GraphDisplay {
    Graph graph;
    JPanel MAIN_PANEL;
    JFrame frame;
    CardLayout cl;
    AssignCategories assignCategories;
    Stack<AssignCategories> PREVIOUS_STATE;
    ArrayList<String> ADMIN_LOG;
    DefaultView DEFAULT_VIEW;
    UndoClass undoClass;

    public GraphDisplay(AssignCategories assignCategories, UndoClass undoClass,
JPanel MAIN_PANEL, JFrame frame, CardLayout cl) {
        this.MAIN_PANEL = MAIN_PANEL;
        this.frame = frame;
        this.cl = cl;
        this.assignCategories = assignCategories;
        this.undoClass = undoClass;
        PREVIOUS_STATE = new Stack<>();
        ADMIN_LOG = new ArrayList<>();
        MAIN_PANEL.add(createGraph(), "Graph");
        cl.show(MAIN_PANEL, "Graph");
    }

    public JPanel createGraph() {
```

```
        JPanel graphPanel = new JPanel();

        //GRAPH CODE
        System.setProperty("org.graphstream.ui", "swing");

        graph = new SingleGraph("embedded");
        SwingViewer viewer = new SwingViewer(graph, Viewer.ThreadingModel.
GRAPH_IN_GUI_THREAD);

        DefaultView view = (DefaultView) viewer.addDefaultView(false);
        DEFAULT_VIEW = view;
        view.setPreferredSize(new Dimension(500, 500));
        viewer.enableAutoLayout();
        graph.setAttribute("ui.quality");
        graph.setAttribute("ui.antialias");

        // Add custom MouseManager for panning
        MouseManager mouseManager = new DefaultMouseManager() {
            private double lastX;
            private double lastY;
            private Node draggedNode;

            @Override
            public void mousePressed(MouseEvent event) {
                lastX = event.getX();
                lastY = event.getY();

                GraphicElement element = view.findGraphicElementAt(EnumSet.
of(InteractiveElement.NODE), event.getX(), event.getY());
                if (element instanceof Node) {
                    draggedNode = (Node) element;
                } else {
                    draggedNode = null;
                }
            }

            @Override
            public void mouseClicked(MouseEvent e) {
                double x = e.getX();
                double y = e.getY();

                GraphicElement element = view.findGraphicElementAt(EnumSet.
of(InteractiveElement.NODE), x, y);
                if (element instanceof Node node && ((Node) element).getId() != null) {
                    if (node.getId().startsWith("Principal category")) {
                        String catName = node.getId().substring(node.getId().indexOf(':') + 2);
                        PrincipalCategory curr = HelperClass.
getCategoryByName(assignCategories.getPrincipalCategories(), catName);
                        if(curr != null){
```

```
                    if(e.getButton() == MouseEvent.BUTTON1) {
                        Set<Principal> categoryPrincipals = new HashSet<>();
                        for(PrincipalCategory pc: assignCategories.getPrincipalCategories()
){
                            if(HelperClass.findAllJuniorCategories(pc).contains(curr)){
                                categoryPrincipals.addAll(pc.getPrincipals());
                            }
                        }
                        List<String> categoryMembers = new ArrayList<>();
                        for (Principal p: categoryPrincipals) {
                            categoryMembers.add(p.getName());
                        }
                        List<String> categoryRules = new ArrayList<>();
                        curr.getStringRules().forEach(rule -> categoryRules.add(rule.
toString()));
                        curr.getIntegerRules().forEach(rule -> categoryRules.add(rule.
toString()));
                        curr.getDateRules().forEach(rule -> categoryRules.add(rule.
toString()));

                        HelperClass.showDoubleDialog(frame, "Info", categoryRules,
categoryMembers, "Rules", "Members");
                    } else if(e.getButton() == MouseEvent.BUTTON3) {
                        Set<ResourceAction> categoryActions = new HashSet<>();
                        List<String> juniorCategories = new ArrayList<>();
                        for(PrincipalCategory pc: HelperClass.findAllJuniorCategories(curr))
{
                            categoryActions.addAll(pc.getActions());
                            if(!pc.equals(curr)){
                                juniorCategories.add(pc.getName());
                            }
                        }
                        List<String> actions = new ArrayList<>();
                        for (ResourceAction a: categoryActions) {
                            actions.add(a.toString());
                        }
                        HelperClass.showDoubleDialog(frame, "Info", actions,
juniorCategories, "Permissions", "Junior categories");
                    }
                }
            } else if(node.getId().startsWith("Principal")){
                String principalName = node.getId().substring(node.getId().indexOf(':') +
2);
                Principal curr = HelperClass.getPrincipalByName(assignCategories.
getPrincipals(), principalName);
                if(curr != null){
                    if(e.getButton() == MouseEvent.BUTTON1) {
                        List<String> attributes = new ArrayList<>();
                        curr.getStringAttributeList().forEach(c -> attributes.add(c.toString()))
```

```
;
                        curr.getIntegerAttributeList().forEach(c -> attributes.add(c.toString())
);
                        curr.getDateAttributeList().forEach(c -> attributes.add(c.toString()));
                        HelperClass.showListDialog(frame, "Attributes ", attributes);
                    } else if(e.getButton() == MouseEvent.BUTTON3){
                        Set<PrincipalCategory> associatedCategories = new HashSet<>();
                        Set<ResourceAction> associatedPermissions = new HashSet<>();
                        for(PrincipalCategory pc: assignCategories.getPrincipalCategories()
){
                            if(pc.getPrincipals().contains(curr)){
                                associatedCategories.add(pc);
                                associatedCategories.addAll(HelperClass.
findAllJuniorCategories(pc));
                            }
                        }
                        for(PrincipalCategory pc: associatedCategories){
                            associatedPermissions.addAll(pc.getActions());
                        }
                        List<String> categoryList = new ArrayList<>();
                        for (PrincipalCategory pc: associatedCategories) {
                            categoryList.add(pc.getName());
                        }
                        List<String> permissionList = new ArrayList<>();
                        for (ResourceAction a: associatedPermissions) {
                            permissionList.add(a.toString());
                        }
                        HelperClass.showDoubleDialog(frame, "Info", categoryList,
permissionList, "Categories", "Actions");
                    }
                }
            }  else if(node.getId().startsWith("Action")){
                String resourceName = "";
                ResourceAction curr = null;
                Node[] neighborNodeArray = node.neighborNodes().
toArray(Node[]::new);
                for(Node neighborNode : neighborNodeArray){
                    if(neighborNode.getId().startsWith("Resource")){
                        String nodeId = node.getId() + neighborNode.getId();
                        if(graph.getEdge(nodeId) != null){
                            resourceName = neighborNode.getId().substring(neighborNode.
getId().indexOf(':') + 2);
                        }
                    }
                }
                String actionName = node.getId().substring(node.getId().indexOf(':') + 2).
replace(resourceName, "").trim();
                curr = HelperClass.getActionByName(assignCategories.
getResourceActions(), actionName, resourceName);
```

```
                    if(curr != null){
                        Set<Principal> authorisedPrincipals = new HashSet<>();
                        for(PrincipalCategory pc: assignCategories.getPrincipalCategories()
){
                            for(PrincipalCategory junior: HelperClass.
findAllJuniorCategories(pc)){
                                if(junior.getActions().contains(curr)){
                                    authorisedPrincipals.addAll(pc.getPrincipals());
                                }
                            }
                        }
                        List<String> tempList = new ArrayList<>();
                        for (Principal p: authorisedPrincipals) {
                            tempList.add(p.getName());
                        }
                        HelperClass.showListDialog(frame, "Principals ", tempList);
                    }
                }else if(node.getId().startsWith("Resource")){
                    String resourceName = node.getId().substring(node.getId().indexOf(':') +
2);
                    Resource curr = HelperClass.getResourceByName(assignCategories.
getResources(), resourceName);
                    if(curr != null){
                        Set<Principal> authorisedPrincipals = new HashSet<>();
                        for(ResourceAction a : HelperClass.
getAllResourceActions(assignCategories.getResourceActions(), curr)) {
                            for (PrincipalCategory pc : assignCategories.
getPrincipalCategories()) {
                                for (PrincipalCategory junior : HelperClass.
findAllJuniorCategories(pc)) {
                                    if (junior.getActions().contains(a)) {
                                        authorisedPrincipals.addAll(pc.getPrincipals());
                                    }
                                }
                            }
                        }
                        List<String> tempList = new ArrayList<>();
                        for (Principal p: authorisedPrincipals) {
                            tempList.add(p.getName());
                        }
                        HelperClass.showListDialog(frame, "Principals ", tempList);
                    }
                }
            }
        }

        @Override
        public void mouseDragged(MouseEvent event) {
            double currentX = event.getX();
```

```
            double currentY = event.getY();
            double dx = (currentX - lastX) * 0.01; // reduce panning speed by a factor of
10*10
            double dy = (currentY - lastY) * 0.01;
            if (draggedNode == null) {

                view.getCamera().setViewCenter(
                        view.getCamera().getViewCenter().x - dx / view.getCamera().
getViewPercent(),
                        view.getCamera().getViewCenter().y + dy / view.getCamera().
getViewPercent(),
                        0);

            }
            lastX = currentX;
            lastY = currentY;
        }

    };
    view.setMouseManager(mouseManager);

    view.addMouseWheelListener(e -> {
        e.consume();
        int i = e.getWheelRotation();
        double factor = Math.pow(1.25, i);
        Camera cam = view.getCamera();
        double zoom = cam.getViewPercent() * factor;
        Point2 pxCenter = cam.transformGuToPx(cam.getViewCenter().x, cam.
getViewCenter().y, 0);
        Point3 guClicked = cam.transformPxToGu(e.getX(), e.getY());
        double newRatioPx2Gu = cam.getMetrics().ratioPx2Gu / factor;
        double x = guClicked.x + (pxCenter.x - e.getX()) / newRatioPx2Gu;
        double y = guClicked.y - (pxCenter.y - e.getY()) / newRatioPx2Gu;
        cam.setViewCenter(x, y, 0);
        cam.setViewPercent(zoom);
    });

    graphPanel.setLayout(new BorderLayout());
    graphPanel.add(view, BorderLayout.CENTER);

    JButton addPrincipalButton = new JButton("<html>Update/Create<br>principal</
html>");
    addPrincipalButton.addActionListener(e -> addPrincipal());
    JButton removePrincipalButton = new JButton("<html>Remove<br>principal</
html>");
    removePrincipalButton.addActionListener(e -> removePrincipal());
    JButton updateCategoryRulesButton = new JButton("<html>Update/
Create<br>category</html>");
    updateCategoryRulesButton.addActionListener(e -> updateCategoryRules());
```

```
        JButton deleteCategoryButton = new JButton("<html>Delete<br>category</html>")
;
        deleteCategoryButton.addActionListener(e -> deleteCategory());
        JButton refreshGraphButton = new JButton("<html>Refresh<br>graph</html>");
        refreshGraphButton.addActionListener(e -> updateGraph());
        JButton centerGraphButton = new JButton("<html>Center<br>graph");
        centerGraphButton.addActionListener(e -> centerGraph());
        JButton addResourceButton = new JButton("<html>Add<br>resource</html>");
        addResourceButton.addActionListener(e -> addResource());
        JButton removeResourceButton = new JButton("<html>Remove<br>resource</
html>");
        removeResourceButton.addActionListener(e -> removeResource());
        JButton addActionButton = new JButton("<html>Add<br>action</html>");
        addActionButton.addActionListener(e -> addAction());
        JButton removeActionButton = new JButton("<html>Remove<br>action</html>");
        removeActionButton.addActionListener(e -> removeAction());
        JButton checkCategoryRulesButton = new JButton("
<html>Configure<br>permissions</html>");
        checkCategoryRulesButton.addActionListener(e -> configurePermissions());
        JButton updateHierarchyButton = new JButton("<html>Update<br>hierarchy</
html>");
        updateHierarchyButton.addActionListener(e -> updateHierarchy());
        JButton saveToDBButton = new JButton("<html>Save to<br>database</html>");
        saveToDBButton.addActionListener(e -> saveToDB(saveToDBButton));
        JButton checkPARButton = new JButton("<html>Check<br>PAR</html>");
        checkPARButton.addActionListener(e -> checkPar());
        JButton infoIcon = new JButton("<html>Query<br>Information</html>");
        infoIcon.addActionListener(e -> showInfoPanel());
        JButton undoActionButton = new JButton("<html>Undo<br>action</html>");
        undoActionButton.addActionListener(e -> undoAction());
        JButton viewLogButton = new JButton("<html>View<br>log</html>");
        viewLogButton.addActionListener(e -> viewLog());

        String[] graphViewOptions = {"Normal view", "PCA", "Unassigned", "Hide
unassigned"};
        JComboBox<String> graphViewDropdown = new
JComboBox<>(graphViewOptions);
        graphViewDropdown.setMaximumSize(new Dimension(150, 30));
        graphViewDropdown.addActionListener(e -> filterGraphView(graphViewDropdown.
getSelectedIndex()));

        JButton policy2 = new JButton("2nd query");
        policy2.addActionListener(e -> secondQuery());
        JButton policy3 = new JButton("3rd query");
        policy3.addActionListener(e -> thirdQuery());
        JButton policy4 = new JButton("4th query");
        policy4.addActionListener(e -> fourthQuery());
        JButton policy5 = new JButton("5th query");
        policy5.addActionListener(e -> fifthQuery());
```

```
        JButton policy6 = new JButton("6th query");
        policy6.addActionListener(e -> sixthQuery());

        updateGraph();

        JPanel topButtonPanel = new JPanel();
        JPanel leftButtonPanel = new JPanel();
        leftButtonPanel.setLayout(new BoxLayout(leftButtonPanel, BoxLayout.Y_AXIS));
        leftButtonPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10)); //
Add spacing to the left panel

        JPanel rightButtonPanel = new JPanel();
        rightButtonPanel.setLayout(new BoxLayout(rightButtonPanel, BoxLayout.Y_AXIS))
;
        rightButtonPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10)); //
Add spacing to the left panel

        graphPanel.add(topButtonPanel, BorderLayout.NORTH);
        graphPanel.add(rightButtonPanel, BorderLayout.EAST);
        graphPanel.add(leftButtonPanel, BorderLayout.WEST);

        JButton[] topButtons = {
                addPrincipalButton, removePrincipalButton, addResourceButton,
removeResourceButton, addActionButton,
                removeActionButton, updateCategoryRulesButton, deleteCategoryButton,
checkCategoryRulesButton, updateHierarchyButton
        };
        for(JButton jButton : topButtons){
            topButtonPanel.add(jButton);
        }
        JButton[] leftButtons = {
                refreshGraphButton, centerGraphButton
        };
        for(JButton jButton : leftButtons){
            leftButtonPanel.add(jButton);
            leftButtonPanel.add(Box.createVerticalStrut(10));
        }
        leftButtonPanel.add(graphViewDropdown);
        graphViewDropdown.setMaximumSize(new Dimension(150, 30));

        JButton[] rightButtons = {
                saveToDBButton, checkPARButton, policy2, policy3, policy4, policy5, policy6,
 viewLogButton, undoActionButton, infoIcon
        };
        for(JButton jButton : rightButtons){
            rightButtonPanel.add(jButton);
            rightButtonPanel.add(Box.createVerticalStrut(10));
        }
        return graphPanel;
```

```
    }


    public void secondQuery() { // "Can all resources be accessed by at least one user?"
        boolean res = true;
        List<String> names = new ArrayList<>();
        for(Resource r : assignCategories.getResources()){
            Set<Principal> authorisedPrincipals = new HashSet<>();
            List<ResourceAction> correspondingActions = HelperClass.
getAllResourceActions(assignCategories.getResourceActions(), r);
            for(ResourceAction curr : correspondingActions){
                for(PrincipalCategory pc: assignCategories.getPrincipalCategories()){
                    for(PrincipalCategory junior: HelperClass.findAllJuniorCategories(pc)){
                        if(junior.getActions().contains(curr)){
                            authorisedPrincipals.addAll(pc.getPrincipals());
                        }
                    }
                }
            }
            if(authorisedPrincipals.isEmpty()){
                names.add(r.getName());
                res = false;
            }
        }
        String output = (res) ? "True" : "False" + " -> " + names;
        JOptionPane.showMessageDialog(frame, output);
    }

    public void thirdQuery() { // "Can all users access at least some of the resources?"
        boolean res = true;
        List<String> names = new ArrayList<>();
        for (Principal curr : assignCategories.getPrincipals()) {
            Set<PrincipalCategory> associatedCategories = new HashSet<>();
            Set<ResourceAction> associatedPermissions = new HashSet<>();
            for (PrincipalCategory pc : assignCategories.getPrincipalCategories()) {
                if (pc.getPrincipals().contains(curr)) {
                    associatedCategories.add(pc);
                    associatedCategories.addAll(HelperClass.findAllJuniorCategories(pc));
                }
            }
            for (PrincipalCategory pc : associatedCategories) {
                associatedPermissions.addAll(pc.getActions());
            }
            if (associatedPermissions.isEmpty()) {
                names.add(curr.getName());
                res = false;
            }
        }
        String output = (res) ? "True" : "False" + " -> " + names;
```

```
        JOptionPane.showMessageDialog(frame, output);
    }

    public void fourthQuery(){ // "Are all the principals associated with at least one
category?"
        boolean res = true;
        List<String> names = new ArrayList<>();
        for (Principal curr : assignCategories.getPrincipals()) {
            Set<PrincipalCategory> associatedCategories = new HashSet<>();
            for (PrincipalCategory pc : assignCategories.getPrincipalCategories()) {
                if (pc.getPrincipals().contains(curr)) {
                    associatedCategories.add(pc);
                    associatedCategories.addAll(HelperClass.findAllJuniorCategories(pc));
                }
            }
            if(associatedCategories.isEmpty()){
                names.add(curr.getName());
                res = false;
            }
        }
        String output = (res) ? "True" : "False" + " -> " + names;
        JOptionPane.showMessageDialog(frame, output);
    }

    public void fifthQuery(){ // "Are there permissions associated with each category?"
        boolean res = true;
        List<String> names = new ArrayList<>();
        for(PrincipalCategory curr : assignCategories.getPrincipalCategories()) {
            Set<ResourceAction> categoryActions = new HashSet<>();
            for (PrincipalCategory pc : HelperClass.findAllJuniorCategories(curr)) {
                categoryActions.addAll(pc.getActions());
            }
            if(categoryActions.isEmpty()){
                names.add(curr.getName());
                res = false;
            }
        }
        String output = (res) ? "True" : "False" + " -> " + names;
        JOptionPane.showMessageDialog(frame, output);
    }

    public void sixthQuery() { // "Are rule definitions compatible with axiom c0 in paper
Specification and Analysis of ABAC Policies via the Category-Based Metamodel"
        List<String> outputList = new ArrayList<>();
        for (PrincipalCategory pc : assignCategories.getPrincipalCategories()) {
            for (PrincipalCategory jr : pc.getJuniorCategories()) {
                if (!HelperClass.isRuleSubset(jr, pc)) {
                    outputList.add(pc.getName() + " -> " + jr.getName());
                }
```

```
            }
        }
        if (outputList.isEmpty()) {
            JOptionPane.showMessageDialog(frame, "true");
        } else {
            HelperClass.showListDialog(frame, "Categories", outputList);
        }
    }

    public void showInfoPanel(){
        String infoString = "<html>" +
                "<br>2nd query - Can all resources be accessed by at least one user?" +
                "<br>3rd query - Can all users access at least some of the resources?" +
                "<br>4th query - Are all the principals associated with at least one category?"
+
                "<br>5th query - Are there permissions associated with each category? "
                +"<br>6th query - If a category is senior, does it's defining condition
<br>imply membership to it's junior categories?"
                + "</html>";
        JOptionPane.showMessageDialog(frame, infoString);
    }

    public void filterGraphView(int index){
        updateGraph();
        SwingUtilities.invokeLater(() -> {
            if(index == 1) {
                graph.nodes().forEach(node -> {
                    if(! (node.getId().startsWith("Principal") || node.getId().startsWith("
Principal category")) ||
                            (node.neighborNodes().toArray(Node[]::new)).length == 0){
                        node.setAttribute("ui.hide");
                        node.edges().forEach(edge ->{
                            edge.setAttribute("ui.hide");
                        });
                    }
                });
            } else if(index == 2){
                graph.nodes().forEach(node -> {
                    Node[] nodeArray = node.neighborNodes().toArray(Node[]::new);
                    if(nodeArray.length > 0){
                        node.setAttribute("ui.hide");
                        node.edges().forEach(edge -> edge.setAttribute("ui.hide"));
                    }
                });
            } else if(index == 3){
                graph.nodes().forEach(node -> {
                    Node[] nodeArray = node.neighborNodes().toArray(Node[]::new);
                    if(nodeArray.length == 0){
                        node.setAttribute("ui.hide");
```

```
                }
            });
        }
    });
    centerGraph();
}

public void centerGraph() {
    SwingUtilities.invokeLater(() -> DEFAULT_VIEW.getCamera().resetView());
}

public void addAction(){
    JDialog dialog = new JDialog();
    JPanel panel = new JPanel();
    dialog.add(panel);
    panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

    JLabel label1 = new JLabel("Resource name");
    JTextField field1 = new JTextField(10);
    panel.add(label1);
    panel.add(field1);

    JLabel label2 = new JLabel("Action name");
    JTextField field2 = new JTextField(10);
    panel.add(label2);
    panel.add(field2);

    JButton okButton = new JButton("OK");
    okButton.addActionListener(e -> {
        String formattedResourceText = field1.getText().toLowerCase().strip();
        String formattedActionText = field2.getText().toLowerCase().strip();
        Resource r = HelperClass.getResourceByName(assignCategories.
getResources(), formattedResourceText);
        boolean error = false;
        if(r != null){
            for(ResourceAction a : assignCategories.getResourceActions()){
                if(a.getResource().equals(r) && a.getName().strip().
equalsIgnoreCase(formattedActionText)){
                    error = true;
                }
            }
        } else {
            error = true;
        }
        if(!error){
            ResourceAction newAction = new ResourceAction(formattedActionText, r);
            undoClass.addAddAction(newAction);
            assignCategories.getResourceActions().add(newAction);
            updateGraph();
```

```
            dialog.dispose();
        } else {
            JOptionPane.showMessageDialog(frame, "Resource not found or action
already exists");
        }
    });
    panel.add(okButton);
    dialog.setTitle("Add action");
    panel.setPreferredSize(new Dimension(300, 300));
    dialog.pack();
    dialog.setLocationRelativeTo(null);
    dialog.setVisible(true);
  }

  public void removeAction(){
    JDialog dialog = new JDialog();
    JPanel panel = new JPanel();
    dialog.add(panel);
    panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

    JLabel label1 = new JLabel("Resource name");
    JTextField field1 = new JTextField(10);
    panel.add(label1);
    panel.add(field1);

    JLabel label2 = new JLabel("Action name");
    JTextField field2 = new JTextField(10);
    panel.add(label2);
    panel.add(field2);

    JButton okButton = new JButton("OK");
    okButton.addActionListener(e -> {
        String formattedResourceText = field1.getText().toLowerCase().strip();
        String formattedActionText = field2.getText().toLowerCase().strip();
        ResourceAction toRemove = HelperClass.getActionByName(assignCategories.
getResourceActions(),formattedActionText, formattedResourceText);
        if(toRemove != null  && HelperClass.getAllResourceActions(assignCategories.
getResourceActions(), toRemove.getResource()).size() > 1){
            List<PrincipalCategory> nameList = new ArrayList<>();
            assignCategories.getPrincipalCategories().forEach(principalCategory -> {
                    if (principalCategory.getActions().contains(toRemove)) {
                        nameList.add(principalCategory);
                    }
                    principalCategory.removeAction(toRemove);
                });
            undoClass.addRemoveAction(toRemove, nameList);
            assignCategories.getResourceActions().remove(toRemove);
            updateGraph();
            dialog.dispose();
```

```
            } else {
                JOptionPane.showMessageDialog(frame, "<html>Not found or only one
action<br>on resource remaining</html>");
            }
        });
        panel.add(okButton);
        dialog.setTitle("Remove action");
        panel.setPreferredSize(new Dimension(300, 300));
        dialog.pack();
        dialog.setLocationRelativeTo(null);
        dialog.setVisible(true);
    }

    public void undoAction() {
        if (!undoClass.getActionTracker().isEmpty()) {
            int responseInt = JOptionPane.showConfirmDialog(frame, "Are you sure you
want to undo?");
            if (responseInt == JOptionPane.YES_OPTION) {
                List<Object> lastEntry = undoClass.getActionTracker().get(undoClass.
getActionTracker().size() - 1);
                try {
                    UndoClass.UNDO_TYPE actionType = (UndoClass.UNDO_TYPE)
lastEntry.get(0);
                    if (actionType == UndoClass.UNDO_TYPE.UPDATE_PRINCIPAL) {
                        Principal oldPrincipal = (Principal) lastEntry.get(1);
                        Principal newPrincipal = (Principal) lastEntry.get(2);
                        assignCategories.removePrincipal(newPrincipal);
                        assignCategories.addPrincipal(oldPrincipal);
                        assignCategories.evaluatePrincipalCategories();

                    } else if (actionType == UndoClass.UNDO_TYPE.CREATE_PRINCIPAL) {
                        Principal principal = (Principal) lastEntry.get(1);
                        assignCategories.removePrincipal(HelperClass.
getPrincipalByName(assignCategories.getPrincipals(), principal.getName()));
                        assignCategories.evaluatePrincipalCategories();

                    } else if (actionType == UndoClass.UNDO_TYPE.REMOVE_PRINCIPAL) {
                        Principal oldPrincipal = (Principal) lastEntry.get(1);
                        assignCategories.addPrincipal(oldPrincipal);
                        assignCategories.evaluatePrincipalCategories();

                    } else if (actionType == UndoClass.UNDO_TYPE.ADD_RESOURCE) {
                        Resource resource = (Resource) lastEntry.get(1);
                        Resource fixedRes = HelperClass.
getResourceByName(assignCategories.getResources(), resource.getName());
                        List<ResourceAction> resourceActions = HelperClass.
getAllResourceActions(assignCategories.getResourceActions(), fixedRes);
                        assignCategories.getPrincipalCategories().forEach(principalCategory ->
principalCategory.getActions().removeAll(resourceActions));
```

```
                assignCategories.getResourceActions().removeAll(resourceActions);
                assignCategories.getResources().remove(fixedRes);
                assignCategories.evaluatePrincipalCategories();

            } else if (actionType == UndoClass.UNDO_TYPE.REMOVE_RESOURCE)
{
                Resource toAdd = (Resource) lastEntry.get(1);
                Map<ResourceAction, List<PrincipalCategory>> assignedPerms =
(Map<ResourceAction, List<PrincipalCategory>>) lastEntry.get(2);
                assignCategories.addResource(toAdd);
                for (ResourceAction a : assignedPerms.keySet()) {
                    ResourceAction fixedAction = new ResourceAction(a.getName(),
toAdd);
                    assignCategories.getResourceActions().add(fixedAction);
                    List<PrincipalCategory> categories = assignedPerms.get(a);
                    for (PrincipalCategory currentPrincipalCategory : categories) {
                        PrincipalCategory fixedRef = HelperClass.
getCategoryByName(assignCategories.getPrincipalCategories(),
                                currentPrincipalCategory.getName());
                        fixedRef.addAction(a);
                    }
                }

            } else if (actionType == UndoClass.UNDO_TYPE.ADD_ACTION) {
                ResourceAction a = (ResourceAction) lastEntry.get(1);
                ResourceAction ref = HelperClass.getActionByName(assignCategories.
getResourceActions(), a.getName(), a.getResource().getName());
                assignCategories.getPrincipalCategories().forEach(principalCategory ->
principalCategory.removeAction(ref));
                assignCategories.getResourceActions().remove(ref);

            } else if (actionType == UndoClass.UNDO_TYPE.REMOVE_ACTION) {
                ResourceAction a = (ResourceAction) lastEntry.get(1);
                Resource resourceRef = HelperClass.
getResourceByName(assignCategories.getResources(), a.getResource().getName());
                ResourceAction actionRef = new ResourceAction(a.getName(),
resourceRef);
                assignCategories.getResourceActions().add(actionRef);
                List<PrincipalCategory> categories = (List<PrincipalCategory>)
lastEntry.get(2);
                for (PrincipalCategory category : categories) {
                    PrincipalCategory fixedRef = HelperClass.
getCategoryByName(assignCategories.getPrincipalCategories(), category.getName());
                    fixedRef.addAction(actionRef);
                }

            } else if (actionType == UndoClass.UNDO_TYPE.UPDATE_CATEGORY) {
                PrincipalCategory oldCategory = (PrincipalCategory) lastEntry.get(1);
                PrincipalCategory newCategory = (PrincipalCategory) lastEntry.get(2);
```

```
                newCategory.setStringRules(oldCategory.getStringRules());
                newCategory.setIntegerRules(oldCategory.getIntegerRules());
                newCategory.setDateRules(oldCategory.getDateRules());
                newCategory.getPrincipals().clear();
                newCategory.setName(oldCategory.getName());
                assignCategories.evaluatePrincipalCategories();

            } else if (actionType == UndoClass.UNDO_TYPE.CREATE_CATEGORY) {
                PrincipalCategory category = (PrincipalCategory) lastEntry.get(1);
                PrincipalCategory fixedRef = HelperClass.
getCategoryByName(assignCategories.getPrincipalCategories(), category.getName());
                assignCategories.removePrincipalCategory(fixedRef);
                for (PrincipalCategory pc : assignCategories.getPrincipalCategories()) {
                    pc.getJuniorCategories().remove(fixedRef);
                }

            } else if (actionType == UndoClass.UNDO_TYPE.REMOVE_CATEGORY)
{
                PrincipalCategory oldCategory = (PrincipalCategory) lastEntry.get(1);
                List<PrincipalCategory> oldSeniorCategories =
(List<PrincipalCategory>) lastEntry.get(2);
                oldCategory.getPrincipals().clear();
                List<ResourceAction> fixedActions = new ArrayList<>();
                for (ResourceAction oldAction : oldCategory.getActions()) {
                    ResourceAction actionRef = HelperClass.
getActionByName(assignCategories.getResourceActions(), oldAction.getName(),
oldAction.getResource().getName());
                    fixedActions.add(actionRef);
                }
                oldCategory.setActions(fixedActions);
                List<Principal> fixedPrincipals = new ArrayList<>();
                for (Principal p : oldCategory.getPrincipals()) {
                    Principal principalRef = HelperClass.
getPrincipalByName(assignCategories.getPrincipals(), p.getName());
                    fixedPrincipals.add(principalRef);
                }
                oldCategory.setPrincipals(fixedPrincipals);
                List<PrincipalCategory> fixedJrCategories = new ArrayList<>();
                for (PrincipalCategory jr : oldCategory.getJuniorCategories()) {
                    PrincipalCategory jrRef = HelperClass.
getCategoryByName(assignCategories.getPrincipalCategories(), jr.getName());
                    fixedJrCategories.add(jrRef);
                }
                oldCategory.setJuniorCategories(fixedJrCategories);
                for (PrincipalCategory oldSenior : oldSeniorCategories) {
                    PrincipalCategory oldRef = HelperClass.
getCategoryByName(assignCategories.getPrincipalCategories(), oldSenior.getName());
                    oldRef.addJuniorCategory(oldCategory);
                }
```

```
                    assignCategories.addPrincipalCategory(oldCategory);
                    assignCategories.evaluatePrincipalCategories();

                } else if (actionType == UndoClass.UNDO_TYPE.
UPDATE_PERMISSIONS) {
                    PrincipalCategory curr = (PrincipalCategory) lastEntry.get(1);
                    List<ResourceAction> oldActions = (List<ResourceAction>) lastEntry.
get(2);
                    curr.setActions(oldActions);

                } else if (actionType == UndoClass.UNDO_TYPE.UPDATE_HIERARCHY)
{
                    PrincipalCategory curr = (PrincipalCategory) lastEntry.get(1);
                    PrincipalCategory categoryRef = HelperClass.
getCategoryByName(assignCategories.getPrincipalCategories(), curr.getName());
                    categoryRef.getJuniorCategories().clear();
                    List<PrincipalCategory> oldJrCategories = (List<PrincipalCategory>)
lastEntry.get(2);
                    for (PrincipalCategory oldJr : oldJrCategories) {
                        PrincipalCategory jrRef = HelperClass.
getCategoryByName(assignCategories.getPrincipalCategories(), oldJr.getName());
                        categoryRef.addJuniorCategory(jrRef);
                    }

                }
            } catch(Exception err){
                err.printStackTrace();
                JOptionPane.showMessageDialog(frame, JOptionPane.
ERROR_MESSAGE);
            }
            undoClass.getActionTracker().remove(lastEntry);
            if(!ADMIN_LOG.isEmpty()) {
                ADMIN_LOG.remove(ADMIN_LOG.get(ADMIN_LOG.size() - 1));
            }
            updateGraph();
        }
    } else {
        JOptionPane.showMessageDialog(frame, "Empty");
    }
}

public void checkPar(){
    JDialog dialog = new JDialog();
    JPanel panel = new JPanel();
    dialog.add(panel);
    panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

    JLabel label1 = new JLabel("Principal name");
    JTextField field1 = new JTextField(10);
```

```
        panel.add(label1);
        panel.add(field1);

        JLabel label2 = new JLabel("Action name");
        JTextField field2 = new JTextField(10);
        panel.add(label2);
        panel.add(field2);

        JLabel label3 = new JLabel("Resource name");
        JTextField field3 = new JTextField(10);
        panel.add(label3);
        panel.add(field3);

        JButton okButton = new JButton("OK");
        okButton.addActionListener(e -> {
            Principal principal = HelperClass.getPrincipalByName(assignCategories.
getPrincipals(), field1.getText());
            ResourceAction action = HelperClass.getActionByName(assignCategories.
getResourceActions(), field2.getText(), field3.getText());
            if(principal != null && action != null){
                Set<ResourceAction> perms = new HashSet<>();
                for(PrincipalCategory pc : assignCategories.getPrincipalCategories()){
                    List<PrincipalCategory> allCategories = HelperClass.
findAllJuniorCategories(pc);
                    if(pc.getPrincipals().contains(principal)) {
                        allCategories.forEach(category -> perms.addAll(category.getActions()));
                    }
                }
                String isAuthorised = (perms.contains(action)) ? "Authorised " : " Not
authorised";
                JOptionPane.showMessageDialog(frame, isAuthorised);
            } else {
                JOptionPane.showMessageDialog(frame, "Does not exist");
            }
        });
        panel.add(okButton);
        dialog.setTitle("Check PAR");
        panel.setPreferredSize(new Dimension(300, 300));
        dialog.pack();
        dialog.setLocationRelativeTo(null);
        dialog.setVisible(true);
    }

    public void saveToDB(JButton button){
        SaveAssignCategoriesWorker saveAssignCategoriesWorker = new
SaveAssignCategoriesWorker(frame, assignCategories, undoClass, button);
        saveAssignCategoriesWorker.execute();
    }
```

```
    public void viewLog() {
        try {
            List<String> logList = new ArrayList<>();
            for (List<Object> innerList : undoClass.getActionTracker()) {
                UndoClass.UNDO_TYPE firstElem = (UndoClass.UNDO_TYPE) innerList.
get(0);
                if (firstElem == UndoClass.UNDO_TYPE.UPDATE_PRINCIPAL) {
                    logList.add(firstElem + " " + ((Principal) innerList.get(2)).getName());
                } else if (firstElem == UndoClass.UNDO_TYPE.CREATE_PRINCIPAL) {
                    logList.add(firstElem + " " +  ((Principal) innerList.get(1)).getName());
                } else if (firstElem == UndoClass.UNDO_TYPE.REMOVE_PRINCIPAL) {
                    logList.add(firstElem + " " +  ((Principal) innerList.get(1)).getName());
                } else if (firstElem == UndoClass.UNDO_TYPE.ADD_RESOURCE) {
                    logList.add(firstElem + " " +  ((Resource) innerList.get(1)).getName());
                } else if (firstElem == UndoClass.UNDO_TYPE.REMOVE_RESOURCE) {
                    logList.add(firstElem + " " +  ((Resource) innerList.get(1)).getName());
                } else if (firstElem == UndoClass.UNDO_TYPE.ADD_ACTION) {
                    logList.add(firstElem + " " +  "Action: " + ((ResourceAction) innerList.get(1)).
getName() +
                            " Resource: " + ((ResourceAction) innerList.get(1)).getResource().
getName());
                } else if (firstElem == UndoClass.UNDO_TYPE.REMOVE_ACTION) {
                    logList.add(firstElem + " " +  "Action: " + ((ResourceAction) innerList.get(1)).
getName() +
                            " Resource: " + ((ResourceAction) innerList.get(1)).getResource().
getName());
                } else if (firstElem == UndoClass.UNDO_TYPE.UPDATE_CATEGORY) {
                    logList.add(firstElem + " " +  ((PrincipalCategory) innerList.get(2)).
getName());
                } else if (firstElem == UndoClass.UNDO_TYPE.CREATE_CATEGORY) {
                    logList.add(firstElem + " " +  ((PrincipalCategory) innerList.get(1)).
getName());
                } else if (firstElem == UndoClass.UNDO_TYPE.REMOVE_CATEGORY) {
                    logList.add(firstElem + " " +  ((PrincipalCategory) innerList.get(1)).
getName());
                } else if (firstElem == UndoClass.UNDO_TYPE.UPDATE_PERMISSIONS) {
                    logList.add(firstElem + " " +  ((PrincipalCategory) innerList.get(1)).
getName());
                } else if (firstElem == UndoClass.UNDO_TYPE.UPDATE_HIERARCHY) {
                    logList.add(firstElem + " " +  ((PrincipalCategory) innerList.get(1)).
getName());
                }
            }
            HelperClass.showListDialog(frame, "Admin log", logList);
        } catch(Exception err){
            err.printStackTrace();
            JOptionPane.showMessageDialog(frame, JOptionPane.ERROR_MESSAGE);
        }
    }
```

```
    public void deleteCategory() {
        String text = JOptionPane.showInputDialog("Enter category name");
        if (text != null && !text.isBlank()) {
            PrincipalCategory toDelete = HelperClass.
getCategoryByName(assignCategories.getPrincipalCategories(), text);
            if (toDelete != null) {
                List<PrincipalCategory> superiorCategories = new ArrayList<>();
                for(PrincipalCategory pc: assignCategories.getPrincipalCategories()){
                    if(!pc.equals(toDelete) && HelperClass.findAllJuniorCategories(pc).
contains(toDelete)){
                        superiorCategories.add(pc);
                    }
                }
                undoClass.addRemoveCategory(toDelete, superiorCategories);
                assignCategories.removePrincipalCategory(toDelete);
                for(PrincipalCategory pc: assignCategories.getPrincipalCategories()){
                    pc.getJuniorCategories().remove(toDelete);
                }
                updateGraph();
            }
        } //helloageag
    }

    public void updateHierarchy() {
        String text = JOptionPane.showInputDialog("Enter category name");
        if (text != null && !text.isBlank()) {
            PrincipalCategory principalCategory = HelperClass.
getCategoryByName(assignCategories.getPrincipalCategories(), text);
            if (principalCategory != null) {
                JDialog dialog = new JDialog();
                JPanel mainPanel = new JPanel(new BorderLayout());
                mainPanel.setPreferredSize(new Dimension(500, 700));
                dialog.add(mainPanel);
                JPanel gridPanel = new JPanel(new GridLayout(assignCategories.
getPrincipalCategories().size(), 1));
                Map<PrincipalCategory, JCheckBox> selectedCatMap = new HashMap<>();
                for(PrincipalCategory pc: assignCategories.getPrincipalCategories()){
                    if(!pc.equals(principalCategory)) {
                        boolean isJuniorOf = false;
                        String deleteThisString = "";
                        List<PrincipalCategory> allJunior = HelperClass.
findAllJuniorCategories(pc);
                        if(allJunior.contains(principalCategory)){
                            isJuniorOf = true;
                            deleteThisString = pc.getName();
                        }
                        if(!isJuniorOf) {
                            JCheckBox curr = new JCheckBox(pc.getName());
```

```
                    selectedCatMap.put(pc, curr);
                    gridPanel.add(curr);
                } else {
                    JLabel textLabel = new JLabel("<html>senior<br>category<br>"
+deleteThisString + "</html>");
                    gridPanel.add(textLabel);
                }
            }
        }
        mainPanel.add(gridPanel, BorderLayout.CENTER);
        JButton submitButton = new JButton("Submit");
        mainPanel.add(submitButton, BorderLayout.NORTH);
        submitButton.addActionListener(e -> {
            List<PrincipalCategory> oldJrCategories = new
ArrayList<>(principalCategory.getJuniorCategories());
            principalCategory.getJuniorCategories().clear();
            for(PrincipalCategory pc: assignCategories.getPrincipalCategories()){
                JCheckBox currBox = selectedCatMap.get(pc);
                if(currBox != null){
                    if(currBox.isSelected()){
                        principalCategory.addJuniorCategory(pc);
                    } else {
                        principalCategory.removeJuniorCategory(pc);
                    }
                }
            }
            undoClass.addUpdateHierarchy(principalCategory, oldJrCategories);
            updateGraph();
            dialog.dispose();
        });

        dialog.pack();
        dialog.setLocationRelativeTo(frame);
        dialog.setModal(true);
        dialog.setVisible(true);
    }
  }
}

  public void configurePermissions() {
    String text = JOptionPane.showInputDialog("Enter category name");
    if (text != null && !text.isBlank()) {
        PrincipalCategory principalCategory = HelperClass.
getCategoryByName(assignCategories.getPrincipalCategories(), text);
        if (principalCategory != null) {
            PrincipalRulePanel prp = new PrincipalRulePanel(assignCategories);
            Set<ResourceAction> juniorActions = new HashSet<>();
            for(PrincipalCategory jr: HelperClass.
findAllJuniorCategories(principalCategory)){
```

```
                if(!jr.equals(principalCategory)) {
                    juniorActions.addAll(jr.getActions());
                }
            }
            for (ResourceAction a : juniorActions) {
                JCheckBox curr = prp.getActionJCheckBoxMap().get(a);
                if (curr != null) {
                    curr.setText("Inherited permission");
                }
            }
            for(ResourceAction a :  principalCategory.getActions()){
                JCheckBox curr = prp.getActionJCheckBoxMap().get(a);
                if(curr != null){
                    if(juniorActions.contains(a)){
                        curr.setText("<html>Inherited permission<br>and explicitly assigned</
html>");
                    } else {
                        curr.setText("Explicitly assigned");
                    }
                }
            }
            JDialog dialog = new JDialog();
            JPanel tempPanel = new JPanel(new BorderLayout());
            JButton submitButton = new JButton("Submit");
            dialog.add(tempPanel);
            prp.getPanel().setPreferredSize(new Dimension(400, 400));
            tempPanel.add(prp.getPanel(), BorderLayout.CENTER);
            tempPanel.add(submitButton, BorderLayout.NORTH);
            submitButton.addActionListener(e -> {
                List<ResourceAction> oldActionList = new ArrayList<>(principalCategory.
getActions());
                principalCategory.getActions().clear();
                for (ResourceAction a : assignCategories.getResourceActions()) {
                    JCheckBox currCheckBox = prp.getActionJCheckBoxMap().get(a);
                    if (currCheckBox != null) {
                        if (currCheckBox.isSelected()) {
                            principalCategory.addAction(a);
                        } else {
                            principalCategory.removeAction(a);
                        }
                    }
                }
                undoClass.addUpdatePermissions(principalCategory, oldActionList);
                updateGraph();
                dialog.dispose();
            });

            dialog.pack();
            dialog.setLocationRelativeTo(frame);
```

```
            dialog.setModal(true);
            dialog.setResizable(true);
            dialog.setVisible(true);
        } else {
            JOptionPane.showMessageDialog(frame, "not found");
        }
    }
}

public void addResource() {
    JDialog dialog = new JDialog();
    JPanel mainPanel = new JPanel(new BorderLayout());
    ResourceRulePanel rrp = new ResourceRulePanel();
    mainPanel.add(rrp.getPanel(), BorderLayout.CENTER);
    dialog.add(mainPanel);
    rrp.getPanel().setPreferredSize(new Dimension(500, 500));

    JButton submitButton = new JButton("Submit");
    mainPanel.add(submitButton, BorderLayout.NORTH);
    submitButton.addActionListener(e -> {
        boolean error = false;
        List<String> uniqueActionNames = new ArrayList<>();
        List<ResourceAction> resourceActions = new ArrayList<>();
        Resource resource = null;
        JTextField resourceName = rrp.getResourceNameField();
        if (resourceName.getText() != null && !resourceName.getText().isBlank() &&
HelperClass.getResourceByName(assignCategories.getResources(), resourceName.
getText()) == null) {
            String fieldText = resourceName.getText().toLowerCase().strip();
            resource = new Resource(fieldText);

            // add to assigncat
            for (int i = 0; i < rrp.getTextFieldArray().length; i++) {
                JTextField currField = rrp.getTextFieldArray()[i];
                if (currField.getText() != null && !currField.getText().isBlank()) {
                    if (!uniqueActionNames.contains(currField.getText().toLowerCase().
strip())) {
                        String currActionName = currField.getText().toLowerCase().strip();
                        uniqueActionNames.add(currField.getText().toLowerCase().strip());
                        resourceActions.add(new ResourceAction(currActionName, resource)
);
                    } else {
                        error = true;
                    }
                }
            }
        } else {
            error = true;
        }
```

```
            if (!error) {
                undoClass.addAddResource(resource);
                assignCategories.addResource(resource);
                assignCategories.getResourceActions().addAll(resourceActions);
                updateGraph();
                dialog.dispose();
            } else {
                JOptionPane.showMessageDialog(frame, "error");
            }
        });

        dialog.pack();
        dialog.setLocationRelativeTo(frame);
        dialog.setModal(true);
        dialog.setResizable(true);
        dialog.setVisible(true);
    }

    public void removeResource() {
        String text = JOptionPane.showInputDialog("Enter resource name:");
        if (text != null && !text.isBlank()) {
            Resource toRemove = HelperClass.getResourceByName(assignCategories.
getResources(), text);
            if (toRemove != null) {
                List<ResourceAction> resourceActions = HelperClass.
getAllResourceActions(assignCategories.getResourceActions(), toRemove);
                Map<ResourceAction, List<PrincipalCategory>> actionCategoryNames =
new HashMap<>();
                for(PrincipalCategory pc : assignCategories.getPrincipalCategories()){
                    List<ResourceAction> categoryActions = pc.getActions();
                    for(ResourceAction a : resourceActions){
                        if(categoryActions.contains(a)){
                            List<PrincipalCategory> categoryList;
                            if(actionCategoryNames.containsKey(a)){
                                categoryList = actionCategoryNames.get(a);
                            } else {
                                categoryList = new ArrayList<>();
                            }
                            categoryList.add(pc);
                            actionCategoryNames.put(a, categoryList);
                        }
                    }
                }
                undoClass.addRemoveResource(toRemove, actionCategoryNames);
                ADMIN_LOG.add("resource " + toRemove.getName() + " deleted "  +
HelperClass.getCurrentTime());

                assignCategories.getPrincipalCategories().forEach(principalCategory ->
principalCategory.getActions().removeAll(resourceActions));
```

```
                assignCategories.getResourceActions().removeAll(resourceActions);
                assignCategories.getResources().remove(toRemove);
                updateGraph();
            } else {
                JOptionPane.showMessageDialog(frame, "Resource not found!", "Error",
JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    public void addPrincipal() {
        String text = JOptionPane.showInputDialog("Enter principal name:");
        if (text != null && !text.isBlank()) {
            Principal currPrincipal = HelperClass.getPrincipalByName(assignCategories.
getPrincipals(), text);
            boolean IS_NEW_PRINCIPAL = (currPrincipal == null);
            JDialog dialog = new JDialog();
            JPanel newPrincipalPanel = new JPanel(new BorderLayout());
            newPrincipalPanel.setPreferredSize(new Dimension(1000, 700));
            JButton submitButton = new JButton("Submit");
            newPrincipalPanel.add(submitButton, BorderLayout.SOUTH);

            RegistrationForm currForm = new RegistrationForm();
            JPanel firstPanel = currForm.getRightPanels()[0][0];
            for(Component c: firstPanel.getComponents()){
                if(c instanceof JTextField textField){
                    textField.setText(text.toLowerCase().strip()); // set first text field to input
text for convenience
                    if(!IS_NEW_PRINCIPAL){
                        textField.setEnabled(false);
                    }
                }
            }
            JPanel regPanel = currForm.getPanel();
            newPrincipalPanel.add(regPanel, BorderLayout.CENTER);
            if(currPrincipal != null){
                newPrincipalPanel.add(new JLabel("Editing " + currPrincipal.getName() ),
BorderLayout.NORTH);
            }

            dialog.add(newPrincipalPanel);

            submitButton.addActionListener(e1 -> {
                boolean isValid = validate(currForm, IS_NEW_PRINCIPAL);
                if (isValid) {
                    boolean error = false;
                    List<StringAttribute> stringAttributeList = new ArrayList<>();
                    List<IntegerAttribute> integerAttributeList = new ArrayList<>();
                    List<DateAttribute> dateAttributeList = new ArrayList<>();
```

```
                for (int i = 0; i < currForm.getNUM_OF_ROWS(); i++) {
                    JPanel leftPanel = currForm.getLeftPanels()[i];
                    JCheckBox checkBox = HelperClass.
getCheckBoxFromComponents(leftPanel.getComponents());
                    JComboBox<?> comboBox = HelperClass.
getComboBoxFromComponents(leftPanel.getComponents());
                    JTextField textField = HelperClass.
getTextFieldFromComponents(leftPanel.getComponents());
                    if (checkBox != null && comboBox != null && textField != null) {
                        int selectedIndex = comboBox.getSelectedIndex();
                        JPanel rightPanel = currForm.getRightPanels()[i][selectedIndex];
                        if (checkBox.isSelected()) {
                            String attributeName = textField.getText().toLowerCase().strip();
                            String attributeValue = HelperClass.
getRightPanelAttributeValue(rightPanel);
                            error = HelperClass.addCorrespondingAttribute(stringAttributeList,
integerAttributeList, dateAttributeList,
                                    selectedIndex, attributeName, attributeValue);
                        }
                    }
                }

                String name = stringAttributeList.get(0).getValue();
                Principal principal = null;
                if(currPrincipal != null){
                    principal = currPrincipal;
                    principal.setName(name);
                } else {
                    principal = new Principal(name);
                }

                if (!error) {
                    ADMIN_LOG.add("Principal " + principal.getName() + " created "  +
HelperClass.getCurrentTime());
                    Principal oldPrincipal = new Principal(principal);
                    principal.setStringAttributeList(stringAttributeList);
                    principal.setIntegerAttributeList(integerAttributeList);
                    principal.setDateAttributeList(dateAttributeList);
                    if(IS_NEW_PRINCIPAL) {
                        undoClass.addCreatePrincipal(principal);
                        assignCategories.addPrincipal(principal);
                    } else {
                        undoClass.addUpdatePrincipal(oldPrincipal, principal);
                    }
                    assignCategories.evaluatePrincipalCategories();
                    updateGraph();
                    dialog.dispose();
                } else {
                    JOptionPane.showMessageDialog(frame, "Error parsing");
```

```
                }
            } else {
                JOptionPane.showMessageDialog(frame, "Invalid");
            }
        });

        dialog.pack();
        dialog.setLocationRelativeTo(frame);
        dialog.setModal(true);
        dialog.setResizable(true);
        dialog.setVisible(true);
    }
  }

  public void removePrincipal() {
      String text = JOptionPane.showInputDialog("Enter principal name:");
      if (text != null && !text.isBlank()) {
          Principal toRemove = HelperClass.getPrincipalByName(assignCategories.
getPrincipals(), text);
          if (toRemove != null) {
              undoClass.addRemovePrincipal(toRemove);
              ADMIN_LOG.add("Principal " + toRemove.getName() + " deleted "  +
HelperClass.getCurrentTime());

              assignCategories.removePrincipal(toRemove);
              for(PrincipalCategory pc: assignCategories.getPrincipalCategories()){
                  pc.getPrincipals().remove(toRemove);
              }
//              assignCategories.evaluatePrincipalCategories();
              updateGraph();
          } else {
              JOptionPane.showMessageDialog(frame, "Principal not found!", "Error",
JOptionPane.ERROR_MESSAGE);
          }
      }
  }

  public void updateCategoryRules() {
      String text = JOptionPane.showInputDialog("Enter principal category name:");
      if (text != null && !text.isBlank()) {
          PrincipalCategory currCat = HelperClass.
getCategoryByName(assignCategories.getPrincipalCategories(), text);
          boolean IS_NEW_CATEGORY = (currCat == null);

          List<String> principalsBefore = new ArrayList<>();
          if(currCat != null){
              for(Principal p : currCat.getPrincipals()){
                  principalsBefore.add(p.getName());
              }
```

```
        }
        List<String> principalsAfter = new ArrayList<>();

        JDialog dialog = new JDialog();
        JPanel tempPanel = new JPanel(new BorderLayout());
        tempPanel.setPreferredSize(new Dimension(1000, 700));

        JButton submitButton = new JButton("Submit");
        tempPanel.add(submitButton, BorderLayout.SOUTH);
        List<StringAttribute> bl1 = HelperClass.
getBlankStringAttributeList(assignCategories.getPrincipals());
        List<IntegerAttribute> bl2 = HelperClass.
getBlankIntegerAttributeList(assignCategories.getPrincipals());
        List<DateAttribute> bl3 = HelperClass.
getBlankDateAttributeList(assignCategories.getPrincipals());

        CategoryRulePanel currRulePanel = new CategoryRulePanel(bl1, bl2, bl3);
        JPanel cPanel = currRulePanel.getPanel();
        tempPanel.add(cPanel, BorderLayout.CENTER);

        String updatedName = (currCat != null) ? currCat.getName() : text.
toLowerCase().strip();
        currRulePanel.getCategoryNameField().setText(updatedName);
        if(!IS_NEW_CATEGORY) {
            currRulePanel.getCategoryNameField().setEnabled(false);
        }

        String infoLabelText = (currCat != null) ? "Modifying category " + currCat.
getName() : "Creating new category";
        tempPanel.add(new JLabel(infoLabelText), BorderLayout.NORTH);

        submitButton.addActionListener(al -> {
            PrincipalCategory currModifiedCategory = currCat;

            boolean error = false;
            PrincipalCategory oldCategory = (!IS_NEW_CATEGORY) ? new
PrincipalCategory(currModifiedCategory) : null;
            if (currRulePanel.getCategoryNameField().getText() != null) {
                String proposedName = currRulePanel.getCategoryNameField().getText().
strip().toLowerCase();
                if (!proposedName.isBlank()) {
                    if ((currModifiedCategory != null && proposedName.strip().
equalsIgnoreCase(currModifiedCategory.getName().strip())) ||
                            HelperClass.getCategoryByName(assignCategories.
getPrincipalCategories(), proposedName) == null) {
                        if (IS_NEW_CATEGORY) {
                            currModifiedCategory = new PrincipalCategory(proposedName);
                        } else {
                            currModifiedCategory.setName(proposedName);
```

```
                    }
                    currModifiedCategory.getStringRules().clear();
                    currModifiedCategory.getIntegerRules().clear();
                    currModifiedCategory.getDateRules().clear();
                    for (int i = 0; i < currRulePanel.getPanelArray().length; i++) {
                        JCheckBox currentRowBox = currRulePanel.getIsEnabledMap().
get(i);
                        if (currentRowBox != null && currentRowBox.isSelected()) {
                            if (i < bl1.size()) {
                                StringAttribute currAttribute = bl1.get(i);
                                error = error || HelperClass.evaluateStringAttr(currAttribute,
currRulePanel, currModifiedCategory);
                            } else {
                                int fixedIndex = i - bl1.size();
                                if (fixedIndex < bl2.size()) {
                                    IntegerAttribute currAttribute = bl2.get(fixedIndex);
                                    error = error || HelperClass.evaluateIntegerAttr(currAttribute,
 currRulePanel, currModifiedCategory);
                                } else {
                                    int finalFixedIndex = i - bl1.size() - bl2.size();
                                    if(finalFixedIndex < bl3.size()) {
                                        DateAttribute currAttribute = bl3.get(finalFixedIndex);
                                        error = error || HelperClass.
evaluateDateAttr(currAttribute, currRulePanel, currModifiedCategory);
                                    }
                                }
                            }
                        }
                    }
                    if (!error) {

                        currModifiedCategory.getPrincipals().clear();
                        if(IS_NEW_CATEGORY) {
                            assignCategories.addPrincipalCategory(currModifiedCategory);
                            undoClass.addCreateCategory(currModifiedCategory);
                        } else {
                            undoClass.addUpdateCategory(oldCategory,
currModifiedCategory);
                        }
                        assignCategories.evaluatePrincipalCategories();
                        for(Principal prin : currModifiedCategory.getPrincipals()){
                            principalsAfter.add(prin.getName());
                        }

                        String currentLogAction = (IS_NEW_CATEGORY) ? "Created new
category " + currModifiedCategory.getName() : " Updated category " + proposedName;
                        ADMIN_LOG.add(currentLogAction + " " + HelperClass.
getCurrentTime());
                        HelperClass.showDoubleDialog(frame, "Category " +
```

```
currModifiedCategory.getName(), principalsBefore, principalsAfter, "Principals before", "
Principals after");

                        updateGraph();
                        dialog.dispose();
                    } else {
                        JOptionPane.showMessageDialog(frame, "Error parsing");
                    }
                } else {
                    JOptionPane.showMessageDialog(frame, "attempting to overwrite
existing.. not possible");
                }
            }
        }
        });
        dialog.add(tempPanel);
        dialog.pack();
        dialog.setLocationRelativeTo(frame);
        dialog.setModal(true);
        dialog.setResizable(true);
        dialog.setVisible(true);
    }
  }

   public boolean isRedundant_PC_Edge(Principal principal, PrincipalCategory
category, List<Object> graphNodes, boolean[][] adjacencyMatrix) {
      List<PrincipalCategory> superiorCategories = new ArrayList<>();
      for(PrincipalCategory pc: assignCategories.getPrincipalCategories()){
         if(!pc.equals(category) && HelperClass.findAllJuniorCategories(pc).
contains(category)){
            superiorCategories.add(pc);
         }
      }
      int principalIndex = HelperClass.getPrincipalIndex(graphNodes, principal);
      for (PrincipalCategory superiorCategory : superiorCategories) {
         int superiorCategoryIndex = HelperClass.getCategoryIndex(graphNodes,
superiorCategory);
         if (!superiorCategory.equals(category) &&
adjacencyMatrix[principalIndex][superiorCategoryIndex]) {
            return true;
         }
      }
      return false;
   }

   public boolean isRedundant_CC_Edge(PrincipalCategory category1,
PrincipalCategory category2, List<Object> graphNodes, boolean[][] adjacencyMatrix) {
      int category1Index = HelperClass.getCategoryIndex(graphNodes, category1);
```

```
        List<PrincipalCategory> superiorCategories = new ArrayList<>();
        for(PrincipalCategory pc: assignCategories.getPrincipalCategories()){
            if(!pc.equals(category2) && HelperClass.findAllJuniorCategories(pc).
contains(category2)){
                superiorCategories.add(pc);
            }
        }
        for (PrincipalCategory superiorCategory : superiorCategories) {
            int superiorCategoryIndex = HelperClass.getCategoryIndex(graphNodes,
superiorCategory);
            if (!superiorCategory.equals(category2) &&
adjacencyMatrix[category1Index][superiorCategoryIndex]) {
                return true;
            }
        }
        return false;
    }

    public boolean isRedundant_CA_Edge(PrincipalCategory category, ResourceAction
action, List<Object> graphNodes, boolean[][] adjacencyMatrix) {
        int actionIndex = HelperClass.getActionIndex(graphNodes, action);

        List<PrincipalCategory> juniorCategories = HelperClass.
findAllJuniorCategories(category);
        for (PrincipalCategory juniorCategory : juniorCategories) {
            int juniorCategoryIndex = HelperClass.getCategoryIndex(graphNodes,
juniorCategory);
            if (!juniorCategory.equals(category) &&
adjacencyMatrix[juniorCategoryIndex][actionIndex]) {
                return true;
            }
        }
        return false;
    }

    public void updateGraph() {
        SwingUtilities.invokeLater(() -> {
            // Clear the existing graph
            graph.clear();
            List<Object> graphNodes = new ArrayList<>();

            for (Principal p : assignCategories.getPrincipals()) {
                addNode("Principal: " + p.getName(), "grey");
                graphNodes.add(p);
            }
            for (PrincipalCategory p : assignCategories.getPrincipalCategories()) {
                addNode("Principal category: " + p.getName(), "green");
                graphNodes.add(p);
            }
```

```
        for (ResourceAction a : assignCategories.getResourceActions()) {
            addNode("Action: " + a.getName() + " " + a.getResource().getName(), "pink");
            graphNodes.add(a);
        }
        for(Resource r: assignCategories.getResources()){
            addNode("Resource: " + r.getName(), "red");
        }

        // Create the adjacency matrix
        boolean[][] adjacencyMatrix = new boolean[graphNodes.size()][graphNodes.
size()];
        for(int i = 0; i < adjacencyMatrix.length; i++){
            for (int j = 0; j < adjacencyMatrix[i].length; j++) {
                Object firstNode = graphNodes.get(i);
                Object secondNode = graphNodes.get(j);

                // Check if there is an edge between the two nodes
                boolean hasEdge = false;
                if (firstNode instanceof Principal && secondNode instanceof
PrincipalCategory) {
                    hasEdge = HelperClass.getPrincipalByName(((PrincipalCategory)
secondNode).getPrincipals(), ((Principal) firstNode).getName()) != null;
                } else if (firstNode instanceof PrincipalCategory && secondNode
instanceof Principal) {
                    hasEdge = HelperClass.getPrincipalByName(((PrincipalCategory)
firstNode).getPrincipals(), ((Principal) secondNode).getName()) != null;
                } else if (firstNode instanceof PrincipalCategory && secondNode
instanceof PrincipalCategory) {
                    hasEdge = HelperClass.getCategoryByName(((PrincipalCategory)
firstNode).getJuniorCategories(), ((PrincipalCategory) secondNode).getName()) != null;
                } else if (firstNode instanceof PrincipalCategory && secondNode
instanceof ResourceAction) {
                    hasEdge = HelperClass.getActionByName(((PrincipalCategory)
firstNode).getActions(), ((ResourceAction) secondNode).getName(), ((ResourceAction)
secondNode).getResource().getName()) != null;
                }
                // Store the relationship between the two nodes in the matrix
                adjacencyMatrix[i][j] = hasEdge;
            }
        }


        for (PrincipalCategory p : assignCategories.getPrincipalCategories()) {
            String id1 = "Principal category: " + p.getName();
            for (Principal pr : p.getPrincipals()) {
                String id2 = "Principal: " + pr.getName();
                String edgeId = id1 + id2;

                // Check if the edge is redundant
```

```
                if (!isRedundant_PC_Edge(pr, p, graphNodes, adjacencyMatrix) && graph.
getEdge(edgeId) == null) {
                    graph.addEdge(edgeId, id1, id2);
                }
            }
        }

        for (ResourceAction a : assignCategories.getResourceActions()) {
            String id1 = "Action: " + a.getName() + " " + a.getResource().getName();
            String id2 = "Resource: " + a.getResource().getName();
            String edgeId = id1 + id2;
            if (graph.getEdge(edgeId) == null) {
                graph.addEdge(edgeId, id1, id2);
            }
        }

        for (PrincipalCategory p : assignCategories.getPrincipalCategories()) {
            String id1 = "Principal category: " + p.getName();
            for (ResourceAction a : p.getActions()) {
                String id2 = "Action: " + a.getName() + " " + a.getResource().getName();
                String edgeId = id1 + id2;

                // Check if the edge is redundant
                if (!isRedundant_CA_Edge(p, a, graphNodes, adjacencyMatrix) && graph.
getEdge(edgeId) == null) {
                    graph.addEdge(edgeId, id1, id2);
                }
            }
        }

        for (PrincipalCategory pc : assignCategories.getPrincipalCategories()) {
            String id1 = "Principal category: " + pc.getName();
            for (PrincipalCategory juniorPC : pc.getJuniorCategories()) {
                String id2 = "Principal category: " + juniorPC.getName();
                String edgeId = id1 + id2;

                // Check if the edge is redundant
                if (!isRedundant_CC_Edge(pc, juniorPC, graphNodes, adjacencyMatrix)
&& graph.getEdge(edgeId) == null) {
                    Edge edge = graph.addEdge(edgeId, id1, id2, true);
                    edge.setAttribute("ui.style", "arrow-shape: arrow; arrow-size: 10px;");
                }
            }
        }
    });
    centerGraph();
}

public boolean validate(RegistrationForm currForm, boolean IS_NEW_PRINCIPAL) {
```

```
        boolean isValid = true;
        String principalName = "";
        List<String> attributeNameList = new ArrayList<>();
        for (int i = 0; i < currForm.getNUM_OF_ROWS(); i++) {
            JPanel leftPanel = currForm.getLeftPanels()[i];
            JCheckBox checkBox = HelperClass.getCheckBoxFromComponents(leftPanel.
getComponents());
            JComboBox<?> comboBox = HelperClass.
getComboBoxFromComponents(leftPanel.getComponents());
            JTextField textField = HelperClass.getTextFieldFromComponents(leftPanel.
getComponents());
            if (checkBox != null && comboBox != null && textField != null) {
                int selectedIndex = comboBox.getSelectedIndex();
                JPanel rightPanel = currForm.getRightPanels()[i][selectedIndex];
                if (checkBox.isSelected()) {
                    if (textField.getText() != null && !textField.getText().isBlank() &&
!(attributeNameList.contains(textField.getText().toLowerCase().strip()))) {
                        String attributeName = textField.getText().toLowerCase().strip();
                        for (Component c : rightPanel.getComponents()) {
                            if (c instanceof JTextField) {
                                if (((JTextField) c).getText() == null || ((JTextField) c).getText().
isBlank()) {
                                    isValid = false;
                                } else if (i == 0) {
                                    principalName = ((JTextField) c).getText().toLowerCase().strip();
                                }
                            }
                        }
                        attributeNameList.add(attributeName);
                    } else {
                        isValid = false;
                    }
                }
            } else {
                isValid = false;
            }
        }
        if(IS_NEW_PRINCIPAL && HelperClass.getPrincipalByName(assignCategories.
getPrincipals(), principalName) != null){
            isValid = false; // principal exists
        }
        return isValid;
    }

    private void addNode(String nodeName, String color) {
        if (graph.getNode(nodeName) == null) {
            Node n = graph.addNode(nodeName);
            n.setAttribute("ui.style", "shape: circle;");
            n.setAttribute("ui.style", "size: 40px,25px;");
```

```
            n.setAttribute("ui.style", "fill-color: " + color + ";");
            n.setAttribute("ui.style", "text-size: 16px;");
            String nodeLabel = nodeName;
            if(nodeLabel.startsWith("Principal category")){
                nodeLabel = nodeLabel.replace("Principal category: ", "");
            } else if(nodeLabel.startsWith("Resource")){
                nodeLabel = nodeLabel.replace("Resource: ", "");
            } else if(nodeLabel.startsWith("Principal")){
                nodeLabel = nodeLabel.replace("Principal: ", "");
            } else if(nodeLabel.startsWith("Action")){
                String myString = nodeLabel.replace("Action: ", "");
                String[] words = myString.split("\\s"); // split the string by whitespace
                if(words.length >= 1 ) {
                    nodeLabel = words[0];
                }
            }
            n.setAttribute("ui.label", nodeLabel);
            n.setAttribute("layout.weight", 8.0);
        }
    }
}
```

```
package guipanels;

import categories.ResourceAction;
import categories.*;
import categoryrules.*;
import com.toedter.calendar.JCalendar;
import principal_resource_attributes.*;

import javax.swing.*;
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.text.SimpleDateFormat;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;
import java.util.*;
import java.util.List;

public class HelperClass {

    public static boolean isFrameOpen = false;

    public static boolean addCorrespondingAttribute(List<StringAttribute>
stringAttributeList, List<IntegerAttribute> integerAttributeList,
                                                    List<DateAttribute> dateAttributeList, int selectedIndex,
 String attributeName, String attributeValue){
        boolean errorParsing = false;
        if(selectedIndex == 0){
            stringAttributeList.add(new StringAttribute(attributeName, attributeValue));
        } else if(selectedIndex == 1){
            try{
                integerAttributeList.add(new IntegerAttribute(attributeName, Integer.
parseInt(attributeValue)));
            } catch(Exception exception){
                exception.printStackTrace();
                errorParsing = true;
            }
        } else if(selectedIndex == 2){
            try{
                SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
                dateAttributeList.add(new DateAttribute(attributeName, dateFormat.
parse(attributeValue)));
            } catch(Exception exception){
                exception.printStackTrace();
                errorParsing = true;
            }
        }
        return errorParsing;
    }
```

```
    public static int getPrincipalIndex(List<Object> graphNodes, Principal principal){
        for (int i = 0; i < graphNodes.size(); i++) {
            if(graphNodes.get(i) instanceof Principal curr){
                if(curr.equals(principal)){
                    return i;
                }
            }
        }
        return -1;
    }

    public static int getCategoryIndex(List<Object> graphNodes, PrincipalCategory
principalCategory){
        for (int i = 0; i < graphNodes.size(); i++) {
            if(graphNodes.get(i) instanceof PrincipalCategory curr){
                if(curr.equals(principalCategory)){
                    return i;
                }
            }
        }
        return -1;
    }

    public static int getActionIndex(List<Object> graphNodes, ResourceAction action){
        for (int i = 0; i < graphNodes.size(); i++) {
            if(graphNodes.get(i) instanceof ResourceAction){
                ResourceAction curr = (ResourceAction) graphNodes.get(i);
                if(action.equals(curr)){
                    return i;
                }
            }
        }
        return -1;
    }

    public static ResourceAction getActionByName(List<ResourceAction> inList, String
actionName, String resourceName){
        ResourceAction toReturn = null;
        String formattedName = actionName.toLowerCase().strip() + "|" + resourceName.
toLowerCase().strip();
        for(ResourceAction a: inList){
            String currName = a.getName().toLowerCase().strip() + "|" + a.getResource().
getName().toLowerCase().strip();
            if(currName.equalsIgnoreCase(formattedName)){
                toReturn = a;
            }
        }
        return toReturn;
```

```
    }

    public static PrincipalCategory getCategoryByName(List<PrincipalCategory> inList,
String text){
        PrincipalCategory toReturn = null;
        for (PrincipalCategory c : inList) {
            if (c.getName().strip().equalsIgnoreCase(text.strip())) {
                toReturn = c;
            }
        }
        return toReturn;
    }

    public static Resource getResourceByName(List<Resource> inList, String text){
        Resource toReturn = null;
        for (Resource r : inList) {
            if (r.getName().strip().equalsIgnoreCase(text.strip())) {
                toReturn = r;
            }
        }
        return toReturn;
    }

    public static Principal getPrincipalByName(List<Principal> inList, String text){
        Principal toReturn = null;
        for (Principal p : inList) {
            if (p.getName().strip().equalsIgnoreCase(text.strip())) {
                toReturn = p;
            }
        }
        return toReturn;
    }

    public static int[] findGridSize(int n) {
        if (n <= 0) {
            return new int[] { 0, 0 };
        }
        int rows = (int) Math.ceil(Math.sqrt(n));
        int cols = (int) Math.ceil((double) n / rows);
        return new int[] { rows, cols };
    }

    public static List<StringAttribute> getBlankStringAttributeList(List<Principal>
principals) {
        List<StringAttribute> blankPrincipalAttributeList = new ArrayList<>();
        for (Principal p : principals) {
            for (StringAttribute a : p.getStringAttributeList()) {
                boolean exists = false;
                for (StringAttribute innerA : blankPrincipalAttributeList) {
```

```
                if (innerA.getName().strip().equalsIgnoreCase(a.getName().strip())) {
                    exists = true;
                }
            }
            if (!exists) {
                blankPrincipalAttributeList.add(new StringAttribute(a.getName(), ""));
            }
        }
    }
    return blankPrincipalAttributeList;
}
public static List<IntegerAttribute> getBlankIntegerAttributeList(List<Principal>
principals) {
    List<IntegerAttribute> blankPrincipalAttributeList = new ArrayList<>();
    for (Principal p : principals) {
        for (IntegerAttribute a : p.getIntegerAttributeList()) {
            boolean exists = false;
            for (IntegerAttribute innerA : blankPrincipalAttributeList) {
                if (innerA.getName().strip().equalsIgnoreCase(a.getName().strip())) {
                    exists = true;
                }
            }
            if (!exists) {
                blankPrincipalAttributeList.add(new IntegerAttribute(a.getName(), 0));
            }
        }
    }
    return blankPrincipalAttributeList;
}
public static List<DateAttribute> getBlankDateAttributeList(List<Principal> principals)
{
    List<DateAttribute> blankPrincipalAttributeList = new ArrayList<>();
    for (Principal p : principals) {
        for (DateAttribute a : p.getDateAttributeList()) {
            boolean exists = false;
            for (DateAttribute innerA : blankPrincipalAttributeList) {
                if (innerA.getName().strip().equalsIgnoreCase(a.getName().strip())) {
                    exists = true;
                }
            }
            if (!exists) {
                blankPrincipalAttributeList.add(new DateAttribute(a.getName(), new Date())
);
            }
        }
    }
    return blankPrincipalAttributeList;
}
```

```
    public static String getRightPanelAttributeValue(JPanel rightPanel){
        for(Component c: rightPanel.getComponents()){
            if(c instanceof JTextField){
                return ((JTextField) c).getText().toLowerCase().strip();
            }
        }
        return "";
    }

    // evaluate user input from CategoryRulePanel and create corresponding rule object
that will be added to the PrincipalCategory object.
    // If an error occurs during any of these processes (e.g. invalid input format), return
false
    public static boolean evaluateStringAttr(StringAttribute currAttribute,
CategoryRulePanel selectedRulePanel, PrincipalCategory currPrincipalCategory){
        boolean error = false;
        JTextField currField = selectedRulePanel.getStringAttributeJTextFieldMap().
get(currAttribute);
            if (currField != null && currField.getText() != null && !currField.getText().isBlank()
){
                String[] values = currField.getText().split(",");
                for (int tempI = 0; tempI < values.length; tempI++) {
                    values[tempI] = values[tempI].toLowerCase().strip();
                }
                Set<String> valuesSet = new HashSet<>();
                Collections.addAll(valuesSet, values);
                List<String> valuesList = new ArrayList<>(valuesSet);
                StringRule tempStrRule = new StringRule(currAttribute, valuesList);
                currPrincipalCategory.getStringRules().add(tempStrRule);
            } else {
                error = true;
            }
            return error;
    }
    public static boolean evaluateIntegerAttr(IntegerAttribute intAttr, CategoryRulePanel
selectedRulePanel, PrincipalCategory currPrincipalCategory){
        boolean error = false;
            int lowerBound = Integer.MAX_VALUE;
            int upperBound = 0;
            JPanel currRangePanel = selectedRulePanel.getIntegerRangeValueMap().
get(intAttr);
            for (Component c : currRangePanel.getComponents()) {
                if (c instanceof JTextField) {
                    if (((JTextField) c).getText() != null && ! (((JTextField) c).getText().isBlank())
 ){
                        try {
                            int currVal = Integer.parseInt(((JTextField) c).getText().strip());
                            if (lowerBound == Integer.MAX_VALUE) {
                                lowerBound = currVal;
```

```
                    } else {
                        upperBound = currVal;
                    }
                } catch (Exception exception) {
                    exception.printStackTrace();
                    error = true;
                }
            } else {
                error = true;
            }
        }
    }
    if(lowerBound > upperBound){
        error = true;
    } else if(!error) {
        IntegerRule tempIntRule = new IntegerRule(intAttr, lowerBound, upperBound)
;
        currPrincipalCategory.getIntegerRules().add(tempIntRule);
    }
    return error;
  }
  public static boolean evaluateDateAttr(DateAttribute dateAttr, CategoryRulePanel
selectedRulePanel, PrincipalCategory currPrincipalCategory){
    boolean error = false;
    Date lowerBound = null;
    Date upperBound = null;
    JPanel currBetweenPanel = selectedRulePanel.getDateBetweenValueMap().
get(dateAttr);
    for(Component c: currBetweenPanel.getComponents()) {
        if (c instanceof JPanel) {
            for(Component innerC: ((JPanel) c).getComponents()){
                if(innerC instanceof JTextField currField){
                    if (currField.getText() != null && !currField.getText().isBlank()) {
                        try {
                            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-
MM-dd");
                            Date currVal = dateFormat.parse(currField.getText());
                            if(lowerBound == null){
                                lowerBound = currVal;
                            } else {
                                upperBound = currVal;
                            }
                        } catch (Exception exception) {
                            error = true;
                        }
                    } else {
                        error = true;
                    }
                }
```

```
            }
        }
    }
    if(lowerBound != null && upperBound != null && !error && upperBound.
after(lowerBound)){
        DateRule tempDateRule = new DateRule(dateAttr, lowerBound, upperBound);
        currPrincipalCategory.getDateRules().add(tempDateRule);
    } else {
        error = true;
    }
    return error;
}

public static void showCalendar(JTextField inField) {
    SwingUtilities.invokeLater(() -> {
        isFrameOpen = true;
        JCalendar customCalendar = new JCalendar();
        JDialog dialog = new JDialog();
        dialog.setModal(true);
        dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
        dialog.addWindowListener(new WindowAdapter() {
            public void windowClosed(WindowEvent e) {
                // reset the flag when the JDialog is closed
                isFrameOpen = false;
            }
        });
        dialog.setSize(300, 300);

        JButton submitButton = new JButton("Submit");
        submitButton.addActionListener(e -> {
            Date selectedDate = customCalendar.getDate();
            if (selectedDate != null) {
                Calendar selectedCalendar = Calendar.getInstance();
                selectedCalendar.setTime(selectedDate);
                Date selectedDateTime = selectedCalendar.getTime();
                SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
                String dateWithTime = dateFormat.format(selectedDateTime);
                if(inField.isEnabled()) {
                    inField.setText(dateWithTime);
                }
            }
            dialog.dispose();
        });
        JPanel calendarPanel = new JPanel();
        calendarPanel.add(customCalendar);
        calendarPanel.add(submitButton);
        dialog.add(calendarPanel);
        dialog.setResizable(false);
        dialog.setVisible(true);
```

```
        });
    }

    public static JTextField getTextFieldFromComponents(Component[] components) {
        for (Component c : components) {
            if (c instanceof JTextField) {
                return (JTextField) c;
            }
        }
        return null;
    }

    public static JComboBox<?> getComboBoxFromComponents(Component[]
components) {
        for (Component c : components) {
            if (c instanceof JComboBox<?>) {
                return (JComboBox<?>) c;
            }
        }
        return null;
    }

    public static JCheckBox getCheckBoxFromComponents(Component[] components) {
        for (Component c : components) {
            if (c instanceof JCheckBox) {
                return (JCheckBox) c;
            }
        }
        return null;
    }

    public static void showListDialog(JFrame frame, String title, List<String> items) {
        DefaultListModel<String> model = new DefaultListModel<>();
        for (String item : items) {
            model.addElement(item);
        }
        JList<String> list = new JList<>(model);
        JScrollPane scrollPane = new JScrollPane(list, JScrollPane.
VERTICAL_SCROLLBAR_AS_NEEDED, JScrollPane.
HORIZONTAL_SCROLLBAR_ALWAYS);
        scrollPane.setPreferredSize(new Dimension(400, 400));
        JPanel mainPanel = new JPanel(new BorderLayout());
        JPanel panel = new JPanel();
        panel.add(scrollPane);
        mainPanel.add(panel, BorderLayout.CENTER);
        JDialog dialog = new JDialog(frame, title, true);
        dialog.add(mainPanel);
        dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
        JButton okButton = new JButton("OK");
```

```
        okButton.addActionListener(e -> dialog.dispose());
        mainPanel.add(okButton, BorderLayout.SOUTH);
        dialog.pack();
        dialog.setLocationRelativeTo(frame);
        dialog.setVisible(true);
    }


    public static void showDoubleDialog(Frame frame, String dialogTitle, List<String>
list1, List<String> list2, String label1, String label2) {
        DefaultListModel<String> model1 = new DefaultListModel<>();
        for (String s : list1) {
            model1.addElement(s);
        }
        JList<String> jList1 = new JList<>(model1);
        JScrollPane scrollPane1 = new JScrollPane(jList1);

        DefaultListModel<String> model2 = new DefaultListModel<>();
        for (String s : list2) {
            model2.addElement(s);
        }
        JList<String> jList2 = new JList<>(model2);
        JScrollPane scrollPane2 = new JScrollPane(jList2);

        JPanel mainPanel = new JPanel(new BorderLayout());
        JPanel panel = new JPanel(new GridLayout(1, 2));
        panel.add(scrollPane1);
        panel.add(scrollPane2);
        mainPanel.add(panel, BorderLayout.CENTER);

        JDialog dialog = new JDialog(frame, dialogTitle, true);
        dialog.add(mainPanel);
        dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);

        JButton okButton = new JButton("OK");
        okButton.addActionListener(acl -> dialog.dispose());
        mainPanel.add(okButton, BorderLayout.SOUTH);

        JPanel labelPanel = new JPanel(new GridLayout(1, 2));
        labelPanel.add(new JLabel(label1));
        labelPanel.add(new JLabel(label2));
        mainPanel.add(labelPanel, BorderLayout.NORTH);

        dialog.pack();
        dialog.setLocationRelativeTo(frame);
        dialog.setVisible(true);
    }

    public static String getCurrentTime() {
```

```
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm:ss");
        LocalTime now = LocalTime.now();
        return now.format(formatter);
    }

    public static boolean hasCycle(PrincipalCategory category) {
        List<PrincipalCategory> visited = new ArrayList<>();
        List<PrincipalCategory> onStack = new ArrayList<>();
        Stack<PrincipalCategory> stack = new Stack<>();
        stack.push(category);

        while (!stack.isEmpty()) {
            PrincipalCategory current = stack.pop();
            visited.add(current);
            onStack.add(current);
            List<PrincipalCategory> juniorCategories = current.getJuniorCategories();
            for (PrincipalCategory juniorCategory : juniorCategories) {
                if (!visited.contains(juniorCategory)) {
                    stack.push(juniorCategory);
                } else if (onStack.contains(juniorCategory)) {
                    return true;
                }
            }
            onStack.remove(current);
        }
        return false;
    }

    public static List<PrincipalCategory> findAllJuniorCategories(PrincipalCategory
category) {
        if (hasCycle(category)) {
            JOptionPane.showMessageDialog(null, "HAS CYCLE");
            return new ArrayList<>();
        } else {
            List<PrincipalCategory> juniorCategories = new ArrayList<>();
            Set<PrincipalCategory> visited = new HashSet<>();
            Queue<PrincipalCategory> queue = new LinkedList<>();
            queue.add(category);
            visited.add(category);
            while (!queue.isEmpty()) {
                PrincipalCategory currentCategory = queue.poll();
                juniorCategories.add(currentCategory);
                List<PrincipalCategory> subcategories = currentCategory.
getJuniorCategories();
                for (PrincipalCategory subcategory : subcategories) {
                    if (!visited.contains(subcategory)) {
                        queue.add(subcategory);
                        visited.add(subcategory);
                    }
```

```
            }
        }
        return juniorCategories;
    }
}

    public static List<ResourceAction> getAllResourceActions(List<ResourceAction>
actions, Resource r){
        List<ResourceAction> toReturn = new ArrayList<>();
        for(ResourceAction a: actions){
            if(a.getResource().equals(r)){
                toReturn.add(a);
            }
        }
        return toReturn;
    }

    public static AssignCategories fixAssignCategories(AssignCategories bad) {

        List<Principal> principalList = new ArrayList<>(bad.getPrincipals());
        List<Resource> resourceList = new ArrayList<>(bad.getResources());
        List<ResourceAction> resourceActionList = new ArrayList<>();

        for (ResourceAction a : bad.getResourceActions()) {
            for (Resource r : resourceList) {
                if (a.getResource().equals(r)) {
                    ResourceAction temp = new ResourceAction(a.getName(), r);
                    resourceActionList.add(temp);
                }
            }
        }
        List<PrincipalCategory> allPrincipalCategories = bad.getPrincipalCategories();
        List<PrincipalCategory> fixedPrincipalCategories = new ArrayList<>();

        for (PrincipalCategory pc : allPrincipalCategories) {
            PrincipalCategory temp = new PrincipalCategory(pc.getName());

            for (Principal p : pc.getPrincipals()) {
                for (Principal ref : principalList) {
                    if (p.equals(ref)) {
                        temp.addPrincipal(ref);
                    }
                }
            }

            for (ResourceAction a : pc.getActions()) {
                for (ResourceAction ref : resourceActionList) {
                    if (a.equals(ref)) {
                        temp.addAction(ref);
```

```
                }
            }
        }

        temp.setStringRules(pc.getStringRules());
        temp.setIntegerRules(pc.getIntegerRules());
        temp.setDateRules(pc.getDateRules());
        fixedPrincipalCategories.add(temp);
    }

    for (PrincipalCategory pc : allPrincipalCategories) {
        for (PrincipalCategory ref : fixedPrincipalCategories) {
            if (pc.getName().equals(ref.getName())) {
                for (PrincipalCategory juniors : pc.getJuniorCategories()) {
                    for (PrincipalCategory temp : fixedPrincipalCategories) {
                        if (juniors.getName().equals(temp.getName())) {
                            ref.addJuniorCategory(temp);
                        }
                    }
                }
            }
        }
    }

    AssignCategories fixed = new AssignCategories(principalList,
fixedPrincipalCategories);
    fixed.setResources(resourceList);
    fixed.setResourceActions(resourceActionList);
    return fixed;
  }


  public static boolean isRuleSubset(PrincipalCategory junior, PrincipalCategory
senior) {
    return junior.getPrincipals().containsAll(senior.getPrincipals());
  }



}
```

```
import categories.*;
import categories.ResourceAction;
import guipanels.HelperClass;
import guipanels.NumericTextField;
import guipanels.RegistrationForm;
import org.junit.jupiter.api.BeforeEach;
import principal_resource_attributes.*;

import javax.swing.*;
import java.awt.*;
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.*;
import java.util.List;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

public class HelperClassTest {

    private List<Object> graphNodes;
    private Principal principal1, principal2;
    private PrincipalCategory category1, category2;
    private ResourceAction action1, action2;
    private Resource resource1, resource2;

    @BeforeEach
    public void setUp() {
        graphNodes = new ArrayList<>();
        principal1 = new Principal("Alice");
        principal2 = new Principal("Bob");
        category1 = new PrincipalCategory("Admins");
        category2 = new PrincipalCategory("Users");
        resource1 = new Resource("Book");
        resource2 = new Resource("program");
        action1 = new ResourceAction("read", resource1);
        action2 = new ResourceAction("write", resource2);

        graphNodes.add(principal1);
        graphNodes.add(category1);
        graphNodes.add(principal2);
        graphNodes.add(category2);
        graphNodes.add(action1);
        graphNodes.add(action2);
    }

    @Test
```

```
    public void testGetPrincipalIndex() {
        assertEquals(0, HelperClass.getPrincipalIndex(graphNodes, principal1));
        assertEquals(2, HelperClass.getPrincipalIndex(graphNodes, principal2));
        assertEquals(-1, HelperClass.getPrincipalIndex(graphNodes, new Principal("
Charlie")));
    }

    @Test
    public void testGetCategoryIndex() {
        assertEquals(1, HelperClass.getCategoryIndex(graphNodes, category1));
        assertEquals(3, HelperClass.getCategoryIndex(graphNodes, category2));
        assertEquals(-1, HelperClass.getCategoryIndex(graphNodes, new
PrincipalCategory("Guests")));
    }

    @Test
    public void testGetActionIndex() {
        assertEquals(4, HelperClass.getActionIndex(graphNodes, action1));
        assertEquals(5, HelperClass.getActionIndex(graphNodes, action2));
        assertEquals(-1, HelperClass.getActionIndex(graphNodes, new ResourceAction("
execute", resource2)));
    }

    @Test
    public void testGetLeftComponents(){
        RegistrationForm r = new RegistrationForm();
        List<JCheckBox> checkBoxes = new ArrayList<>();
        List<JComboBox<?>> comboBoxes = new ArrayList<>();
        List<JTextField> textFields = new ArrayList<>();
        for(int i = 0; i<r.getLeftPanels().length; i++){
            JPanel leftPanel = r.getLeftPanels()[i];
            checkBoxes.add(HelperClass.getCheckBoxFromComponents(leftPanel.
getComponents()));
            comboBoxes.add(HelperClass.getComboBoxFromComponents(leftPanel.
getComponents()));
            textFields.add(HelperClass.getTextFieldFromComponents(leftPanel.
getComponents()));
        }
        assertEquals(6, checkBoxes.size()); // 6 ATTRIBUTES ALLOWED PER
PRINCIPAL, INCLUDING NAME (COMPULSORY)
        assertEquals(6, comboBoxes.size());
        assertEquals(6, textFields.size());
    }

    @Test
    public void testFillFormWithData() {
        RegistrationForm registrationForm = new RegistrationForm();
        DateTimeFormatter dateFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
```

```
        int[] attributeTypes = {0, 1, 2}; // 0 for String, 1 for Integer, 2 for Date

        for (int i = 0; i < registrationForm.getNUM_OF_ROWS(); i++) {
            JCheckBox enabledCheckbox = (JCheckBox) registrationForm.getLeftPanels()
[i].getComponent(2);
            if (!enabledCheckbox.isSelected()) {
                enabledCheckbox.doClick();
            }

            JComboBox<String> attributeTypeComboBox = (JComboBox<String>)
registrationForm.getLeftPanels()[i].getComponent(0);
            int attributeTypeIndex = attributeTypes[Math.min(i, attributeTypes.length - 1)];
            attributeTypeComboBox.setSelectedIndex(attributeTypeIndex);

            JPanel rightPanel = registrationForm.getRightPanels()[i][attributeTypeIndex];
            Component inputComponent = rightPanel.getComponent(0);

            if (inputComponent instanceof NumericTextField numericTextField ) {
                numericTextField.setText(String.valueOf(i * 100));
            } else if (inputComponent instanceof JTextField textField) {
                textField.setText("Sample Text " + (i + 1));
            } else {
                continue;
            }

            String selectedItem = String.valueOf(attributeTypeComboBox.getSelectedItem())
;
            if (selectedItem.equals("Date attribute")) {
                JTextField dateField = (JTextField) registrationForm.getRightPanels()[i][2].
getComponent(0);
                dateField.setText(dateFormatter.format(LocalDate.now()));
            }
        }

        // You can add assertions here to verify the form data is filled correctly.
        // Example:
        for (int i = 0; i < registrationForm.getNUM_OF_ROWS(); i++) {
            JComboBox<String> attributeTypeComboBox = (JComboBox<String>)
registrationForm.getLeftPanels()[i].getComponent(0);
            String selectedItem = String.valueOf(attributeTypeComboBox.getSelectedItem())
;

            JPanel rightPanel = registrationForm.getRightPanels()
[i][attributeTypeComboBox.getSelectedIndex()];
            Component inputComponent = rightPanel.getComponent(0);

            if (inputComponent instanceof NumericTextField numericTextField) {
                assertTrue(numericTextField.getText().matches("\\d+"));
            } else if (inputComponent instanceof JTextField textField ) {
```

```
            assertTrue(textField.getText().startsWith("Sample Text") || textField.getText().
matches("\\d{4}-\\d{2}-\\d{2}"));
        }
      }
   }


   @Test
   public void testAddCorrespondingAttribute() {
      List<StringAttribute> stringAttributeList = new ArrayList<>();
      List<IntegerAttribute> integerAttributeList = new ArrayList<>();
      List<DateAttribute> dateAttributeList = new ArrayList<>();

      // Test adding a string attribute
      boolean errorParsing = HelperClass.addCorrespondingAttribute(stringAttributeList,
integerAttributeList, dateAttributeList, 0, "Attribute 1", "Value 1");
      assertFalse(errorParsing);
      assertEquals(stringAttributeList.size(), 1);
      assertEquals(stringAttributeList.get(0).getName(), "attribute 1"); // AS THIS
APPLICATION DOES NOT USE UPPERCASE
      assertEquals(stringAttributeList.get(0).getValue(), "value 1");

      // Test adding an integer attribute
      errorParsing = HelperClass.addCorrespondingAttribute(stringAttributeList,
integerAttributeList, dateAttributeList, 1, "Attribute 2", "123");
      assertFalse(errorParsing);
      assertEquals(integerAttributeList.size(), 1);
      assertEquals(integerAttributeList.get(0).getName(), "attribute 2".toLowerCase());
      assertEquals(integerAttributeList.get(0).getValue(), 123);

      // Test adding a date attribute
      errorParsing = HelperClass.addCorrespondingAttribute(stringAttributeList,
integerAttributeList, dateAttributeList, 2, "Attribute 3", "2022-01-01");
      assertFalse(errorParsing);
      assertEquals(dateAttributeList.size(), 1);
      assertEquals(dateAttributeList.get(0).getName(), "attribute 3".toLowerCase());
      SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
      assertEquals(dateFormat.format(dateAttributeList.get(0).getValue()), "2022-01-01")
;
   }

   @Test
   public void testGetActionByName() {
      List<ResourceAction> actionList = new ArrayList<>();
      Resource resource1 = new Resource("Resource 1");
      Resource resource2 = new Resource("Resource 2");
      actionList.add(new ResourceAction("Action 1", resource1));
      actionList.add(new ResourceAction("Action 2", resource1));
      actionList.add(new ResourceAction("Action 1", resource2));
```

```
        actionList.add(new ResourceAction("Action 2", resource2));

        // Test getting an existing action
        ResourceAction action = HelperClass.getActionByName(actionList, "Action 1", "
Resource 1");
        assertNotNull(action);
        assertEquals(action.getName(), "Action 1");
        assertEquals(action.getResource().getName(), "Resource 1");

        // Test getting a non-existing action
        action = HelperClass.getActionByName(actionList, "Action 3", "Resource 3");
        assertNull(action);
    }

    @Test
    public void testGetCategoryByName() {
        List<PrincipalCategory> categoryList = new ArrayList<>();
        categoryList.add(new PrincipalCategory("Category 1"));
        categoryList.add(new PrincipalCategory("Category 2"));

        // Test getting an existing category
        PrincipalCategory category = HelperClass.getCategoryByName(categoryList, "
Category 1");
        assertNotNull(category);
        assertEquals(category.getName(), "Category 1");

        // Test getting a non-existing category
        category = HelperClass.getCategoryByName(categoryList, "Category 3");
        assertNull(category);
    }

    @Test
    public void testGetResourceByName() {
        List<Resource> resourceList = new ArrayList<>();
        resourceList.add(new Resource("Resource 1"));
        resourceList.add(new Resource("Resource 2"));
        // Test getting an existing resource
        Resource resource = HelperClass.getResourceByName(resourceList, "Resource
1");
        assertNotNull(resource);
        assertEquals(resource.getName(), "Resource 1");

// Test getting a non-existing resource
        resource = HelperClass.getResourceByName(resourceList, "Resource 3");
        assertNull(resource);
    }

    @Test
    public void testGetPrincipalByName() {
```

```
        List<Principal> principalList = new ArrayList<>();
        principalList.add(new Principal("Principal 1"));
        principalList.add(new Principal("Principal 2"));
        // Test getting an existing principal
        Principal principal = HelperClass.getPrincipalByName(principalList, "Principal 1");
        assertNotNull(principal);
        assertEquals(principal.getName(), "Principal 1");

// Test getting a non-existing principal
        principal = HelperClass.getPrincipalByName(principalList, "Principal 3");
        assertNull(principal);
    }

    @Test
    public void testFindGridSize() {
// Test with n = 0
        int[] gridSize = HelperClass.findGridSize(0);
        assertNotNull(gridSize);
        assertEquals(gridSize[0], 0);
        assertEquals(gridSize[1], 0);
        // Test with n = 1
        gridSize = HelperClass.findGridSize(1);
        assertNotNull(gridSize);
        assertEquals(gridSize[0], 1);
        assertEquals(gridSize[1], 1);

// Test with n = 4
        gridSize = HelperClass.findGridSize(4);
        assertNotNull(gridSize);
        assertEquals(gridSize[0], 2);
        assertEquals(gridSize[1], 2);

// Test with n = 5
        gridSize = HelperClass.findGridSize(5);
        assertNotNull(gridSize);
        assertEquals(gridSize[0], 3);
        assertEquals(gridSize[1], 2);
    }

    @Test
    public void testGetBlankStringAttributeList2() {
// Create principal list with attributes
        Principal principal1 = new Principal("Principal 1");
        principal1.getStringAttributeList().add(new StringAttribute("Attribute 1", ""));
        principal1.getStringAttributeList().add(new StringAttribute("Attribute 2", ""));
        Principal principal2 = new Principal("Principal 2");
        principal2.getStringAttributeList().add(new StringAttribute("Attribute 1", ""));
        principal2.getStringAttributeList().add(new StringAttribute("Attribute 3", ""));
        List<Principal> principalList = new ArrayList<>();
```

```
        principalList.add(principal1);
        principalList.add(principal2);
        // Test getting blank string attribute list
        List<StringAttribute> blankStringAttributeList = HelperClass.
getBlankStringAttributeList(principalList);
        assertNotNull(blankStringAttributeList);
        assertEquals(blankStringAttributeList.size(), 3);
        assertTrue(blankStringAttributeList.contains(new StringAttribute("Attribute 1", "")));
        assertTrue(blankStringAttributeList.contains(new StringAttribute("Attribute 2", "")));
        assertTrue(blankStringAttributeList.contains(new StringAttribute("Attribute 3", "")));
    }
        @Test
        public void testGetResourceByName2() {
            List<Resource> resourceList = new ArrayList<>();
            resourceList.add(new Resource("Resource 1"));
            resourceList.add(new Resource("Resource 2"));
// Test getting an existing resource
            Resource resource = HelperClass.getResourceByName(resourceList, "
Resource 1");
            assertNotNull(resource);
            assertEquals(resource.getName(), "Resource 1");

// Test getting a non-existing resource
            resource = HelperClass.getResourceByName(resourceList, "Resource 3");
            assertNull(resource);
        }

        @Test
        public void testGetPrincipalByName2() {
            List<Principal> principalList = new ArrayList<>();
            principalList.add(new Principal("Principal 1"));
            principalList.add(new Principal("Principal 2"));

// Test getting an existing principal
            Principal principal = HelperClass.getPrincipalByName(principalList, "Principal 1")
;
            assertNotNull(principal);
            assertEquals(principal.getName(), "Principal 1");

// Test getting a non-existing principal
            principal = HelperClass.getPrincipalByName(principalList, "Principal 3");
            assertNull(principal);
        }

        @Test
        public void testFindGridSize2() {
// Test with n = 0
            int[] gridSize = HelperClass.findGridSize(0);
            assertNotNull(gridSize);
```

```
            assertEquals(gridSize[0], 0);
            assertEquals(gridSize[1], 0);
// Test with n = 1
            gridSize = HelperClass.findGridSize(1);
            assertNotNull(gridSize);
            assertEquals(gridSize[0], 1);
            assertEquals(gridSize[1], 1);

// Test with n = 4
            gridSize = HelperClass.findGridSize(4);
            assertNotNull(gridSize);
            assertEquals(gridSize[0], 2);
            assertEquals(gridSize[1], 2);

// Test with n = 5
            gridSize = HelperClass.findGridSize(5);
            assertNotNull(gridSize);
            assertEquals(gridSize[0], 3);
            assertEquals(gridSize[1], 2);
        }

        @Test
        public void testGetBlankStringAttributeList() {
// Create principal list with attributes
            Principal principal1 = new Principal("Principal 1");
            principal1.getStringAttributeList().add(new StringAttribute("Attribute 1", ""));
            principal1.getStringAttributeList().add(new StringAttribute("Attribute 2", ""));
            Principal principal2 = new Principal("Principal 2");
            principal2.getStringAttributeList().add(new StringAttribute("Attribute 1", ""));
            principal2.getStringAttributeList().add(new StringAttribute("Attribute 3", ""));
            List<Principal> principalList = new ArrayList<>();
            principalList.add(principal1);
            principalList.add(principal2);
// Test getting blank string attribute list
            List<StringAttribute> blankStringAttributeList = HelperClass.
getBlankStringAttributeList(principalList);
            assertNotNull(blankStringAttributeList);
            assertEquals(blankStringAttributeList.size(), 3);
            assertTrue(blankStringAttributeList.contains(new StringAttribute("Attribute 1", "")
));
            assertTrue(blankStringAttributeList.contains(new StringAttribute("Attribute 2", "")
));
            assertTrue(blankStringAttributeList.contains(new StringAttribute("Attribute 3", "")
));
        }

        @Test
        public void testGetBlankIntegerAttributeList() {
// Create principal list with attributes
```

```
        Principal principal1 = new Principal("Principal 1");
        principal1.getIntegerAttributeList().add(new IntegerAttribute("Attribute 1", 0));
        principal1.getIntegerAttributeList().add(new IntegerAttribute("Attribute 2", 0));
        Principal principal2 = new Principal("Principal 2");
        principal2.getIntegerAttributeList().add(new IntegerAttribute("Attribute 1", 0));
        principal2.getIntegerAttributeList().add(new IntegerAttribute("Attribute 3", 0));
        List<Principal> principalList = new ArrayList<>();
        principalList.add(principal1);
        principalList.add(principal2);
        // Test getting blank integer attribute list
        List<IntegerAttribute> blankIntegerAttributeList = HelperClass.
getBlankIntegerAttributeList(principalList);
        assertNotNull(blankIntegerAttributeList);
        assertEquals(blankIntegerAttributeList.size(), 3);
        assertTrue(blankIntegerAttributeList.contains(new IntegerAttribute("Attribute 1",
0)));
        assertTrue(blankIntegerAttributeList.contains(new IntegerAttribute("Attribute 2",
0)));
        assertTrue(blankIntegerAttributeList.contains(new IntegerAttribute("Attribute 3",
0)));
    }

    @Test
    public void testGetBlankDateAttributeList() {
// Create principal list with attributes
        Principal principal1 = new Principal("Principal 1");
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
        Date date = new Date();
        principal1.getDateAttributeList().add(new DateAttribute("Attribute 1", date));
        principal1.getDateAttributeList().add(new DateAttribute("Attribute 2", date));
        Principal principal2 = new Principal("Principal 2");
        principal2.getDateAttributeList().add(new DateAttribute("Attribute 1", date));
        principal2.getDateAttributeList().add(new DateAttribute("Attribute 3", date));
        List<Principal> principalList = new ArrayList<>();
        principalList.add(principal1);
        principalList.add(principal2);

// Test getting blank date attribute list
        List<DateAttribute> blankDateAttributeList = HelperClass.
getBlankDateAttributeList(principalList);
        assertNotNull(blankDateAttributeList);
        assertEquals(blankDateAttributeList.size(), 3);
        assertTrue(blankDateAttributeList.contains(new DateAttribute("Attribute 1", new
Date())));
        assertTrue(blankDateAttributeList.contains(new DateAttribute("Attribute 2", new
Date())));
        assertTrue(blankDateAttributeList.contains(new DateAttribute("Attribute 3", new
Date())));
    }
```

```
}
```

```
package principal_resource_attributes;

import java.util.Objects;

public class IntegerAttribute {
    private int value;
    private String name;

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        IntegerAttribute that = (IntegerAttribute) o;
        return value == that.value &&
                Objects.equals(name, that.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(value, name);
    }

    public IntegerAttribute(String name, int value) {
        this.name = name.toLowerCase();
        this.value = value;
    }
    public IntegerAttribute(IntegerAttribute other) {
        this.name = other.name;
        this.value = other.value;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name.toLowerCase();
    }

    public int getValue() {
        return value;
    }

    public void setValue(int value) {
        this.value = value;
    }

    @Override
    public String toString() {
```

```
        return "Type: Integer attribute | " + name.toString() + " | Value: " + value;
    }
}
```

```
package categoryrules;

import principal_resource_attributes.IntegerAttribute;

import java.util.Objects;

public class IntegerRule {
    private int lowerBound;
    private int upperBound;
    private IntegerAttribute integerAttribute;

    public IntegerRule(IntegerRule other) {
        this.lowerBound = other.lowerBound;
        this.upperBound = other.upperBound;
        this.integerAttribute = new IntegerAttribute(other.integerAttribute);
    }

    public IntegerAttribute getAttribute() {
        return integerAttribute;
    }

    public void setAttribute(IntegerAttribute integerAttribute) {
        this.integerAttribute = integerAttribute;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        IntegerRule that = (IntegerRule) o;
        return lowerBound == that.lowerBound &&
                upperBound == that.upperBound &&
                Objects.equals(integerAttribute, that.integerAttribute);
    }

    @Override
    public int hashCode() {
        return Objects.hash(lowerBound, upperBound, integerAttribute);
    }

    public IntegerRule(IntegerAttribute attribute, int lowerBound, int upperBound){
        this.integerAttribute = attribute;
        this.lowerBound = lowerBound;
        this.upperBound = upperBound;
    }

    public int getLowerBound() {
        return lowerBound;
    }
```

```
    public void setLowerBound(int lowerBound) {
        this.lowerBound = lowerBound;
    }

    public int getUpperBound() {
        return upperBound;
    }

    public void setUpperBound(int upperBound) {
        this.upperBound = upperBound;
    }

    @Override
    public String toString(){
        return integerAttribute.getName() +  "[ " + getLowerBound() + " - " +
getUpperBound() + " ]";
    }
}
```

```
package database;

import categories.AssignCategories;
import com.mongodb.client.MongoClients;
import guipanels.GraphDisplay;
import guipanels.HelperClass;

import javax.swing.*;
import java.awt.*;

import java.util.HashMap;
import java.util.Map;

public class LoadDataWorker extends SwingWorker<Map<String, Object>, Void> {
    private final JFrame frame;
    private final JPanel START_PANEL;
    private final CardLayout cl;
    private final JButton tempButton;
    private final JButton otherButton;

    public LoadDataWorker(JFrame frame, JPanel startPanel, CardLayout cl, JButton
tempButton, JButton otherButton) {
        this.frame = frame;
        this.START_PANEL = startPanel;
        this.cl = cl;
        this.tempButton = tempButton;
        this.otherButton = otherButton;
    }
    @Override
    protected Map<String, Object> doInBackground() throws Exception {
        tempButton.setEnabled(false);
        MongoMain mongoMain = new MongoMain(MongoClients.
create(MongoDB_CONFIG.DATABASE_URL));
        if (mongoMain.databaseEmpty()) {
            JOptionPane.showMessageDialog(frame, "Database empty, create new policy!")
;
            return null;
        } else {
            AssignCategories currentState = mongoMain.getAssignCategories();
            AssignCategories fixedAfter = HelperClass.fixAssignCategories(currentState);
            UndoClass undoClass = mongoMain.getUndoClass();
            Map<String, Object> result = new HashMap<>();
            result.put("assignCategories", fixedAfter);
            result.put("undoClass", undoClass);
            return result;
        }
    }

    @Override
```

```
    protected void done() {
        try {
            Map<String, Object> result = get();
            if (result != null) {
                AssignCategories fixedAfter = (AssignCategories) result.get("
assignCategories");
                UndoClass undoClass = (UndoClass) result.get("undoClass");
                // Do something with undoClass if needed
                new GraphDisplay(fixedAfter, undoClass, START_PANEL, frame, cl);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            tempButton.setEnabled(true);
            otherButton.setEnabled(true);
        }
    }

}
```

```
package database;

import categories.AssignCategories;
import com.mongodb.client.MongoClients;
import guipanels.GraphDisplay;

import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;

public class LoadFileWorker extends SwingWorker<Void, Void> {
    private final JFrame frame;
    private final JPanel START_PANEL;
    private final CardLayout cl;
    private final JButton loadFileButton;
    private final JButton otherButton;
    private final AssignCategories assignCategories;
    private final UndoClass undoClass;

    public LoadFileWorker(JFrame frame, JPanel startPanel, CardLayout cl, JButton
loadFileButton, JButton otherButton,
                          AssignCategories assignCategories, UndoClass undoClass) {
        this.frame = frame;
        this.START_PANEL = startPanel;
        this.cl = cl;
        this.loadFileButton = loadFileButton;
        this.otherButton = otherButton;
        this.undoClass = undoClass;
        this.assignCategories = assignCategories;
    }

    @Override
    protected Void doInBackground() throws Exception {
        loadFileButton.setEnabled(false);
        otherButton.setEnabled(false);
        MongoMain mongoMain = new MongoMain(MongoClients.
create(MongoDB_CONFIG.DATABASE_URL));
            mongoMain.saveAssignCategories(assignCategories, undoClass);
        return null;
    }

    @Override
    protected void done() {
        try {
            new GraphDisplay(assignCategories, undoClass, START_PANEL, frame, cl);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            loadFileButton.setEnabled(true);
```

```
            otherButton.setEnabled(true);
        }
    }
}
```