Assignment #2

Nathan Balon

CIS 550

November 7, 2004

Design Decisions

The program is used to create people in a university and then add them to a directory so information can be stored and retrieved for all members of the university. The program contain two main hierarchies' a person hierarchy and a unit hierarchy which from the main structure of the information that must be stored. Along with these two hierarchies, a directory is used to store information about the people in the university. A number of design decisions were made when implementing the program.

Person Hierarchy

The person class hierarchy contains the superclass Person and has a number of subclasses of Person. All classes in the Person hierarchy implement the interfaces Cloneable and Serializable. The Serializable interface is used so that objects can be serialized and written to a file. The Cloneable interface is used so that objects of the type Person can be cloned. The Cloneable interface was used so a deep copy could be performed on the people in the university directory. The classes at the top of the person hierarchy are Person, Student and Employee and are all declared abstract classes. These classes were created abstract rather then be defined as an interface, because it is possible to reuse the code in Person rather then make all subclasses define the methods, such as getName and setName. All of the methods in Person, besides toString and clone are declared final so it is not possible to override these methods. All subclasses of person, also follow this convention of define methods as final that don't need to be overridden by subclasses. Person also uses the "has a" abstraction. The class Person uses composition; the class has the data members Unit and Address which are other classes in the program, so objects of those types will be contained in Person. The class Address was created as separated class rather than having a Person contain one long string containing address information, so it would be possible to search for members based on information in their address easily, such as which people from the university are from Detroit. The Person class defines a constructor that has the parameters name, gender, address, and unit. When a subclass of Person calls the Person constructor these data members will be initialized along with a unique id. The Person constructor uses the singleton ID class to get the next id available. This ensures that each person will have unique id number that can be used to search for a person in the directory.

Subclasses of Person

There are six subclasses of Person which are Student, Employee, Undergraduate, Graduate, Staff and Faculty. Both Student and Employee are direct subclasses of Person. The classes Graduate and Undergraduate are subclasses of Student. The classes Faculty and Staff are subclasses of Employee. Each of these classes defines new data members and methods. The student class defines the data member allowedToRegister that can be set if the student is allowed to register for classes. Student has the methods setAllowedToRegister and isAllowedToRegister which are declared final. The constructor set the allowedToRegister data member to true when a subclass of Student is constructed. The final abstract class in the Person hierarchy is Employee. It defines the data member salary, which is used to hold an employee of the university's salary. The methods get and set salary are defined final this is especially important that these methods are declared final. If someone created a subclass of employee other than the ones that are included in the program if these

methods were not final the person could override these methods and circumvent any checks that were placed on salaries. For instance, if salaries are not allowed over a certain number of dollars someone could define a new class and avoid this constraint. The abstract classes were declared abstract rather than interface because these classes have implementation that can be reused by there subclasses.

The rest of the classes in the hierarchy can be instantiated. Each of these classes defines new methods that are specific to the behavior to that class of object. For instance Graduate defines the methods getLevel and setLevel along with the overridden method toString and clone. The methods toString and clone are overridden in each class in the hierarchy. The get and set methods accept an argument of the type GraduateLevel which is an enum of valid Graduate levels MASTERS and PHD. Enums are used for other leaf classes in the hierarchy such as the enum StaffStatus that is used to indicate if a staff member is PARTTIME or FULLTIME. The rest of the classes Undergraduate, Faculty and Staff are implemented similarly.

Unit Hierarchy

The Unit hierarchy has the superclass Unit that is declared abstract the class has no abstract methods, but is declared abstract, because it would not make sense to be able to declare a generic unit in the university program. The Unit class has the data members name and address. A get and set method is defined for name along with a get method for name. The set method for name was left out since the name of a unit wouldn't change after it is added to the university. These methods are again declared final. The other abstract classes in the unit hierarchy are Admin and Academic. There are five concrete classes in the Unit hierarchy they are Library, HR, CIS, Education, and Admissions. All of the concrete classes in the Unit hierarchy are singleton classes since there will only be one of each of these for the university. It wouldn't make sense to allow to instances of computer science to be created for the university this is the reason for the singleton pattern. Most of the classes in the Unit hierarchy have limited implementation besides the methods related to the address and name of the Unit. The Library class has uses a Book class to create books and store them in a list. Also, the CIS class has a list that of all the programming languages taught by the department.

Main Method of Application

The class UniversityApplication is the main entry point to the program. The main purpose of the application is to create people and add them to the directory. The main method is implemented by first checking if a file of the directory exists and if it doesn't the file is created. The file is used to people can be stored persistently. After the file is either opened or created, the main method contains a loop that continues to run until user decides to exit the program. The main menu is generated by using a static class Menu, which is used to display various menus in the program. After the menu is displayed the user can then, enter there choice to perform specific action on the directory. The options that the user has is: to add a person to the university directory, to list all people who are in the directory, to list all people who are of a certain type such as all undergraduate students, to list all people who belong to a certain unit such as all people who are affiliated with the computer science department, to list the number of entries of each unit

and categories for a example the number of people who are graduate students, to search the directory for a person by name or id, and last to exit the program. Based on the users input a method will be called and the result will be returned to the user. When the program is exited, the directory is written to a file so the next time the program is ran the data can be retrieved.

Person Factory

Since it is not know at the time when a person is to be created what type of person it will be a factory method is used to create people. The factory pattern is used by the client calling the static method createPerson of the class PersonFactory. The PersonFactory is another hierarchy of classes with PersonFactory the superclass. The class hierarchy contains StudentFactory, EmployeeFactory, GraduateFactory, UndergraduateFactory, StaffFactory and FacultyFactory. The concrete classes in the hierarchy are GraduateFactory, UndergraduateFactory, StaffFactory and FacultyFactory which all implement the interface Build which has a single method build. Once one of Factories that implement the interface build is called the Factory calls a factory method of its superclass which in turn classes a Factory method in the Person class which is buildPersonInfo. These methods each read the users input from the console to create a new Person. Once all the information is correctly entered the build method will create and return a Person to the method createPerson which will then return the Person to the client. Using the factory pattern avoid the problem of not knowing which type of person to create.

University Directory

The University Directory uses the facade pattern. There is a class called UniversityFacade that is used in the main method of the application. Along with this class there are two other classes that actually store the people of the university they are UnitDirectory and CategoryDirectory. By using the facade pattern objects can be added to these two directories without the client being aware that the directory is actually composed of more than one directory. Within the CategoryDirectory and UnitDirectory are a number of list that contain the people of the university. Separate list were created so that when people are added, they are contained in a list specific to there type. So for instance, if a person is affiliated with the unit CIS they will be added to the cisList in the UnitDirectory. Two Hashmaps are also used to find the list that a person is in when searching for them. A person can be search by both name and id so the Hashmap contain both an “id unit lookup” and a “name unit lookup”. So if all that is known is a persons id number, it can be looked up in the Hashmap and determined what unit the person belongs to, then it is only necessary to search though one list to locate the persons information. This has the benefit of avoiding have to search through all lists till a person is found.

The DirectoryFacade class also declares a constructor that accepts the argument of the type DirectoryFacade. This is used to create a new directory object by performing a deep copy of another directory object. The constructor creates a new UnitDirectory and CategoryDirecotry objects by call the clone method on these objects. These two objects implement the clone method; the deep copy is performed by going through each list they contain and calling the clone method on a Person object and then storing the objects in a new list then returning the object. The clone method was not provided for the Unit classes

since they are singleton classes and can have only on instance so in this case the Person clone method only copies a reference.

It would have been possible to create just two lists for the entire directory, but it would have poor performance for searching. This is the reason for using a separate list for each category and unit. The drawback to using separate lists and Hashmaps is the storage requirements will increase. An alternative could be creating a class for each type of list so there would have been nine classes in total to store the directory information one for each type of list, this really wouldn't have reduced the amount of code needed but may make it easier to add new classes to the program that need to be stored. Another possibility would have been to combine both the visitor and facade pattern. The visitor pattern would have made it easy to add new types of objects to the directory. Since, the type of classes that need to be held in the directory will remain static just the facade pattern was used. The drawback though is if a new unit class was added the methods in the Unit directory would need to be modified.

Implementation Concepts

i) Polymorphic variable

A polymorphic variable is that can hold a different type. The static type of a variable may be one type when the variables dynamic type may differ from the static type. One case where polymorphic variable is used in the university directory is the person variable. The person variable may be holding another type such as a graduate student. The benefit of this is when the toString method is called on person it will call the method from graduate student.

ii) True polymorphism

True polymorphism occurs when an argument to a method is a polymorphic variable. The addPerson method of UniversityDirectory is one case where this occurs. Different types of people can be added to the university directory and based on there type they will be add to the correct list. Based on the dynamic type of a variable passed to the function add different action will be performed.

iii) Composition (for code reuse)

Composition is a form of the "has a" abstraction. Address is one instance of where composition for reuse occurs in the program. Address is a data abstraction that is used as part of another class. In the case of the university program, both the Person class and the Unit class in the university have an address associated with them.

iv) Subtyping

A subtype is when an instance of one class can be substituted for an instance of another class with no observable effects. One case where subtyping exist is faculty is a subtype of person. So, a Faculty object can be substituted for a Person object in the program.

v) Inheritance

Inheritance is used extensively in the university directory. One instance where inheritance is used in the program is inheritance for specialization. There is a hierarchy of different types of people in the university. Each subclass of Person adds more specific behavior. So an Employee class is a subtype of Person and adds additional methods such getSalary and setSalary.

vi) Overriding

Overriding occurs when a method in a subclass has the same signature as a method in a parent class and is used to override the behavior of the method in the parent class. One case where overriding is used is the toString method. The method is overridden in each of the subclasses of person. So depending on which subclass of person the method is call on different output will be returned. So it is possible to print out the information about a person and then override the method to add specific information for a staff member such as there salary.

vii) Overloading (Scope or Type signature?)

Overloading a method occurs when a method with the same name occurs more then once in a class but differs in the method arguments. About the only case where overloading was used is in the constructors. For instance person has both a default constructor and a constructor that takes four arguments the name, sex, address and unit.

viii) Coercion/conversion during substitution

One instance of where substitution as conversion takes place is with the method add of the directory. The add method add a person to the directory and the parameter is a person but it is possible to pass other objects to that method other than a person object. For example it is possible to pass a graduate student to the method add even though the method is defined to accept a person.

ix) Copying (shallow or deep)

Copying can be either a sallow copy or a deep copy. In the case of a shallow copy the pointers are copied so two classes will share an object. When a deep copy occurs instead of copying the pointers new objects are created. By using a deep copy if changes are made to an object the changes won't affect the other object that was involved in the copy operation. In the university program uses a deep copy when coping an object of the directory type. Each person in the university clone method is called that implements a deep copy, On the other hand the for the unit a shallow copy is performed since they unit is a singleton object and creating a deep copy would have no effect since there can only be one unit object of each type.

x) Abstract class and methods

An abstract class is a class that can not be instantiated but used as a base class for the purpose of inheritance. There are a number of abstract classes in the program. Some of the abstract classes are Person, Unit, Student, Employee, Academic, Admin, PersonFactory, EmployeeFactory, and StudentFactory. The only place that abstract methods were used is in the interface in the program for instance the interface Build has a build method that is abstract.

xi) Identify the type(s) of inheritance used

A number of different types of inheritance were used in the program. Subclassing for specialization was used extensively in the program. Some examples of its use are the subclasses of Person. Subclassing for specification was used. The Build interface defines a build method and each factory that implements the interface defines the build method. Syubclassing for extension is also used in the program for instance Library is a subclass of Unit but it add additional methods that are not available in Unit.

xii) Default constructor

A default constructor is a constructor that takes no arguments. One instance of the use of a default constructor is in the singleton ID class. When the default constructor is invoked it initializes the id to 1.

xiii) Refinement

Refinement occurs when an overridden method combines code from the parent class with code in the child class. The toString methods of all the classes that inherit from Person call the `super.toString()` method and also add there specific information to the string. The code in the parent and child is the merged together.

xiv) Reverse polymorphism

Reverse polymorphism is used when a `Person` is added to the university directory. In the `UniversityDirectory` there are separate list to store the different types of people. The lists use generics and in order to add a person to the correct list it must be cast back to its dynamic type.

xv) Design Pattern

A few of the design patterns that were used were the singleton pattern so that only one instance of `ID` can exist in the program so each `ID` will be unique. The factory pattern was also used to construct the people of the university. Since it is not known at compile time what type of person needs to be created the factory pattern was used. The facade pattern was also used for the two different directories. The façade pattern allows multiple directories to exist and the user of the directory only need to know about one directory. So façade pattern hides some of the implementation details from a user.