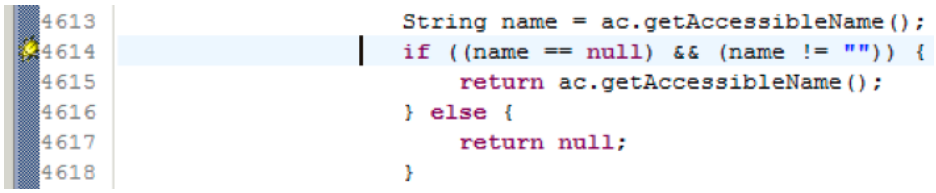
Categories:

- **Pointers/References**
- **Multi-threading**
- **Null/Pointers/References**
- **Dead Code**
- **Generics**
- **Inheritance/Polymorphism**
- **Serialization**
- **Test Coverage**

Notification 1 (string comparison using == or !=):

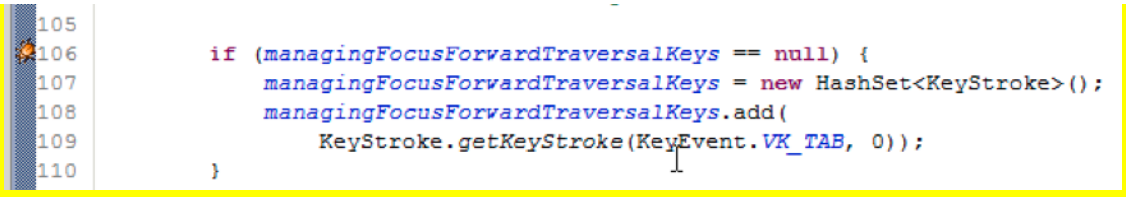


```
4613                    String name = ac.getAccessibleName();
4614                    if ((name == null) && (name != "")) {
4615                        return ac.getAccessibleName();
4616                    } else {
4617                        return null;
4618                    }
```

This code compares java.lang.String objects for reference equality using the == or != operators. Unless both strings are either constants in a source file, or have been interned using the String.intern() method, the same string value may be represented by two different String objects. Consider using the equals(Object) method instead.

You shouldn't compare strings using == or != because it is only comparing the reference not the actual string itself. Comparing strings is done using the .equals() method.

Notification 2 (incorrect lazy initialization):



```
105
106         if (managingFocusForwardTraversalKeys == null) {
107             managingFocusForwardTraversalKeys = new HashSet<KeyStroke>();
108             managingFocusForwardTraversalKeys.add(
109                 KeyStroke.getKeyStroke(KeyEvent.VK_TAB, 0));
110         }
```

This method contains an unsynchronized lazy initialization of a static field. After the field is set, the object stored into that location is further updated or accessed. The setting of the field is visible to other threads as soon as it is set. If the futher accesses in the method that set the field serve to initialize the object, then you

have a very serious multithreading bug, unless something else prevents any other thread from accessing the stored object until it is fully initialized.

Even if you feel confident that the method is never called by multiple threads, it might be better to not set the static field until the value you are setting it to is fully populated/initialized.

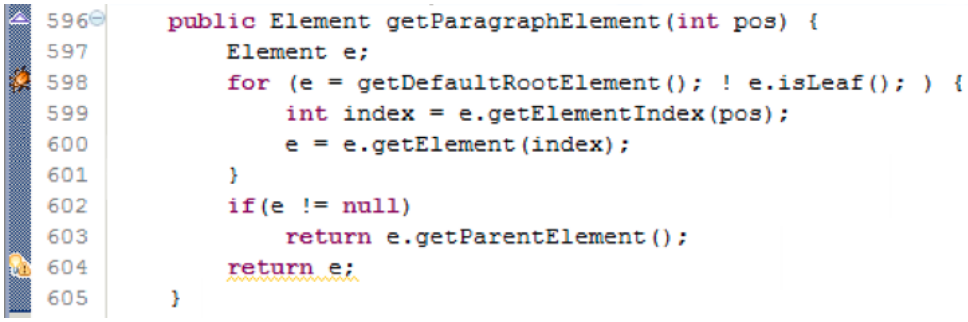
You are initializing a static variable without a synchronizing it, which if you trying to do lazy initialization is incorrect as the way the code is written now more than one of these objects can be created.

Notification 3 (Synchronize on a mutable field):

This method synchronizes on an object referenced from a mutable field. This is unlikely to have useful semantics, since different threads may be synchronizing on different objects.

It is possible for more than one of these objects to have been created or changed before it is actually synchronized on so there is no telling what it is in fact being synchronized.

Notification 4 (Redundant null check):

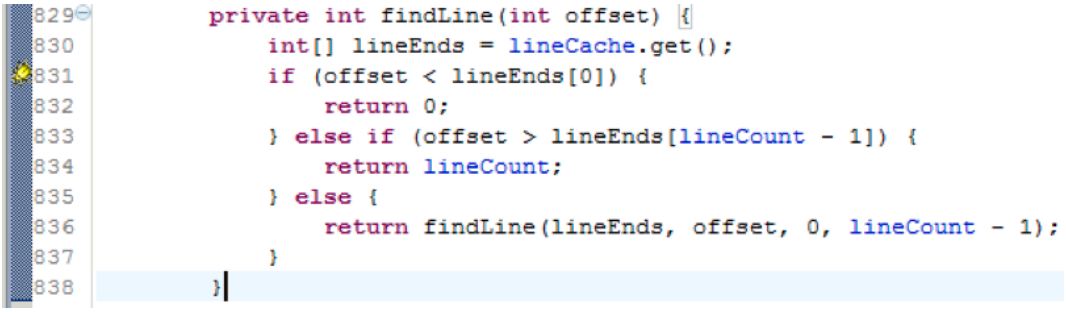


A value is checked here to see whether it is null, but this value can't be null

because it was previously dereferenced and if it were null a null pointer exception would have occurred at the earlier dereference. Essentially, this code and the previous dereference disagree as to whether this value is allowed to be null. Either the check is redundant or the previous dereference is erroneous.

The null check you are doing is not needed or misplaced. If e was null the code would break before reaching the null check. You should consider removing the null check and handing potential exception when e is dereferenced

Notification 5 (Possible null pointer dereference):



```
829         private int findLine(int offset) {
830             int[] lineEnds = lineCache.get();
831             if (offset < lineEnds[0]) {
832                 return 0;
833             } else if (offset > lineEnds[lineCount - 1]) {
834                 return lineCount;
835             } else {
836                 return findLine(lineEnds, offset, 0, lineCount - 1);
837             }
838         }
```

The return value from a method is dereferenced without a null check, and the return value of that method is one that should generally be checked for null. This may lead to a `NullPointerException` when the code is executed.

You are trying to access data that may not exist. You should check lineEnds[0] for null before trying to access it.

Notification 6 (Unused code):

```
45     private org.omg.CORBA.ORB _orb;
46     private Vector<String> _contexts;
```

```
45     The value of the field ContextListImpl._orb is not used
46         private Vector<String> _contexts;
47
```

You are not using (or reading from) this variable anywhere in this class (it's a private variable so it's not being used outside this class either). You could remove it to get rid of the error and the code would work the same.

Notification 7 (Parameterized/Raw type):

```
46      private Vector<String> _contexts;
47
48      public ContextListImpl(org.omg.CORBA.ORB orb)
49      {
50          // Note: This orb could be an instanceof ORBSingleton or ORB
51          //TODO COMP Task 2
52          _orb = orb;
53          _contexts = new Vector(INITIAL_CAPACITY, CAPACITY_INCREMENT);
54      }
```

```
50          _orb = orb;
51  Multiple markers at this line
52    - Type safety: The expression of type Vector needs unchecked conversion to conform to Vector<String>
53    - Vector is a raw type. References to generic type Vector<E> should be parameterized
```

You created a generic object Vector<String> but did not properly initialize it. The new Vector should be new Vector<String>.

Notification 8 (unimplemented methods):

```
37  class DirectByteBuffer extends MappedByteBuffer
38                         implements DirectBuffer
39
```

```
37  The type DirectByteBuffer must implement the inherited abstract method DirectBuffer.viewedBuffer()
38                         implements DirectBuffer
```

You are implementing a class (DirectBuffer) but not implementing all the required methods (viewedBuffer). If you implement this method the error will go away.

Notification 9 (serializable class needs serial ID):

```
44  public class DynAnyBasicImpl extends DynAnyImpl
45  {
46
```

```
44  The serializable class DynAnyBasicImpl does not declare a static final serialVersionUID field of type long
45  {
```

Somewhere down the line of classes/interfaces being implemented/extended from this class, Serializable is being implemented. Proper usage of this interface requires a serialversionUID during deserialization to ensure that the classes loaded are compatible with respect to serialization.

Notification 10 (unimplemented methods):

```
159            interruptor = new Interruptible() {
160                    public void interrupt(Thread target) {
161                        synchronized (closeLock) {
162                            if (!open)
163                                return;
164                            open = false;
165                            interrupted = target;
```

```
159  The type new AbstractInterruptibleChannel.Interruptible(){} must implement the inherited abstract method AbstractInterruptibleChannel.Interruptible.interrupt()
160                    public void interrupt(Thread target) {
161                        synchronized (closeLock) {
162                            if (!open)
163                                return;
164                            open = false;
```

There are methods from the interface you are trying to instantiated as an anonymous class that you are not implementing. You should implement an interrupt method with no parameters or change the method signature in the interface.

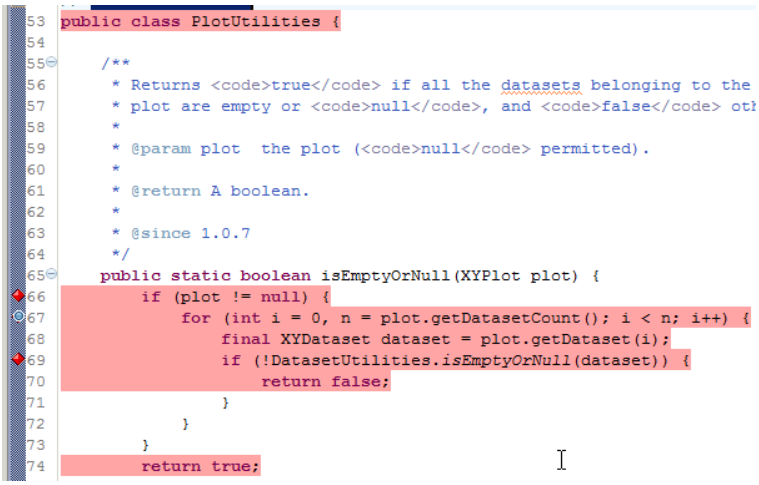
Notification 11 (method not applicable for arguments):

```
178            if (me.isInterrupted())
179                interruptor.interrupt(me);
180        }
```

```
178            if (me.isInterrupted())
179  The method interrupt() in the type AbstractInterruptibleChannel.Interruptible is not applicable for the arguments (Thread)
180        }
```

You are trying to call an interrupt method that is not expected for this Interruptible object. You should either call the method you implemented with no parameters or make sure the method in the interface matches this one.

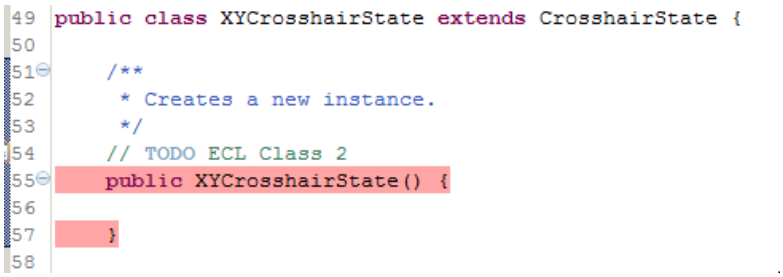
Notification 12 (Red class with red header):



```
53 public class PlotUtilities {
54
55     /**
56      * Returns <code>true</code> if all the datasets belonging to the
57      * plot are empty or <code>null</code>, and <code>false</code> ot
58      *
59      * @param plot  the plot (<code>null</code> permitted).
60      *
61      * @return A boolean.
62      *
63      * @since 1.0.7
64      */
65     public static boolean isEmptyOrNull(XYPlot plot) {
66         if (plot != null) {
67             for (int i = 0, n = plot.getDatasetCount(); i < n; i++) {
68                 final XYDataset dataset = plot.getDataset(i);
69                 if (!DatasetUtilities.isEmptyOrNull(dataset)) {
70                     return false;
71                 }
72             }
73         }
74         return true;
```

You have not instantiated an instance of this class (default constructor) nor have you called any of the methods.

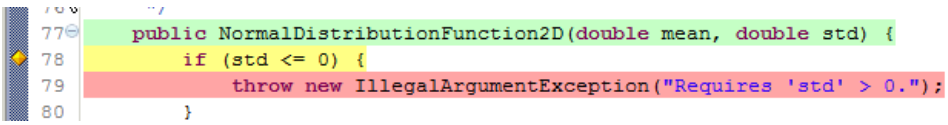
Notification 13 (Red class--constructor only):



```
49 public class XYCrosshairState extends CrosshairState {
50
51     /**
52      * Creates a new instance.
53      */
54     // TODO ECL Class 2
55     public XYCrosshairState() {
56
57     }
58
```

You have not created/instantiated an instance of this class (implemented constructor)

Notification 14 (Simple if statement 1 of 2 branches):



```
77     public NormalDistributionFunction2D(double mean, double std) {
78         if (std <= 0) {
79             throw new IllegalArgumentException("Requires 'std' > 0.");
80         }
```

```
77    public NormalDistributionFunction2D(double mean, double std) {
78  1 of 2 branches missed. <= 0) {
79            throw new IllegalArgumentException("Requires 'std' > 0.");
80        }
81        this.mean = mean;
82        this.std = std;
```

You are only executing one branch of this 2 branch if statement (the false branch). You should run the method with input(s) that will execute the true branch of the if.

Notification 15 (Short circuit return statement):

```
176    protected boolean isLinePass(int pass) {
177        return pass == 0 || pass == 1;
178    }
```

In the case of the notifications they look at, the methods are not being called. However each return statement mentions branches; 2 branches means you need to test pass = that number and also pass != to that number. 4 branches may mean when each part of the return statement returns true and false.

Notification 16 (try/catch -- no exception caught):

```
178        try {
179            ByteArrayOutputStream buffer = new ByteArrayOutputStream();
180            ObjectOutput out = new ObjectOutputStream(buffer);
181            out.writeObject(w1);
182            out.close();
183
184            ObjectInput in = new ObjectInputStream(
185                    new ByteArrayInputStream(buffer.toByteArray()));
186            w2 = (Week) in.readObject();
187            in.close();
188        }
189        catch (Exception e) {
190            e.printStackTrace();
191        }
```

The try block has executed and no exception was caught so the catch block did not execute.

Notification 17 (try/finally -- exception thrown, partial coverage in finally):

```
296         try {
297             Week w = new Week(new Date(11361098300001),
298                     TimeZone.getTimeZone("GMT"));
299             assertEquals(2005, w.getYearValue());
300             assertEquals(52, w.getWeek());
301             assertFalse(true);
302         }
303         finally {
304             Locale.setDefault(saved);
305         }
306     }
```

The try attempted to execute but failed, which led to the finally being executed and then exiting the method. Because only failure of the try and execution of the finally was tested, the inside of the finally is yellow. If this same test were called twice, once with an exception and once without, presumably at least the inside of the finally would be green. The red bracket at the end of the method suggests that the method exited after executing the finally.

Notification 18 (try/finally -- try executed, partial coverage in finally):

```
314         try {
315             TimeZone zone = TimeZone.getTimeZone("GMT");
316             GregorianCalendar gc = new GregorianCalendar(zone);
317             gc.set(2005, Calendar.JANUARY, 1, 12, 0, 0);
318             Week w = new Week(gc.getTime(), zone);
319             assertEquals(53, w.getWeek());
320             assertEquals(new Year(2004), w.getYear());
321         }
322         finally {
323             Locale.setDefault(saved);
324         }
325     }
```

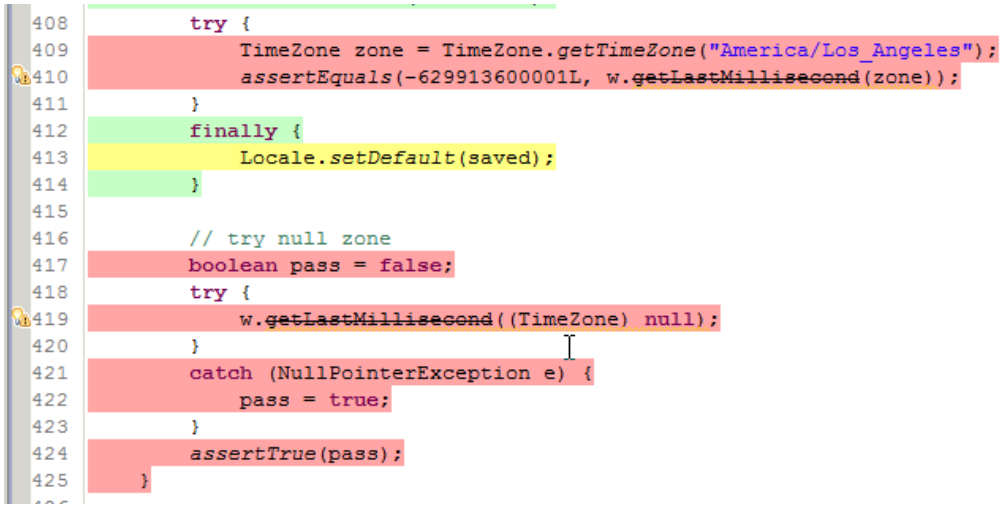
This is the opposite of N17. The try did execute which means the finally does not. Because this code was only called once (with no exception) the inside of the finally is yellow.

Notification 19 (try/catch -- exception caught):

```
378         try {
379             w.getFirstMillisecond((Calendar) null);
380         }
381         catch (NullPointerException e) {
382             pass = true;
383         }
```

The try attempted to executed but failed; an exception was thrown and caught so the catch block was executed.

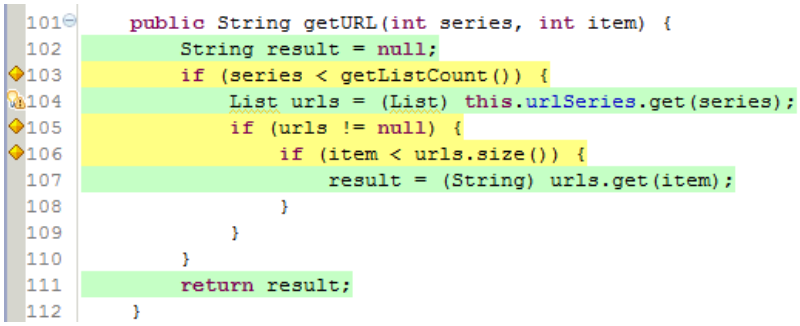
Notification 20 (try/finally -- method exits):



```
408         try {
409             TimeZone zone = TimeZone.getTimeZone("America/Los_Angeles");
410             assertEquals(-629913600001L, w.getLastMillisecond(zone));
411         }
412         finally {
413             Locale.setDefault(saved);
414         }
415
416         // try null zone
417         boolean pass = false;
418         try {
419             w.getLastMillisecond((TimeZone) null);
420         }
421         catch (NullPointerException e) {
422             pass = true;
423         }
424         assertTrue(pass);
425     }
```

This is similar to N17 except here there is more code in the method instead of just the closing bracket so we can see more clearly that the method exited once the finally executed.

Notification 21 (Nested if statements):



```
101     public String getURL(int series, int item) {
102         String result = null;
103         if (series < getListCount()) {
104             List urls = (List) this.urlSeries.get(series);
105             if (urls != null) {
106                 if (item < urls.size()) {
107                     result = (String) urls.get(item);
108                 }
109             }
110         }
111         return result;
112     }
```

```
101    public String getURL(int series, int item) {
102        String result = null;
103    1 of 2 branches missed.ies < getListCount()) {
104            List urls = (List) this.urlSeries.get(series);
105            if (urls != null) {
106                if (item < urls.size()) {
107                    result = (String) urls.get(item);
108                }
109            }
110        }
111        return result;
112    }
```

Only the true branches of each conditional executed (the only path that does anything).