Survey on Multi-language Design Smells

Thank you for agreeing to participate, it will take around 30 minutes to complete.

Study Policy:

- Participation in this study is completely voluntary. If you decide not to participate there will not be any negative consequences. If you decide to participate, you may stop participating at any time and withdraw entirely your participation or you may decide not to answer any specific question.
- Your identity and the data collected thanks to your participation will remain anonymous and will never be released to the public. Only anonymous data (aggregated or not) will be published in scientific articles, ensuring that the data cannot be linked back to a particular participant. The data will be kept by the principal investigator for five years before being destroyed.
- By submitting this survey, you are indicating that you have read the description of the study, are over the age of 18, and that you agree to the terms and consent as described in https://drive.google.com/file/d/1aZfHRCr0bEX0i33l_oQHlS9ui9h6rlC5/view?usp=sharing

If you have any questions, please contact us at mouna.abidi@polymtl.ca

Study Design: The purpose of this study is to investigate the prevalence of design smells related to multi-language systems. These systems are developed using more than one programming language. We aim to investigate the perceived prevalence and impact of the design smells detailed below. Our main goal is to improve the quality of those systems.

Definition of terminologies:

Not Handling Exceptions	The exceptions are not handled, developers generally rely on the exceptions provided by the other language
Assuming Safe Return Value	A value is returned to the other language without being checked. Thus, the interaction between both languages may not be correctly performed
Excessive Inter-language Communication	A wrong partitioning in both languages leads to many calls in a way or the other. It adds complexity takes more time to run and may indicate a bad separation of concerns
Too Much Clustering	The multi-language code is concentrated in a few classes, regardless of their concerns and responsibilities.
Too Much Scattering	Many classes are scarcely used in multi-language communication
Hard Coding Libraries	When different libraries are needed depending on the operating system, they are not loaded with conditions on the operating system, but for instance, with a try-catch mechanism, making it hard to know which library has really been loaded
Local References Abuse	The developer does not manage the memory in the native space properly and does not release local and global references
Memory Management Mismatch	Reference types passed from one language to another are not released in a language that does not handle the management of memory causing memory leaks
Not Caching Objects	A method is called to retrieve a field every time this field is needed, although the field's ID or value could have been cached.
Not Securing Libraries	The code loads a foreign library without any security check or restriction privilege
Not Using Relative Path	A library is loaded using only the name not the path. It cannot be accessed in the same way from everywhere
Excessive Objects	A whole object is passed as an argument, although only some of the fields were needed, and it would have been better for the system performance to pass only these fields
Unused Method Declaration	A method is declared in the host language but not implemented in the foreign language
Unused Method Implementation	A method is declared in the host language and implemented in the foreign language, but never called from the host language
Unused Parameters	Some arguments of a function are used neither in its body nor in the other language.

(Khomh, F., & Gueheneuce, Y. G. (2008, April). Do design patterns impact software quality positively?. In Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on (pp. 274-278).

IEEE.)

− Expandability: The degree to which the design of a system can be extended.

− Simplicity: The degree to which the design of a system can be understood easily.

− Reusability: The degree to which a piece of design can be reused in another design.

− Learnability: The degree to which the code source of a system is easy to learn.

− Understandability: The degree to which the code source can be understood easily.

− Performance: The degree to which the code meets its requirements for timeliness.

− Modularity: The degree to which the implementation of the functions of a system is independent of one another.

Thank you.

Best regards,

*** 1. How often do you encounter the following design smells in your project(s)?**
Please check the definitions provided above before answering this questions

	1 Very Often	2 Often	3 Rarely	N/A
Not Handling Exceptions	○	○	○	○
Assuming Safe Return Value	○	○	○	○
Excessive Inter-language Communication	○	○	○	○
Too Much Clustering	○	○	○	○
Too Much Scattering	○	○	○	○
Hard Coding Libraries	○	○	○	○
Local References Abuse	○	○	○	○
Memory Management Mismatch	○	○	○	○
Not Caching Objects	○	○	○	○
Not Securing Libraries	○	○	○	○
Not Using Relative Path	○	○	○	○
Excessive Objects	○	○	○	○
Unused Method Declaration	○	○	○	○
Unused Method Implementation	○	○	○	○
Unused Parameters	○	○	○	○

*** 2. How do you evaluate the impact of the following design smells in those software quality attributes?**

Please carefully read the definition of the smells provided bellow and the reference provided.
(VN: Very Negative, N: Negative, NS: Not significant/Neutral, P: Positive, and VP: Very Positive)

	Expandability	Simplicity	Reusability	Learnability	Understandability	Performance	Modularity	N/A
Not Handling Exceptions								○
Assuming Safe Return Value								○
Excessive Inter-language Communication								○
Too Much Clustering								○
Too Much Scattering								○
Hard Coding Libraries								○
Local References Abuse								○
Memory Management Mismatch								○
Not Caching Objects								○
Not Securing Libraries								○
Not Using Relative Path								○
Excessive Objects								○
Unused Method Declaration								○
Unused Method Implementation								○
Unused Parameters								○

*** 3. Please rank the following design smells from the most harmful to the less harmful**
(Most harmful to the less harmful: 15 -> 1)

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Not Handling Exceptions

Assuming Safe Return Value

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Excessive Inter-language Communication

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Too Much Clustering

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Too Much Scattering

Hard Coding Libraries

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Local References Abuse

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Memory Management Mismatch

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Not Caching Objects

Not Securing Libraries

Not Using Relative Path

Excessive Objects

Unused Method Declaration

Unused Method Implementation

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Unused Parameters

* 4. Task:

a) In your opinion, does the following code(s) contain any occurrence of design smell(implementation and-or design problem)?

```
JavaMethod::JavaMethod(JNIEnv* env, jobject const& obj, const char* method_name, const char* signature)
{
jclass cls = env->GetObjectClass(obj);
m_method_id = env->GetMethodID(cls, method_name, signature);
REALM_ASSERT_RELEASE_EX(m_method_id, method_name, signature);
}
```

○ Yes ○ No

5. **b) If YES, please provide an explanation or specify the design smell(s) involved?**

6. **c) If YES, (In your opinion,) What is the motivation behind using this specific way of implementation?**

* 7. **d) Please rate the severity of the implementation problem (if any), from 1 (Very Low) to 5 (Very High)**

1 Very Low	2 Low	3 Medium	4 High	5 Very High	N/A
◯	◯	◯	◯	◯	◯

*** 8. e) If YES, would you apply this refactored solution?**

```
JavaMethod::JavaMethod(JNIEnv* env, jobject const& obj, const char* method_name, const char* signature)
{
jclass cls = env->GetObjectClass(obj);
m_method_id = env->GetMethodID(cls, method_name, signature);
if (m_method_id == nullptr {
jclass exception = (*env)->FindClass(env, "java/lang/NoSuchMethodException");
(*env)->ThrowNew(env, exception, "callback(String) not found");
}
REALM_ASSERT_RELEASE_EX(m_method_id, method_name, signature);
}
```

◯ Yes (Refactor with this solution) ◯ Yes (Refactor with an alternative solution)

◯ No (No refactoring)

*** 9. <u>Task:</u>**

a) In your opinion, does the following code(s) contain any occurrence of design smell(implementation and-or design problem)?

```
JNIEXPORT void JNICALL Java_io_realm_SyncManager_nativeSimulateSyncError(JNIEnv* env, jclass, jlong j_app_ptr, jstring local_realm_path, jint err_code,
jstring err_message, jboolean is_fatal)
{
try {
auto app = *reinterpret_cast(j_app_ptr);
JStringAccessor path(env, local_realm_path);
JStringAccessor message(env, err_message);

auto session = app->sync_manager()->get_existing_active_session(path);
if (!session) {
ThrowException(env, IllegalArgument, concat_stringdata("Session not found: ", path));
return;
}
std::error_code code = std::error_code{static_cast(err_code), realm::sync::protocol_error_category()};
}
CATCH_STD()
}
```

◯ Yes ◯ No

10. b) If YES, please provide an explanation or specify the design smell(s) involved?

11. c) If YES, (In your opinion,) What is the motivation behind using this specific way of implementation?

*** 12. d) Please rate the severity of the implementation problem (if any), from 1 (Very Low) to 5 (Very High)**

1 Very Low	2 Low	3 Medium	4 High	5 Very High	N/A
○	○	○	○	○	○

*** 13. e) If YES, would you apply this refactored solution?**

```
JNIEXPORT void JNICALL Java_io_realm_SyncManager_nativeSimulateSyncError(JNIEnv* env, jclass, jlong j_app_ptr, jstring local_realm_path, jint err_code,
jstring err_message)
{
try {
auto app = *reinterpret_cast*>(j_app_ptr);
JStringAccessor path(env, local_realm_path);
JStringAccessor message(env, err_message);

auto session = app->sync_manager()->get_existing_active_session(path);
if (!session) {
ThrowException(env, IllegalArgument, concat_stringdata("Session not found: ", path));
return;
}
std::error_code code = std::error_code{static_cast(err_code), realm::sync::protocol_error_category()};
}
CATCH_STD()
}
```

○ Yes (Refactor with this solution) ○ Yes (Refactor with an alternative solution)

○ No (No refactoring)

*** 14. <u>Task:</u>**

a) In your opinion, does the following code(s) contain any occurrence of design smell(implementation and-or design problem)?

```
private static String loadCorrectLibrary(String... libraryCandidateNames) {
for (String libraryCandidateName : libraryCandidateNames) {
try {
System.loadLibrary(libraryCandidateName);
return libraryCandidateName;
} catch (Throwable ignored) {
}
}
return null;
}
```

○ Yes ○ No

15. b) If YES, please provide an explanation or specify the design smell(s) involved?

16. c) If YES, (In your opinion,) What is the motivation behind using this specific way of implementation?

*** 17. d) Please rate the severity of the implementation problem (if any), from 1 (Very Low) to 5 (Very High)**

1 Very Low	2 Low	3 Medium	4 High	5 Very High	N/A
○	○	○	○	○	○

*** 18. e) If YES, would you apply this refactored solution?**

```
private static String loadCorrectLibrary(String... libraryCandidateNames) {
for (String libraryCandidateName : libraryCandidateNames) {
try {
static {
AccessController.doPrivileged( new PrivilegedAction() {
public Void run() {
System.loadLibrary("libraryCandidateName");
return libraryCandidateName; } } );
}
} catch (Throwable ignored) {
}
}
return null;
}
```

○ Yes (Refactor with this solution) ○ Yes (Refactor with an alternative solution)

○ No (No refactoring)

*** 19. Task:**

a) In your opinion, does the following code(s) contain any occurrence of design smell(implementation and-or design problem)?

```
JavaGlobalRefByMove JavaClass::get_jclass(JNIEnv* env, const char* class_name)
{
jclass cls = env->FindClass(class_name);
REALM_ASSERT_RELEASE_EX(cls, class_name);

JavaGlobalRefByMove cls_ref(env, cls, true);
return cls_ref;
}
```

○ Yes ○ No

20. b) If YES, please provide an explanation or specify the design smell(s) involved?

21. c) If YES, (In your opinion,) What is the motivation behind using this specific way of implementation?

* 22. d) Please rate the severity of the implementation problem (if any), from 1 (Very Low) to 5 (Very High)

1 Very Low	2 Low	3 Medium	4 High	5 Very High	N/A
○	○	○	○	○	○

* 23. e) If YES, would you apply this refactored solution?

```
JavaGlobalRefByMove JavaClass::get_jclass(JNIEnv* env, const char* class_name)
{
jclass cls = env->FindClass(class_name);
REALM_ASSERT_RELEASE_EX(cls, class_name);

JavaGlobalRefByMove cls_ref(env, cls, true);

if (cls_ref != nullptr){
return cls_ref;
}}
```

○ Yes (Refactor with this solution) ○ Yes (Refactor with an alternative solution)
○ No (No refactoring)

* 24. Task:

a) In your opinion, does the following code(s) contain any occurrence of design smell(implementation and-or design problem)?

```
{
if (j_custom_headers_array) {
jsize count = env->GetArrayLength(j_custom_headers_array);
for (int i = 0; i < count; i = i + 2) {
JStringAccessor key(env, (jstring) env->GetObjectArrayElement(j_custom_headers_array, i));
JStringAccessor value(env, (jstring) env->GetObjectArrayElement(j_custom_headers_array, i + 1));
config.sync_config->custom_http_headers[std::string(key)] = std::string(value);
}
}
return to_jstring(env, "");
}
CATCH_STD()
return nullptr;
}
}
```

○ Yes ○ No

25. **b) If YES, please provide an explanation or specify the design smell(s) involved?**

26. **c) If YES, (In your opinion,) What is the motivation behind using this specific way of implementation?**

* 27. **d) Please rate the severity of the implementation problem (if any), from 1 (Very Low) to 5 (Very High)**

1 Very Low	2 Low	3 Medium	4 High	5 Very High	N/A
○	○	○	○	○	○

* 28. **e) If YES, would you apply this refactored solution?**

```
{
if (j_custom_headers_array) {
jsize count = env->GetArrayLength(j_custom_headers_array);
for (int i = 0; i < count; i = i + 2) {
JStringAccessor key(env, (jstring) env->GetObjectArrayElement(j_custom_headers_array, i));
JStringAccessor value(env, (jstring) env->GetObjectArrayElement(j_custom_headers_array, i + 1));
config.sync_config->custom_http_headers[std::string(key)] = std::string(value);
}
}
return to_jstring(env, "");
}
CATCH_STD()
return nullptr;
}

(*env)->ReleaseObjectArrayElement(env, j_custom_headers_array, value, key);
}
```

○ Yes (Refactor with this solution)
○ Yes (Refactor with an alternative solution)
○ No (No refactoring)

* 29. **Task:**

a) In your opinion, does the following code(s) contain any occurrence of design smell(implementation and-or design problem)?

```
public OsSyncUser[] allUsers() {
long[] nativeUsers = nativeGetAllUsers(nativePtr);
OsSyncUser[] osSyncUsers = new OsSyncUser[nativeUsers.length];
for (int i = 0; i < nativeUsers.length; i++) {
osSyncUsers[i] = new OsSyncUser(nativeUsers[i]);
}
return osSyncUsers;
}
```

Yes No

30. b) If YES, please provide an explanation or specify the design smell(s) involved?

31. c) If YES, (In your opinion,) What is the motivation behind using this specific way of implementation?

*** 32. d) Please rate the severity of the implementation problem (if any), from 1 (Very Low) to 5 (Very High)**

1 Very Low	2 Low	3 Medium	4 High	5 Very High	N/A
○	○	○	○	○	○

*** 33. e) If YES, would you apply this refactored solution?**

```
public OsSyncUser[] allUsers() {
long[] nativeUsers = nativeGetAllUsers(nativePtr);
OsSyncUser[] osSyncUsers = new OsSyncUser[nativeUsers.length];
for (int i = 0; i < nativeUsers.length; i++) {
osSyncUsers[i] = new OsSyncUser(nativeUsers[i]);
}
if osSyncUsers !== null)){
return osSyncUsers;
}}
```

Yes (Refactor with this solution) Yes (Refactor with an alternative solution)
No (No refactoring)

*** 34. <u>Task:</u>**

a) In your opinion, does the following code(s) contain any occurrence of design smell(implementation and-or design problem)?

```
class Table{
public static native void nativeSetLong(long nativeTableRefPtr, long columnKey, long rowKey, long value, boolean isDefault);
public static native void nativeSetBoolean(long nativeTableRefPtr, long columnKey, long rowKey, boolean value, boolean isDefault);
public static native void nativeSetFloat(long nativeTableRefPtr, long columnKey, long rowKey, float value, boolean isDefault);
public static native void nativeSetDouble(long nativeTableRefPtr, long columnKey, long rowKey, double value, boolean isDefault);
public static native void nativeSetTimestamp(long nativeTableRefPtr, long columnKey, long rowKey, long dateTimeValue, boolean isDefault);
public static native void nativeSetString(long nativeTableRefPtr, long columnKey, long rowKey, String value, boolean isDefault);
public static native void nativeSetNull(long nativeTableRefPtr, long columnKey, long rowKey, boolean isDefault);
public static native void nativeSetByteArray(long nativePtr, long columnKey, long rowKey, byte[] data, boolean isDefault);
public static native void nativeSetDecimal128(long nativeTableRefPtr, long columnKey, long rowKey, long low, long high, boolean isDefault);
public static native void nativeSetObjectId(long nativeTableRefPtr, long columnKey, long rowKey, String data, boolean isDefault);
public static native void nativeSetLink(long nativeTableRefPtr, long columnKey, long rowKey, long value, boolean isDefault);
private static native boolean nativeSetEmbedded(long nativeTableRefPtr, boolean isEmbedded);
private native long nativeAddColumn(long nativeTableRefPtr, int type, String name, boolean isNullable);
private native long nativeAddPrimitiveListColumn(long nativeTableRefPtr, int type, String name, boolean isNullable);
private native long nativeAddColumnLink(long nativeTableRefPtr, int type, String name, long targetTablePtr);
private native void nativeRenameColumn(long nativeTableRefPtr, long columnKey, String name);
private native void nativeRemoveColumn(long nativeTableRefPtr, long columnKey);
private native boolean nativeIsColumnNullable(long nativePtr, long columnKey);
private native void nativeConvertColumnToNullable(long nativeTableRefPtr, long columnKey, boolean isPrimaryKey);
private native void nativeAddSearchIndex(long nativePtr, long columnKey);
private native void nativeRemoveSearchIndex(long nativePtr, long columnKey);
private native boolean nativeHasSearchIndex(long nativePtr, long columnKey);
private native boolean nativeIsNullLink(long nativePtr, long columnKey, long rowKey);
public static native void nativeNullifyLink(long nativePtr, long columnKey, long rowKey);
private native boolean nativeIsNull(long nativePtr, long columnKey, long rowKey);
private native boolean nativeHasSameSchema(long thisTable, long otherTable);
private static native long nativeFreeze(long frozenSharedRealmPtr, long nativeTableRefPtr);
private static native boolean nativeIsEmbedded(long nativeTableRefPtr);
public static native void nativeIncrementLong(long nativeTableRefPtr, long columnKey, long rowKey, long value);
private native void nativeConvertColumnToNotNullable(long nativePtr, long columnKey, boolean isPrimaryKey);
private native long nativeSize(long nativeTableRefPtr);
private native void nativeClear(long nativeTableRefPtr);
private native boolean nativeIsValid(long nativeTableRefPtr);
private native long nativeCountLong(long nativePtr, long columnKey, long value);
private native long nativeCountFloat(long nativePtr, long columnKey, float value);
private native long nativeCountDouble(long nativePtr, long columnKey, double value);
private native long nativeCountString(long nativePtr, long columnKey, String value);
private native long nativeWhere(long nativeTableRefPtr);
public static native long nativeFindFirstInt(long nativeTableRefPtr, long columnKey, long value);
private native long nativeFindFirstBool(long nativePtr, long columnKey, boolean value);
private native long nativeFindFirstFloat(long nativePtr, long columnKey, float value);
private native long nativeFindFirstDouble(long nativePtr, long columnKey, double value);
private native long nativeFindFirstTimestamp(long nativeTableRefPtr, long columnKey, long dateTimeValue);
public static native long nativeFindFirstString(long nativeTableRefPtr, long columnKey, String value);
public static native long nativeFindFirstDecimal128(long nativeTableRefPtr, long columnKey, long low, long high);
public static native long nativeFindFirstObjectId(long nativeTableRefPtr, long columnKey, String value);
public static native long nativeFindFirstNull(long nativeTableRefPtr, long columnKey);
}
```

○ Yes ○ No

35. b) If YES, please provide an explanation or specify the design smell(s) involved?

36. c) If YES, (In your opinion,) What is the motivation behind using this specific way of implementation?

* 37. d) Please rate the severity of the implementation problem (if any), from 1 (Very Low) to 5 (Very High)

1 Very Low	2 Low	3 Medium	4 High	5 Very High	N/A

○	○	○	○	○	○

*** 38. e) If YES, would you apply this refactored solution?**

```
class NativeGetter{
private native long nativeGetColumnCount(long nativeTableRefPtr);
private native String nativeGetColumnName(long nativeTableRefPtr, long columnKey);
private native String[] nativeGetColumnNames(long nativeTableRefPtr);
private native long nativeGetColumnKey(long nativeTableRefPtr, String columnName);
private native int nativeGetColumnType(long nativeTableRefPtr, long columnKey);
private native void nativeMoveLastOver(long nativeTableRefPtr, long rowKey);
private native long nativeGetLong(long nativeTableRefPtr, long columnKey, long rowKey);
private native boolean nativeGetBoolean(long nativeTableRefPtr, long columnKey, long rowKey);
private native float nativeGetFloat(long nativeTableRefPtr, long columnKey, long rowKey);
private native double nativeGetDouble(long nativeTableRefPtr, long columnKey, long rowKey);
private native long nativeGetTimestamp(long nativeTableRefPtr, long columnKey, long rowKey);
private native String nativeGetString(long nativePtr, long columnKey, long rowKey);
private native byte[] nativeGetByteArray(long nativePtr, long columnKey, long rowKey);
private native long nativeGetLink(long nativePtr, long columnKey, long rowKey);
private native long nativeGetLinkTarget(long nativePtr, long columnKey);
private native long[] nativeGetDecimal128(long nativePtr, long columnKey, long rowKey);
private native String nativeGetObjectId(long nativePtr, long columnKey, long rowKey);
private native String nativeGetName(long nativeTableRefPtr);
private static native long nativeGetFinalizerPtr();
native long nativeGetRowPtr(long nativePtr, long objKey);
}

class NativeSetter{
public static native void nativeSetLong(long nativeTableRefPtr, long columnKey, long rowKey, long value, boolean isDefault);
public static native void nativeSetBoolean(long nativeTableRefPtr, long columnKey, long rowKey, boolean value, boolean isDefault);
public static native void nativeSetFloat(long nativeTableRefPtr, long columnKey, long rowKey, float value, boolean isDefault);
public static native void nativeSetDouble(long nativeTableRefPtr, long columnKey, long rowKey, double value, boolean isDefault);
public static native void nativeSetTimestamp(long nativeTableRefPtr, long columnKey, long rowKey, long dateTimeValue, boolean isDefault);
public static native void nativeSetString(long nativeTableRefPtr, long columnKey, long rowKey, String value, boolean isDefault);
public static native void nativeSetNull(long nativeTableRefPtr, long columnKey, long rowKey, boolean isDefault);
public static native void nativeSetByteArray(long nativePtr, long columnKey, long rowKey, byte[] data, boolean isDefault);
public static native void nativeSetDecimal128(long nativeTableRefPtr, long columnKey, long rowKey, long low, long high, boolean isDefault);
public static native void nativeSetObjectId(long nativeTableRefPtr, long columnKey, long rowKey, String data, boolean isDefault);
public static native void nativeSetLink(long nativeTableRefPtr, long columnKey, long rowKey, long value, boolean isDefault);
private static native boolean nativeSetEmbedded(long nativeTableRefPtr, boolean isEmbedded);
}


class ColumnManager{
private native long nativeAddColumn(long nativeTableRefPtr, int type, String name, boolean isNullable);
private native long nativeAddPrimitiveListColumn(long nativeTableRefPtr, int type, String name, boolean isNullable);
private native long nativeAddColumnLink(long nativeTableRefPtr, int type, String name, long targetTablePtr);
private native void nativeRenameColumn(long nativeTableRefPtr, long columnKey, String name);
private native void nativeRemoveColumn(long nativeTableRefPtr, long columnKey);
private native boolean nativeIsColumnNullable(long nativePtr, long columnKey);
private native void nativeConvertColumnToNullable(long nativeTableRefPtr, long columnKey, boolean isPrimaryKey);
private native void nativeAddSearchIndex(long nativePtr, long columnKey);
private native void nativeRemoveSearchIndex(long nativePtr, long columnKey);
private native boolean nativeHasSearchIndex(long nativePtr, long columnKey);
}
```

```
class NativeObject{
private native boolean nativeIsNullLink(long nativePtr, long columnKey, long rowKey);
public static native void nativeNullifyLink(long nativePtr, long columnKey, long rowKey);
private native boolean nativeIsNull(long nativePtr, long columnKey, long rowKey);
private native boolean nativeHasSameSchema(long thisTable, long otherTable);
private static native long nativeFreeze(long frozenSharedRealmPtr, long nativeTableRefPtr);
private static native boolean nativeIsEmbedded(long nativeTableRefPtr);
public static native void nativeIncrementLong(long nativeTableRefPtr, long columnKey, long rowKey, long value);
private native void nativeConvertColumnToNotNullable(long nativePtr, long columnKey, boolean isPrimaryKey);
private native long nativeSize(long nativeTableRefPtr);
private native void nativeClear(long nativeTableRefPtr);
private native boolean nativeIsValid(long nativeTableRefPtr);
}

class Counter{
private native long nativeCountLong(long nativePtr, long columnKey, long value);
private native long nativeCountFloat(long nativePtr, long columnKey, float value);
private native long nativeCountDouble(long nativePtr, long columnKey, double value);
private native long nativeCountString(long nativePtr, long columnKey, String value);
private native long nativeWhere(long nativeTableRefPtr);
}


class Finder{
public static native long nativeFindFirstInt(long nativeTableRefPtr, long columnKey, long value);
private native long nativeFindFirstBool(long nativePtr, long columnKey, boolean value);
private native long nativeFindFirstFloat(long nativePtr, long columnKey, float value);
private native long nativeFindFirstDouble(long nativePtr, long columnKey, double value);
private native long nativeFindFirstTimestamp(long nativeTableRefPtr, long columnKey, long dateTimeValue);
public static native long nativeFindFirstString(long nativeTableRefPtr, long columnKey, String value);
public static native long nativeFindFirstDecimal128(long nativeTableRefPtr, long columnKey, long low, long high);
public static native long nativeFindFirstObjectId(long nativeTableRefPtr, long columnKey, String value);
public static native long nativeFindFirstNull(long nativeTableRefPtr, long columnKey);
}
```

○ Yes (Refactor with this solution) ○ Yes (Refactor with an alternative solution)
○ No (No refactoring)

*** 39.** **Task:**

a) In your opinion, does the following code(s) contain any occurrence of design smell(implementation and-or design problem)?

```
package io.realm.internal;
final class NativeObjectReference extends PhantomReference {
void cleanup() {
synchronized (context) {
nativeCleanUp(nativeFinalizerPtr, nativePtr);
}
referencePool.remove(this);
}
private static native void nativeCleanUp(long nativeFinalizer, long nativePointer);
}

package io.realm.internal;
public class OsSchemaInfo{
private static native long nativeCreateFromList(long[] objectSchemaPtrs);
private static native long nativeGetFinalizerPtr();
private static native long nativeGetObjectSchemaInfo(long nativePtr, String className);
}

package io.realm.internal;
public class OsCollectionChangeSet {
private static native long nativeGetFinalizerPtr();
private static native int[] nativeGetRanges(long nativePtr, int type);
private static native int[] nativeGetIndices(long nativePtr, int type);
}
```

○ Yes ○ No

40. b) If YES, please provide an explanation or specify the design smell(s) involved?

41. c) If YES, (In your opinion,) What is the motivation behind using this specific way of implementation?

*** 42. d) Please rate the severity of the implementation problem (if any), from 1 (Very Low) to 5 (Very High)**

1 Very Low	2 Low	3 Medium	4 High	5 Very High	N/A
○	○	○	○	○	○

*** 43. e) If YES, would you apply this refactored solution?**

```
package io.realm.internal;
public class NativeObject{
void cleanup() {
synchronized (context) {
nativeCleanUp(nativeFinalizerPtr, nativePtr);
}
referencePool.remove(this);
}
private static native void nativeCleanUp(long nativeFinalizer, long nativePointer);
private static native long nativeCreateFromList(long[] objectSchemaPtrs);
private static native long nativeGetFinalizerPtr();
private static native long nativeGetObjectSchemaInfo(long nativePtr, String className);
private static native long nativeGetFinalizerPtr();
private static native int[] nativeGetRanges(long nativePtr, int type);
private static native int[] nativeGetIndices(long nativePtr, int type);
}
```

○ Yes (Refactor with this solution) ○ Yes (Refactor with an alternative solution)
○ No (No refactoring)

*** 44. Task:**

a) In your opinion, does the following code(s) contain any occurrence of design smell(implementation and-or design problem)?

```
private static native long[] nativeCreate(long nativeSharedRealmPtr, long nativeRowPtr, long columnKey);

public OsList(UncheckedRow row, long columnKey) {
OsSharedRealm sharedRealm = row.getTable().getSharedRealm();
long[] ptrs = nativeCreate(sharedRealm.getNativePtr(), row.getNativePtr(), columnKey);

this.nativePtr = ptrs[0];
this.context = sharedRealm.context;
context.addReference(this);

if (ptrs[1] != 0) {
targetTable = new Table(sharedRealm, ptrs[1]);
} else {
targetTable = null;
}
}
```

○ Yes ○ No

45. b) If YES, please provide an explanation or specify the design smell(s) involved?

46. c) If YES, (In your opinion,) What is the motivation behind using this specific way of implementation?

*** 47. d) Please rate the severity of the implementation problem (if any), from 1 (Very Low) to 5 (Very High)**

1 Very Low	2 Low	3 Medium	4 High	5 Very High	N/A
○	○	○	○	○	○

*** 48. e) If YES, would you apply this refactored solution?**

```
private static native long[] nativeCreate(long nativeSharedRealmPtr, long nativeRowPtr, long columnKey);

public OsList(UncheckedRow row, long columnKey) {
OsSharedRealm sharedRealm = row.getTable().getSharedRealm();
long[] ptrs = nativeCreate(sharedRealm.getNativePtr(), row.getNativePtr(), columnKey);

this.nativePtr = ptrs[0];
this.context = sharedRealm.context;
context.addReference(this);

if (ptrs[1] != 0) {
targetTable = new Table(sharedRealm, ptrs[1]);
} else {
targetTable = null;
}
}
```

○ Yes (Refactor with this solution) ○ Yes (Refactor with an alternative solution)

○ No (No refactoring)

* **49.** **Task:**

a) In your opinion, does the following code(s) contain any occurrence of design smell(implementation and-or design problem)?

```
private native void nativeBetween(long nativeQueryPtr, long[] columnIndex, float value1, float value2);
private native void nativeBetweenTimestamp(long nativeQueryPtr, long[] columnIndex, long value1, long value2);

public TableQuery between(long[] columnKey, Date value1, Date value2) {
if (value1 == null || value2 == null) {
throw new IllegalArgumentException("Date values in query criteria must not be null.");
}
nativeBetweenTimestamp(nativePtr, columnKey, value1.getTime(), value2.getTime());
queryValidated = false;
return this;
}
//C++
JNIEXPORT void JNICALL Java_io_realm_internal_TableQuery_nativeBetweenTimestamp(JNIEnv* env, jobject,
jlong nativeQueryPtr,
jlongArray columnKeys,
jlong value1, jlong value2)
{
JLongArrayAccessor col_key_arr(env, columnKeys);
jsize arr_len = col_key_arr.size();
try {
if (arr_len == 1) {
if (!TYPE_VALID(env, Q(nativeQueryPtr)->get_table(), col_key_arr[0], type_Timestamp)) {
return;
}
Q(nativeQueryPtr)
->greater_equal(ColKey(col_key_arr[0]), from_milliseconds(value1))
.less_equal(ColKey(col_key_arr[0]), from_milliseconds(value2));
}
else {
ThrowException(env, IllegalArgument, "between() does not support queries using child object fields.");
}
}
CATCH_STD()
}
```

○ Yes ○ No

50. **b) If YES, please provide an explanation or specify the design smell(s) involved?**

51. **c) If YES, (In your opinion,) What is the motivation behind using this specific way of implementation?**

*** 52. d) Please rate the severity of the implementation problem (if any), from 1 (Very Low) to 5 (Very High)**

1 Very Low	2 Low	3 Medium	4 High	5 Very High	N/A
○	○	○	○	○	○

*** 53. e) If YES, would you apply this refactored solution?**

```
private native void nativeBetweenTimestamp(long nativeQueryPtr, long[] columnIndex, long value1, long value2);

public TableQuery between(long[] columnKey, Date value1, Date value2) {
if (value1 == null || value2 == null) {
throw new IllegalArgumentException("Date values in query criteria must not be null.");
}
nativeBetweenTimestamp(nativePtr, columnKey, value1.getTime(), value2.getTime());
queryValidated = false;
return this;
}
//C++
JNIEXPORT void JNICALL Java_io_realm_internal_TableQuery_nativeBetweenTimestamp(JNIEnv* env, jobject,
jlong nativeQueryPtr,
jlongArray columnKeys,
jlong value1, jlong value2)
{
JLongArrayAccessor col_key_arr(env, columnKeys);
jsize arr_len = col_key_arr.size();
try {
if (arr_len == 1) {
if (!TYPE_VALID(env, Q(nativeQueryPtr)->get_table(), col_key_arr[0], type_Timestamp)) {
return;
}
Q(nativeQueryPtr)
->greater_equal(ColKey(col_key_arr[0]), from_milliseconds(value1))
.less_equal(ColKey(col_key_arr[0]), from_milliseconds(value2));
}
else {
ThrowException(env, IllegalArgument, "between() does not support queries using child object fields.");
}
}
CATCH_STD()
}
```

○ Yes (Refactor with this solution) ○ Yes (Refactor with an alternative solution)

○ No (No refactoring)

*** 54. Task:**

a) In your opinion, does the following code(s) contain any occurrence of design smell(implementation and-or

design problem)?

```
public class OsObjectBuilder implements Closeable {

private static native void nativeAddByteArrayListItem(long listPtr, byte[] val);
private static native void nativeAddDateListItem(long listPtr, long val);
private static native void nativeAddObjectListItem(long listPtr, long rowPtr);


private static ItemCallback byteArrayItemCallback = new ItemCallback() {
public void handleItem(long listPtr, byte[] item) {
nativeAddByteArrayListItem(listPtr, item);
}
};

private static ItemCallback dateItemCallback = new ItemCallback() {
public void handleItem(long listPtr, Date item) {
nativeAddDateListItem(listPtr, item.getTime());
}
};
}
//// C++

JNIEXPORT void JNICALL Java_io_realm_internal_objectstore_OsObjectBuilder_nativeAddByteArrayListItem
(JNIEnv* env, jclass, jlong list_ptr, jbyteArray j_value)
{
try {
auto data = OwnedBinaryData(JByteArrayAccessor(env, j_value).transform());
const JavaValue value(data);
add_list_element(list_ptr, value);
}
CATCH_STD()
}

JNIEXPORT void JNICALL Java_io_realm_internal_objectstore_OsObjectBuilder_nativeAddDateListItem
(JNIEnv* env, jclass, jlong list_ptr, jlong j_value)
{
try {
const JavaValue value(from_milliseconds(j_value));
add_list_element(list_ptr, value);
}
CATCH_STD()
}

JNIEXPORT void JNICALL Java_io_realm_internal_objectstore_OsObjectBuilder_nativeAddObjectListItem
(JNIEnv* env, jclass, jlong list_ptr, jlong row_ptr)
{
try {
const JavaValue value(reinterpret_cast(row_ptr));
add_list_element(list_ptr, value);
}
CATCH_STD()
}
```

○ Yes ○ No

55. b) If YES, please provide an explanation or specify the design smell(s) involved?

56. c) If YES, (In your opinion,) What is the motivation behind using this specific way of implementation?

*** 57. d) Please rate the severity of the implementation problem (if any), from 1 (Very Low) to 5 (Very High)**

1 Very Low	2 Low	3 Medium	4 High	5 Very High	N/A
○	○	○	○	○	○

*** 58. e) If YES, would you apply this refactored solution?**

```
public class OsObjectBuilder implements Closeable {

private static native void nativeAddByteArrayListItem(long listPtr, byte[] val);
private static native void nativeAddDateListItem(long listPtr, long val);


private static ItemCallback byteArrayItemCallback = new ItemCallback() {
public void handleItem(long listPtr, byte[] item) {
nativeAddByteArrayListItem(listPtr, item);
}
};

private static ItemCallback dateItemCallback = new ItemCallback() {
public void handleItem(long listPtr, Date item) {
nativeAddDateListItem(listPtr, item.getTime());
}
};
}
//// C++

JNIEXPORT void JNICALL Java_io_realm_internal_objectstore_OsObjectBuilder_nativeAddByteArrayListItem
(JNIEnv* env, jclass, jlong list_ptr, jbyteArray j_value)
{
try {
auto data = OwnedBinaryData(JByteArrayAccessor(env, j_value).transform());
const JavaValue value(data);
add_list_element(list_ptr, value);
}
CATCH_STD()
}

JNIEXPORT void JNICALL Java_io_realm_internal_objectstore_OsObjectBuilder_nativeAddDateListItem
(JNIEnv* env, jclass, jlong list_ptr, jlong j_value)
{
try {
const JavaValue value(from_milliseconds(j_value));
add_list_element(list_ptr, value);
}
CATCH_STD()
}
```

○ Yes (Refactor with this solution) ○ Yes (Refactor with an alternative solution)
○ No (No refactoring)

Your responses have been registered!

Thank you for taking the time to complete the survey, your input is valuable to us.