Homework #1

Nathan Balon
CIS 550
Object Oriented Programming
October 3, 2004

Homework 1 – Intro to OO

Due: Sep. 28, 2004

1- Consider a university campus with the following community: Staff, Faculty and Students. Staff includes full-time and part-time employees. Faculty can be Adjunct or Tenure-track faculty. And students can be Undergraduate or Graduate. Graduate students can further be Masters or PhD, and Undergraduate students can be in their Junior or senior. Staff and Faculty are employees of the university, and, along with students, form the "people" in campus.

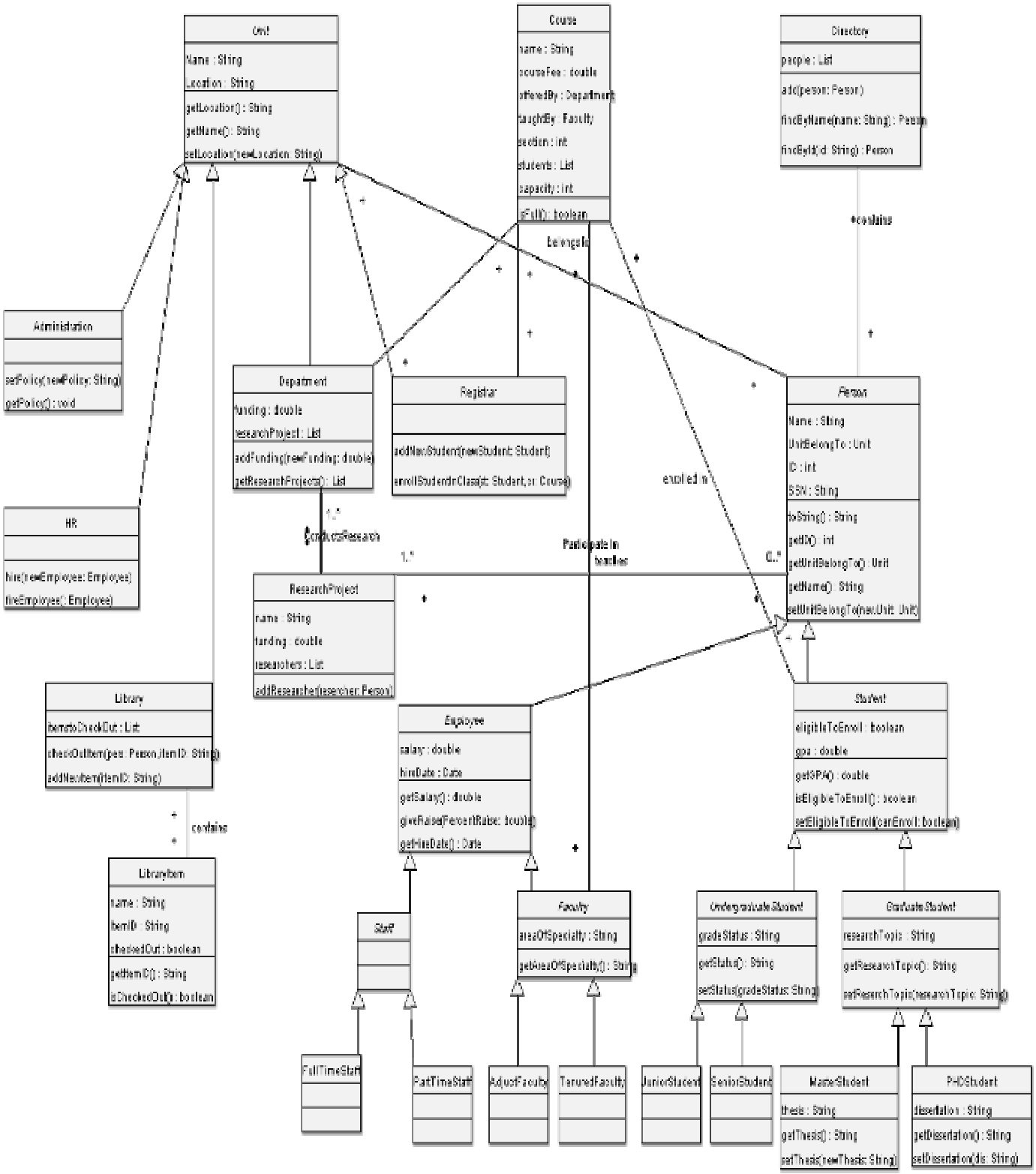
All employees and students of the university have their information stored in a *Directory*, that is used to view the information of different members of the campus. The search for an individual can be done by their name or ID. Once a member is located, all their information (except for confidential info, such as SSN) is displayed.

The university also has different units, such as the library and all the academic departments, such as CIS, Education, Science, etc., the Human Resources and Administration buildings.

Every member of the campus is affiliated to one unit, i.e., their office location or the department a student is enrolled in.

Your task is to model a university system based on the above information. You classes include: People, Units, Directory, Faculty, Staff, Students, Employee, Library, HR, Administration, Registrar, etc.. These are some of the issues you should address:

University Class Diagram



a) Which classes are abstract classes?

An abstract class is a class that can not be instantiated. For instance, if class Person was declared abstract, it would be an error to try to instantiate an object of type person as follows: `Person p = new Person()`. Abstract classes are used when it doesn't make sense to create an object of a certain type. A classical example used to explain abstract classes is a shape class with a draw method. It doesn't make sense for a shape to be able to draw itself, so the draw method is declared abstract. On the other hand a class called line could inherit from shape and then implement the method draw. The abstract classes for the university are: Person, Unit, Employee, Faculty, Staff, Student, UndergraduateStudent, and GraduateStudent.

b) What type of abstractions will you use to model this problem? Identify where "is-a", "has-a", "part-of", etc. are used. Illustrate inheritance through an inheritance tree.

The University software will use the "is-a" and "has-a" types of abstraction. The Person class is a super-class of many of the other class in the university system. For instance Employee and Student are both subclass of Person, because a student is a person and an employee is a person. Furthermore, GraduateStudent is a sub-class of Student. The GraduateStudent class passes the "is-a" test since a graduate student is a student. The University also uses the "has-a" abstraction. The directory of the university has the persons of the university contained in it. Also, the library has library items, which it to loans out.

c) Identify the attributes and methods of each class

The attributes and methods that a class has are identified in the class diagram above. It is possible to add additional methods and attributes to the class diagram. Depending on how the software will be used the number of methods needed may increase or decrease. Depending on the domain that the software would be used in the number of classes for the university could grow dramatically. For instance, if an actual universities information system was to be created from the design a lower level of abstraction would be needed. It would be necessary to create additional class such as the library may have a number of different classes for the items it loans out. On the other hand, if the software was design to just simulate a university the level of detail given in the class diagram above may suffice. For some of the classes, I'm not really sure what the difference in behavior and attributes are on such case is the difference between an adjunct professor and a tenured professor. Also, for the undergraduate students it may be possible to eliminate the senior and

junior student classes. It may not be beneficial to have separate classes for the four types of undergraduate students it would probably be more beneficial to add an attribute for the student grade level. This subtype may or may not provide any benefit when actually constructing the software.

d) When a student or employee is located in the directory, the information displayed always has their name and location, for example. Illustrate how one method called *describe* or *toString* can be invoked to display the info of all types of objects.

Each class should define a toString method. The method toString is included in the abstract class Person. Each subclass of Person then can override the method toString, to produce class dependent behavior. For instance the Employee class could return along with their name and id, the number of years employed for the university. Having each class provide a toString method will allow the method to be call polymorphically.

e) How are the constructors defined? What do they initialize?

Constructors are method of a class that initializes an object to a consistent state. The constructor for a class has the same name as the class and has no return type. A class may have more than one constructors defined as long as there signatures are different. When an object is instantiated a constructor is called to initialize an object. A constructor is used to initialize the data members of an object, so the object will be in a consistent state when it is used. The primary benefit of using constructor is they guarantee that an object is never used that has not be initialized. In the case of the university class diagrams the constructors were left out of the diagram to provide clarity. The person class should have a constructor of the form Person(String name, Unit unit, String ssn) .

f) How is a new employee added to the directory – i.e., what method is required and where should it be located?

The class Directory should have a method named add that adds people to the directory. The method could have the signature: void add(Person). By only declaring a method such as add it will allow the programmer to have different choices in the data structure to use to store the people of the university in the directory. The data structure of used for the directory could be changed in the future without affecting the users of the class since they are just using an interface which is a method.

g) What type(s) of inheritance is used in your solution? Does it allow subtyping? Give examples.

Sub-typing is used in the solution to allow for substitution of objects. The solution uses sub-classing for extension, sub-classing for specialization and sub-classing for specification. The system does allow sub-typing. For instance, UndergraduateStudent is a subtype of Student. Also, TenuredFaculty is a subtype of Faculty. This design allows the substitution of an instance. An example of this is the substitution of a MasterStudent for an instance of a GraduateStudent.

h) Would it be appropriate/efficient/recommended to have *Directory* as an interface or abstract class? Why? What about *People?*

It would be appropriate to have the directory defined as an interface. If the directory was defined as an interface it would be easy to add new class that implemented a directory interface for the university. Classes would just have to implement the methods that are defined in the directory interface. Using the interface would allow a number of classes to implement a directory interface and be used interchangeably. In the case of the person class it should be abstract. Methods such as getname() will be the same for all classes that inherit from person. Using an abstract class here over an interface is mostly a convenience issue. So instead of making each class implement the person interface and implement the method it would make more sense to put the implementation in an abstract class and have other classes inherit from it.

i) Illustrate three different abstraction **levels** of your system.

The highest level of abstraction is the services provided by the university such as registering new students or hiring teachers. The next level of abstraction contains how classes in the system relate to each other and how messages are sent between classes. This level of abstraction can be shown using UML and a class diagram. The lowest level of abstraction contains how the actual methods in the system are implemented. Issues at this level are those such as: what type of data structure to use for the directory, and how the employees are added to the directory. At this level, the actual implementation of methods is considered.

2- Insert something at "insert here" in the following C++ code:

```
#include <iostream.h>

using namespace std;

class question2{
  public:

    question2(){
       cout << "Initialize\n";
```

```
        }

        virtual ~question2(){
           cout << "Clean up\n";
        }
    };

    main()
    {
        question2 q2;
        cout << "Hello, world\n";
       delete q2;
    }
```

so that it produces the output:

Initialize
Hello, world
Clean up

Do NOT modify *main()* in any way.

3- Given the following C# code:

```
class Animal {
        public virtual void WhoAreYou() { Console.WriteLine("Iam an animal");}
}

class Dog: Animal {
        public override void WhoAreYou(){Console.WriteLine("I am a dog"); }
}
```

Add a *Cat* and *Cow* subclasses to the *Animal* class. Each subclass should implement a method *Likes(string food*), which returns true if the animal likes the food that was passed as a parameter. Each animal should also have a method *Speak()*, which returns a string with the sound that is produced by this animal (e.g., "woof" for *Dog*). An animal farm should then be usable as follows:

```
foreach (Animal a in farm) // where fram is an array: Animal[] farm
   if (a.likes("fish")) a.Speak();   // this shoud return "woof" or "moo" or a message stating that the given animal does not like the food passed as parameter.
```

The array *farm* can be initialized as:

```
static Animal[] farm = { new Cow(), new Dog(), new Cat(), new Dog()};
```

You should also have a *name* method that will return the name of the animal. For example, in the *Dog* class, you would have:

public string Name { get {return “dog”;}) // using property

Since all dog, cow and cat classes inherit from Animal, they will all *override* the methods from *Animal*. Thus, all the methods in *animal* will be abstract, i.e., without any implementation.

```
using System;

namespace Ass1Prob3
{
	/// <summary>
	/// Summary description for Class1.
	/// </summary>
	///
	abstract class Animal
	{
		abstract public void WhoAreYou();
		abstract public bool Likes(string food);
		abstract public string Speak();
	}

	class Cat: Animal
	{
		public string Name
		{
			get
			{
				return "cat";
			}
		}

		public override void WhoAreYou()
		{
			Console.WriteLine("I am a cat");
		}

		public override bool Likes(string food)
		{
			if(food.Equals("fish"))
			{
				return true;
			}
			else
			{
				return false;
			}
		}

		public override string Speak()
		{
			return "Meow";
		}
	}
```

```
class Cow: Animal
{

	public string Name
	{
		get
		{
			return "cow";
		}
	}

	public override void WhoAreYou()
	{
		Console.WriteLine("I am a cow");
	}

	public override bool  Likes(string food)
	{
		if(food.Equals("grass"))
		{
			return true;
		}
		else
		{
			return false;
		}
	}

	public override string Speak()
	{
		return "Moo";
	}
}

class Dog: Animal
{
	public string Name
	{
		get
		{
			return "dog";
		}
	}

	public override void WhoAreYou()
	{
		Console.WriteLine("I am a dog");
	}

	public override bool  Likes(string food)
	{
		if(food.Equals("bone"))
		{
			return true;
		}
		else
		{
			return false;
		}
```

```
            }

            public override string Speak()
            {
                  return "Woof";
            }
      }

      class AnimalImplementation
      {
            /// <summary>
            /// The main entry point for the application.
            /// </summary>
            [STAThread]
            static void Main(string[] args)
            {
                  Animal[] farm = {new Cow(), new Dog(), new Cat(), new
Dog()};

                  foreach (Animal a in farm) // where farm is an array:
Animal[] farm
                  {

                        a.WhoAreYou();
                        if(a.Likes("fish"))
                        {
                              // this shoud return “woof” or “moo” or a
                              // message stating that the given animal
                              // likes the food passed as parameter
                              Console.WriteLine(a.Speak());
                        }

                  }
                  Console.ReadLine();
            }
      }
   }
```

4- Define a class “parent” so that the lines of code below are either okay, or give errors, as described by the comments:

```
class child: public parent
{
public:
      play()
      {     f(); // should be okay
            g(); // should be okay
            h(); // should give error
      }
};

void another(void)
{
      parent trap;
```

```
        trap.f(); // should be okay
        trap.g(); // should give error
        trap.h(); // should give error
}
```

How would a C++ code differ from Java in this? What about C#?

```
class parent{
        public:
                void f(){ }
        protected:
                void g(){ }
        private:
                void h(){ }
};
```

C++ contains three types of access control public, protected and private. In C# the default for a data field is private as opposed to Java where the default is package. Java on the other hand has four types of access control public, protected, private and the default package. In Java if no access specifier is included then it will have package level access control, then the method or variable can be accessed by all class in the package. In C# also fields marked as private can be accessed through public properties. C# defines 5 types of access modifiers. The access modifiers are: public, private, protected, internal and protected internal. The first three are basically the same as those in C++. Internal allows other classes to access a member if it is in the same assembly. This is similar to Java's package access. Protected internal allows subclasses to access a member and other class in the assembly to access a variable. These are the main differences between the access modifiers in the three languages.