Practical 4

Janco Spies, u21434159

September 26, 2022

Task 1

1.1 #3 The error produced by the program is an Arithmetic exception at line 19. The function arguments were a=-2, b=0.

1.1 #5 The error occurs in the *improve* function that is called by the main function at line 13.

1.1 #8 *highest* = 0

1.1 #9 The error occurs because the *improve* function is called with the arguments *a*=-2 and *highest*=0. This causes a division by 0.

1.2 #3 Process ID: 2175

1.2 #4 Invalid write of size 4.

1.2 #5 It tells me that the error comes from the *capture* method at line 4 which is called by the *main* method at line 9.

1.2 #6 An integer is 4 bytes long and an array of 10 integers has not been freed. Therefore 40 bytes have been lost.

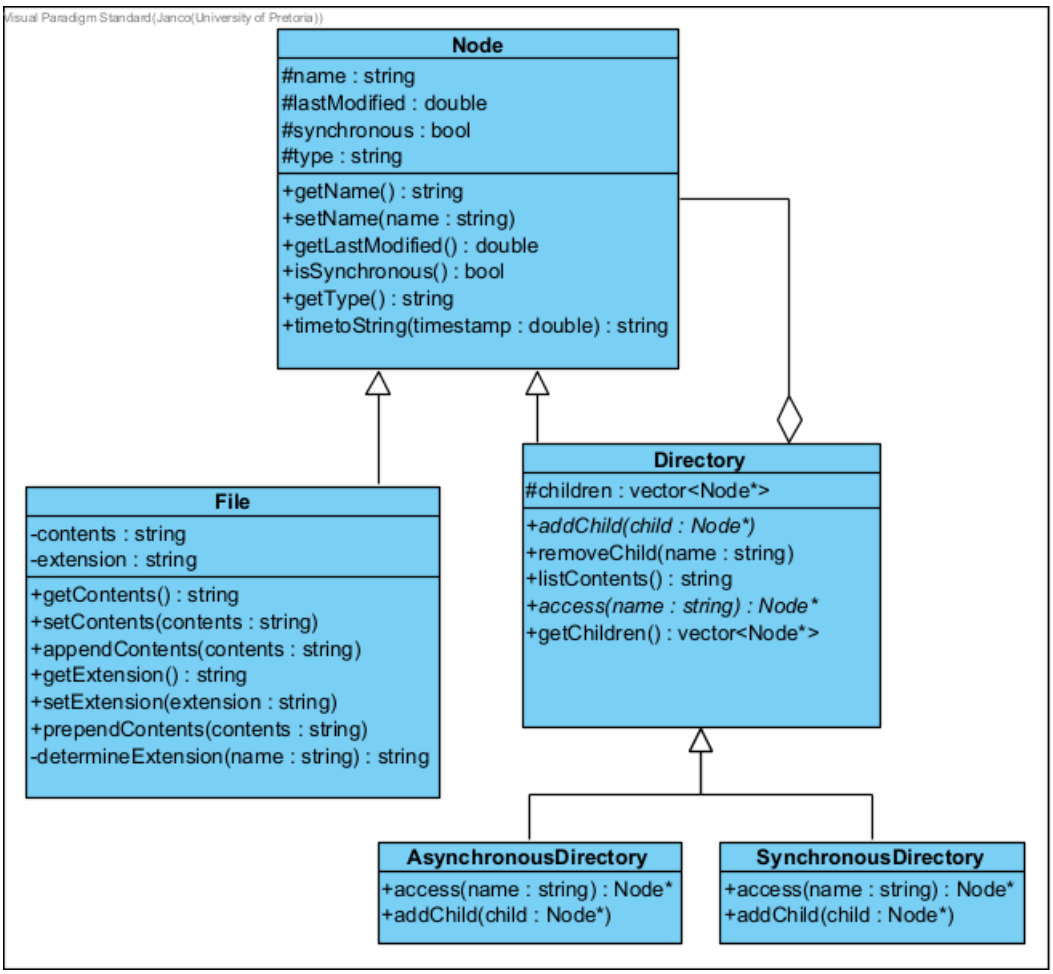
1.2 #7 I would deallocate the integer array that is created in the *capture* method at line 3.

Task 2

2.1 The first approach is to use the Composite design pattern to model the way directories are made up of files together with the Template method pattern to model the difference between asynchronous directories and synchronous directories. The second approach is to use the Template method pattern to model directories and files having shared attributes and the Strategy pattern to model the difference between asynchronous directories and synchronous directories. The first approach directly imitates the tree structure we want in our file system and therefore makes the operations to be applied to the system more intuitive. The second approach is less intuitive since the tree structure of the program will have to be mimicked by other classes which adds extra complexity to the system.

2.2 I will follow the first approach by using the Composite design pattern to model the way directories are made up of files together with the Template method pattern to model the difference between asynchronous directories and synchronous directories.

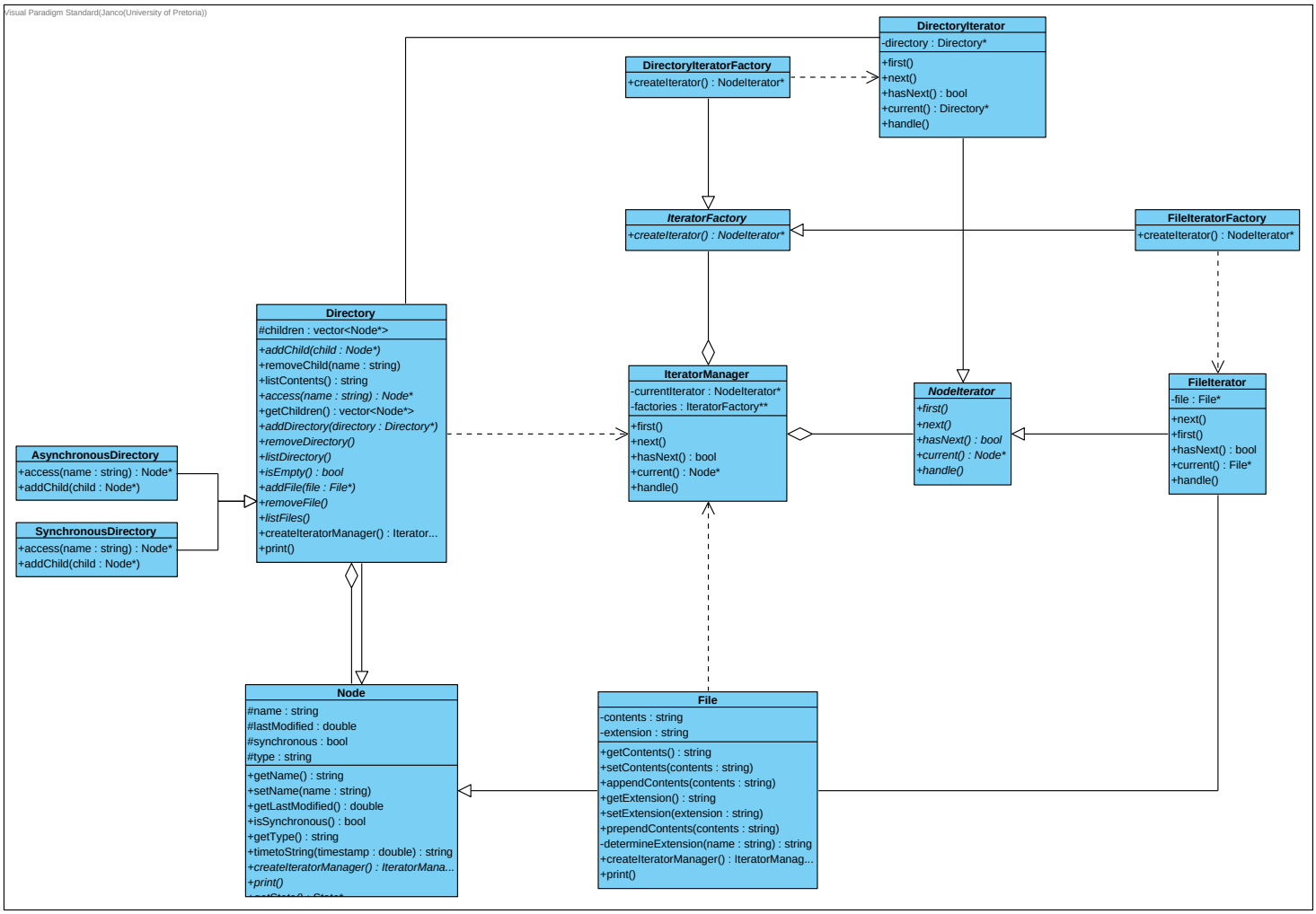
2.3



Task 3

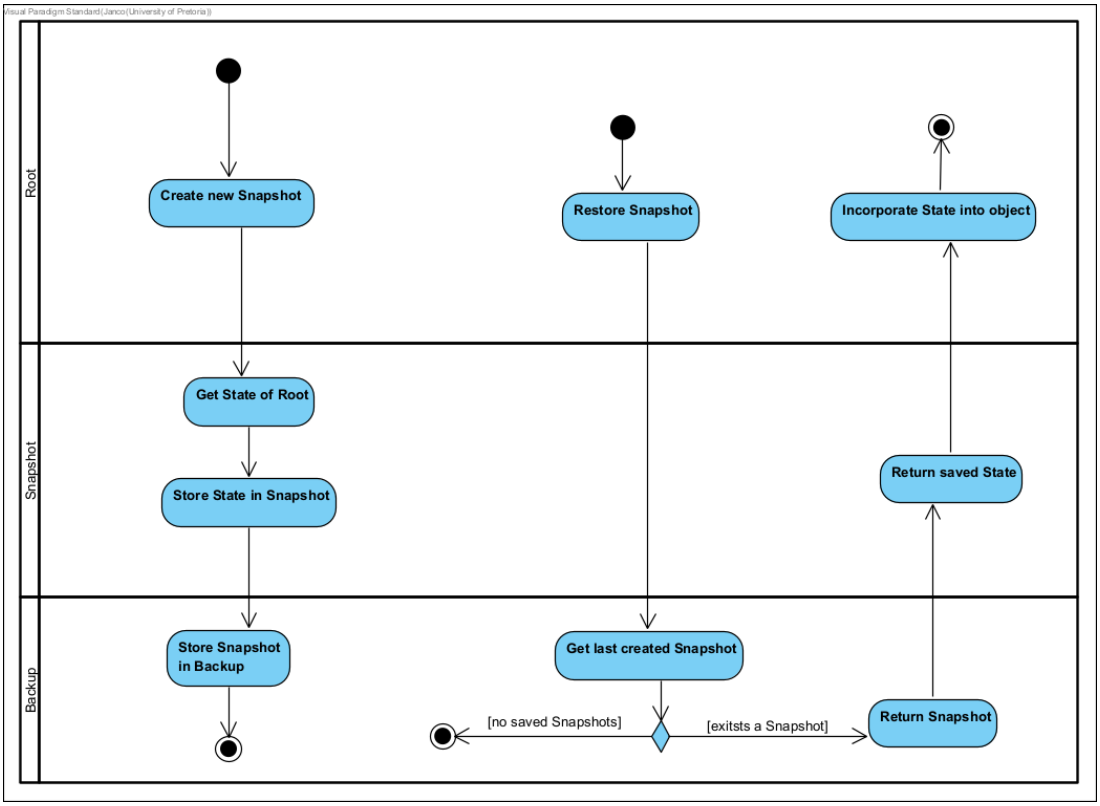
3.3 The Factory method pattern will work best to allow the user to create two different iterators easily by having a *Creator* class that creates NodeIterator objects with *concreteCreators* DirectoryIteratorFactory and FileIteratorFactory.

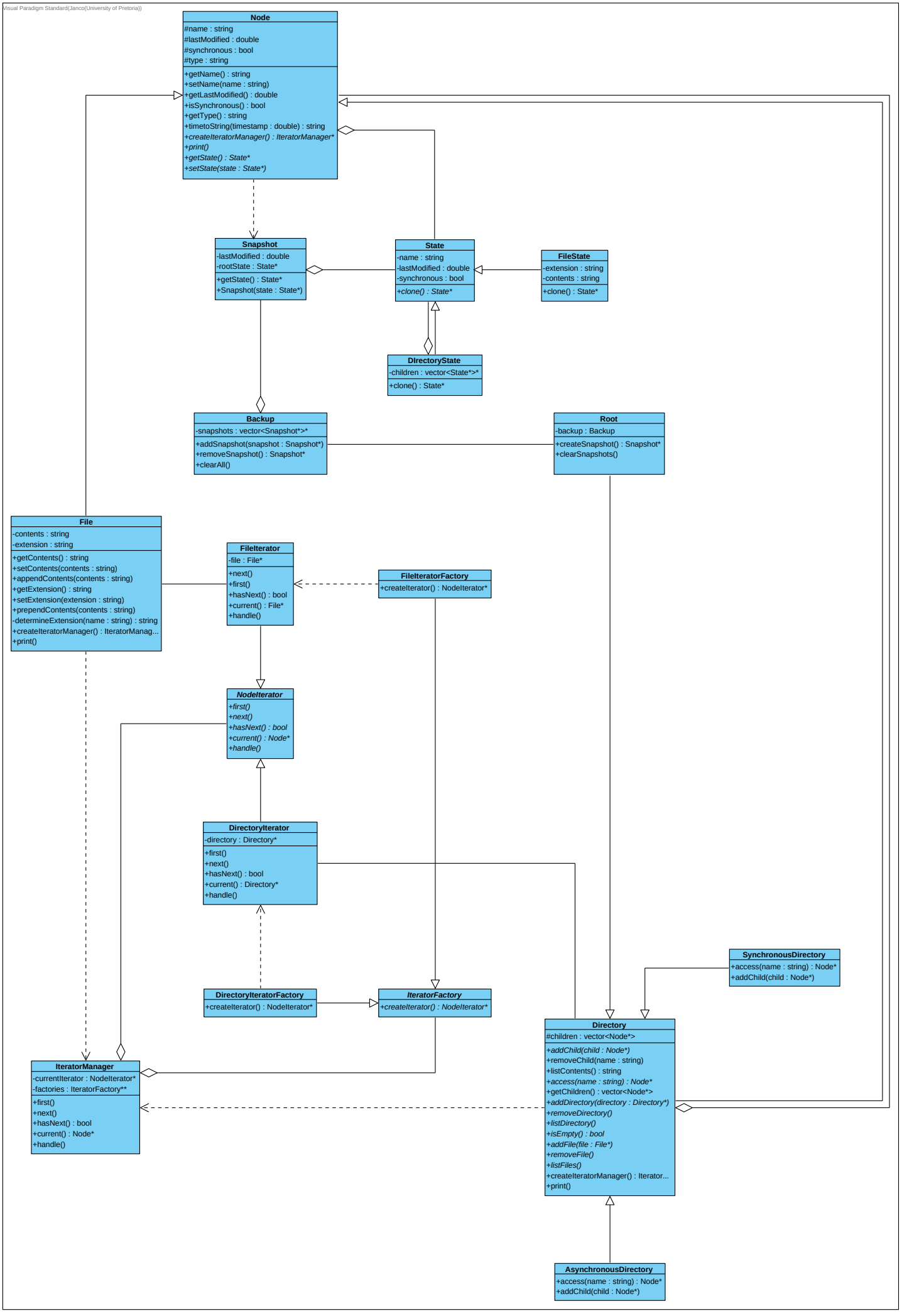
3.4 The first approach is to use the State pattern to switch between the *File* and *Directory* iterators depending on the current Node. The second approach is to set the return type on both classes to *Node** and then cast the returned object to the correct type. This approach is less safe since the user will have to cast the returned object to the correct type, but polymorphism can be exploited.


Visual Paradigm Standard(Janco(University of Pretoria))
DirectoryIterator
-directory : Directory*
+first()
+next()
+hasNext() : bool
+current() : Directory*
+handle()
DirectoryIteratorFactory
+createIterator() : NodeIterator*
IteratorFactory
+createIterator() : NodeIterator*
FileIteratorFactory
+createIterator() : NodeIterator*
Directory
#children : vector<Node*>
+addChild(child : Node*)
+removeChild(name : string)
+listContents() : string
+access(name : string) : Node*
+getChildren() : vector<Node*>
+addDirectory(directory : Directory*)
+removeDirectory()
+listDirectory()
+isEmpty() : bool
+addFile(file : File*)
+removeFile()
+listFiles()
+createIteratorManager() : Iterator...
+print()
IteratorManager
-currentIterator : NodeIterator*
-factories : IteratorFactory**
+first()
+next()
+hasNext() : bool
+current() : Node*
+handle()
NodeIterator
+first()
+next()
+hasNext() : bool
+current() : Node*
+handle()
FileIterator
-file : File*
+next()
+first()
+hasNext() : bool
+current() : File*
+handle()
AsynchronousDirectory
+access(name : string) : Node*
+addChild(child : Node*)
SynchronousDirectory
+access(name : string) : Node*
+addChild(child : Node*)
Node
#name : string
#lastModified : double
#synchronous : bool
#type : string
+getName() : string
+setName(name : string)
+getLastModified() : double
+isSynchronous() : bool
+getType() : string
+timetoString(timestamp : double) : string
+createIteratorManager() : IteratorMana...
+print()
File
-contents : string
-extension : string
+getContents() : string
+setContents(contents : string)
+appendContents(contents : string)
+getExtension() : string
+setExtension(extension : string)
+prependContents(contents : string)
-determineExtension(name : string) : string
+createIteratorManager() : IteratorManag...
+print()

Task 4

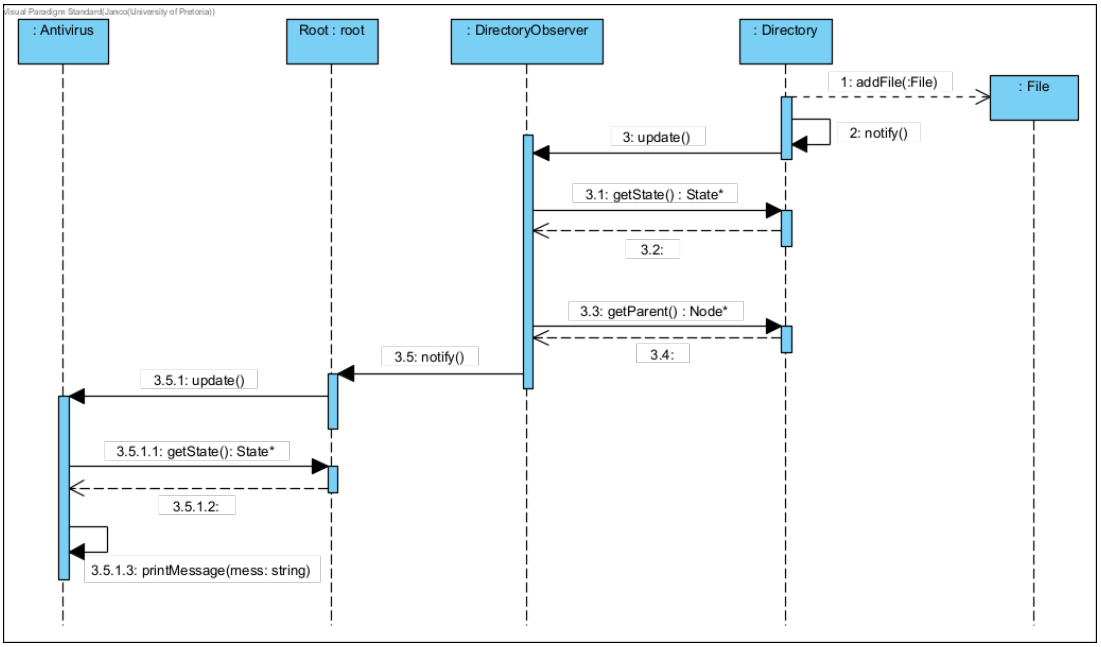
4.2


Root
Snapshot
Backup
Create new Snapshot
Restore Snapshot
Incorporate State into object
Get State of Root
Store State in Snapshot
Return saved State
Store Snapshot in Backup
Get last created Snapshot
[no saved Snapshots]
[exitsts a Snapshot]
Return Snapshot



Task 5

5.3



Task 6

This is the main menu that the driver program opens on. From here different operations can be performed on the file system.

```
WELCOME TO THE DRIVER PROGRAM
You are currently in: root
Please choose an operation to perform:
[0] Add new Directory
[1] Add new File
[2] Delete Directory
[3] Delete File
[4] Move to sub-directory
[5] Move to root directory
[6] Create new snapshot of the current system
[7] Delete all snapshots
[8] Restore a previous snapshot
[9] View contents of current directory
[10] Quit
Option:
```

To remove a file, select option 3 from the main menu and follow the prompts to select the file to be removed.

```
WELCOME TO THE DRIVER PROGRAM
You are currently in: root
Please choose an operation to perform:
[0] Add new Directory
[1] Add new File
[2] Delete Directory
[3] Delete File
[4] Move to sub-directory
[5] Move to root directory
[6] Create new snapshot of the current system
[7] Delete all snapshots
[8] Restore a previous snapshot
[9] View contents of current directory
[10] Quit
Option: 3
root files:
file1
file2
Enter the name of the file you would like to delete.
Name: file1
File/directory file1 has been deleted.
Press Enter to continue. . .
```

To create a new snapshot of the file system, select option 6 from the main menu.

```
WELCOME TO THE DRIVER PROGRAM
You are currently in: root
Please choose an operation to perform:
[0] Add new Directory
[1] Add new File
[2] Delete Directory
[3] Delete File
[4] Move to sub-directory
[5] Move to root directory
[6] Create new snapshot of the current system
[7] Delete all snapshots
[8] Restore a previous snapshot
[9] View contents of current directory
[10] Quit
Option: 6
Snapshot created successfully
Press Enter to continue. . .
```

To restore the file system to a previous snapshot, select option 7 from the main menu.

```
WELCOME TO THE DRIVER PROGRAM
You are currently in: root
Please choose an operation to perform:
[0] Add new Directory
[1] Add new File
[2] Delete Directory
[3] Delete File
[4] Move to sub-directory
[5] Move to root directory
[6] Create new snapshot of the current system
[7] Delete all snapshots
[8] Restore a previous snapshot
[9] View contents of current directory
[10] Quit
Option: 8
File/directory sub1 has been deleted.
File/directory sub2 has been deleted.
File/directory file2 has been deleted.
New file/directory sub1 has been scanned.
New file/directory sub2 has been scanned.
New file/directory file2 has been scanned.
Snapshot restored successfully
Press Enter to continue. . .
```

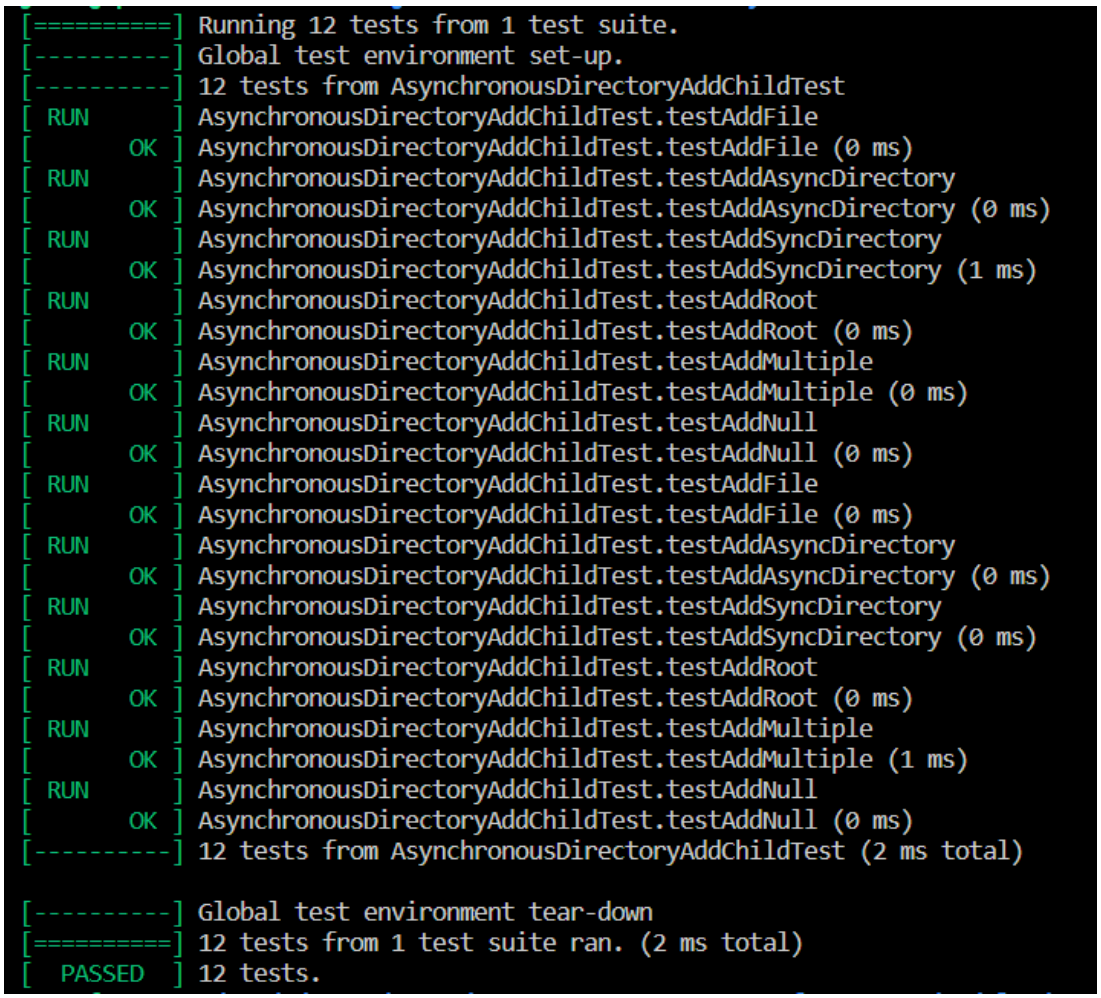
Task 7

I have decided to test the *addChild* function in the AsynchronousDirectory class. The testing code can be seen below followed by a screenshot of the produced output.

```
#include <limits.h>
#include <stdexcept>
#include "AsynchronousDirectory.h"
#include "SynchronousDirectory.h"
#include "File.h"
#include "Root.h"
#include "gtest/gtest.h"
namespace
{
    TEST(AsynchronousDirectoryAddChildTest, testAddFile)
    {
        AsynchronousDirectory *d = new AsynchronousDirectory("d");
        File *f = new File("f.txt");

        EXPECT_EQ(0, d->getChildrenCount());
```

```
    d->addChild(f);
    EXPECT_EQ(1, d->getChildrenCount());
    EXPECT_EQ("f.txt", d->getChild(0)->getName());
    EXPECT_EQ("File", d->getChild(0)->getType());
    EXPECT_EQ(false, d->getChild(0)->isSynchronous());
    delete d;
}

TEST(AsynchronousDirectoryAddChildTest, testAddAsyncDirectory)
{
    AsynchronousDirectory *d = new AsynchronousDirectory("d");
    AsynchronousDirectory *d2 = new AsynchronousDirectory("d2");

    EXPECT_EQ(0, d->getChildrenCount());
    d->addChild(d2);
    EXPECT_EQ(1, d->getChildrenCount());
    EXPECT_EQ("d2", d->getChild(0)->getName());
    EXPECT_EQ("Directory", d->getChild(0)->getType());
    EXPECT_EQ(false, d->getChild(0)->isSynchronous());
    delete d;
}

TEST(AsynchronousDirectoryAddChildTest, testAddSyncDirectory)
{
    AsynchronousDirectory *d = new AsynchronousDirectory("d");
    SynchronousDirectory *d2 = new SynchronousDirectory("d2");

    EXPECT_EQ(0, d->getChildrenCount());
    try
    {
        d->addChild(d2);
        FAIL();
    }
    catch (std::invalid_argument &e)
    {
        EXPECT_EQ(e.what(), std::string("Cannot add a synchronous node to an asynchronous
            directory"));
    }
    catch (...)
    {
        FAIL();
    }
    EXPECT_EQ(0, d->getChildrenCount());
    delete d2;
    delete d;
}

TEST(AsynchronousDirectoryAddChildTest, testAddRoot)
{
    AsynchronousDirectory *d = new AsynchronousDirectory("d");
    Root *r = new Root();
    EXPECT_EQ(0, d->getChildrenCount());
    try
    {
        d->addChild(r);
        FAIL();
    }
    catch (std::invalid_argument &e)
    {
        EXPECT_EQ(e.what(), std::string("Cannot add a synchronous node to an asynchronous
            directory"));
    }
    catch (...)
    {
        FAIL();
    }
```

```
        EXPECT_EQ(0, d->getChildrenCount());
        delete d;
        delete r;
    }

    TEST(AsynchronousDirectoryAddChildTest, testAddMultiple)
    {
        AsynchronousDirectory *d = new AsynchronousDirectory("d");
        EXPECT_EQ(0, d->getChildrenCount());
        d->addChild(new File("f1.txt"));
        EXPECT_EQ(1, d->getChildrenCount());
        d->addChild(new File("f2.txt"));
        EXPECT_EQ(2, d->getChildrenCount());
        d->addChild(new AsynchronousDirectory("d1"));
        EXPECT_EQ(3, d->getChildrenCount());
        d->addChild(new AsynchronousDirectory("d2"));
        EXPECT_EQ(4, d->getChildrenCount());
        try {
            d->addChild(new SynchronousDirectory("d3"));
            FAIL();
        }
        catch (std::invalid_argument &e)
        {
            EXPECT_EQ(e.what(), std::string("Cannot add a synchronous node to an asynchronous
                directory"));
        }
        catch (...)
        {
            FAIL();
        }
        EXPECT_EQ(4, d->getChildrenCount());
        delete d;
    }
    TEST(AsynchronousDirectoryAddChildTest, testAddNull)
{
    AsynchronousDirectory *d = new AsynchronousDirectory("d");
    EXPECT_EQ(0, d->getChildrenCount());
    try
    {
        d->addChild(NULL);
        FAIL();
    }
    catch (std::invalid_argument &e)
    {
        EXPECT_EQ(e.what(), std::string("Cannot add a null node"));
    }
    catch (...)
    {
        FAIL();
    }
    EXPECT_EQ(0, d->getChildrenCount());
    delete d;
}
}
```


[==========] Running 12 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 12 tests from AsynchronousDirectoryAddChildTest
[RUN   ] AsynchronousDirectoryAddChildTest.testAddFile
[OK] AsynchronousDirectoryAddChildTest.testAddFile (0 ms)
[RUN   ] AsynchronousDirectoryAddChildTest.testAddAsyncDirectory
[OK] AsynchronousDirectoryAddChildTest.testAddAsyncDirectory (0 ms)
[RUN   ] AsynchronousDirectoryAddChildTest.testAddSyncDirectory
[OK] AsynchronousDirectoryAddChildTest.testAddSyncDirectory (1 ms)
[RUN   ] AsynchronousDirectoryAddChildTest.testAddRoot
[OK] AsynchronousDirectoryAddChildTest.testAddRoot (0 ms)
[RUN   ] AsynchronousDirectoryAddChildTest.testAddMultiple
[OK] AsynchronousDirectoryAddChildTest.testAddMultiple (0 ms)
[RUN   ] AsynchronousDirectoryAddChildTest.testAddNull
[OK] AsynchronousDirectoryAddChildTest.testAddNull (0 ms)
[RUN   ] AsynchronousDirectoryAddChildTest.testAddFile
[OK] AsynchronousDirectoryAddChildTest.testAddFile (0 ms)
[RUN   ] AsynchronousDirectoryAddChildTest.testAddAsyncDirectory
[OK] AsynchronousDirectoryAddChildTest.testAddAsyncDirectory (0 ms)
[RUN   ] AsynchronousDirectoryAddChildTest.testAddSyncDirectory
[OK] AsynchronousDirectoryAddChildTest.testAddSyncDirectory (0 ms)
[RUN   ] AsynchronousDirectoryAddChildTest.testAddRoot
[OK] AsynchronousDirectoryAddChildTest.testAddRoot (0 ms)
[RUN   ] AsynchronousDirectoryAddChildTest.testAddMultiple
[OK] AsynchronousDirectoryAddChildTest.testAddMultiple (1 ms)
[RUN   ] AsynchronousDirectoryAddChildTest.testAddNull
[OK] AsynchronousDirectoryAddChildTest.testAddNull (0 ms)
[----------] 12 tests from AsynchronousDirectoryAddChildTest (2 ms total)
[----------] Global test environment tear-down
[==========] 12 tests from 1 test suite ran. (2 ms total)
[PASSED] 12 tests.