Feature Envy : TagDao.java (books-core com.sismics.books.core.dao.jpa)

Original Code:

```java
/**
 * Tag DAO.
 *
 * @author bgamard
 */
public class TagDao {
    /**
     * Gets a tag by its ID.
     *
     * @param id Tag ID
     * @return Tag
     */
    public Tag getById(String id) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        try {
            return em.find(Tag.class, id);
        } catch (NoResultException e) {
            return null;
        }
    }

    /**
     * Returns the list of all tags.
     *
     * @return List of tags
     */
    @SuppressWarnings("unchecked")
    public List<Tag> getByUserId(String userId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("select t from Tag t where t.userId =
:userId and t.deleteDate is null order by t.name");
        q.setParameter("userId", userId);
        return q.getResultList();
    }

    /**
     * Update tags on a user book.
     *
     * @param userBookId
     * @param tagIdSet
     */
    public void updateTagList(String userBookId, Set<String> tagIdSet) {
        // Delete old tag links
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("delete UserBookTag bt where bt.userBookId
= :userBookId");
        q.setParameter("userBookId", userBookId);
        q.executeUpdate();

        // Create new tag links
        for (String tagId : tagIdSet) {
            UserBookTag userBookTag = new UserBookTag();
            userBookTag.setId(UUID.randomUUID().toString());
            userBookTag.setUserBookId(userBookId);
            userBookTag.setTagId(tagId);
            em.persist(userBookTag);
        }

    }

    /**
     * Returns tag list on a user book.
     * @param userBookId
     * @return
     */
    @SuppressWarnings("unchecked")
    public List<TagDto> getByUserBookId(String userBookId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("select t.TAG_ID_C,
t.TAG_NAME_C, t.TAG_COLOR_C from T_USER_BOOK_TAG bt ");
        sb.append(" join T_TAG t on t.TAG_ID_C = bt.BOT_IDTAG_C ");
        sb.append(" where bt.BOT_IDUSERBOOK_C = :userBookId and
t.TAG_DELETEDATE_D is null ");
        sb.append(" order by t.TAG_NAME_C ");

        // Perform the query
        Query q = em.createNativeQuery(sb.toString());
        q.setParameter("userBookId", userBookId);
        List<Object[]> l = q.getResultList();

        // Assemble results
        List<TagDto> tagDtoList = new ArrayList<TagDto>();
        for (Object[] o : l) {
            int i = 0;
            TagDto tagDto = new TagDto();
            tagDto.setId((String) o[i++]);
            tagDto.setName((String) o[i++]);
            tagDto.setColor((String) o[i++]);
            tagDtoList.add(tagDto);
        }
        return tagDtoList;
    }

    /**
     * Creates a new tag.
     *
     * @param tag Tag
     * @return New ID
     * @throws Exception
     */
    public String create(Tag tag) {
        // Create the UUID
        tag.setId(UUID.randomUUID().toString());

        // Create the tag
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        tag.setCreateDate(new Date());
        em.persist(tag);

        return tag.getId();
    }

    /**
     * Returns a tag by name.
     * @param userId User ID
     * @param name Name
     * @return Tag
     */
    public Tag getByName(String userId, String name) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("select t from Tag t where t.name = :name
and t.userId = :userId and t.deleteDate is null");
        q.setParameter("userId", userId);
        q.setParameter("name", name);
        try {
            return (Tag) q.getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }

    /**
     * Returns a tag by ID.
     * @param userId User ID
     * @param tagId Tag ID
     * @return Tag
     */
    public Tag getByTagId(String userId, String tagId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("select t from Tag t where t.id = :tagId
and t.userId = :userId and t.deleteDate is null");
        q.setParameter("userId", userId);
        q.setParameter("tagId", tagId);
        try {
            return (Tag) q.getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }

    /**
     * Deletes a tag.
     *
     * @param tagId Tag ID
     */
    public void delete(String tagId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();

        // Get the tag
        Query q = em.createQuery("select t from Tag t where t.id = :id and
t.deleteDate is null");
        q.setParameter("id", tagId);
        Tag tagDb = (Tag) q.getSingleResult();

        // Delete the tag
        Date dateNow = new Date();
        tagDb.setDeleteDate(dateNow);

        // Delete linked data
        q = em.createQuery("delete UserBookTag bt where bt.tagId =
:tagId");
        q.setParameter("tagId", tagId);
        q.executeUpdate();
    }

    /**
     * Search tags by name.
     *
     * @param name Tag name
     * @return List of found tags
     */
    @SuppressWarnings("unchecked")
    public List<Tag> findByName(String userId, String name) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("select t from Tag t where t.name like
:name and t.userId = :userId and t.deleteDate is null");
        q.setParameter("userId", userId);
        q.setParameter("name", "%" + name + "%");
        return q.getResultList();
    }
}
```

Manually Refactored Code:

```
public class TagDao {
    private final EntityManager em;

    public TagDao() {
        this.em = ThreadLocalContext.get().getEntityManager();
    }

    public Tag getById(String id) {
        return findTagById(id);
    }

    private Tag findTagById(String id) {
        try {
            return em.find(Tag.class, id);
        } catch (NoResultException e) {
            return null;
        }
    }

    public List<Tag> getByUserId(String userId) {
        return findTagsByUserId(userId);
    }

    private List<Tag> findTagsByUserId(String userId) {
        Query q = em
                .createQuery("select t from Tag t where t.userId = :userId
and t.deleteDate is null order by t.name");
        q.setParameter("userId", userId);
        return q.getResultList();
    }

    public void updateTagList(String userBookId, Set<String> tagIds) {
        deleteOldTagLinks(userBookId);
        createNewTagLinks(userBookId, tagIds);
    }

    private void deleteOldTagLinks(String userBookId) {
        Query q = em.createQuery("delete UserBookTag bt where bt.userBookId
= :userBookId");
        q.setParameter("userBookId", userBookId);
        q.executeUpdate();
    }

    private void createNewTagLinks(String userBookId, Set<String> tagIds) {
        for (String tagId : tagIds) {
            UserBookTag userBookTag = new UserBookTag();
            userBookTag.setId(UUID.randomUUID().toString());
            userBookTag.setUserBookId(userBookId);
            userBookTag.setTagId(tagId);
            em.persist(userBookTag);
        }
    }

    public List<TagDto> getByUserBookId(String userBookId) {
        return assembleTagDtosFromUserBookId(userBookId);
    }

    private List<TagDto> assembleTagDtosFromUserBookId(String userBookId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("select t.TAG_ID_C,
t.TAG_NAME_C, t.TAG_COLOR_C from T_USER_BOOK_TAG bt ");
        sb.append(" join T_TAG t on t.TAG_ID_C = bt.BOT_IDTAG_C ");
        sb.append(" where bt.BOT_IDUSERBOOK_C = :userBookId and
t.TAG_DELETEDATE_D is null ");
        sb.append(" order by t.TAG_NAME_C ");

        // Perform the query
        Query q = em.createNativeQuery(sb.toString());
        q.setParameter("userBookId", userBookId);
        List<Object[]> l = q.getResultList();

        // Assemble results
        List<TagDto> tagDtoList = new ArrayList<TagDto>();
        for (Object[] o : l) {
            int i = 0;
            TagDto tagDto = new TagDto();
            tagDto.setId((String) o[i++]);
            tagDto.setName((String) o[i++]);
            tagDto.setColor((String) o[i++]);
            tagDtoList.add(tagDto);
        }
        return tagDtoList;
    }

    public String create(Tag tag) {
        tag.setId(UUID.randomUUID().toString());
        tag.setCreateDate(new Date());
        em.persist(tag);
        return tag.getId();
    }

    public Tag getByName(String userId, String name) {
        return findTagByNameAndUserId(userId, name);
    }

    private Tag findTagByNameAndUserId(String userId, String name) {
        Query q = em.createQuery(
                "select t from Tag t where t.name = :name and t.userId =
:userId and t.deleteDate is null");
        q.setParameter("userId", userId);
        q.setParameter("name", name);
        try {
            return (Tag) q.getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }

    public Tag getByTagId(String userId, String tagId) {
        return findTagByIdAndUserId(userId, tagId);
    }

    private Tag findTagByIdAndUserId(String userId, String tagId) {
        Query q = em
                .createQuery("select t from Tag t where t.id = :tagId and
t.userId = :userId and t.deleteDate is null");
        q.setParameter("userId", userId);
        q.setParameter("tagId", tagId);
        try {
            return (Tag) q.getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }

    public void delete(String tagId) {
        Tag tagDb = findTagById(tagId);
        if (tagDb != null) {
            Date dateNow = new Date();
            tagDb.setDeleteDate(dateNow);
            deleteLinkedData(tagId);
        }
    }

    private void deleteLinkedData(String tagId) {
        Query q = em.createQuery("delete UserBookTag bt where bt.tagId =
:tagId");
        q.setParameter("tagId", tagId);
        q.executeUpdate();
    }

    public List<Tag> findByName(String userId, String name) {
        return findTagsByNameAndUserId(userId, name);
    }

    private List<Tag> findTagsByNameAndUserId(String userId, String name) {
        Query q = em.createQuery(
                "select t from Tag t where t.name like :name and t.userId =
:userId and t.deleteDate is null");
        q.setParameter("userId", userId);
        q.setParameter("name", "%" + name + "%");
        return q.getResultList();
    }
}
```

LLM Given Code:

```
/**
 * Data Access Object (DAO) for managing Tag entities.
 * Handles CRUD operations and queries related to tags.
 * Author: bgamard
 */
public class TagDao {

    /**
     * Retrieves a tag by its ID.
     *
     * @param id Tag ID
     * @return Tag object if found, null otherwise
     */
    public Tag getById(String id) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        try {
            return em.find(Tag.class, id);
        } catch (NoResultException e) {
            return null;
        }
    }

    /**
     * Retrieves all tags associated with a specific user.
     *
     * @param userId User ID
     * @return List of tags associated with the user
     */
    @SuppressWarnings("unchecked")
    public List<Tag> getByUserId(String userId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("SELECT t FROM Tag t WHERE t.userId =
:userId AND t.deleteDate IS NULL ORDER BY t.name");
        q.setParameter("userId", userId);
        return q.getResultList();
    }

    /**
     * Updates tags associated with a user book.
     *
     * @param userBookId User book ID
     * @param tagIdSet Set of tag IDs to be associated with the user book
     */
    public void updateTagsForUserBook(String userBookId, Set<String>
tagIdSet) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        // Delete old tag links
        Query deleteQuery = em.createQuery("DELETE FROM UserBookTag bt
WHERE bt.userBookId = :userBookId");
        deleteQuery.setParameter("userBookId", userBookId);
        deleteQuery.executeUpdate();

        // Create new tag links
        for (String tagId : tagIdSet) {
            UserBookTag userBookTag = new UserBookTag();
            userBookTag.setId(UUID.randomUUID().toString());
            userBookTag.setUserBookId(userBookId);
            userBookTag.setTagId(tagId);
            em.persist(userBookTag);
        }
    }

    /**
     * Retrieves tag list associated with a user book.
     *
     * @param userBookId User book ID
     * @return List of TagDto objects representing tags associated with the
user book
     */
    @SuppressWarnings("unchecked")
    public List<TagDto> getTagsByUserBookId(String userBookId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        StringBuilder sb = new StringBuilder("SELECT t.TAG_ID_C,
t.TAG_NAME_C, t.TAG_COLOR_C FROM T_USER_BOOK_TAG bt ");
        sb.append("JOIN T_TAG t ON t.TAG_ID_C = bt.BOT_IDTAG_C ");
        sb.append("WHERE bt.BOT_IDUSERBOOK_C = :userBookId AND
t.TAG_DELETEDATE_D IS NULL ");
        sb.append("ORDER BY t.TAG_NAME_C ");

        // Perform the query
        Query q = em.createNativeQuery(sb.toString());
        q.setParameter("userBookId", userBookId);
        List<Object[]> resultList = q.getResultList();

        // Assemble results
        List<TagDto> tagDtoList = new ArrayList<>();
        for (Object[] result : resultList) {
            int index = 0;
            TagDto tagDto = new TagDto();
            tagDto.setId((String) result[index++]);
            tagDto.setName((String) result[index++]);
            tagDto.setColor((String) result[index++]);
            tagDtoList.add(tagDto);
        }
        return tagDtoList;
    }

    /**
     * Creates a new tag.
     *
     * @param tag Tag object to be created
     * @return New ID of the created tag
     */
    public String createTag(Tag tag) {
        // Generate UUID
        tag.setId(UUID.randomUUID().toString());

        // Create the tag
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        tag.setCreateDate(new Date());
        em.persist(tag);

        return tag.getId();
    }

    /**
     * Retrieves a tag by its name.
     *
     * @param userId User ID
     * @param name Tag name
     * @return Tag object if found, null otherwise
     */
    public Tag getByName(String userId, String name) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("SELECT t FROM Tag t WHERE t.name = :name
AND t.userId = :userId AND t.deleteDate IS NULL");
        q.setParameter("userId", userId);
        q.setParameter("name", name);
        try {
            return (Tag) q.getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }

    /**
     * Retrieves a tag by its ID.
     *
     * @param userId User ID
     * @param tagId Tag ID
     * @return Tag object if found, null otherwise
     */
    public Tag getByTagId(String userId, String tagId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("SELECT t FROM Tag t WHERE t.id = :tagId
AND t.userId = :userId AND t.deleteDate IS NULL");
        q.setParameter("userId", userId);
        q.setParameter("tagId", tagId);
        try {
            return (Tag) q.getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }

    /**
     * Deletes a tag.
     *
     * @param tagId Tag ID
     */
    public void deleteTag(String tagId) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();

        // Get the tag
        Query selectQuery = em.createQuery("SELECT t FROM Tag t WHERE t.id
= :id AND t.deleteDate IS NULL");
        selectQuery.setParameter("id", tagId);
        Tag tagDb = (Tag) selectQuery.getSingleResult();

        // Delete the tag
        Date dateNow = new Date();
        tagDb.setDeleteDate(dateNow);

        // Delete linked data
        Query deleteQuery = em.createQuery("DELETE FROM UserBookTag bt
WHERE bt.tagId = :tagId");
        deleteQuery.setParameter("tagId", tagId);
        deleteQuery.executeUpdate();
    }

    /**
     * Searches tags by name.
     *
     * @param userId User ID
     * @param name Tag name (or part of it) to search for
     * @return List of found tags
     */
    @SuppressWarnings("unchecked")
    public List<Tag> findByName(String userId, String name) {
        EntityManager em = ThreadLocalContext.get().getEntityManager();
        Query q = em.createQuery("SELECT t FROM Tag t WHERE t.name LIKE
:name AND t.userId = :userId AND t.deleteDate IS NULL");
        q.setParameter("userId", userId);
        q.setParameter("name", "%" + name + "%");
        return q.getResultList();
    }
}
```

ChatGPT Chat Link: https://chat.openai.com/share/3b246ad1-11ce-427a-9b11-b2d2b8e19158

Differences in Refactoring Approach

1. **Entity Manager Initialization:**
    - **Manual:** Uses a class-level `EntityManager`, reducing redundancy.
    - **LLM:** Keeps the original method-level `EntityManager` access.
2. **Method Structuring:**
    - **Manual:** Extracts private helper methods for common queries, improving code reuse.
    - **LLM:** Executes queries directly in public methods, simpler but less reusable.
3. **Code Readability and Maintainability:**
    - **Manual:** Enhances readability and maintainability by centralizing query logic.
    - **LLM:** Each method is self-contained, straightforward but potentially harder to maintain.
4. **Efficiency Considerations:**
    - **Both:** No significant efficiency difference, though manual might slightly edge out due to centralized `EntityManager` management.

Scenarios Where LLMs Excel vs. Manual Intervention

- **LLMs Excel:**
    - Quick boilerplate code generation.
    - Immediate, contextually relevant suggestions.
    - Maintaining existing coding style consistency.
- **Manual Intervention Preferable:**
    - Deep architectural refactoring.
    - Adherence to specific coding or architectural patterns.
    - Complex refactoring involving performance, readability, and maintainability trade-offs.

Conclusion

While manual refactoring is better for deep, maintainable changes, LLMs are great for quick fixes and maintaining style consistency. A combination of both methods often works best.