Exercise No. 2

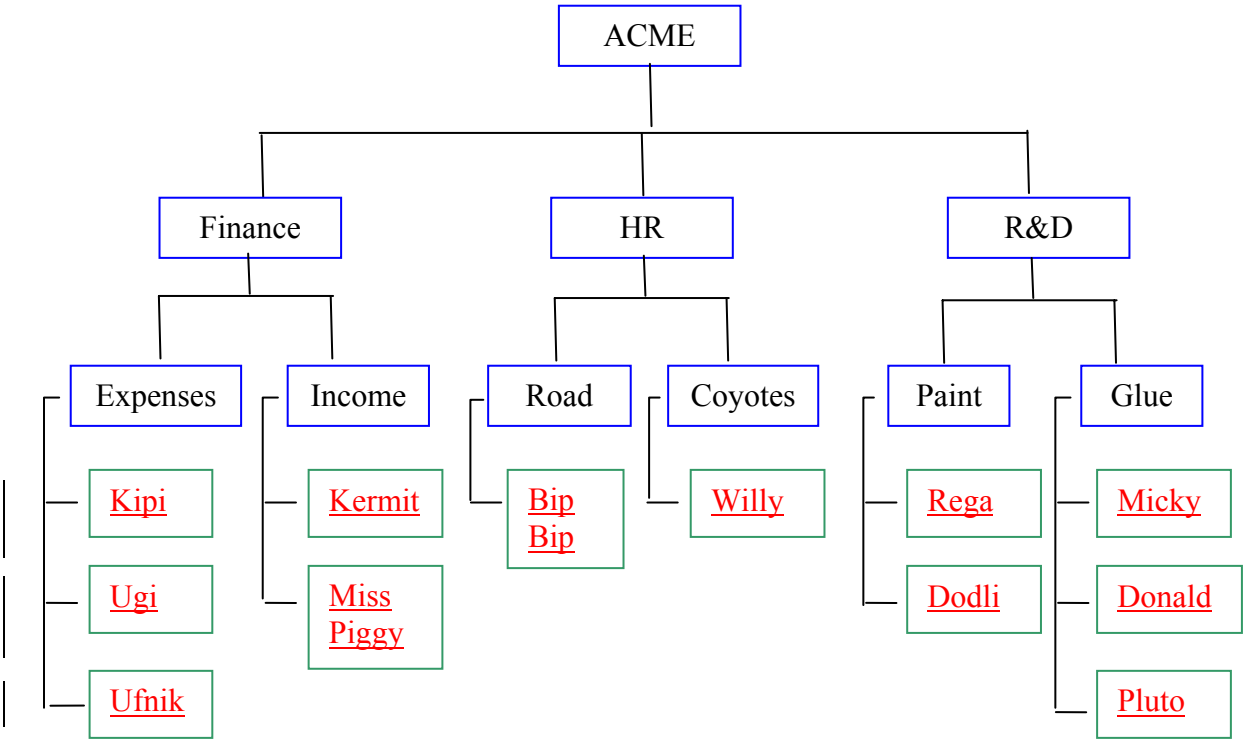
Due Date: 03.08.2006, 17:00

Part 1:

The purpose of this part of the exercise is to practice using objects.

On the course web-site you will find a jar[1] file with a set of classes representing a simple model of a company. There are two classes: **Department** and **Worker**. They each have a set of attributes that describe them and methods that maintain the relationship between them.

The Acme company is founded with the following initial structure:



1. Implement the method found() in the class AcmeCompany. The method should create a Department object that represents the entire company, as shown in the diagram above. This method should construct many objects and connect them as appropriate.
2. Implement a method findWorker(String name) that returns a worker given his/her name, and a method findDepartment(String name) that returns a department given its name. These two should use recursion.
3. Implement a method moveUgi() that moves the worker Ugi from Expenses to Income.

[1] An explanation about jar files can be found in the Eclipse handouts on the course web-site.

4. Implement a method fireKermit() that removes Kermit from the company.
5. Implement a method reorganize() that closes the Glue department, fires Miki and Donald, and moves Pluto to the Coyotes department.

Part 2:

The main purpose of this part of the assignment is to practice Design By Contract, both in implementation and proof.

You will implement a class named DistjointSets that represents a set S of disjoint sets: S = {S1,S2, ...,Sn}, where each Si is a set of nonnegative integers. The class will support the following public methods:

```
// @pre x is not in any of the sets Si
// @post S = old(S) U {{x}}
public void makeSet(int x);

// @pre x in Si , y in Sj
// @return true iff i == j
public boolean equiv(int x, int y);

// @pre x in Si , y in Sj , i !=j
// @post S = old(S) - Si - Sj U {Sij} where Sij = Si U Sj
public void joinSets(int x, int y);

// @pre nothing
// @return true iff x is in some Si
public boolean inASet(int x);
```

One way to implement the class is by representing each set Si as a tree, where each node (associated with a nonnegative integer x) points to his parent. In this way, the root of a tree uniquely represents a set Si. An array named parent can be used to hold all these pointers. Specifically, for each number x, the value of parent[x] is the number of the parent node in the tree or -1 if the number x does not belong to any set Si. The root of a tree points to itself. The length of the array should be bigger than any number x that currently in one of the sets Si. Thus, there is a need to replace the current array with a bigger one when a new large number is added.

Given an object of the DisjointSets class, its abstract state is a set of disjoint sets of nonnegative integers, {S1,S2, ...,Sn}, where the abstraction function can be defined as follows:

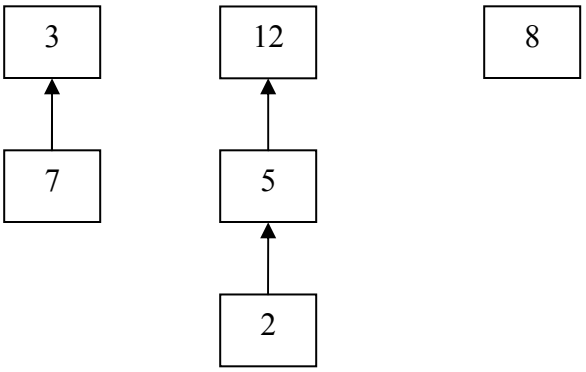
We define a function r(m,x) as follows:
r(1,x) = parent[x],
for m>0 r(m+1,x)= r(m,parent[x]) if parent[x] ≠ -1, otherwise r(m,parent[x]) = -1

S(this) = {S_1, S_2, … S_n} such that
For all x, [for all i, $1 \leq i \leq n$, $x \notin Si$] iff [$x \geq$ parent.length or parent[x] = -1]
For all x,y $0 \leq x,y <$ parent.length Exists i, 1 <=i <= n x,y $\in$ Si
iff there exist m1 and m2, such that r(m1,x) = r(m2,y)

F

-1	-1	5	3	-1	12	-1	3	8	-1	-1	-1	12
0	1	2	3	4	5	6	7	8	9	10	11	12

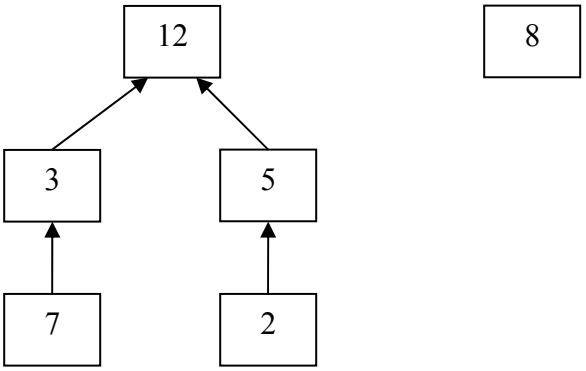
the associated tress are:



and the abstract state is S = { {3, 7} {12, 5, 2} {8} }. Applying `equiv(3,5)` will return `false`. If we apply `joinSets(7,5)` the parent array will be

-1	-1	5	12	-1	12	-1	3	8	-1	-1	-1	12
0	1	2	3	4	5	6	7	8	9	10	11	12

where the associated trees are



and the abstract state is S = { {3, 7, 12, 5, 2} {8} }. If we apply `makeSet(4)` the abstract state will be S = { {3, 7, 12, 5, 2} {8} {4}}.

Your assignment is to:

- Implement the DisjointSets class efficiently using the parent array (a template can be found on the course web site). Note that for your convenient you may need to define a few private methods. For example,

```
// @return the root of the tree to which the node with
// number n belongs
private int root(int n)

// If needed, replace parent by an array large enough,
// with same content.
private void adjustArraySize(int n) {
     if (n >= parent.length) {
                ……
     }
}
```

  Also, the `Arrays` class in the `jave.util` package may be very helpful.
- Define the representation invariant of the class

Comments about the Submission:

1. Create a new java project, named `ex2`, with the default properties.
2. Under this project create a package, named `il.ac.tau.cs.<your user name>.ex2`
3. Download the template files for the exercise (for both parts) from the course web-site and import it into your package.
4. Implement the required classes. Note that you are allowed to add helper methods, but do not change the signature (visibility, name etc.) of the methods specified above.
5. Add proper documentation (as comments) that describes the contract of the classes.
6. You should hand-in a printout of the source files and a ZIP containing the same files as explained in the submission guidelines document on the course web-site.