Warmup Task

Karl Hajal

1 Violations in each example Java file in JavaMOP's examples/agent/many directory

1.1 HasNext_1

Output:

```
! hasNext() has not been called before calling next() for
    an iterator
! hasNext() has not been called before calling next() for
    an iterator
! hasNext() has not been called before calling next() for
    an iterator
sum: 15
```

The violations occured because next() was called four times in a row after checking hasNext() only once at the beginning. The property in HasNext.mop specifies that hasNext() should always be called before next().

It can be argued that this isn't a bug since the Vector was initialized with 4 elements and we accessed exactly four consecutive elements right after that. However, the violations help us think of a better and safer way to do the same thing which would work independently of how the Vector was initialized:

```
while(i.hasNext()) {
  sum += (Integer)i.next();
}
```

1.2 HasNext_2

Output:

```
sum: 15
```

There are no violations here since the code was improved to look like what was suggested earlier.

1.3 HasNext_3

Output:

```
! hasNext() has not been called before calling next() for
    an iterator
sum = 3
! hasNext() has not been called before calling next() for
    an iterator
```

The violations occur since after each hasNext() check, next() is called twice. This is definitely a bug since we are looping with hasNext() as a condition; if the Vector has an odd number of elements, the program fails.

1.4 SafeEnum_1

Output:

```
improper enumeration usage at SafeEnum_1.main(SafeEnum_1.
    java:24)
sum: 26
```

The violation occured given that 11 was added to the Vector while the Enumeration associated to it was still in use and used to call nextElement() after the Vector's modification.
Even though we happened to get a correct output (assuming that we wanted to include the 11 in the sum), modifying a collection during iteration is unsafe. Modifying the collection during iteration leads to unspecified behavior, which is unacceptable.

1.5 SafeEnum_2

Output:

```
sum: 15
```

No violations here given that 11 was added after having iterated through the entire Vector and was excluded from the sum.

1.6 SafeFile_1

Output:

```
begin
open
close
end
begin
open
```

```
end
improper use of files
improper use of files
begin
close
improper use of files
end
improper use of files
improper use of files
Program has ended.
Program has ended.
```

The violations here occur given that the filereader is used to open a file in one method, and then closes the file in another. This violates the software engineering practice of closing files within the method they were opened.

1.7 SafeFile_2

Output:

```
begin
open
close
end
begin
open
close
end
begin
open
close
end
begin
open
close
end
begin
open
close
end
begin
open
close
end
begin
open
close
end
begin
open
close
end
begin
open
```

```
close
end
begin
open
close
end
Program has ended.
Program has ended.
Program has ended.
Program has ended.
Program has ended.
Program has ended.
Program has ended.
Program has ended.
Program has ended.
```

No violations

1.8 SafeMapIterator_1

Output:

```
unsafe iterator usage!
unsafe iterator usage!
! hasNext() has not been called before calling next() for
    an iterator
java found the problem too
```

There are two kinds of violations:

1. The first one is due to adding an element to the map and then iterating over the set of keys using an iterator that was previously instantiated. This is a bug that was caught by java as well.

2. The second one is due to calling next() without checking hasNext() first. This can be considered not to be a bug since there is no loop, and the Map was initialized with 3 elements before this call and the developer is aware of that.

1.9 SafeMapIterator_2

Output:

```
! hasNext() has not been called before calling next() for
    an iterator
Bar
```

The violation here is similar the second one above. It isn't really a bug since the Map was explicitly initialized with two elements and next() was called once; a hasNext() check would've been redundant from the eyes of the developer.

2 Violations in apache commons-fileupload

1. Specification Closeable_MeaninglessClose has been violated on line org.apache.commons.fileupload.util.Streams.copy(Streams.java:111).

   The violation occured because close() was called on an instance of ByteArrayOutputStream, which has no effect. This violation makes sure the user understands that calling close() in some classes (ByteArrayInputStream, ByteArrayOutputStream, CharArrayWriter, StringWriter) is meaningless.
   This can potentially help the user understand the code's behavior since methods can be called following close() without any IOExceptions.

2. Specification Closeable_MeaninglessClose has been violated on line org.apache.commons.io.IOUtils.closeQuietly(IOUtils.java:285).

   Same as the above.

3. Specification Closeable_MultipleClose has been violated on line org.apache.commons.fileupload.FileUploadBase$FileItemIteratorImpl$FileItemStreamImpl.close(FileUploadBase.java:870).

   This violation occured due to closing the same Closeable twice. Although this isn't harmful, it can help developers understand how their code is functioning, which can help find other bugs that are occuring.

4. Specification Closeable_MultipleClose has been violated on line org.apache.commons.fileupload.MultipartStream$ItemInputStream.close(MultipartStream.java:950).

   Same as the above.

5. Specification Dictionary_Obsolete has been violated on line org.apache.commons.fileupload.portlet.MockPortletActionRequest.<init>(MockPortletActionRequest.java:68).

   This violation occured due to the usage of the Hashtable class which extends Dictionary. The Dictionary class is obsolete in favour of the Map class.

6. Specification Integer_StaticFactory has been violated on line org.apache.commons.fileupload.ProgressListenerTest$ProgressListenerImpl.update(ProgressListenerTest.java:59).

This violation occured due to the usage of the constructor of Integer. While this does not cause a glitch, the static factory is recommended instead for performance reasons.

7. `Specification Iterator_HasNext has been violated on line org.apache.commons.fileupload.FileItemHeadersTest.testFileItemHeaders(FileItemHeadersTest.java:50).`

   The violation occured because next() was called without checking hasNext() first. This is not a bug since the collection was explicitly initialized with 6 elements whose values are then being used to test properties of FileItemHeaders.

8. `Specification Iterator_HasNext has been violated on line org.apache.commons.fileupload.SizesTest.testFileUpload(SizesTest.java:77).`

   Here also a file is being written to a certain number of times, and then next() is being called this exact number of times before finally checking that hasNext() returns false. While this can be considered not to be a bug, the test could have been written in a safer way with hasNext() as the condition to keep looping while counting the iterations, which would have made sure that the spec isn't violated.

9. `Specification Iterator_HasNext has been violated on line org.apache.commons.fileupload.StreamingTest.testFileUpload(StreamingTest.java:55).`

   This is also similar to the previous case where the developer seems to know the exact number of times the iterations should happen and checks after all is done that hasNext() returns false. Again, hasNext() could have been used as the condition to keep looping while counting the iterations and then checking then asserting the count.

10. `Specification Long_BadParsingArgs has been violated on line org.apache.commons.fileupload.FileUploadBase$FileItemIteratorImpl.getContentLength(FileUploadBase.java:1093).`

    This violation occured because a seemingly empty header was passed to Long.parselong(String s). According to documentation, s should neither be null nor of length 0, so this would have been a bug if the code wasn't enclosed in a try catch block which will catch a NumberFormatException whenever the passed string cannot be parsed properly into a Long.

11. Specification Long_BadParsingArgs has been violated on line org.apache.commons.fileupload.portlet.PortletRequestContext.contentLength(PortletRequestContext.java:101).

    Same as the above with the method being enclosed in a try catch block which will catch the NumberFormatException.

12. Specification Long_BadParsingArgs has been violated on line org.apache.commons.fileupload.servlet.ServletRequestContext.contentLength(ServletRequestContext.java:99).

    Same as the above.

13. Specification Long_StaticFactory has been violated on line org.apache.commons.fileupload.ProgressListenerTest$ProgressListenerImpl.update(ProgressListenerTest.java:57).

    The violation occurs because Long's constructor is used instead of the static factory which is preferred for performance reasons.

14. Specification ObjectOutput_Close has been violated on line (Unknown).

    The violation tells us that an ObjectOutput instance wasn't closed. This is definitely a bug, but we are unfortunately not told where it occured.

15. Specification OutputStream_ManipulateAfterClose has been violated on line org.apache.commons.io.output.ThresholdingOutputStream.close(ThresholdingOutputStream.java:158).

    The violation occured because flush() was called after the OutputStream was closed, which does not throw an IOException. According to the documentation, that is a bug.

16. Specification Serializable_UID has been violated on line org.apache.commons.io.filefilter.AndFileFilter.<clinit>(AndFileFilter.java:1).
    Specification Serializable_UID has been violated on line org.apache.commons.io.filefilter.DirectoryFileFilter.<clinit>(DirectoryFileFilter.java:54).
    Specification Serializable_UID has been violated on line org.apache.commons.io.filefilter.NameFileFilter.<clinit>(NameFileFilter.java:1).

```
Specification Serializable_UID has been violated on
    line org.apache.commons.io.filefilter.NotFileFilter
    .<clinit>(NotFileFilter.java:1).
Specification Serializable_UID has been violated on
    line org.apache.commons.io.filefilter.OrFileFilter.<
    clinit>(OrFileFilter.java:1).
Specification Serializable_UID has been violated on
    line org.apache.commons.io.filefilter.TrueFileFilter
    .<clinit>(TrueFileFilter.java:42).
Specification Serializable_UID has been violated on
    line org.apache.commons.io.output.
    StringBuilderWriter.<clinit>(StringBuilderWriter.
    java:1).
```

These seem to be false alarms since, after checking the code, it seems that all these classes explicitly declare a static final long serialVersionUID.
It was added to them in the following commit:
https://github.com/apache/commons-io/commit/016222a00fbcaf6870f4552d5f27b7a375412340
547861034f3db97643e6f8937062bf25