









Plagiarism Detector UML and Java Interfaces

CodeX		UserID	File1	Settings Log Out	
File Name	File Name	% Similar	TimeStamp	Get Report	
 User 2	File1	70 %	15 Sep, 8:56 AM (2013)	Details	
 User 2	File2	70 %	15 Sep, 7:12 AM (2013)	Details	
 User 3	File3	70 %	15 Sep, 4:32 AM (2013)	Details	
 User 4	File1	70 %	15 Sep, 2:08 AM (2013)	Details	
 User 4	File1	70 %	15 Sep, 8:56 AM (2013)	Details	
 User 4	File1	10 %	15 Sep, 7:12 AM (2013)	Details	
 User 5	File1	70 %	15 Sep, 4:34 AM (2013)	Details	
 User 5	File1	70 %	15 Sep, 2:08 AM (2013)	Details	

Phase B
Team 102

Content

Objective	3
Design Pattern Used	4
UML	5
Java Interface	6
References	9

Objective

The primary objective of this document is to reflect our component design and relationships between each through a UML diagram. In the next phase we will use this as starting point for our implementation.

Abstract:

- User logs into the system. The user can be an Admin, Professor or student. New users have to register first.
- Student uploads homework.
- Admin can perform CRUD operations on other users.
- Professor can check for plagiarism by taking a Student's submission with others.
- We implemented strategy design pattern to allow Professor to choose different strategies at runtime. As of now we will be implementing AST strategy.
- We will generate AST for a given python file using respective parsers.
- These generated ASTs will be passed to the comparator for plagiarism check, which gives a detailed summary along with the percentage match.
- Using this summary a report is generated which can be viewed and/or downloaded by the Professor.

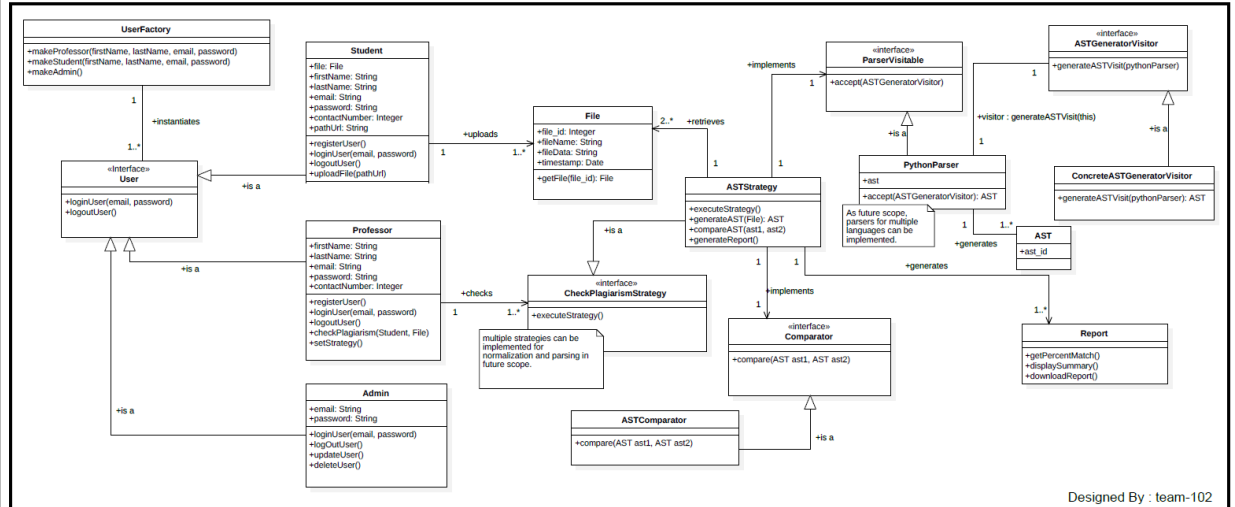
Design Pattern Used

We have used three design patterns here.

- ✓ **Factory Pattern:** We will use factory pattern to instantiate the different types of System users. Admin, Student and Professor implements user interface. We will create a UserFactoryClass to return the object of specific classes. This helps in creation of different objects through a shared common factory which helps in attaining a single point of control for multiple products.
- ✓ **Strategy Design Pattern:** We will use the strategy design pattern so that we can enable selecting an algorithm at runtime. As of now we will be implementing AST Strategy to check plagiarism. At later point of time we can implement different strategies and we can set particular strategy and implement the strategy we want to in checkPlaiigiarism Interface which in turns executes the strategy which we want to implement.
- ✓ **Visitor Design Pattern:** We will use the visitor design pattern so that we can design reusable software. Here if we want to implement the functionality to build and AST for files in other languages we would just need to add another class and implement the visit method specific for building the AST for that class. The same has been represented in the UML class diagram.

UML

CODEX - Plagiarism Detection Tool (UML Class Diagram)



Java Interfaces

ASTGeneratorVisitor.java:

```
package interfaces;

/**
 * This is an interface which has method(s) to provide functionality to
 * to generate AST of language specific tokens.
 * All the classes implementing this interface should provide implementation
for
 * all the methods declared in this interface.
 */
public interface ASTGeneratorVisitor {

    /**
     * This method accepts language specific tokens and generates AST
     * based on those tokens
     * @param pythonParser An object of type Parser
     * @return ASTClass representing AST of given ParserVisitable object
     */
    ASTClass generateASTVisit(ParserVisitable pythonParser);
}
```

CheckPlagiarismStrategy.java:

```
package interfaces;

/**
 * This is an interface which has method(s) to provide functionality to
 * execute specific strategy to implement plagiarism detection
 * All the classes implementing this interface should provide implementation
for
 * all the methods declared in this interface.
 */
public interface CheckPlagiarismStrategy {

    /**
     * This method executes specific plagiarism strategies
     */
    void executeStrategy();
}
```

Comparator.java:

```
package interfaces;

/**
 * This is an interface which has method(s) to provide functionality to
 * compare two ASTs.
 * All the classes implementing this interface should provide implementation
 * for
 * all the methods declared in this interface.
 */
public interface Comparator {

    /**
     * This method compares given to two AST's and generates a string
     * representing the comparison of passed AST's.
     * @param firstAST AST object generated after parsing code file
     * @param secondAST AST object generated after parsing code file
     */
    String compare(ASTClass firstAST, ASTClass secondAST);
}
```

ParserVisitable.java:

```
package interfaces;

/**
 * This is an interface which has method(s) to provide functionality to
 * generate language specific tokens and AST.
 * All the classes implementing this interface should provide implementation
 * for
 * all the methods declared in this interface.
 */
public interface ParserVisitable {

    /**
     * This method returns an AST representation of given language specific
     * parser
     * file
     * @return AST representation of passed language parser object
     */
    ASTClass accept(ASTGeneratorVisitor languageParser);
}
```

User.java:

```
package interfaces;

/**
 * This is an interface which has method(s) to provide functionality to
 * different types of system users.
 * All the classes implementing this interface should provide implementation
 * for
 * all the methods declared in this interface.
 */
public interface User {

    /**
     * This method allows system users to log into the system
     * @param email unique email address of the user
     * @param password user password for given email address
     */
    void LoginUser(String email, String password);

    /**
     * This method allows system users to log out of the system
     * @param userId unique user id for specific user
     */
    void LogoutUser(int userId);
}
```


References

1. <http://www.newthinktank.com/2012/11/visitor-design-pattern-tutorial/>
2. https://en.wikipedia.org/wiki/Visitor_pattern
3. https://en.wikipedia.org/wiki/Strategy_pattern