**chap 7 Objection interaction: an introduction**

**Objectives**

* Learn the meaning of Java keyword null
* Learn how to create a collection of objects using Java class List
* Know that List is a collection type of class, and it is also the static binding (generic programming) under the context of polymorphism
* Learn how to use for-each loop, and what is the difference between for-each and a regular for loop
* Know how to define and use superclass (parent class) and subclass (child class)

**Schedule:** This lesson covers Moodle folder “chap 7”. Please follow the steps below.

1, work on sec 7.1 ~ 7.10, which include scenario autumn-1 and autumn-2.

2, notice the difference between for-each loop and a regular for loop:

* for-each loop is easier to write, because the navigation of the array is done automatically.
* for-each loop has to go through every element in the array, and a regular while loop/for loop allows you to skip certain elements because there is a loop control variable in them that you can play with, such as statement ***for(int i=0; i< 10; i+=2)*** will skip the odd index element in the array.
* for-each loop actually makes a copy of the next element in the array, and then manipulate on the element copy in the loop body. Thus you cannot modify the array with for-each loop, because even if you modify the element copy, it will not affect the original array element. In comparison, the regular for loop/while loop can modify any element in the array.

3, in a collection type such as List, we need to plug in another data type such as Leaf inside the pair of < >, to make List a concrete class. The collection class with angle bracket <> is one form of polymorphism called: generic programming, because plugging different classes into < > results in different concrete classes, thus List itself is generic, and it is not attached to any particular class. Generic programming is also called static binding. Please refer to file “chap1-schedule.docx”, the last page, and see the different context of polymorphism. Static binding is one context of polymorphism.

4, work on textbook exercises 7.15, 7.16, 7.17 and 7.18. These exercises are not homework assignment for this week, but it is important to finish them, so that you can get a better understanding of the generic program concept in this chapter. You need to first open the existing scenario “autumn-2” in this chapter, then start working on exercise 7.15.

**There is a bonus homework assigned for chapter 7.** Refer to file ***“bonusHomework-autumnClick.docs”*** for detail explanation. Follow the instructions in this document, in order to work on this bonus homework.

5, download and unzip “**inheritanceExample.zip**”, and study its java source code examples. The inheritance relationship is illustrated by the table below:

|  |  |  |
| --- | --- | --- |
| Superclass |  | Subclass |
| private data | become =========> | inherited data |
| public methods | become =========> | public methods |

which implies, when a subclass inherits from a superclass,

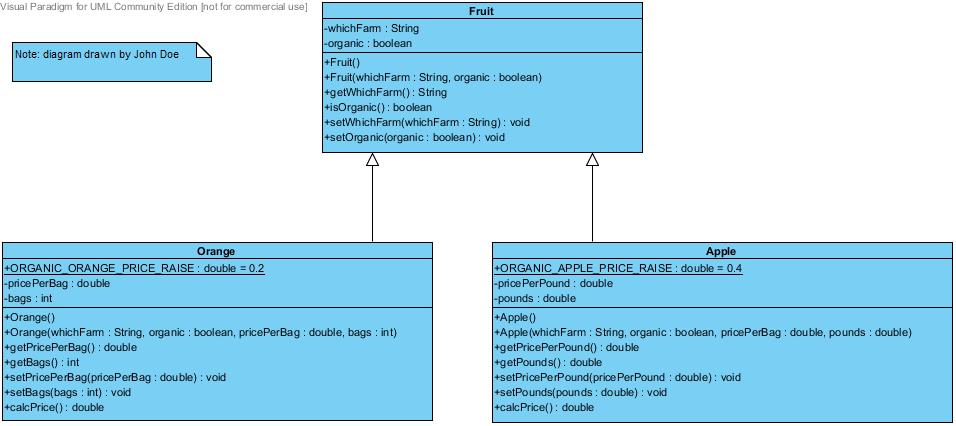
* the public methods of the superclass become the public methods of subclass;
* the private data of the superclass become the inherited data of subclass.

For inherited data in the subclass, even the inherited data exist in each instance of subclass, but because these data fields are defined as private in the superclass, so for the subclass to read, write, or make use of these inherited data fields, the subclass has to call public getters or public setters or public effectors in the superclass.

This is what the OOP principle “encapsulation” is all about:

* private data in class A can only be accessed directly (means without the use of public getters, setters, or effectors) in the class it is defined, i.e., in class A;
* therefore, any other class such as class B has to rely on the public getters/setters/effectors of class A, to access the private data of class A, even if class B inherits from class A.

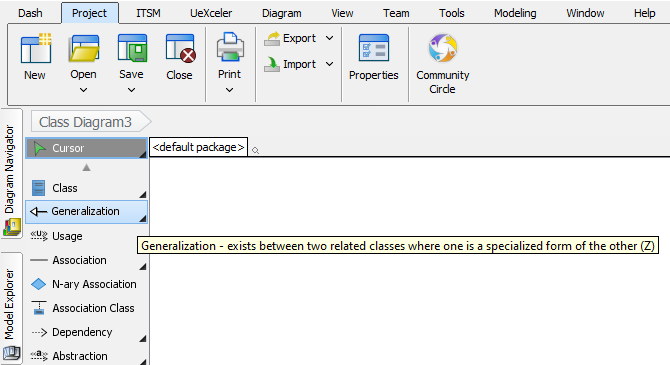
The UML class diagram of the Fruit family is as below, with superclass Fruit and two subclasses Orange and Apple.



In UML class diagram, the superclass is connected to the subclass using a solid line with a triangle shape pointing to the superclass. This connection line is the Generalization icon in UML class diagram. When you use Generalization line to connect a superclass (such as Fruit) to a subclass (such as Orange) , you need to select the Generalization icon first, then draw a line from superclass to subclass, so that the triangle shape points to the superclass, as shown in the UML class diagram above. Do NOT reverse the direction the triangle head is pointing to, and if you let it point to the subclass, it is wrong.

Question: where is the Generalization icon containing a triangle shape at the top with a solid dash line?

Answer: the Generalization icon is right below the Class icon in Visual Paradigm, as indicated below:



Generalization icon represents the inheritance relationship, and it corresponds to the java keyword “extends”. Therefore, we can apply the “is-a” relationship between the child class (subclass) and the parent class (superclass).

In general, in an inheritance relationship, where a subclass extends from a superclass, we can say subclass “is-a” superclass. For example, we can say: Apple “is-a” Fruit, Orange “is-a” Fruit, given the fact that class Apple inherits from class Fruit, and class Orange inherits from class Fruit.

6, there are two usages of Java keyword ***super***, and the first usage is explained as below:

Keyword super is used inside the constructor of a subclass, to invoke a constructor in the parent class (superclass).

Syntax: super(formal parameter list for the constructor in superclass);

Example: file ***Orange.java***, line 25. **super(whichFarm, organic);**

In the above example, ***Orange.java*** is included in file “**inheritanceExample.zip**”.

In file ***Orange.java***, line 25, the super keyword invoke the **Fruit(String, int) constructor** from its superclass **Fruit**, because class **Orange** inherits from class **Fruit**.

7, now you are ready for homework 7, and please follow the instructions in file “**homework7.docx**”, and work on this homework. After you finish homework 7, you need to submit the solution in zip file to its Moodle drop box. When you are coding your homework, please follow the rules in file “RulesForIndentAndAlignCode.docx”. The steps below are not related to homework 7, however, we still need to work on them, and they are part of the OOP principles and will be used in the following homework assignments.

8, the second usage of java keyword ***super*** is explained as below:

Keyword super is used inside a method in the subclass, to invoke a method in the superclass that has **the same method name and method parameters** (i.e., the **same method signature**) in the superclass. In other words, to invoke an ***overridden*** method from the superclass.

Syntax: super**.**overriddenMethodName(formal parameter list for this overridden method);

Example: file ***SmoothMover.java***, line 44**. super.setLocation((int) exactX, (int) exactY);**

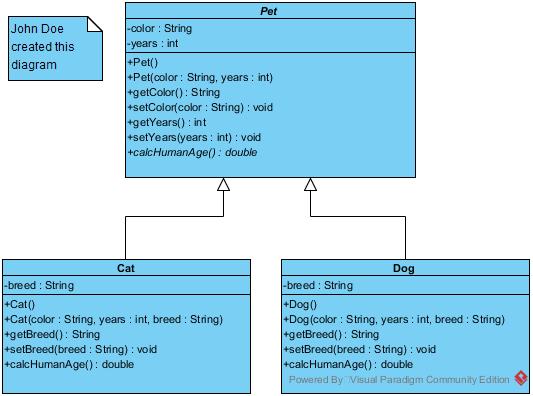
Example: file ***SmoothMover.java***, line 54. **super.setLocation((int) x, (int) y);**

Example: file ***SmoothMover.java***, line 65. **super.setLocation(x, y);**

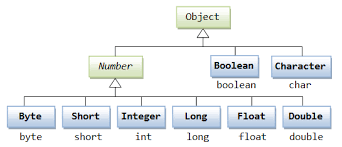
In the above example, ***SmoothMover.java*** can be found in Greenfoot sample project “Newtons-Lab-3” in textbook chapter 8. In ***SmoothMover.java***, line, 44, 54, and 65, the super keyword invoke the **setLocaltion(int, int) method** from its superclass **Actor**, because class **SmoothMover** inherits from class **Actor**.

There is a summary of the use of java keyword this and keyword super on page #16 of “Test1StudyGuide.docx”.

9, download and unzip file “***dynamicBinding-staticBinding.zip***”, then read the associated file “dynamicBinding-staticBinding.docx” that explains dynamic binding and static binding. Follow the instructions in this WORD document and compile and run all the java code examples in this zip file. We introduce the concept of abstract class in this step, and abstract class will help you understand the java interface that will be introduced in next lesson. Also we introduce the dynamic binding, and it will be covered in more detail in next lesson. Be aware that, in homework 7, we don’t use abstract class or dynamic binding yet, but in next homework, we will.

We can also apply the “is-a” relationship to the class family of Pet in folder “exampleOfDynamicBindg”, where both class Cat and Dog inherits from class Pet. So we can say: Cat “is-a” Pet, and Dog “is-a” Pet.

10, for generic class like ArrayList< ElementType>, the ElementType must be a class type, and it cannot be a primitive type. Therefore we have 8 wrapper class types in Java, corresponding to the 8 primitive types, as illustrated by the picture below:



We can see from the picture above, for each primitive type starting with lower case, there is a corresponding wrapper class type starting with upper case. For example, boolean is a primitive type, while Boolean is a wrapper class type, therefore, we can have ArrayList<***Boolean***>, and we CANNOT have ArrayList<***boolean***>.

11, keep working on the questions in file “Test2StudyGuide.docx”, and prepare for test 2. This file is available in Moodle folder “test 2 review lesson”.

12, use Eclipse, work on coding exercise 4.1 to 4.5 on inheritance principle of OOP from this link:

<http://www.ntu.edu.sg/home/ehchua/programming/java/J3f_OOPExercises.html#show-toc>

For this link, first work on exercise 4.1 by yourself. You need to generate the source code based on the given UML class diagram for the Cylinder class, which inherits from the Circle class. Circle class is already available from exercise 2.1. Then you can compare your answer with the answer given on the website. Then you need to create an application class called TestCylinder, which has the main method, and in the main method, you create three objects of Cylinder class using three different constructors, respectively, and then you need to call some methods of the Cylinder class using the Cylinder objects you just create, and output the results of the method callings. Again, compare your TestCylinder class with the answer given on the website.

Then you can work on exercise 4.2 to 4.5, and for each exercise, you need to first create the OOP class based on the given UML class diagram of the OOP class, and then for each OOP class, you need to create an application class that has the main method, and then create several objects of the OOP class type using different constructors, respectively, and then call some method of the OOP class using the objects you just created, and output the results of the method callings, like what you did in exercise 4.1 for class TestCylinder.

Another link addressing inheritance is: <http://www.ntu.edu.sg/home/ehchua/programming/java/J3b_OOPInheritancePolymorphism.html#zz-2> , and there are more explanations about the inheritance principle of OOP in section 2 of this link.

13, next lesson, we will be working on Moodle folder “chap 8”, please study it in advance.