Composite pattern

```
public class Drive {
    protected String name;
    protected Collection<File> fileList = new ArrayList<File>();
    protected Collection<Directory> dirList = new ArrayList<Directory>();

    public Drive(String name) {
      this.name = name;
    }
    public void addFile(File file){
      fileList.add(file);
     }

    public void addDir(Directory directory){
      dirList.add(directory);
    }


    public int computeSize(){
      int sizeInBytes=0;
      for (File file : fileList){
            sizeInBytes+=file.getSizeInBytes();
      }
      for (Directory dir : dirList){
            sizeInBytes+=dir.computeSize();
      }
      return sizeInBytes;
    }

    public void print(){
      System.out.println("- drive "+name);
      for (Directory dir : dirList){
            dir.print();
      }
      for (File file : fileList){
            file.print();
      }
    }
}

public class File {
    private String name;
    private int sizeInBytes;

    public File(String name, int sizeInBytes) {
            this.name = name;
            this.sizeInBytes = sizeInBytes;
      }

      public int getSizeInBytes() {
            return sizeInBytes;
      }

    public void print(){
      System.out.println("--- file "+name+" size="+getSizeInBytes()+"
bytes");
    }
}
```

```java
public class Directory {

    protected String name;
    protected Collection<File> fileList = new ArrayList<File>();
    protected Collection<Directory> dirList = new ArrayList<Directory>();


    public Directory(String name) {
      this.name = name;
    }

    public void print(){
      System.out.println("-- dir "+name+" size="+computeSize()+" bytes");
      for (Directory dir : dirList){
            dir.print();
      }
      for (File file : fileList){
            file.print();
      }
    }
    public void addFile(File file){
      fileList.add(file);
     }

    public void addDir(Directory directory){
      dirList.add(directory);
    }


    public int computeSize(){
      int sizeInBytes=0;
      for (File file : fileList){
            sizeInBytes+=file.getSizeInBytes();
      }
      for (Directory dir : dirList){
            sizeInBytes+=dir.computeSize();
      }
      return sizeInBytes;
    }
}

public class Application {
      public static void main(String[] args) {
            Drive cdrive = new Drive("C");
            Directory appdir = new Directory("applications");
            Directory datadir = new Directory("my data");
            Directory coursedir = new Directory("cs525");
            File excelfile = new File("msexcel.exe", 2353256);
            File wordfile = new File("msword.exe", 3363858);
            File studentsfile = new File("students.doc", 34252);
            cdrive.addDir(appdir);
            cdrive.addDir(datadir);
            datadir.addDir(coursedir);
            appdir.addFile(excelfile);
            appdir.addFile(wordfile);
            coursedir.addFile(studentsfile);
            cdrive.print();
      }
}
```

If we execute Application.java, we get the following output:

```
- drive C
-- dir applications size=5717114 bytes
--- file msexcel.exe size=2353256 bytes
--- file msword.exe size=3363858 bytes
-- dir my data size=34252 bytes
-- dir cs525 size=34252 bytes
--- file students.doc size=34252 bytes
```

This file system contains a tree structure of drives, directories and files. The problem with this code is that if we want to add a new class to this tree structure, for example a Link, which is a link to a remote drive, directory or file, then we have to change a lot of code. The Directory class would also need a list of Link objects. We learned that the Composite pattern can be applied when we have tree structures of objects.
Redesign the application with the Composite pattern applied to the application.
In your solution it should be easy to add new objects to the tree structure. So it should be easy to add a Link object to the file system.

Draw the class diagram with the Composite pattern applied.