**Team <Scentaur>**

**Ga Jun Young** (ga.young@ucdconnect.ie)

Royal Thomas (royal.thomas@ucdconnect.ie)

Zheng Ju (zheng.ju@ucdconnect.ie)

William Ikenna-Nwosu (william.ikenna-nwosu@ucdconnect.ie)

Use a splash page image here [optional]

Use *LaTeX* if you wish but use the general spacing and font/style you find here (1.5 spacing, 12-point font for text, etc.).

Be sure to submit a PDF (not a .DOC file) as your report. Overall it should be 12 to 15 pages, including diagrams.

**1. Introduction**

Scentaur is a Java specific “code smell” detector. The team behind Scentaur believes that many current code smell detectors are not particularly easy to use and understand (i.e. JDeodrant and infusion) thus the team envisions Scentaur to be a user-friendly code smell detector tool.

Team Scentaur plans to achieve in making a web-based code smell detector tool. Allowing quick, easy and maintainable way of detecting smells and even enabling users to refracture fragments of code that was detected by Scentaur. Scentaur aims to reduce the amount of time required to detect smells unlike JDeodrant which has a significantly long time in detecting smells. As it contains many types of detectors (22) along with the ability to refracture smelly code.

Team Scentaur has set several main goals in mind. Here are the following:

1. To learn and understand the different types of code smells.
2. To understand and take responsibility in a larger team.
3. To enable Scentaur for future uses outside of the assignment background.
4. To enable easy implementations and execution of other code smells in the future.

Scentaur strives to provide easy access, usage, lookup for all types of users. Being able to switch between a detailed and broad descriptor on the detected code smells. Followed by a color-coded scheme to display each individual smell. Team Scentaur encourages young Java users to understand the purpose of code smells by presenting easy to understand visualization of code smells, along with a brief description of the smells involved. Scentaur also provides for those that are more experienced with Java and data. By presenting a more in-depth explanation of code smells detected and help users to track down their smelly code.

**2. Specification**

**2.a Analyze Project**

Scentaur is planned to become a web application through Spring. Therefore, users are enabled to either submit a zip folder or java files directly to a web server. If a Zip folder is submitted, its contents will be extracted to a directory. Otherwise, files will be placed into a directory.

Designate a folder directory to contain the location of where Scentaur should analyze the project.

* For testing purposes, Scentaur will sniff out code from the *“testProject”* directory.
  + Note: Purposely made code smells will be available in *“testProject”*
* This will be a temporary directory to enable multi-users to run Scentaur.
* Once a user is done with Scentaur, the contents of the directory is wiped.

In order to analyze the entire directory given, Scentaur will be using **JavaParser.**

* A Parser class will take in the root directory path as a string.
* configureSymbolSolver method will set the symbols required to sniff out java files.
* The constructor will call configureSymbolSolver and parse all source file based on JavaParser-JUG-Milano slides.
* Parser will have a method that returns the compilation units of all java files within its root and sub directories.
  + Note: Information on compilation units is given in **Software Overview**

**2.b Detect Code Smells**

* An abstract Smell Superclass will generalize all code smells.
* Sub directories will be made to accommodate smells that are categorized
  + These categories include:
    - Bloater, Abuser, Coupler, Dispensable
* The following interfaces will be made: *Smellable, Abusable, Bloatable, Coupleable and Dispensable.* To ensure that we enable plug-in-play system for the smells within the categories.

A sample hierarchy is shown to display the hierarchy specification.

*Smell* extends *VoidVisitorAdapter<Void>* inherits *Smellable*

*Bloater* extends *Smell* inherits *Bloatable*

*LongParameterList* extends *Bloater*

*VoidVisitorAdapter<Void>* enables code smell detectors to visit nodes for a compilation unit related to the code smell. *E.g. LongParameterList visits methods of a class and checks if the method has a long parameter*

The following is possible due to the hierarchy above:

*Smell* longParameterList = new LongParameterList();

**2.c Generate Report**

The report will be available in multiple different forms.

* A Report class will take in all the smells that were detected
* It will have an object inside the report to enable calculations in generating data for the smells. This is to **show distribution of different smells** that exists within the code.
  + - Sample text: Bloater Smells – 23  
       LongParameterList – 15  
       Long Method - 8
    - Generate percentages in terms of smells.

Bloater/Total Problems \* 100%

Abusers/Total Problems \* 100%

Coupler/Total Problems \* 100%

Dispensable/Total Problems \* 100%

Furthermore, the report can be obtained in a text file. Displayed on a table like manner.

* The report can also generate classes in either text or java format with comments added to wherever the code smells existed.

**2.d Visualize the Code-Base & Identify Trouble-Spots**

The code base is planned to be visualized on the web browser using spring and CSS.

* Each smell will have their own identifying color in hexadecimal for CSS to interpret.
* Scentaur plans to visualize problems by either commenting problems above the smell or color coding the specific problems.
* Using the compilation unit to locate the line at which the problems exists we can add color to it.
* Problems will be highlighted.
* A cross comparison will be shown where the left-hand side displays the original code and the right-hand side displays the updated version containing comments or highlighted text describing the code smell.
* Users will be enabled to choose different smells to detect from through a selection bar. The default option is the one where all smells are sniffed for. [Drop Down Menu]
* Percentages like the calculations in the report will be shown in terms of pie charts, histograms and other visual representations.

**3. Software Overview**

Provide a schematic view of your design here. A UML diagram might be useful. What interfaces are you putting in place to ensure different team members are working toward a coherent body of software elements are can be coherently integrated and tested.

**4. Major Responsibilities and Work Breakdown**

Provide a modular view of the work here, with an assignment of responsibilities to each team member. A Gantt diagram is useful here.

**5. Team Communication**

Provide an overview of the tools and methodology you will be using to ensure productive teamwork and a transparent flow of information. Will you be using any groupware applications to coordinate your efforts? How often will you be meeting as a team? How will you use resources like GitHub to maximum effect? How will you deal with disagreements or scheduling conflicts or misalignment of goals or plans?

**6. Concluding Remarks**

Provide a brief statement of your team’s philosophy here. Mention any special of noteworthy aspects of your approach to the problem. Highlight any risks you feel may impact the work, and offer mitigation strategies if necessary.

**Acknowledgements**

Every team member should contribute an equal effort to this report. Use this optional section to provide report credits, or to highlight a special contribution by a given team member.

**References**

List any bibliographical citations here [optional]