Lab - Domain Models using UML Class Diagrams

In this lab you will use the Papyrus UML diagramming tool to create a domain model for a given problem. The domain model is created using class diagram notation. The domain model is a design tool to help bridge the gap from the real-world object to software objects in your code. Your team can use the domain model to help guide the software design process.

In this lab you will also create Java code that matches a given UML class diagram.

Experience with UML modeling will be applicable to potential internships or careers.

What I Want You To Learn

- The basics of using a UML modeling tool to create a class diagram
- The basics of creating code from a UML class diagram

Deliverable

- Part A: UML class diagram
- Part B: Java Code
- Upload both to Sakai as a single export .zip of your Eclipse project

Background

This assignment has two parts. In part A you should compose a domain model using Papyrus's class diagram notation. In part B you should design, write, and test code that implements a given class diagram.

Part A

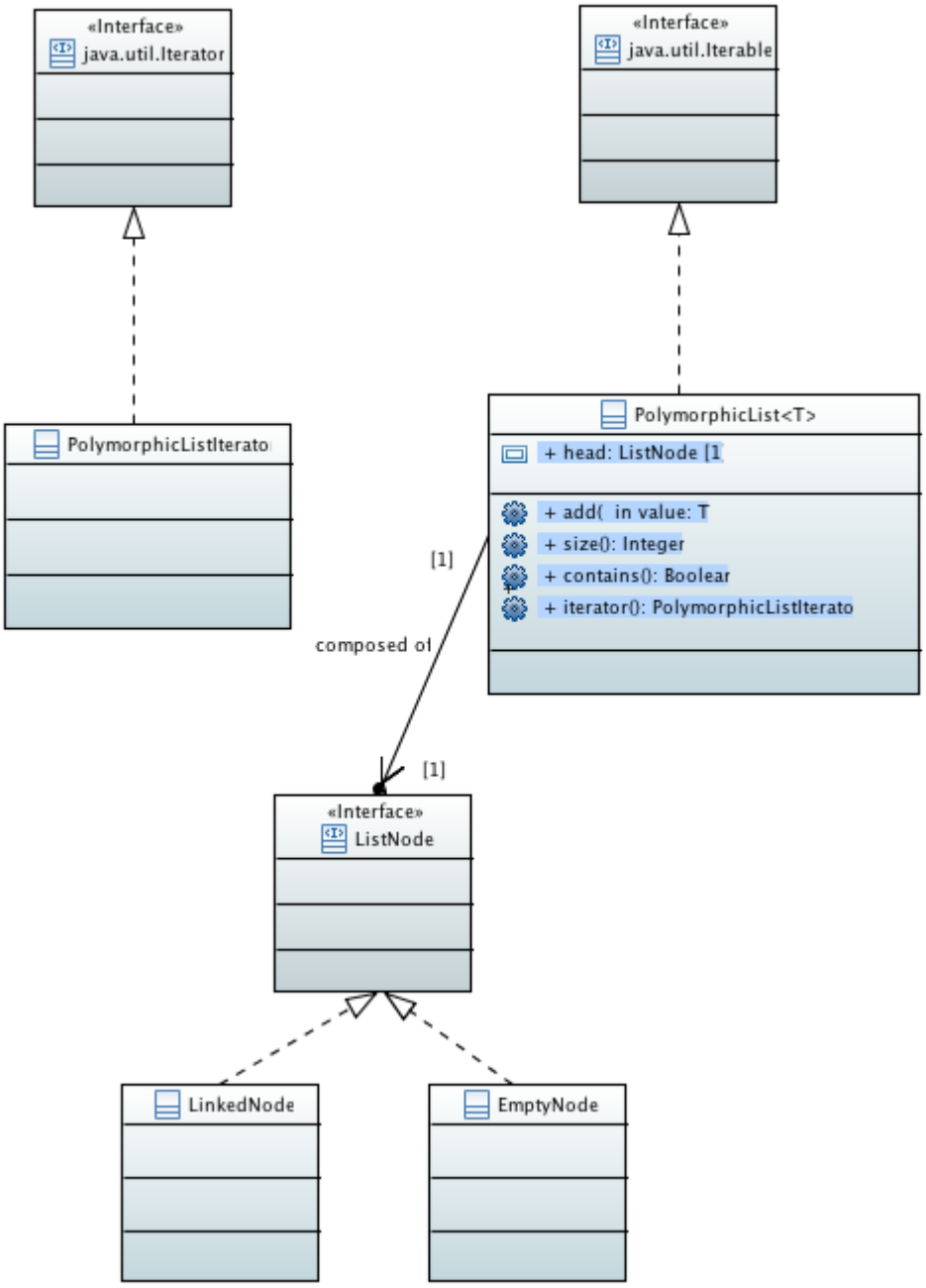
- Create a new **Java** project in Eclipse named UML_class_diagram_lab
- Create a new Papyrus Model named vehicles.di in your src folder.
- Create a new UML Class Diagram in your vehicles.di named Vehicles.
- Add classes (with attributes), interfaces, associations (composition), generalization (inheritance), realization(implements) to your domain model to match the following description:
    - All vehicles have methods forward, reverse, left, right.
    - Cars, vans, and trucks are motor vehicles.
    - Boats and ships are vehicles, but they move and turn in a different way from other vehicles.
    - A truck has methods load, unload.
    - An ambulance is a van that can run red lights.
    - A parking lot holds motor vehicles.
    - A truck can tow 0 or 1 other motor vehicle.
    - A ferry is a boat and can hold up to 40 motor vehicles.

Part B

You are given the class diagram on the following page of a polymorphic list model. You must create the classes, interfaces, and relationships shown in the diagram. A polymorphic list is a single-linked list that uses typed nodes to allow implementation of common operations in a linked list without using loops or conditional execution. Instead, conditions related to the end of the list are handled by polymorphic execution of methods on either LinkedNode or EmptyNode instances.

You are also given the PolymorphicListTest class. Your code must pass the included tests and work in general, but you may not use any conditional constructs (no if statement, ternary operator, or switch statement) or loop constructs (for, while, break/continue). A few hints:

- Your constructor for PolymorphicList should assign head to a new EmptyNode.
- You will need to use recursion on the ListNodes.
- You will need to add methods as needed. The only methods in the diagram that are listed are on PolymorphicList -- you must implement at least these methods so that the tests pass.



Grading Rubric Part A [50 points]

Criteria	Done Well	Need Improvement
Identify a set of conceptual classes that model the problem	A set of conceptual classes is identified that is reasonable for the problem	Conceptual classes are not identified or do not seem reasonable for the problem
Show associations between the classes	Associations are shown that seem reasonable for the problem; associations are named; multiplicity is shown	Associations are missing, or do not seem to be reasonable, or are not named or are missing multiplicity
Show possible attributes and methods for the classes	Reasonable attributes and methods are shown for classes	Some additional attributes or methods could have been identified for the classes
Use the UML class diagram notation as shown in the examples	The diagram follows conventional UML class diagram notation	The diagram does not follow the conventional class diagram notation

Grading Rubric Part B [50 points]

- [20 points] Methods implemented successfully such that tests pass
- [20 points] Code uses polymorphism and not conditionals or loops
- [10 points] Classes and interfaces match diagram