# 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<br>Very Often | 2<br>Often | 3<br>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. <u>**Task:**</u>

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

```
void JNICALL Java_i2jrt_TAOObject__1release(JNIEnv *jni, jobject jThis)
{
jclass clazz = findClass(jni, "i2jrt/TAOObject");
jfieldID fid = jni->GetFieldID(clazz, "_jni_ptr", "J");
jlong _jni_ptr = jni->GetLongField(jThis, fid);
CORBA::Object_ptr o = reinterpret_cast(_jni_ptr);
CORBA::release(o);
jni->SetLongField(jThis, fid, reinterpret_cast(CORBA::Object::_nil()));
}
```

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<br>Very Low | 2<br>Low | 3<br>Medium | 4<br>High | 5<br>Very High | N/A |
|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ |

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

```
void JNICALL Java_i2jrt_TAOObject__1release(JNIEnv *jni, jobject jThis)
{
jclass clazz = findClass(jni, "i2jrt/TAOObject");
if (clazz == NULL) {
jclass newExc = env->FindClass("java/lang/NullPointerException");
env->ThrowNew(newExc, "The native object does not exist.");
return 0;}
jfieldID fid = jni->GetFieldID(clazz, "_jni_ptr", "J");
jlong _jni_ptr = jni->GetLongField(jThis, fid);
CORBA::Object_ptr o = reinterpret_cast(_jni_ptr);
CORBA::release(o);
jni->SetLongField(jThis, fid, reinterpret_cast(CORBA::Object::_nil()));
}
```

○ 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)?**

```
STDMETHODIMP RTDServer::GetIDsOfNames(REFIID riid, OLECHAR **rgszNames, UINT cNames, LCID lcid, DISPID *rgDispId)
{
HRESULT hr = E_FAIL;
if (riid != IID_NULL)
return E_INVALIDARG;
if (m_pTypeInfoInterface != NULL)
hr = m_pTypeInfoInterface->GetIDsOfNames(rgszNames, cNames, rgDispId);
return hr;
}
```

○ 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?

```
STDMETHODIMP RTDServer::GetIDsOfNames(REFIID riid,OLECHAR **rgszNames,UINT cNames,DISPID *rgDispId)
{
HRESULT hr = E_FAIL;
if (riid != IID_NULL)
return E_INVALIDARG;
if (m_pTypeInfoInterface != NULL)
hr = m_pTypeInfoInterface->GetIDsOfNames(rgszNames, cNames, rgDispId);
return hr;
}
```

○ 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 {
String library = "opendds-jms-native";
if (Boolean.getBoolean("opendds.native.debug")) {
library = library.concat("d");
}
System.loadLibrary(library);
}
```

○ 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?**

```
public static void loadLibrary
static {
String library = "opendds-jms-native";
if (Boolean.getBoolean("opendds.native.debug")) {
library = library.concat("d");
}
AccessController.doPrivileged( new PrivilegedAction() {
public Void run() {
System.loadLibrary(library);
}} } );
}
```

○ 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)?**

```
jobject JNICALL Java_i2jrt_TAOObject__1duplicate(JNIEnv *jni, jobject jThis)
{
CORBA::Object_ptr ptr = recoverTaoObject(jni, jThis);

if (CORBA::is_nil(ptr)) return 0;

CORBA::Object_ptr dupl = CORBA::Object::_duplicate(ptr);
jclass clazz = jni->GetObjectClass(jThis);
jmethodID ctor = jni->GetMethodID(clazz, "", "(J)V");
return jni->NewObject(clazz, ctor, reinterpret_cast(dupl));
}
```

○ 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<br>Very Low | 2<br>Low | 3<br>Medium | 4<br>High | 5<br>Very High | N/A |
|---|---|---|---|---|---|

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ |

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

```
jobject JNICALL Java_i2jrt_TAOObject__1duplicate(JNIEnv *jni, jobject jThis)
{
CORBA::Object_ptr ptr = recoverTaoObject(jni, jThis);

if (CORBA::is_nil(ptr)) return 0;

CORBA::Object_ptr dupl = CORBA::Object::_duplicate(ptr);
jclass clazz = jni->GetObjectClass(jThis);
jmethodID ctor = jni->GetMethodID(clazz, "", "(J)V");
if (ctor == nullptr) {
LOG_ERROR("failed to find method %s", methodName);
abort();
}
return jni->NewObject(clazz, ctor, reinterpret_cast(dupl));
}
```

○ 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)?**

```
ACE_TCHAR** to_argv(JNIEnv* env, jobjectArray args, jsize len)
{
ACE_TCHAR** argv = new ACE_TCHAR*[len];

argv[0] = const_cast(ACE_TEXT("JAVA"));

for (int i = 1; i < len; ++i) {
jstring arg = reinterpret_cast
(env->GetObjectArrayElement(args, i - 1));
if (arg == 0) {
throw_exception(env, "java/lang/NullPointerException");
return 0;
}
const char* cs = env->GetStringUTFChars(arg, 0);
argv[i] = ACE_OS::strdup(ACE_TEXT_CHAR_TO_TCHAR(cs));

env->ReleaseStringUTFChars(cs);
}

return argv;
}
```

○ 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?**

```
ACE_TCHAR** to_argv(JNIEnv* env, jobjectArray args, jsize len)
{
ACE_TCHAR** argv = new ACE_TCHAR*[len];

argv[0] = const_cast(ACE_TEXT("JAVA"));

for (int i = 1; i < len; ++i) {
jstring arg = reinterpret_cast
(env->GetObjectArrayElement(args, i - 1));
if (arg == 0) {
throw_exception(env, "java/lang/NullPointerException");
return 0;
}
const char* cs = env->GetStringUTFChars(arg, 0);
argv[i] = ACE_OS::strdup(ACE_TEXT_CHAR_TO_TCHAR(cs));

env->ReleaseStringUTFChars(cs);
(*env)->DeleteLocalRef(env, arg);
}

return argv;
}
```

○ 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 NativeLoader(File parent) throws IOException {
logger.debug("Using native directory: %s", parent.GetFullPathName());
this.parent = Files.verifyDirectory(parent);
}
```

○ 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 NativeLoader(File parent) throws IOException {
logger.debug("Using native directory: %s", parent.getAbsolutePath());
this.parent = Files.verifyDirectory(parent);
}
```

○ 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)?**

```
public static char convertToChar(StreamItem item) throws MessageFormatException {

if (item == null) throw new NullPointerException();
final ItemKind itemKind = item.discriminator();
if (compare(itemKind, ItemKind.CHAR_KIND)) {
return item.charValue();
} else {
throw new MessageFormatException("Cannot convert stream item to char");
}
}
```

○ 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<br>Very Low | 2<br>Low | 3<br>Medium | 4<br>High | 5<br>Very High | N/A |
|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ |

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

```
public static char convertToChar(StreamItem item) throws MessageFormatException {

if (item == null) throw new NullPointerException();
final ItemKind itemKind = item.discriminator();
if (ItemKind  !== null)){
if (compare(itemKind, ItemKind.CHAR_KIND)) {
return item.charValue();
}} else {
throw new MessageFormatException("Cannot convert stream item to char");
}
}
```

○ 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 OpenDDS.DCPS.transport;

public class MulticastInst {
public native boolean getDefaultToIPv6();
public native void setDefaultToIPv6(boolean dtip6);

public native short getPortOffset();
public native void setPortOffset(short po);

public native String getGroupAddress();
public native void setGroupAddress(String ga);

public native boolean getReliable();
public native void setReliable(boolean r);

public native double getSynBackoff();
public native void setSynBackoff(double sb);

public native long getSynInterval();
public native void setSynInterval(long si);

public native long getSynTimeout();
public native void setSynTimeout(long st);

public native int getNakDepth();
public native void setNakDepth(int nd);

public native long getNakInterval();
public native void setNakInterval(long ni);

public native int getNakDelayInterval();
public native void setNakDelayInterval(int ndi);

public native int getNakMax();
public native void setNakMax(int nm);

public native long getNakTimeout();
public native void setNakTimeout(long nt);

public native byte getTimeToLive();
public native void setTimeToLive(byte ttl);

public native int getRcvBufferSize();
public native void setRcvBufferSize(int rbs);
}
```

○ 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<br>Very Low | 2<br>Low | 3<br>Medium | 4<br>High | 5<br>Very High | N/A |
|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ |

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

```
 package OpenDDS.DCPS.transport;
public class SynchTime extends MulticastInst {
public native double getSynBackoff();
public native void setSynBackoff(double sb);
public native long getSynInterval();
public native void setSynInterval(long si);
public native long getSynTimeout();
public native void setSynTimeout(long st);
public native byte getTimeToLive();
public native void setTimeToLive(byte ttl);
}

package OpenDDS.DCPS.transport;
public class Memaddress extends MulticastInst {
public native String getGroupAddress();
public native void setGroupAddress(String ga);
public native boolean getReliable();
public native void setReliable(boolean r);
public native int getRcvBufferSize();
public native void setRcvBufferSize(int rbs);
public native boolean getDefaultToIPv6();
public native void setDefaultToIPv6(boolean dtip6);
public native short getPortOffset();
public native void setPortOffset(short po);
}

package OpenDDS.DCPS.transport;
public class NakManager extends MulticastInst {
public native int getNakDepth();
public native void setNakDepth(int nd);
public native long getNakInterval();
public native void setNakInterval(long ni);
public native int getNakDelayInterval();
public native void setNakDelayInterval(int ndi);
public native int getNakMax();
public native void setNakMax(int nm);
public native long getNakTimeout();
public native void setNakTimeout(long nt);
}
```

○ 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)?**

```
package DDS;
public final class DATAREADER_QOS_DEFAULT {
private DATAREADER_QOS_DEFAULT() {}
public static native DataReaderQos get();
}

package DDS;
public final class DATAREADER_QOS_USE_TOPIC_QOS {
private DATAREADER_QOS_USE_TOPIC_QOS() {}
public static native DataReaderQos get();
}

package DDS;
public final class DATAWRITER_QOS_DEFAULT {
private DATAWRITER_QOS_DEFAULT() {}
public static native DataWriterQos get();
}

package DDS;
```

```
public final class DATAWRITER_QOS_USE_TOPIC_QOS {
private DATAWRITER_QOS_USE_TOPIC_QOS() {}
public static native DataWriterQos get();
}
```

○ 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?**

```
package DDS;
public final class DATAREADER_QOS{
private DATAREADER_QOS() {}
public static native DataReaderQos get();
public static native DataWriterQos get();
}
```

○ 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)?**

```
public class TcpInst {
public native String getLocalAddress();
public native void setLocalAddress(String la);
public native boolean isEnableNagleAlgorithm();
public native void setEnableNagleAlgorithm(boolean ena);
}
public class TransportConfigTest {
protected static void testCreateNewTransportUdp() throws Exception {
final String ID = "Udp2";
TransportInst ti =
TheTransportRegistry.create_inst(ID,
TheTransportRegistry.TRANSPORT_UDP);
ti.setMaxPacketSize(999);
UdpInst sui = (UdpInst) ti;
sui.setLocalAddress("0.0.0.0:1234");

TransportInst ti2 = TheTransportRegistry.get_inst(ID);
assert ti2.getMaxPacketSize() == 999;
UdpInst sui2 = (UdpInst) ti2;
assert sui2.getLocalAddress().endsWith(":1234");
}
}

//cpp
jstring JNICALL Java_OpenDDS_DCPS_transport_TcpInst_getLocalAddress
(JNIEnv * jni, jobject jthis)
{
OpenDDS::DCPS::TcpInst_rch inst = OpenDDS::DCPS::rchandle_from(recoverCppObj(jni, jthis));
jstring retStr = jni->NewStringUTF(inst->local_address_string().c_str());
return retStr;
}

void JNICALL Java_OpenDDS_DCPS_transport_TcpInst_setLocalAddress
(JNIEnv * jni, jobject jthis, jstring val)
{
OpenDDS::DCPS::TcpInst_rch inst = OpenDDS::DCPS::rchandle_from(recoverCppObj(jni, jthis));
JStringMgr jsm_val(jni, val);
inst->local_address(jsm_val.c_str());
}
```

○ 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?**

```
public class TcpInst {
public native String getLocalAddress();
public native void setLocalAddress(String la);
}
public class TransportConfigTest {
protected static void testCreateNewTransportUdp() throws Exception {
final String ID = "Udp2";
TransportInst ti =
TheTransportRegistry.create_inst(ID,
TheTransportRegistry.TRANSPORT_UDP);
ti.setMaxPacketSize(999);
UdpInst sui = (UdpInst) ti;
sui.setLocalAddress("0.0.0.0:1234");

TransportInst ti2 = TheTransportRegistry.get_inst(ID);
assert ti2.getMaxPacketSize() == 999;
UdpInst sui2 = (UdpInst) ti2;
assert sui2.getLocalAddress().endsWith(":1234");
}
}
//cpp
jstring JNICALL Java_OpenDDS_DCPS_transport_TcpInst_getLocalAddress
(JNIEnv * jni, jobject jthis)
{
OpenDDS::DCPS::TcpInst_rch inst = OpenDDS::DCPS::rchandle_from(recoverCppObj(jni, jthis));
jstring retStr = jni->NewStringUTF(inst->local_address_string().c_str());
return retStr;
}
void JNICALL Java_OpenDDS_DCPS_transport_TcpInst_setLocalAddress
(JNIEnv * jni, jobject jthis, jstring val)
{
OpenDDS::DCPS::TcpInst_rch inst = OpenDDS::DCPS::rchandle_from(recoverCppObj(jni, jthis));
JStringMgr jsm_val(jni, val);
inst->local_address(jsm_val.c_str());
}
```

○ 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 native int getMaxSamplesPerPacket();
public native void setMaxSamplesPerPacket(int mspp);
public native int getOptimumPacketSize();
public native void setOptimumPacketSize(int ops);

protected static void testModifyTransportFromFileTCP() throws Exception {
final String ID = "tcp1";
TransportInst ti = TheTransportRegistry.get_inst(ID);
assert ti.getMaxSamplesPerPacket() == 5;
TcpInst ti_tcp = (TcpInst) ti;
assert ti_tcp.getConnRetryAttempts() == 42;

ti.setMaxSamplesPerPacket(6);
ti_tcp.setConnRetryAttempts(49);

TransportInst ti2 = TheTransportRegistry.get_inst(ID);
assert ti2.getMaxSamplesPerPacket() == 6;
TcpInst ti_tcp2 = (TcpInst) ti2;
assert ti_tcp2.getConnRetryAttempts() == 49;
}
jint JNICALL Java_OpenDDS_DCPS_transport_TransportInst_getMaxSamplesPerPacket
(JNIEnv * jni, jobject jthis)
{
OpenDDS::DCPS::TransportInst_rch inst = OpenDDS::DCPS::rchandle_from(recoverCppObj(jni, jthis));
return static_cast(inst->max_samples_per_packet_);
}

void JNICALL Java_OpenDDS_DCPS_transport_TransportInst_setMaxSamplesPerPacket
(JNIEnv * jni, jobject jthis, jint val)
{
OpenDDS::DCPS::TransportInst_rch inst = OpenDDS::DCPS::rchandle_from(recoverCppObj(jni, jthis));
inst->max_samples_per_packet_ = val;
}

jint JNICALL Java_OpenDDS_DCPS_transport_TransportInst_getQueueInitialPools
(JNIEnv * jni, jobject jthis)
{
OpenDDS::DCPS::TransportInst_rch inst = OpenDDS::DCPS::rchandle_from(recoverCppObj(jni, jthis));
return static_cast(inst->queue_initial_pools_);
}
```

○ 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 native int getMaxSamplesPerPacket();
public native void setMaxSamplesPerPacket(int mspp);

protected static void testModifyTransportFromFileTCP() throws Exception {
final String ID = "tcp1";
TransportInst ti = TheTransportRegistry.get_inst(ID);
assert ti.getMaxSamplesPerPacket() == 5;
TcpInst ti_tcp = (TcpInst) ti;
assert ti_tcp.getConnRetryAttempts() == 42;
ti.setMaxSamplesPerPacket(6);
ti_tcp.setConnRetryAttempts(49);

TransportInst ti2 = TheTransportRegistry.get_inst(ID);
assert ti2.getMaxSamplesPerPacket() == 6;
TcpInst ti_tcp2 = (TcpInst) ti2;
assert ti_tcp2.getConnRetryAttempts() == 49;
}
jint JNICALL Java_OpenDDS_DCPS_transport_TransportInst_getMaxSamplesPerPacket
(JNIEnv * jni, jobject jthis)
{
OpenDDS::DCPS::TransportInst_rch inst = OpenDDS::DCPS::rchandle_from(recoverCppObj(jni, jthis));
return static_cast(inst->max_samples_per_packet_);
}

void JNICALL Java_OpenDDS_DCPS_transport_TransportInst_setMaxSamplesPerPacket
(JNIEnv * jni, jobject jthis, jint val)
{
OpenDDS::DCPS::TransportInst_rch inst = OpenDDS::DCPS::rchandle_from(recoverCppObj(jni, jthis));
inst->max_samples_per_packet_ = val;
}
```

○ 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)?**

```
bool composite_mapping::gen_struct(UTL_ScopedName *name, const vector &fields, const char *repoid)
{
for (vector::iterator it(components_.begin());
it != components_.end(); ++it) {
if (!(*it)->gen_struct(name, fields, repoid))
return false;
}
 return true;
}
```

○ 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<br>Very Low | 2<br>Low | 3<br>Medium | 4<br>High | 5<br>Very High | N/A |
|---|---|---|---|---|---|
| ○ | ○ | ○ | ○ | ○ | ○ |

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

```
bool composite_mapping::gen_struct(UTL_ScopedName *name, const vector<AST_Field *> &fields, const char *repoid)
{
for (vector<idl_mapping *>::iterator it(components_.begin());
it != components_.end(); ++it) {
if it != null {
if (!(*it)->gen_struct(name, fields, repoid))
return false;
}}
 return true;
}
```

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

○ No (No refactoring)

Your responses have been registered!