**CS 447 Project**

**Part 2A**

1. **CID:**10065 **Error:** Missing Break In Switch  
   **Classification:** Intentional  
   **Reasoning:** Because there is no “Break” in case 3, the program will fall through and execute case 4. This will not result in an error because of the condition checks internal to each case, but this is generally considered bad practice and adding a “Break” at the end of case 3 would remove this warning in the future.
2. **CID:**10066 **Error:** CN: Bad implementation of cloneable idiom  
   **Classification:** Intentional  
   **Reasoning:** According to Java’s documentation, implementing Cloneable indicates for a particular class that it is legal for the Object.clone() method to make field-for-field copies of that class. The implementing of Cloneable here appears to signal, for lack of a better term, the developer’s intent to access this functionality. However, Object.clone() is a private method so it is convention to override this with a public method. The developers could implement their own clone method, but this could prove tedious, and probably unnecessary given what their intentions appear to be.
3. **CID:**10067 **Error:** Dm: Dubious method used  
   **Classification:** False Positive  
   **Reasoning:** This error is more specifically a reliance on default encoding of PrintWriter. This was likely flagged because of possible different encodings that may exist in different environments, however there appear to be no negative consequences that could flow from this reliance on a given environments default settings.
4. **CID:** 10068  
   **Error:** Dm: Dubious method used  
   **Classification:** Intentional  
   **Reasoning:** This error more specifically complains about the use of nextDouble method in Random being used to generate a random integer when nextInt would be more efficient. This looks like an intentional choice given the cast to int made in the return value and comments preceding the function explaining it’s intent and implementation.
5. **CID:** 10069 **Error:** Eq: Problems with implementation of equals()  
   **Classification:** False Positive  
   **Reasoning:** This warning complains that the equals method, as implemented, only compares class names as opposed to class objects. However, from the comments preceding the function, this is the intended functionality and meets the developer’s definition of equality.
6. **CID:** 10070 **Error:** Eq: Problems with implementation of equals()  
   **Classification:** False Positive  
   **Reasoning:** This case, and the reasoning that follows from the identified warning, is identical to that which was previously identified in bullet 5 (CID 10069).
7. **CID:** 10071 **Error:** ES: Checking String equality using == or !=  
   **Classification:** Intentional  
   **Reasoning:** It is terrible practice to compare strings using == or !=, even if it doesn’t always necessarily result in an error. The developer should instead use the equals() method (and the ‘!’ logical operator where necessary) to ensure good coding practices are in use.
8. **CID:** 10072 **Error:** ES: Checking String equality using == or !=  
   **Classification:** Intentional  
   **Reasoning:** The reasoning is identical to that of bullet 7 (CID 10071)
9. **CID:** 10073 **Error:** ES: Checking String equality using == or !=  
   **Classification:** Intentional  
   **Reasoning:** The reasoning is identical to that of bullet 7 (CID 10071)
10. **CID:** 10074 **Error:** ES: Checking String equality using == or !=  
    **Classification:** Intentional  
    **Reasoning:** The reasoning is identical to that of bullet 7 (CID 10071)
11. **CID:** 10075 **Error:** IM: Questionable integer math  
    **Classification:** Bug  
    **Reasoning:** The given warning specifically warns of a possible integer overflow. This, unfortunately, could happen in line 649, within the while loop, if a large value is assigned to high and low has been previously modified to another suitably large value in lines 652-653. A fix could be to implement range checks using if-statements and pre-defined MAX and MIN values.
12. **CID:** 10076 **Error:** NP: Null pointer dereference  
    **Classification:** False Positive  
    **Reasoning:** This warning complains of null being returned when the return type is Boolean, however this appears to be intentional based on the comments preceding the function, allowing for null to be the value passed to the function. Note that this is allowable when using Boolean as opposed to boolean. This allows the developer to handle the null case explicitly rather than leaving to chance the return value when null is passed.
13. **CID:** 10077 **Error:** NP: Null pointer dereference  
    **Classification:** False positive  
    **Reasoning:** Similar to bullet 12 (CID 10076), the comments preceding the function explicitly allow for null to be returned even while the stated return type is Boolean, which is allowable (as opposed to when using boolean).
14. **CID:** 10078 **Error:** NP: Null pointer dereference  
    **Classification:** False positive  
    **Reasoning:** This warning, and the reasoning that results, is identical to that which was previously identified in bullet 12 (CID 10076).
15. **CID:** 10079 **Error:** NP: Null pointer dereference  
    **Classification:** False positive  
    **Reasoning:** This warning, and the reasoning that results, is identical to that which was previously identified in bullet 13 (CID 10077).
16. **CID:** 10080 **Error:** NP: Null pointer dereference  
    **Classification:** False positive  
    **Reasoning:** As in the previous cases, Boolean is being used as opposed to boolean which allows for null values. The comments preceding the function specify null to be returned under specific conditions.
17. **CID:** 10081 **Error:** NP: Null pointer dereference  
    **Classification:** False positive **Reasoning:** This warning, and the reasoning that results, is identical to that which was previously identified in bullet 16 (CID 10081)
18. **CID:** 10082 **Error:** REC: RuntimeException capture  
    **Classification:** Intentional  
    **Reasoning:** By Exception as opposed to specific exception types, the developer creates a scenario where RuntimeException may be caught, which is generally undesirable. Very little is done with the exception itself in this code, thus this is of little consequence, although better practice would be to try and catch specific types of exceptions (which would admittedly be much more tedious and time consuming during execution).
19. **CID:** 10083 **Error:** Se: Incorrect definition of Serializable class  
    **Classification:** Bug  
    **Reasoning:** Serializing is the process of converting an object state into bytes. To declare an object transient would mean that a member variable is not serialized. Since mRules in line 137 is non-serializable and non-transient this could lead to erroneous states in execution. A simple fix would be to declare mRules to be transient.
20. **CID:** 10084 **Error:** UrF: Unread field  
    **Classification:** Intentional  
    **Reasoning:** This warning complains that the key field is unread, however the preceding comments note that key is to be used for storing each Entry in a table. While unused fields are bad practice, there does appear to be some motivation for leaving this as is.