Lazy Class & Deficient Encapsulation: Constants (com.sismics.books.core.constant)

Identified Code Snippet:

`Constants.java`:

```java
/**
 * Application constants.
 *
 * @author jtremeaux
 */
public class Constants {
    /**
     * Default locale.
     */
    public static final String DEFAULT_LOCALE_ID = "en";

    /**
     * Default timezone ID.
     */
    public static final String DEFAULT_TIMEZONE_ID = "Europe/London";

    /**
     * Default theme ID.
     */
    public static final String DEFAULT_THEME_ID = "default.less";

    /**
     * Administrator's default password ("admin").
     */
    public static final String DEFAULT_ADMIN_PASSWORD =
"$2a$05$6Ny3TjrW3aVAL1or2SlcR.fhuDgPKp5jp.P9fBXwVNePgeLqb4i3C";

    /**
     * Default generic user role.
     */
    public static final String DEFAULT_USER_ROLE = "user";
}
```

`UserDao.java`

```java


/**
 * User DAO.
 *
 * @author jtremeaux
 */
public class UserDao {
    ///
    /// ... something ...
    ///

    /**
     * Creates a new user.
     *
     * @param user User to create
     * @return User ID
     * @throws Exception
     */
    public String create(User user) throws Exception {
        ///
        /// ... something ...
        ///

        user.setTheme(Constants.DEFAULT_THEME_ID);

        ///
        /// ... something ...
        ///
    }

    ///
    /// ... something ...
    ///
}
```

`LocaleUtil.java`

```


/**
 * Locale utilities.
 *
 * @author jtremeaux
 */
public class LocaleUtil {
    ///
    /// ... something ...
    ///

    /**
     * Extracts the ID of the locale from the HTTP Accept-Language header.
     *
     * @param acceptLanguageHeader header
     * @return Locale ID
     */
    public static String getLocaleIdFromAcceptLanguage(String
acceptLanguageHeader) {
        ///
        /// ... something ...
        ///

        if (StringUtils.isNotBlank(localeId)) {
            ///
            /// ... something ...
            ///

            localeId = Constants.DEFAULT_LOCALE_ID;

            ///
            /// ... something ...
            ///
        }
        if (StringUtils.isBlank(localeId)) {
            localeId = Constants.DEFAULT_LOCALE_ID;
        }

        ///
        /// ... something ...
        ///
    }
}
```

`UserResource.java` (Changed to `UserCrudResource.java` after God Smell refactoring)

```


/**
 * User REST resources for CRUD Operations on User.
 *
 * @author jtremeaux
 */
@Path("/user")
public class UserCrudResource extends BaseResource {

    /**
     * Returns the information about the connected user.
     *
     * @return Response
     * @throws JSONException
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response info() throws JSONException {
        JSONObject response = new JSONObject();
        if (!authenticate()) {

            // something

            if (adminUser != null && adminUser.getDeleteDate() == null) {
                response.put("is_default_password",
Constants.DEFAULT_ADMIN_PASSWORD.equals(adminUser.getPassword()));
```

```
            }
        } else {
            // something

            response.put("is_default_password",
hasBaseFunction(BaseFunction.ADMIN) &&
Constants.DEFAULT_ADMIN_PASSWORD.equals(user.getPassword()));
        }

        return Response.ok().entity(response).build();
    }

    /**
     * Creates a new user.
     *
     * @param username User's username
     * @param password Password
     * @param email E-Mail
     * @param localeId Locale ID
     * @return Response
     * @throws JSONException
     */
    @PUT
    @Produces(MediaType.APPLICATION_JSON)
    public Response register(
        @FormParam("username") String username,
        @FormParam("password") String password,
        @FormParam("locale") String localeId,
        @FormParam("email") String email) throws JSONException {

        // ... something ...

        // Create the user
        User user = new User();
        user.setRoleId(Constants.DEFAULT_USER_ROLE);

        // ... something ...

        user.setLocaleId(Constants.DEFAULT_LOCALE_ID);

        // ... something ...
    }

    // something

}
```

`RequestContextFilter.java`:

```


/**
 * Filter used to process a couple things in the request context.
```

```java
 *
 * @author jtremeaux
 */
public class RequestContextFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Force the locale in order to not depend on the execution
environment
        Locale.setDefault(new Locale(Constants.DEFAULT_LOCALE_ID));

        // something
    }

    // ...other methods...
}
```

TokenBasedSecurityFilter.java:

```java


/**
 * This filter is used to authenticate the user having an active session
via an authentication token stored in database.
 * The filter extracts the authentication token stored in a cookie.
 * If the ocokie exists and the token is valid, the filter injects a
UserPrincipal into a request attribute.
 * If not, the user is anonymous, and the filter injects a
AnonymousPrincipal into the request attribute.
 *
 * @author jtremeaux
 */
public class TokenBasedSecurityFilter implements Filter {
    // something

    /**
     * Inject an anonymous user into the request attributes.
     *
     * @param request HTTP request
     */
    private void injectAnonymousUser(HttpServletRequest request) {
        // something


anonymousPrincipal.setDateTimeZone(DateTimeZone.forID(Constants.DEFAULT_TIM
EZONE_ID));

        // something
    }
}
```

Apply Manual Refactoring

1. Removed `Constants.java`

`UserDao.java`

```java
/**
 * User DAO.
 *
 * @author jtremeaux
 */
public class UserDao {


    private String DEFAULT_THEME_ID = "default.less";


    ///
    /// ... something ...
    ///

    /**
     * Creates a new user.
     *
     * @param user User to create
     * @return User ID
     * @throws Exception
     */
    public String create(User user) throws Exception {
        ///
        /// ... something ...
        ///

        user.setTheme(DEFAULT_THEME_ID);

        ///
        /// ... something ...
        ///
    }

    ///
    /// ... something ...
    ///
}
```

`LocaleUtil.java`

```java
/**
```

```
 * Locale utilities.
 *
 * @author jtremeaux
 */
public class LocaleUtil {
    ///
    /// ... something ...
    ///

    private static String DEFAULT_LOCALE_ID = "en";

    /**
     * Extracts the ID of the locale from the HTTP Accept-Language header.
     *
     * @param acceptLanguageHeader header
     * @return Locale ID
     */
    public static String getLocaleIdFromAcceptLanguage(String
acceptLanguageHeader) {
        ///
        /// ... something ...
        ///

        if (StringUtils.isNotBlank(localeId)) {
            ///
            /// ... something ...
            ///

            localeId = DEFAULT_LOCALE_ID;

            ///
            /// ... something ...
            ///
        }
        if (StringUtils.isBlank(localeId)) {
            localeId = DEFAULT_LOCALE_ID;
        }

        ///
        /// ... something ...
        ///
    }
}
```

`UserResource.java` (Changed to `UserCrudResource.java` after God Smell refactoring)

```


/**
 * User REST resources for CRUD Operations on User.
 *
 * @author jtremeaux
 */
```

```
@Path("/user")
public class UserCrudResource extends BaseResource {

    private String DEFAULT_ADMIN_PASSWORD =
"$2a$05$6Ny3TjrW3aVAL1or2SlcR.fhuDgPKp5jp.P9fBXwVNePgeLqb4i3C";
    private String DEFAULT_USER_ROLE = "user";
    private String DEFAULT_LOCALE_ID = "en";

    /**
     * Returns the information about the connected user.
     *
     * @return Response
     * @throws JSONException
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response info() throws JSONException {
        JSONObject response = new JSONObject();
        if (!authenticate()) {

            // something

            if (adminUser != null && adminUser.getDeleteDate() == null) {
                response.put("is_default_password",
DEFAULT_ADMIN_PASSWORD.equals(adminUser.getPassword()));
            }
        } else {
            // something

            response.put("is_default_password",
hasBaseFunction(BaseFunction.ADMIN) &&
DEFAULT_ADMIN_PASSWORD.equals(user.getPassword()));
        }

        return Response.ok().entity(response).build();
    }

    /**
     * Creates a new user.
     *
     * @param username User's username
     * @param password Password
     * @param email E-Mail
     * @param localeId Locale ID
     * @return Response
     * @throws JSONException
     */
    @PUT
    @Produces(MediaType.APPLICATION_JSON)
    public Response register(
        @FormParam("username") String username,
        @FormParam("password") String password,
        @FormParam("locale") String localeId,
        @FormParam("email") String email) throws JSONException {
```

```
        // ... something ...

        // Create the user
        User user = new User();
        user.setRoleId(DEFAULT_USER_ROLE);

        // ... something ...

        user.setLocaleId(DEFAULT_LOCALE_ID);

        // ... something ...
    }

    // something

}
```

`RequestContextFilter.java`:

```
/**
 * Filter used to process a couple things in the request context.
 *
 * @author jtremeaux
 */
public class RequestContextFilter implements Filter {
    private String DEFAULT_LOCALE_ID = "en";

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Force the locale in order to not depend on the execution
environment
        Locale.setDefault(new Locale(DEFAULT_LOCALE_ID));

        // something
    }

    // ...other methods...
}
```

`TokenBasedSecurityFilter.java`:

```
/**
 * This filter is used to authenticate the user having an active session
via an authentication token stored in database.
 * The filter extracts the authentication token stored in a cookie.
 * If the ocokie exists and the token is valid, the filter injects a
```

```
UserPrincipal into a request attribute.
 * If not, the user is anonymous, and the filter injects a
AnonymousPrincipal into the request attribute.
 *
 * @author jtremeaux
 */
public class TokenBasedSecurityFilter implements Filter {
    // something
    private String DEFAULT_TIMEZONE_ID = "Europe/London";

    /**
     * Inject an anonymous user into the request attributes.
     *
     * @param request HTTP request
     */
    private void injectAnonymousUser(HttpServletRequest request) {
        // something


anonymousPrincipal.setDateTimeZone(DateTimeZone.forID(DEFAULT_TIMEZONE_ID))
;

        // something
    }
}
```

LLM Refactoring:

`Constants.java`:

```java
/**
 * Application constants.
 *
 * @author jtremeaux
 */
public class Constants {
    private Constants() {
        // private constructor to prevent instantiation
    }

    private static final String DEFAULT_LOCALE_ID = "en";
    private static final String DEFAULT_TIMEZONE_ID = "Europe/London";
    private static final String DEFAULT_THEME_ID = "default.less";
    private static final String DEFAULT_ADMIN_PASSWORD =
"$2a$05$6Ny3TjrW3aVAL1or2SlcR.fhuDgPKp5jp.P9fBXwVNePgeLqb4i3C";
    private static final String DEFAULT_USER_ROLE = "user";

    public static String getDefaultLocaleId() {
        return DEFAULT_LOCALE_ID;
    }

    public static String getDefaultTimezoneId() {
        return DEFAULT_TIMEZONE_ID;
    }

    public static String getDefaultThemeId() {
        return DEFAULT_THEME_ID;
    }

    public static String getDefaultAdminPassword() {
        return DEFAULT_ADMIN_PASSWORD;
    }

    public static String getDefaultUserRole() {
        return DEFAULT_USER_ROLE;
    }
}
```

`UserDao.java`:

```java
/**
 * User DAO.
 *
 * @author jtremeaux
 */
public class UserDao {
    ///
    /// ... something ...
    ///

    /**
     * Creates a new user.
     *
     * @param user User to create
     * @return User ID
     * @throws Exception
     */
    public String create(User user) throws Exception {
        ///
        /// ... something ...
        ///

        user.setTheme(Constants.getDefaultThemeId());

        ///
        /// ... something ...
        ///
    }

    ///
    /// ... something ...
    ///
}
```

`LocaleUtil.java`:

```
/**
 * Locale utilities.
 *
 * @author jtremeaux
 */
public class LocaleUtil {
    ///
    /// ... something ...
    ///

    /**
     * Extracts the ID of the locale from the HTTP Accept-Language header.
     *
     * @param acceptLanguageHeader header
     * @return Locale ID
     */
    public static String getLocaleIdFromAcceptLanguage(String
acceptLanguageHeader) {
        ///
        /// ... something ...
        ///
        String localeId = ""; // declare localeId here

        if (StringUtils.isNotBlank(localeId)) {
            ///
            /// ... something ...
            ///

            localeId = Constants.getDefaultLocaleId();

            ///
            /// ... something ...
            ///
        }
        if (StringUtils.isBlank(localeId)) {
            localeId = Constants.getDefaultLocaleId();
        }

        ///
        /// ... something ...
        ///
        return localeId;
    }
}
```

UserCrudResource.java (Changed to UserCrudResource.java after God Smell refactoring):

```
/**
 * User REST resources for CRUD Operations on User.
 *
 * @author jtremeaux
 */
@Path("/user")
public class UserCrudResource extends BaseResource {

    /**
     * Returns the information about the connected user.
     *
     * @return Response
     * @throws JSONException
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response info() throws JSONException {
        JSONObject response = new JSONObject();
        if (!authenticate()) {

            // something

            if (adminUser != null && adminUser.getDeleteDate() == null) {
                response.put("is_default_password",
Constants.getDefaultAdminPassword().equals(adminUser.getPassword()));
            }
        } else {
            // something

            response.put("is_default_password",
hasBaseFunction(BaseFunction.ADMIN) &&
Constants.getDefaultAdminPassword().equals(user.getPassword()));
        }

        return Response.ok().entity(response).build();
    }

    /**
     * Creates a new user.
     *
     * @param username User's username
     * @param password Password
     * @param email E-Mail
     * @param localeId Locale ID
     * @return Response
     * @throws JSONException
     */
    @PUT
    @Produces(MediaType.APPLICATION_JSON)
    public Response register(
        @FormParam("username") String username,
        @FormParam("password") String password,
        @FormParam("locale") String localeId,
        @FormParam("email") String email) throws JSONException {

        // ... something ...

        // Create the user
        User user = new User();
        user.setRoleId(Constants.getDefaultUserRole());

        // ... something ...

        user.setLocaleId(Constants.getDefaultLocaleId());

        // ... something ...
    }

    // something

}
```

`RequestContextFilter.java`:

```
/**
 * Filter used to process a couple things in the request context.
 *
 * @author jtremeaux
 */
public class RequestContextFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // Force the locale in order to not depend on the execution
environment
        Locale.setDefault(new Locale(Constants.getDefaultLocaleId()));

        // something
    }

    // ...other methods...
}
```

TokenBasedSecurityFilter.java:

```
/**
 * This filter is used to authenticate the user having an active session
via an authentication token stored in database.
 * The filter extracts the authentication token stored in a cookie.
 * If the ocokie exists and the token is valid, the filter injects a
UserPrincipal into a request attribute.
 * If not, the user is anonymous, and the filter injects a
AnonymousPrincipal into the request attribute.
 *
 * @author jtremeaux
 */
public class TokenBasedSecurityFilter implements Filter {
    // something

    /**
     * Inject an anonymous user into the request attributes.
     *
     * @param request HTTP request
     */
    private void injectAnonymousUser(HttpServletRequest request) {
        // something


anonymousPrincipal.setDateTimeZone(DateTimeZone.forID(Constants.getDefaultT
imezoneId()));

        // something
    }
}
```

Differences between Manual and LLM-generated Refactored Versions:

Manual Refactoring:

- `Removal of Constants Class`: The manual refactoring involved removing the Constants class and directly using the constants within the classes where they were needed.

- `Constants as Instance Variables`: Constants were converted into instance variables within each class where they were used, such as UserDao, LocaleUtil, UserCrudResource, RequestContextFilter, and TokenBasedSecurityFilter.

- `Duplication of Constants`: Some constants were duplicated across different classes, leading to potential inconsistencies and maintenance overhead.

LLM Refactoring:

- `Introduction of Constants Class`: The LLM-generated refactoring reintroduced the Constants class to encapsulate the constants.
- `Constants as Static Methods`: Constants were defined as static methods within the Constants class, providing a centralized location for accessing them.
- `Elimination of Duplication`: By using a centralized Constants class, duplication of constants across multiple classes was avoided, ensuring consistency and reducing maintenance overhead.
- `Method-level Encapsulation`: Constants were encapsulated within methods, providing clear access points and promoting readability and maintainability.

Evaluation:

Manual Refactoring:

Strengths:

- `Direct control over refactoring process`: Manual refactoring allows developers to have full control over the refactoring process, making it easier to implement specific design decisions or optimizations.
- `Flexibility`: Developers can tailor the refactoring process to suit the specific needs and constraints of the project.
- `Immediate understanding`: Since developers are directly involved in the refactoring process, they have a clear understanding of the changes made and their implications.

Weaknesses:

- `Prone to human error`: Manual refactoring relies on developers to make changes correctly, which can introduce human errors, such as forgetting to update all occurrences of a constant or inadvertently introducing new bugs.
- `Time-consuming`: Manual refactoring can be time-consuming, especially for large codebases or complex refactorings, as developers need to carefully analyze the code and make changes manually.
- `Potential inconsistency`: Without strict guidelines or automated checks, manual refactoring can lead to inconsistencies or deviations from best practices across different parts of the codebase.

LLM Refactoring:

Strengths:

- `Automation`: LLM-generated refactoring automates the process of identifying and implementing refactorings, saving developers time and effort.
- `Consistency`: LLMs ensure consistency by applying refactorings uniformly across the codebase, reducing the risk of inconsistencies or errors introduced by manual refactoring.
- `Encapsulation`: LLM-generated refactorings often encapsulate constants within methods or classes, promoting better encapsulation and code organization.
- `Reduced cognitive load`: LLMs can handle repetitive or mundane refactorings, freeing developers to focus on more complex or creative tasks.

Weaknesses:

- `Lack of context awareness`: LLMs may lack contextual understanding, leading to suboptimal refactorings or failure to consider specific project requirements or constraints.
- `Limited customization`: LLMs may not offer the same level of customization or flexibility as manual refactoring, making it challenging to tailor the refactoring process to suit specific needs.
- `Potential for over-engineering`: LLM-generated refactorings may introduce unnecessary complexity or abstraction if not carefully controlled or reviewed by human developers.

Scenarios:

LLMs Excel:

- `Routine Refactorings`: LLMs excel at performing routine refactorings, such as extracting constants or methods, renaming variables, or optimizing imports, freeing developers from repetitive tasks.
- `Consistency Enforcement`: LLMs are valuable for enforcing coding standards and ensuring consistency across large codebases, where manual enforcement would be impractical.
- `Codebase Initialization`: LLMs can be useful for quickly setting up a new codebase or applying standardized patterns and practices, saving time during project initialization.

Manual Intervention is Preferable:

- `Complex Refactorings`: For complex or domain-specific refactorings requiring deep understanding of the codebase or project domain, manual intervention is often preferable to ensure optimal results.
- `Strategic Design Decisions`: When refactoring involves making strategic design decisions or trade-offs, such as choosing between different architectural patterns or optimization strategies, manual intervention allows developers to exercise judgment and expertise.
- `Legacy Codebases`: In legacy codebases with convoluted or poorly documented code, manual intervention may be necessary to untangle dependencies, clarify intent, and ensure that refactorings align with the broader project goals.

In conclusion, while LLMs offer significant advantages in terms of automation, consistency, and efficiency, manual intervention remains essential for complex refactorings, strategic design decisions, and navigating legacy codebases where human expertise and judgment are indispensable. Combining the strengths of LLMs with human oversight and intervention can lead to optimal refactorings that improve code readability, maintainability, and efficiency