

# 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/1aZfHRCr0bEX0i33I\\_oQHIS9ui9h6rIC5/view?usp=sharing](https://drive.google.com/file/d/1aZfHRCr0bEX0i33I_oQHIS9ui9h6rIC5/view?usp=sharing)

If you have any questions, please contact us at [mouna.abidi@polymtl.ca](mailto: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.

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	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Assuming Safe Return Value	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Excessive Inter-language Communication	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Too Much Clustering	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Too Much Scattering	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Hard Coding Libraries	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Local References Abuse	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Memory Management Mismatch	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Not Caching Objects	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Not Securing Libraries	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Not Using Relative Path	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Excessive Objects	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Unused Method Declaration	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Unused Method Implementation	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Unused Parameters	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

**\* 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 below 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	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Assuming Safe Return Value	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Excessive Inter-language Communication	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Too Much Clustering	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Too Much Scattering	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Hard Coding Libraries	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Local References Abuse	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Memory Management Mismatch	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Not Caching Objects	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Not Securing Libraries	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Not Using Relative Path	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Excessive Objects	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Unused Method Declaration	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Unused Method Implementation	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Unused Parameters	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>

**\* 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

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

Not Using Relative Path

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

Excessive Objects

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

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)?**

```
static jmethodID getLogRecordFoundProxyMethodId(JNIEnv* env) {  
    jclass jclass = getClass(env);  
    static jmethodID mid = env->GetMethodID(  
        jclass, "logRecordFoundProxy", "(Ljava/lang/String;JJ)S");  
    assert(mid != nullptr);  
    return mid;  
}
```

☐ 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?

```
static jmethodID getLogRecordFoundProxyMethodId(JNIEnv* env) {  
    jclass jclazz = getJClass(env);  
    if(jclazz == nullptr) {  
        return nullptr;  
    }  
    static jmethodID mid = env->GetMethodID(  
        jclazz, "logRecordFoundProxy", "(Ljava/lang/String;JJ)S");  
    assert(mid != nullptr);  
    return mid;  
}
```

☐ Yes (Refactor with this solution)

☐ Yes (Refactor with an alternative solution)

☐ No (No refactoring)

\* 9. Task:

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

```
Status WriteBatchInternal::Merge(WriteBatch* b, uint32_t column_family_id, const SliceParts& key, const SliceParts& value) {  
    LocalSavePoint save(b);  
    Status s = CheckSlicePartsLength(key, value);  
    if (!s.ok()) {  
        return s;  
    }  
}
```

☐ 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?

```
Status WriteBatchInternal::Merge(WriteBatch* b, const SliceParts& key, const SliceParts& value) {
  LocalSavePoint save(b);
  Status s = CheckSlicePartsLength(key, value);
  if (!s.ok()) {
    return s;
  }
}
```

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

\* 14. Task:

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

```
static {
  RocksDB.loadLibrary();
}
```

- ☐ 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?

```
static {
  AccessController.doPrivileged( new PrivilegedAction() {
    public Void run() {
      RocksDB.loadLibrary(); } } );
}
```

- ☐ 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)?**

```
static jmethodID getLogRecordFoundProxyMethodId(JNIEnv* env) {
jclass jclass = getJClass(env);
if(jclazz == nullptr) {
return nullptr;
}

static jmethodID mid = env->GetMethodID(
jclazz, "logRecordFoundProxy", "(Ljava/lang/String;JJ)S");
return mid;
}
```

☐ 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?**

```
static jmethodID getLogRecordFoundProxyMethodId(JNIEnv* env) {
jclass jclass = getJClass(env);
if(jclazz == nullptr) {
return nullptr;
}

static jmethodID mid = env->GetMethodID(
jclazz, "logRecordFoundProxy", "(Ljava/lang/String;JJ)S");
if(mid != nullptr) {
return mid;
}
}
```

☐ 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)?**

```
std::vector txn_column_families_helper(
JNIEnv* env, jlongArray jcolumn_family_handles, bool* has_exception) {
std::vector cf_handles;
if (jcolumn_family_handles != nullptr) {
const jsize len_cols = env->GetArrayLength(jcolumn_family_handles);
if (len_cols > 0) {
if (env->EnsureLocalCapacity(len_cols) != 0) {
*has_exception = JNI_TRUE;
return std::vector();
}

jlong* jcfh = env->GetLongArrayElements(jcolumn_family_handles, nullptr);
if (jcfh == nullptr) {
*has_exception = JNI_TRUE;
return std::vector();
}
for (int i = 0; i < len_cols; i++) {
auto* cf_handle =
reinterpret_cast(jcfh[i]);
cf_handles.push_back(cf_handle);
}
env->ReleaseLongArrayElements(jcolumn_family_handles, JNI_ABORT);
}}
return cf_handles;
}
```

☐ 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?**

```

std::vector txn_column_families_helper(
JNIEnv* env, jlongArray jcolumn_family_handles, bool* has_exception) {
std::vector cf_handles;
if (jcolumn_family_handles != nullptr) {
const jsize len_cols = env->GetArrayLength(jcolumn_family_handles);
if (len_cols > 0) {
if (env->EnsureLocalCapacity(len_cols) != 0) {
*has_exception = JNI_TRUE;
return std::vector();
}

jlong* jcfh = env->GetLongArrayElements(jcolumn_family_handles, nullptr);
if (jcfh == nullptr) {
*has_exception = JNI_TRUE;
return std::vector();
}
for (int i = 0; i < len_cols; i++) {
auto* cf_handle =
reinterpret_cast(jcfh[i]);
cf_handles.push_back(cf_handle);
}
env->ReleaseLongArrayElements(jcolumn_family_handles, jcfh, JNI_ABORT);
}}
return cf_handles;
}

```

- ☐ 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)?**

```

Void loadLibraryFromJar(final String tmpDir)
throws IOException {
if (!initialized) {
System.load(loadLibraryFromJarToTemp(tmpDir).getAbsolutePath());
initialized = true;
}
}
}

```

- ☐ 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?

```
Void loadLibraryFromJar(final String tmpDir)
throws IOException {
    if (!initialized) {
        System.load(loadLibraryFromJarToTemp(tmpDir).GetFullPathName());
        initialized = true;
    }
}
```

☐ Yes (Refactor with this solution)

☐ Yes (Refactor with an alternative solution)

☐ No (No refactoring)

\* 34. Task:

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

```
static jobject constructWith(JNIEnv* env, const bool direct, const char* buf, const size_t capacity, jclass jbytebuffer_class = nullptr) {
    if (direct) {
        bool allocated = false;
        if (buf == nullptr) {
            buf = new char[capacity];
            allocated = true;
        }
        jobject jbuf = env->NewDirectByteBuffer(const_cast(buf), static_cast(capacity));
        if (jbuf == nullptr) {
            if (allocated) {
                delete[] static_cast(buf);
            }
            return nullptr;
        }
        return jbuf;
    }
}
```

☐ 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?**

```
static jobject constructWith(JNIEnv* env, const bool direct, const char* buf,
const size_t capacity,
jclass jbytebuffer_clazz = nullptr) {
    if (direct) {
        bool allocated = false;
        if (buf == nullptr) {
            buf = new char[capacity];
            allocated = true;
        }
        jobject jbuf = env->NewDirectByteBuffer(const_cast(buf), static_cast(capacity));
        if (jbuf == nullptr) {
            return nullptr;
        }
        return jbuf;
    }
}
```

releaseShortArrayElements(env, array, isCopy);

☐ 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)?**

```
public Options(final DBOptions dbOptions, final ColumnFamilyOptions columnFamilyOptions) {
    super(newOptions(dbOptions.nativeHandle_,
        columnFamilyOptions.nativeHandle_));
    env_ = Env.getDefault();
}
```

☐ 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?

```
public Options(final DBOptions dbOptions, final ColumnFamilyOptions columnFamilyOptions) {  
    super(newOptions(dbOptions.nativeHandle_,columnFamilyOptions.nativeHandle_));  
    if ( columnFamilyOptions != null){  
        env_ = Env.getDefault();  
    }  
}
```

☐ 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)?

```
public class Transaction{  
    private native void setSnapshot(final long handle);  
    private native void setSnapshotOnNextOperation(final long handle);  
    private native void setSnapshotOnNextOperation(final long handle,final long transactionNotifierHandle);  
    private native long getSnapshot(final long handle);  
    private native void clearSnapshot(final long handle);  
    private native void prepare(final long handle) throws RocksDBException;  
    private native byte[] get(final long handle, final long readOptionsHandle,final byte key[], final int keyLength, final long columnFamilyHandle)throws  
    RocksDBException;  
    private native byte[] get(final long handle, final long readOptionsHandle,final byte key[], final int keyLen) throws RocksDBException;  
    private native byte[][] multiGet(final long handle,final long readOptionsHandle, final byte[][] keys,final long[] columnFamilyHandles) throws RocksDBException;  
    private native byte[][] multiGet(final long handle,final long readOptionsHandle, final byte[][] keys)throws RocksDBException;  
    private native byte[] getForUpdate(final long handle, final long readOptionsHandle,final byte key[], final int keyLength, final long columnFamilyHandle, final boolean  
    exclusive,final boolean doValidate) throws RocksDBException;  
    private native byte[] getForUpdate(final long handle, final long readOptionsHandle,final byte key[], final int keyLen, final boolean exclusive, final boolean  
    doValidate)throws RocksDBException;  
    private native byte[][] multiGetForUpdate(final long handle,final long readOptionsHandle, final byte[][] keys,final long[] columnFamilyHandles) throws  
    RocksDBException;  
    private native byte[][] multiGetForUpdate(final long handle,final long readOptionsHandle, final byte[][] keys)throws RocksDBException;  
    private native long getIterator(final long handle,final long readOptionsHandle);  
    private native long getIterator(final long handle,final long readOptionsHandle, final long columnFamilyHandle);  
    private native void put(final long handle, final byte[] key, final int keyLength,final byte[] value, final int valueLength, final long columnFamilyHandle,final boolean  
    assumeTracked) throws RocksDBException;  
    private native void put(final long handle, final byte[] key, final int keyLength, final byte[] value, final int valueLength) throws RocksDBException;  
    private native void put(final long handle, final byte[][] keys, final int keysLength, final byte[][] values, final int valuesLength, final long columnFamilyHandle,final  
    boolean assumeTracked) throws RocksDBException;  
    private native void put(final long handle, final byte[][] keys,final int keysLength, final byte[][] values, final int valuesLength)throws RocksDBException;  
    private native void merge(final long handle, final byte[] key, final int keyLength,final byte[] value, final int valueLength, final long columnFamilyHandle,final boolean  
    assumeTracked) throws RocksDBException;  
    private native void merge(final long handle, final byte[] key,final int keyLength, final byte[] value, final int valueLength)throws RocksDBException;  
    private native void delete(final long handle, final byte[] key, final int keyLength,final long columnFamilyHandle, final boolean assumeTracked) throws  
    RocksDBException;  
    private native void delete(final long handle, final byte[] key,final int keyLength) throws RocksDBException;  
    private native void delete(final long handle, final byte[][] keys, final int keysLength,final long columnFamilyHandle, final boolean assumeTracked) throws  
    RocksDBException;
```

```

private native void delete(final long handle, final byte[][] keys, final int keysLength) throws RocksDBException;
private native void singleDelete(final long handle, final byte[] key, final int keyLength, final long columnFamilyHandle, final boolean assumeTracked) throws RocksDBException;
private native void singleDelete(final long handle, final byte[] key, final int keyLength) throws RocksDBException;
private native void singleDelete(final long handle, final byte[][] keys, final int keysLength, final long columnFamilyHandle, final boolean assumeTracked) throws RocksDBException;
private native void singleDelete(final long handle, final byte[][] keys, final int keysLength) throws RocksDBException;
private native void putUntracked(final long handle, final byte[] key, final int keyLength, final byte[] value, final int valueLength, final long columnFamilyHandle) throws RocksDBException;
private native void putUntracked(final long handle, final byte[] key, final int keyLength, final byte[] value, final int valueLength) throws RocksDBException;
private native void putUntracked(final long handle, final byte[][] keys, final int keysLength, final byte[][] values, final int valuesLength, final long columnFamilyHandle) throws RocksDBException;
private native void putUntracked(final long handle, final byte[][] keys, final int keysLength, final byte[][] values, final int valuesLength) throws RocksDBException;
private native void mergeUntracked(final long handle, final byte[] key, final int keyLength, final byte[] value, final int valueLength, final long columnFamilyHandle) throws RocksDBException;
private native void mergeUntracked(final long handle, final byte[] key, final int keyLength, final byte[] value, final int valueLength) throws RocksDBException;
private native void deleteUntracked(final long handle, final byte[] key, final int keyLength, final long columnFamilyHandle) throws RocksDBException;
private native void deleteUntracked(final long handle, final byte[] key, final int keyLength) throws RocksDBException;
private native void deleteUntracked(final long handle, final byte[][] keys, final int keysLength, final long columnFamilyHandle) throws RocksDBException;
private native void deleteUntracked(final long handle, final byte[][] keys, final int keysLength) throws RocksDBException;
private native void commit(final long handle) throws RocksDBException;
private native void rollback(final long handle) throws RocksDBException;
private native void setSavePoint(final long handle) throws RocksDBException;
private native void rollbackToSavePoint(final long handle) throws RocksDBException;
private native void putLogData(final long handle, final byte[] blob, final int blobLength);
private native void disableIndexing(final long handle);
private native void enableIndexing(final long handle);
private native long getCommitTimeWriteBatch(final long handle);
private native void rebuildFromWriteBatch(final long handle, final long writeBatchHandle) throws RocksDBException;
private native long getNumKeys(final long handle);
private native long getNumPuts(final long handle);
private native long getNumDeletes(final long handle);
private native long getNumMerges(final long handle);
private native long getElapsedTime(final long handle);
private native long getWriteBatch(final long handle);
private native void setLockTimeout(final long handle, final long lockTimeout);
private native long getWriteOptions(final long handle);
private native void setWriteOptions(final long handle, final long writeOptionsHandle);
private native void undoGetForUpdate(final long handle, final byte[] key, final int keyLength, final long columnFamilyHandle);
private native void undoGetForUpdate(final long handle, final byte[] key, final int keyLength);
private native void setName(final long handle, final String name) throws RocksDBException;
private native String getName(final long handle);
private native long getID(final long handle);
private native boolean isDeadlockDetect(final long handle);
private native WaitingTransactions getWaitingTxns(final long handle);
private native byte getState(final long handle);
private native long getID(final long handle);
private native void setLogNumber(final long handle, final long logNumber);
private native long getLogNumber(final long handle);
}

```

☐ 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
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

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

```
class Snapshot{
private native void setSnapshot(final long handle);
private native void setSnapshotOnNextOperation(final long handle);
private native void setSnapshotOnNextOperation(final long handle,final long transactionNotifierHandle);
private native long getSnapshot(final long handle);
private native void clearSnapshot(final long handle);
private native void prepare(final long handle) throws RocksDBException;
private native byte[] get(final long handle, final long readOptionsHandle,final byte key[], final int keyLength, final long columnFamilyHandle)throws
RocksDBException;
private native byte[] get(final long handle, final long readOptionsHandle,final byte key[], final int keyLen) throws RocksDBException;
private native byte[][] multiGet(final long handle,final long readOptionsHandle, final byte[][] keys,final long[] columnFamilyHandles) throws RocksDBException;
private native byte[][] multiGet(final long handle,final long readOptionsHandle, final byte[][] keys)throws RocksDBException;
private native byte[] getForUpdate(final long handle, final long readOptionsHandle,final byte key[], final int keyLength, final long columnFamilyHandle, final boolean
exclusive,final boolean doValidate) throws RocksDBException;
private native byte[] getForUpdate(final long handle, final long readOptionsHandle,final byte key[], final int keyLen, final boolean exclusive, final boolean
doValidate)throws RocksDBException;
private native byte[][] multiGetForUpdate(final long handle,final long readOptionsHandle, final byte[][] keys,final long[] columnFamilyHandles) throws
RocksDBException;
private native byte[][] multiGetForUpdate(final long handle,final long readOptionsHandle, final byte[][] keys)throws RocksDBException;
private native long getIterator(final long handle,final long readOptionsHandle);
private native long getIterator(final long handle,final long readOptionsHandle, final long columnFamilyHandle);
}
class Cleaner{
private native void put(final long handle, final byte[] key, final int keyLength,final byte[] value, final int valueLength, final long columnFamilyHandle,final boolean
assumeTracked) throws RocksDBException;
private native void put(final long handle, final byte[] key, final int keyLength, final byte[] value, final int valueLength) throws RocksDBException;
private native void put(final long handle, final byte[][] keys, final int keysLength, final byte[][] values, final int valuesLength, final long columnFamilyHandle,final
boolean assumeTracked) throws RocksDBException;
private native void put(final long handle, final byte[][] keys,final int keysLength, final byte[][] values, final int valuesLength)throws RocksDBException;
private native void merge(final long handle, final byte[] key, final int keyLength,final byte[] value, final int valueLength, final long columnFamilyHandle,final boolean
assumeTracked) throws RocksDBException;
private native void merge(final long handle, final byte[] key,final int keyLength, final byte[] value, final int valueLength)throws RocksDBException;
private native void delete(final long handle, final byte[] key, final int keyLength,final long columnFamilyHandle, final boolean assumeTracked) throws
RocksDBException;
private native void delete(final long handle, final byte[] key,final int keyLength) throws RocksDBException;
private native void delete(final long handle, final byte[][] keys, final int keysLength,final long columnFamilyHandle, final boolean assumeTracked) throws
RocksDBException;
private native void delete(final long handle, final byte[][] keys,final int keysLength) throws RocksDBException;
private native void singleDelete(final long handle, final byte[] key, final int keyLength,final long columnFamilyHandle, final boolean assumeTracked) throws
RocksDBException;
```

```

private native void singleDelete(final long handle, final byte[] key, final int keyLength) throws RocksDBException;
private native void singleDelete(final long handle, final byte[][] keys, final int keysLength, final long columnFamilyHandle, final boolean assumeTracked) throws RocksDBException;
private native void singleDelete(final long handle, final byte[][] keys, final int keysLength) throws RocksDBException;
}

class UntrackedManager{
private native void putUntracked(final long handle, final byte[] key, final int keyLength, final byte[] value, final int valueLength, final long columnFamilyHandle) throws RocksDBException;
private native void putUntracked(final long handle, final byte[] key, final int keyLength, final byte[] value, final int valueLength) throws RocksDBException;
private native void putUntracked(final long handle, final byte[][] keys, final int keysLength, final byte[][] values, final int valuesLength, final long columnFamilyHandle) throws RocksDBException;
private native void putUntracked(final long handle, final byte[][] keys, final int keysLength, final byte[][] values, final int valuesLength) throws RocksDBException;
private native void mergeUntracked(final long handle, final byte[] key, final int keyLength, final byte[] value, final int valueLength, final long columnFamilyHandle) throws RocksDBException;
private native void mergeUntracked(final long handle, final byte[] key, final int keyLength, final byte[] value, final int valueLength) throws RocksDBException;
private native void deleteUntracked(final long handle, final byte[] key, final int keyLength, final long columnFamilyHandle) throws RocksDBException;
private native void deleteUntracked(final long handle, final byte[] key, final int keyLength) throws RocksDBException;
private native void deleteUntracked(final long handle, final byte[][] keys, final int keysLength, final long columnFamilyHandle) throws RocksDBException;
private native void deleteUntracked(final long handle, final byte[][] keys, final int keysLength) throws RocksDBException;
}

class CommitManager{
private native void commit(final long handle) throws RocksDBException;
private native void rollback(final long handle) throws RocksDBException;
private native void setSavePoint(final long handle) throws RocksDBException;
private native void rollbackToSavePoint(final long handle) throws RocksDBException;
private native void putLogData(final long handle, final byte[] blob, final int blobLength);
private native void disableIndexing(final long handle);
private native void enableIndexing(final long handle);
private native long getCommitTimeWriteBatch(final long handle);
private native void rebuildFromWriteBatch(final long handle, final long writeBatchHandle) throws RocksDBException;
}

class Keys{
private native long getNumKeys(final long handle);
private native long getNumPuts(final long handle);
private native long getNumDeletes(final long handle);
private native long getNumMerges(final long handle);
private native long getElapsedTime(final long handle);
private native long getWriteBatch(final long handle);
private native void setLockTimeout(final long handle, final long lockTimeout);
private native long getWriteOptions(final long handle);
private native void setWriteOptions(final long handle, final long writeOptionsHandle);
private native void undoGetForUpdate(final long handle, final byte[] key, final int keyLength, final long columnFamilyHandle);
private native void undoGetForUpdate(final long handle, final byte[] key, final int keyLength);
}

class Transaction{
private native void setName(final long handle, final String name) throws RocksDBException;
private native String getName(final long handle);
private native long getID(final long handle);
private native boolean isDeadlockDetect(final long handle);
private native WaitingTransactions getWaitingTxns(final long handle);
private native byte getState(final long handle);
private native long getID(final long handle);
private native void setLogNumber(final long handle, final long logNumber);
private native long getLogNumber(final long handle);
}

```

- ☐ 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)?**

```
package org.rocksdb;

public abstract class AbstractCompactionFilter{
private native void disposeInternal(final long handle);
}

package org.rocksdb;
public abstract class AbstractCompactionFilterFactory {
private native long createNewCompactionFilterFactory0();
private native void disposeInternal(final long handle);
}

package org.rocksdb;
public abstract class AbstractComparator {
private native boolean usingDirectBuffers(final long nativeHandle);
private native long createNewComparator(final long comparatorOptionsHandle);
}

package org.rocksdb;
public abstract class AbstractEventListener{
private native long createNewEventListener(final long enabledEventCallbackValues);
private native void disposeInternal(final long handle);
}
```

☐ 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?**

```
package org.rocksdb;

public abstract class AbstractComparator{
private native void disposeInternal(final long handle);
}
```

```

public abstract class AbstractCompactionFilterFactory extends RocksCallbackObject {
private native long createNewCompactionFilterFactory0();
private native void disposeInternal(final long handle);
public abstract class AbstractComparator extends RocksCallbackObject {
private native boolean usingDirectBuffers(final long nativeHandle);
private native long createNewComparator(final long comparatorOptionsHandle);
public abstract class AbstractEventListener extends RocksCallbackObject {
private native long createNewEventListener(final long enabledEventCallbackValues);
private native void disposeInternal(final long handle);
}
}

```

- ☐ 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)?**

```
private static native void add(final long handle, final long compactionJobStatsHandle);
```

```

public List cfPaths() {
final int len = (int) cfPathsLen(nativeHandle__);
if (len == 0) {
return Collections.emptyList();
}
final String paths[] = new String[len];
final long targetSizes[] = new long[len];
cfPaths(nativeHandle__, paths, targetSizes);
final List cfPaths = new ArrayList<>();
for (int i = 0; i < len; i++) {
cfPaths.add(new DbPath(Paths.get(paths[i]), targetSizes[i]));
}
return cfPaths;
}

```

- ☐ 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?

```
private static native void add(final long handle, final long compactionJobStatsHandle, final int len);
```

```
public List cfPaths() {
    final int len = (int) cfPathsLen(nativeHandle__);
    if (len == 0) {
        return Collections.emptyList();
    }
    final String paths[] = new String[len];
    final long targetSizes[] = new long[len];
    cfPaths(nativeHandle__, paths, targetSizes);
    final List cfPaths = new ArrayList<>();
    cfPaths.add(new DbPath(cfPaths, targetSizes[], len));
    return cfPaths;
}
```

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

\* 59. Task:

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

```
private native void putLogData(final long handle, final byte[] blob, final int blobLength);
private native void disableIndexing(final long handle);
private native void enableIndexing(final long handle);
```

```
public void disableIndexing() {
    assert(isOwningHandle());
    disableIndexing(nativeHandle__);
}

public void enableIndexing() {
    assert(isOwningHandle());
    enableIndexing(nativeHandle__);
}
```

```
//cpp
void Java_org_rocksdb_Transaction_disableIndexing(JNIEnv* /*env*/, jobject /*job*/, jlong jhandle) {
    auto* txn = reinterpret_cast<jhandle>();
    txn->DisableIndexing();
}
```

```
void Java_org_rocksdb_Transaction_enableIndexing(JNIEnv* /*env*/, jobject /*job*/, jlong jhandle) {
    auto* txn = reinterpret_cast<jhandle>();
    txn->EnableIndexing();
}
```

- ☐ Yes ☐ No

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

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

\* 62. 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
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

\* 63. e) If YES, would you apply this refactored solution?

```
private native void disableIndexing(final long handle);
private native void enableIndexing(final long handle);
```

```
public void disableIndexing() {
    assert(isOwningHandle());
    disableIndexing(nativeHandle_);
}
```

```
public void enableIndexing() {
    assert(isOwningHandle());
    enableIndexing(nativeHandle_);
}
```

```
//cpp
void Java_org_rocksdb_Transaction_disableIndexing(JNIEnv* /*env*/, jobject /*job*/, jlong jhandle) {
    auto* txn = reinterpret_cast<jhandle>();
    txn->DisableIndexing();
}
```

```
void Java_org_rocksdb_Transaction_enableIndexing(JNIEnv* /*env*/, jobject /*job*/, jlong jhandle) {
    auto* txn = reinterpret_cast<jhandle>();
    txn->EnableIndexing();
}
```

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

\* 64. Task:

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

```
public static DBOptions getDBOptionsFromProps(final Properties properties) {
    DBOptions dbOptions = null;
    final String optionsString = Options.getOptionStringFromProps(properties);
    final long handle = getDBOptionsFromProps(optionsString);
    if (handle != 0) {
        dbOptions = new DBOptions(handle);
    }
}
```

```
return dbOptions;
}
```

☐ Yes

☐ No

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

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

\* 67. 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



\* 68. e) If YES, would you apply this refactored solution?

```
public static DBOptions getDBOptionsFromProps(final Properties properties) {
    DBOptions dbOptions = null;
    final String optionsString = Options.getOptionStringFromProps(properties);
    final long handle = getDBOptionsFromProps(optionsString);
    if (handle != 0) {
        dbOptions = new DBOptions(handle);
    }
    if (handle != null) {
        return dbOptions;
    }
}
```

☐ 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.