PDFSams two class/method smells

1. org.pdfsam.ui.info.SummaryTab class (smells of a God Class)
    1.1. This is a class that is part of the user interface. The class was found to be a God Class by intelliJDeodorant was mainly due to the setFileProperties function and all the label type private variables.
    1.2. The class were detected as a God class because the SummaryTab class has 11 private variables which can be refactored into two groups file property labels and pdf property labels.
    1.3. In the beginning of using intelliJDeodorant I didn't agree with this smell but as I dug into it more, I came to realize the different grouping of the private variables would make more sense. If you were to create two new sub classes called PdfPropertiesTabs and filePropertiesTabs each could then implement their on add functions and eliminate the large function calls in this class where it has to add, set and do other things with each tab. This could be refactored to two private variables and thus reduce the function calls to just two instead of eleven times.

2. Org.pdfsam.support.io.NetUtils function call urlToStream(URL url) smells of a long method
    2.1. This class only has one function call urlToStream and thus that is the one job of the NetUtils class. This is a support class that helps with handling the url from the network.
    2.2. The urlToStream function was marked as a long method because it is 32 lines of code. Which according to intelliJDeodorant passes the threshold and makes the function call too long.
    2.3. I agree with this smell detection as I can see the function having a few subparts that can be refactored out into separate function calls. Especially the if statement that checks if the connection is an HttpURLConnection instance. That whole branch could be refactored into a function call called extractURL(HtppURLConnection connection) which will return a URL class type.

Jedit's two class/method smells

1. Org.gjt.sp.jedit.search.HyperSearchResults class smells of God Class
    1.1. This class contains the HyperSearch results that is created after a hypersearch is done. It contains a highlight tree and many other subclasses that are important for the displaying of the hypersearch results to the user in the GUI
    1.2. The reason that this class was marked has having a God smell class is because it has tons of functions and more than 1100 lines that belong to the class. It also declares 18 sub classes inside of its class rather than declaring the classes outside in other java files all 18 are smashed into the one class HyperSearchResults.
    1.3. Yes, I agree with the IntelliJDeoderant in saying that this class has a bad smell and is a God Class. It is unmaintainable and confusing. There is too much going on especially with there being 18 subclasses that are declared inside of the HyperSearchResults class and most of them are inheriting from different classes which creates a spaghetti code

type feel to the code. This class would take a long time to refactor. It would require going through all the 18 sub classes and grouping them into groups that are similar and creating wrappers around them and making an instance of that in the HyperSearchResults. Also, HyperSearchResults looks like it could use some more subclasses that deal with different jobs. Like one with getting the results, another with holding results and the other with displaying the results to the user.

2. org.gjt.sp.jedit.gui.NumericTextField smells like a God Class
    2.1. Class oversees the text fields in Jedit. It contains several functions and variables that deal with the value of text field that are interrelated. These variables are minValue, maxValue, intergerOnly and positive only.
    2.2. The reason that this was flagged as a God class is because all the variables, I brought up above are cohesive to each other and have a separate objective than the handling of the text field that the rest of the class has. Those variables can be made into a sub class. This will make the NumericTextField more focused in on handling the textField and not worry so much about the value of the number's restraints.
    2.3. Yes, I believe this is a good find and makes sense to refactor the code and pull out the funcitonality of max, min, positive, and integer only into its own class of restraints that will be used by NumericTextField Class.

PDFSam fixing of urlToStream function

1. the overall functionality of this test is covered by the LatestNewsControllerTest.java that is already implemented. Since my function is simply a sub partition of the original function, I believe that the current test is sufficient because my test would be difficult to test individually without all the code for having a working connection and url to use. Since I made the test smaller and did not change the functionality, I believe the current test is sufficient.
2. I tried using the IntelliJDeoderant to remove the long function, but it didn't do a good job refactoring. It simply grabbed a portion of the case structure and put it into a get string function. It looked bad to have a function call that was a partial case structure. That would create a new bad code smell. Instead I refactored the code in way that was more modular.
3. The LatestNewsControllerTest grabs the latest news that are found. It uses and tests the urlToStream function. If I changed the behavior of this method with my internal private method, then it would break the LatestNewsControllerTest Junit test that is provided. I ran the tests and they passed. It was a little confusing at first to see if the test passed because it stated that the test passed but it threw an exception. I looked through the test and there were several cases that were supposed to throw exceptions etc. You can see that via the method calls nullFetchNews() and failingFetchNews() which were expected to fail. That is why when I ran the individual test via the command line "mvn test -DfailIfNoTests=false -Dtest=LatestNewsControllerTest test" it passed. If my function broke the functionality at all accessing the url and etc we would have seen a failure here.
4. I reran IntelliJDeodorant and the smell was successfully removed, and my new method isn't smelly either when I ran intelliJDeodorant and

5. What I did to remove the smell was I grabbed all the code that was used for retrieving relative URLs and checking them into its own private function call getRelativeURL(HttpURLConnection connection, URL url). This reduced the long method smell from 35 lines of code to 22. I removed 13 lines of code and gave it a descriptive name to fit what it was doing.
6. The reason I chose the specific lines of code was because all the lines of code resided in an if branch. I looked through the if branch because I knew the code would be interrelated and saw that it was. I then grabbed all the code in there that was related to the same function of getting a related urls from the same connection and created a private sub function that was called when that branch was exercised. The reason I made it private is because I knew that it should only be used internally by that other function. This thus increased also the cohesion of the class as there were more interrelations between the two functions.
7. I did all the changes manually because IntelliJDeoderant made a refactoring suggestion that introduced a new smell. It grabbed part of a case structure and put into a function call and wanted to name it as a get string method. It seemed to me that it would hurt maintainability of the class, so I abstracted the code into its own function that had a name that described what was going on.

Jedit fixing of NumericTextField class smell

1. There were no specific test cases for this class, so I had to create my own J-unit test called NumericTextFieldTest.java that tested the setting of constraints and getting a value from text using those constraints.
2. I tried using the IntelliJDeoderant to extract the class but I felt like I could do a better job in extracting another class from the file.
3. I reran the above junit test that I created, and it passed with flying colors when I updated a function call to remove the middleman. Since I extracted a class from NumericTextField called NumericTextConstraints, I created another J-Unit test called NumericTextConstraintsTest which tests the getter and setter function and it passed.
4. I reran IntelliJDeodorant and the smell was successfully removed, NumericTextField class is no longer a God Class and the new extracted class that I created doesn't have any smells.
5. I removed the booleans isInteger, isPositive, minValue and maxValue to a new class called NumericTextConstraints and brought over the setMinValue and setMaxValue functions. Next, I updated all references of those variables in the numericTextField class to be the constraint class with the getter functions. I also removed the setMinValue and setMaxValues from that class and created a getter for the constraint class that one could use in setting the constraints for a given text field. This removed making NumericTextField from being middleman.
6. The reason behind using the refactoring method extract class was because NumericTextField was marked as God class by the IntelliJDeoderant software. I also looked through the variables and functions and was able to see that the variables isInteger, isPositive, minValue and maxValue were all interrelated, had high cohesion and dealt with the constraints placed upon the number within the NumericTextField class. Using that logic and seeing the functions related were easy to extract I moved forward in creating a new class called NumericTextConstraints.

7. I did all the changes manually because IntelliJDeodorant did separate and package the constrain variables like I wanted. I also felt like what I could do would be better than the automated refactoring would have done.