Progress Report - Week 2

[Graphical Tool for Refactoring Class Diagrams]

(Akanksha, Ashay, Gaurav, Prateek, Sree)

Week 1 Recap:

1) Finalized on UML tool to integrate refactorings on. - ArgoUML
2) Studied the code and its behavior for class diagrams.
3) Determine the flow with regards to user experience with the tool.

Click on a button :

1. Save the diagram details into an XMI.
2. Refactoring API will run in the background
3. Generate the .pl file
4. Run some constraints check to make sure the diagram is valid.
5. Check the possible refactorings
6. Generate a new .xmi and .pmg file.
7. Argo screen picks the new .xmi and .pmg file and shows the image.

Determination of tasks to be done:

Argo generates two files for each of its diagram. One .xmi file and one .pmg file with position details of the various objects that have been created.

- Team members would be focusing on parsing the xmi file to determine the possible refactorings.
- A team member will need to check how the placements of the classes can be done once the refactoring has been determined.
- Team member to work on the UI to be able to accomplish the above flow.

Week 2 Update:

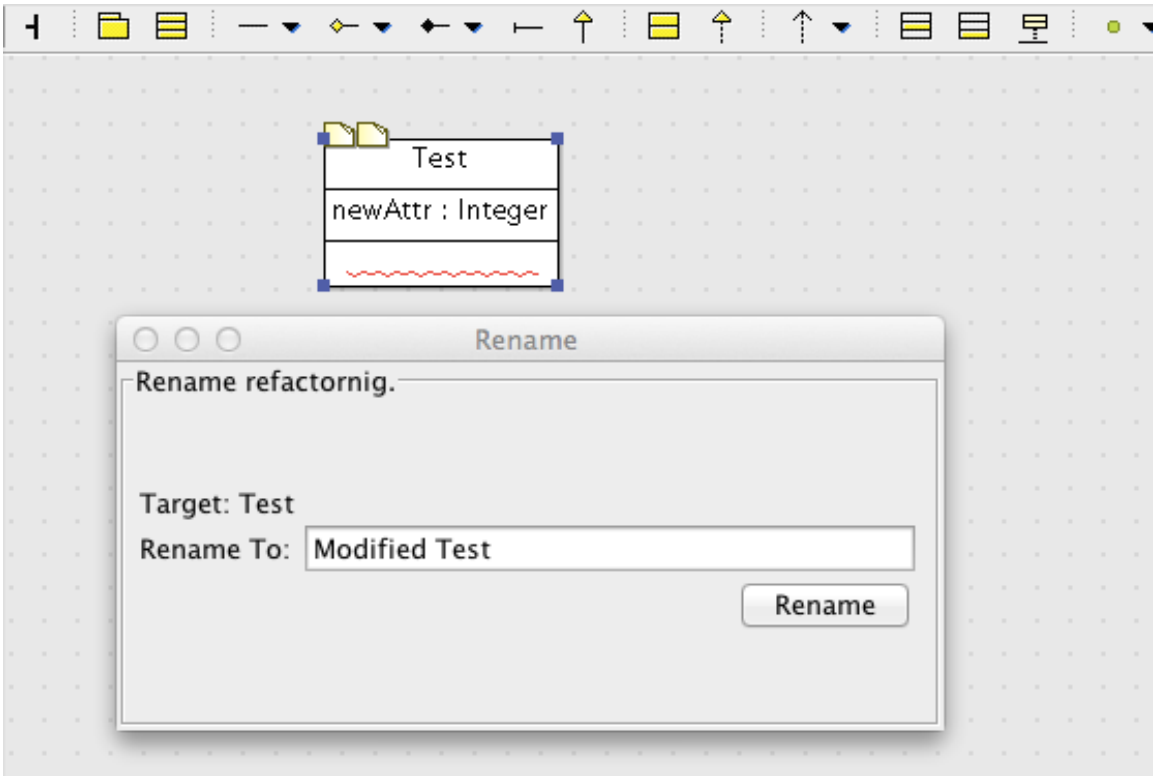
- **Modifying GUI to handle refactorings:**

One task we performed was to understand the ARGO source code and try to modify it to add support for basic refactorings.

Challenges/Tasks:

1. Narrowing down the event handling components required for our task within the total code base. This required understanding of the UML class hierarchy and AWT event propagation.
2. Using learning from 1, figure out how to identify various components drawn on the GUI. Once we are able to get handle on each element, we could perform any refactoring we wanted.
3. Create placeholder for various refactorings and added a form for rename refactoring. User goes and selects a class/arrow on the canvas and then clicks the rename button. Tool would identify the selected element and provide a window to update the name.

Here is a snapshot:



- **Analysis of XMI generated from tool**
  - The XMI is in UML2 format. The Eclipse UML2 jars define the meta-model for the XMI files. They are a part of the argouml-core-model-euml package. The files generated using them are being used by this package to generate the XMI and parse them.

- **XMI parsing and fetching relevant attributes.**
  - Studied the facade() method to get a way to find relevant attributes for our .pl
  - Analyzed which Model implementation has to be used.
    - using : "org.argouml.model.mdr.MDRModelImplementation"

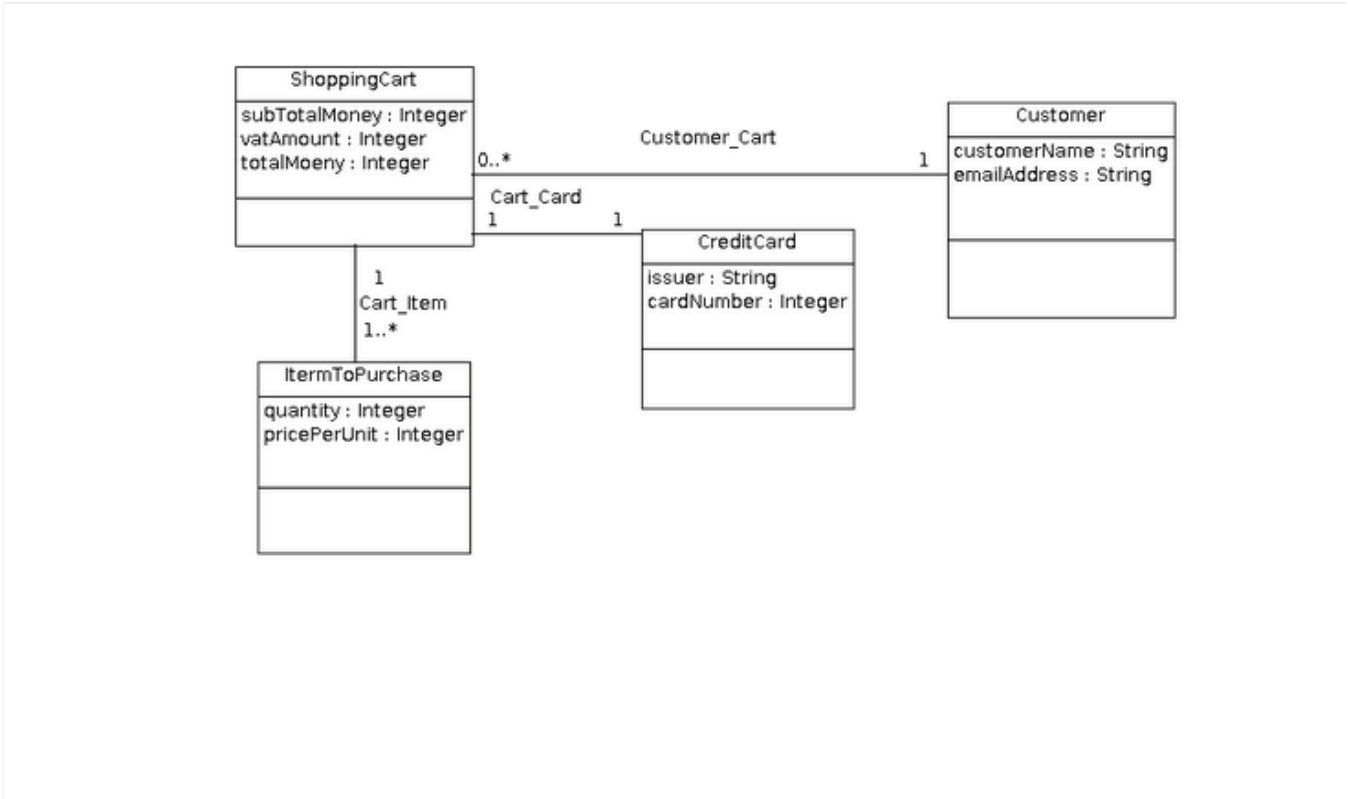
- **Generation of .pl database from XMI.**
  - Using the APIs given from facade() to get appropriate features from XMI.
  - Formatted the .pl to make it parsable directly by swipl.
  - Able to test it on a generated pl, and swipl is able to load it properly.

Note:

The only important thing missing in the .pl is the coordinate positions of the models (as they are captured in a different xml with a totally different format). One approach that i think of is generate .pl without coordinates, run all the constraints and M2M transformations and from that .pl create a new DSL, which we can then use along with ArgoXML writer to make the modified XMI file from within the code.

Example:

Class Diagram



Generated pl:

dbname(ShoppingCart, [ShoppingCart, Customer, CreditCard, ItermToPurchase,]).
class(class_1, ShoppingCart, vk_public).
attribute(attr_1, subTotalMoney, vk_public).
attribute(attr_2, vatAmount, vk_public).
attribute(attr_3, totalMoeny, vk_public).

class(class_2, Customer, vk_public).
attribute(attr_4, customerName, vk_public).
attribute(attr_5, emailAddress, vk_private).

class(class_3, CreditCard, vk_public).
attribute(attr_6, issuer, vk_public).
attribute(attr_7, cardNumber, vk_private).

class(class_4, ItermToPurchase, vk_public).
attribute(attr_8, quantity, vk_public).
attribute(attr_9, pricePerUnit, vk_public).

composition(assoc_1, Customer_Cart).

association(assoc_end_1, class_1, 0..inf).
association(assoc_end_2, class_2, 1..1).

composition(assoc_2, Cart_Item).
association(assoc_end_3, class_1, 1..1).
association(assoc_end_4, class_4, 1..inf).

composition(assoc_3, Cart_Card).
association(assoc_end_5, class_1, 1..1).
association(assoc_end_6, class_3, 1..1).

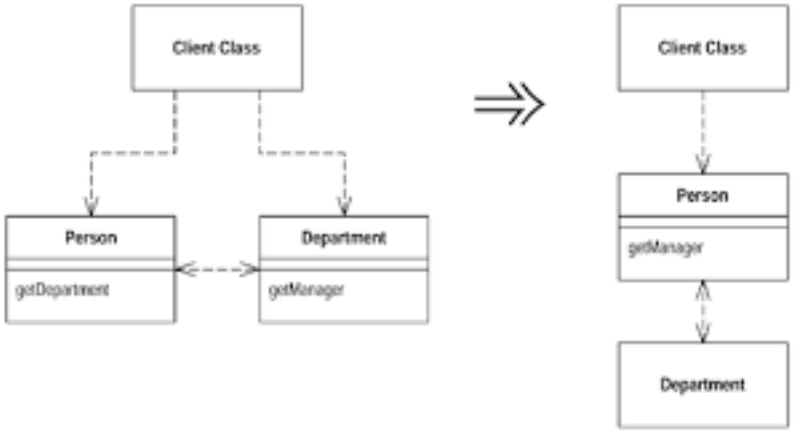
- **Proposed Refactorings:**
  Refactorings, which can be determined from a class diagram:

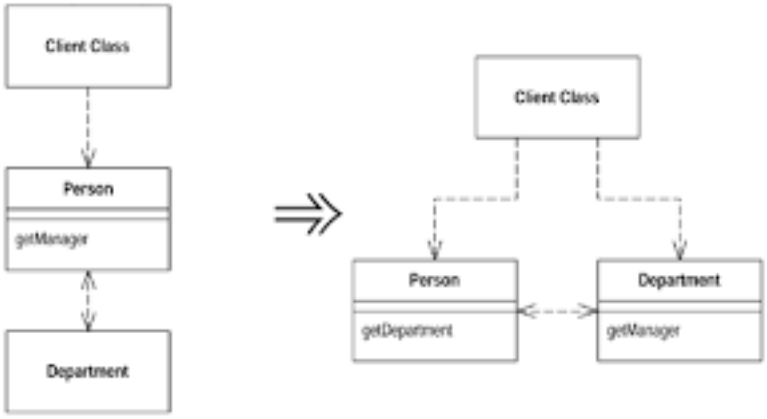
  Extract a new Class:
  When between two objects we see similar functionalities then it makes sense to extract a class out of the two.

  Mechanics:
  - List variables and functions, which are common within classes and then, show the possible list of refactoring to the user for him to select.
  - Make a link from the old to the new class. Do we need to create a back link? Don't think this can be determined from the diagram itself.

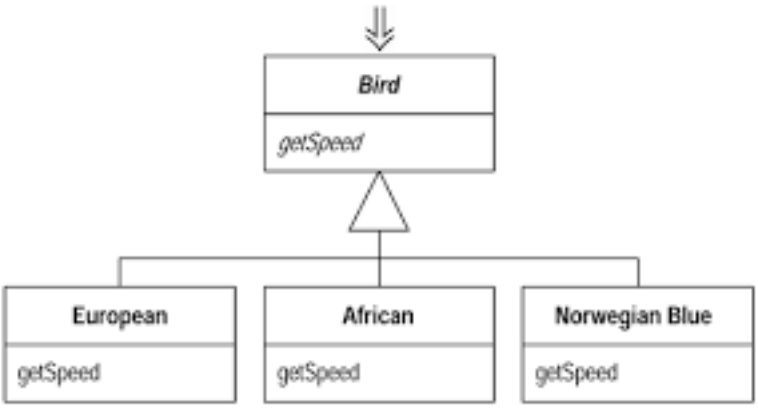
  Delegates:

  We can suggest a below example of refactoring to the user





Extract interface: Can be suggested if the class structure is something like below:



Replacing Delegation with inheritance: