**CODE SMELLS – ELEMENT 4**

**Illustration of the code snippet**

Text

Description automatically generated

**The exact location on the codebase**

This class called ExportComparator has just one method - which is not the constructor - called compare (which compares two lists of strings), so the code smell is **Lazy Class**.

**An explanation of the rationale for identifying this code smell**

Every additional class adds more complexity to the project. More classes just mean more code to maintain. The ExportComparator class is not entirely too useful, has an empty constructor. In conclusion, this class does not do enough to earn the attention of the project. What is made in this class could be done in other classes.

**A refactoring proposal**

A solution could be collapsing the class or possibly combining it with an existing class. Incline Class or Collapse Hierarchy can help clean up lazy classes if the single responsibility principle (a class should have only one reason to change) is being kept. The refactoring proposal could be implementing the compare method in the classes where it is most needed.

**Illustration of the code snippet**

Text

Description automatically generated

**The exact location on the codebase**

This class called VersionPreferences has just two methods: a constructor and one getter method, so the code smell is **Data Class**. Data class refers to a class which contains only fields and crude methods for accessing them (getters and setters). These are simply containers for data used by other classes.

**An explanation of the rationale for identifying this code smell**

This class does not contain any additional functionality and cannot independently operate on the data that own. It’s a normal thing when a newly created class contains only a few public fields (and maybe even a handful of getters/setters). But the true power of objects is that they can contain behaviour types or operations on their data. In conclusion, this class which is a data class, contain only data and no real functionality, only a getter method. This indicates that it may not be a good abstraction or a necessary class.

**A refactoring proposal**

In this case that this class only uses public methods, one solution could be the use of the Encapsulation (which means, ability to conceal object data. Otherwise, all objects would be public and other objects could get and modify the data of the object without any checks and balances) to hide them from direct access and require that access to be performed via getters and setters only.

**Illustration of the code snippet**

Text

Description automatically generated

**The exact location on the codebase**

This class called VersionPreferences has just one variable of the type Version called ignoredVersion. The methods of a class should be interested in the variables and functions of the class they belong to, and not the variables and functions of other classes, so the code smell in this class is **Feature Envy**.

**An explanation of the rationale for identifying this code smell**

This code smell occurs when there is a method that is more interested in the details of a class other than the one it is in. If two methods or classes are always talking to one another and seem as if they should be together, then chances are this is true. The getIgnoredVersion method reaches into the Version object to get the data on which it operates. It “wishes” that it were inside the Version class so that it could have direct access to the variable (ignoredVersion) it is manipulating.

**A refactoring proposal**

In that case, we may consider moving this method (getIgnoredVersion) to the other class it uses (Version). This would make **classes more internally coherent** (because we would move a method to a class which contains all the data used by the method). So, in conclusion, the method would become a part of Version class instead of VersionPreferences class since all it does is to use a variable of the type of Version (ignoredVersion).