Group 8

COMP2021 Project

COMP VIRTUAL FILE SYSTEM

Huang Lingru, Shen Linghui, Tan Rou Xin

INTRODUCTION

Our **COMP Virtual File System** aims to simulate the file system in memory.

Instruct the users to enter **commands** through CLI
Call the corresponding methods sequentially to perform file management.

-create, delete, rename, and list **files**
-create, print, and utilize various types of **criteria**
-save, load, change, and exit the working **system**
-undo and redo the commands

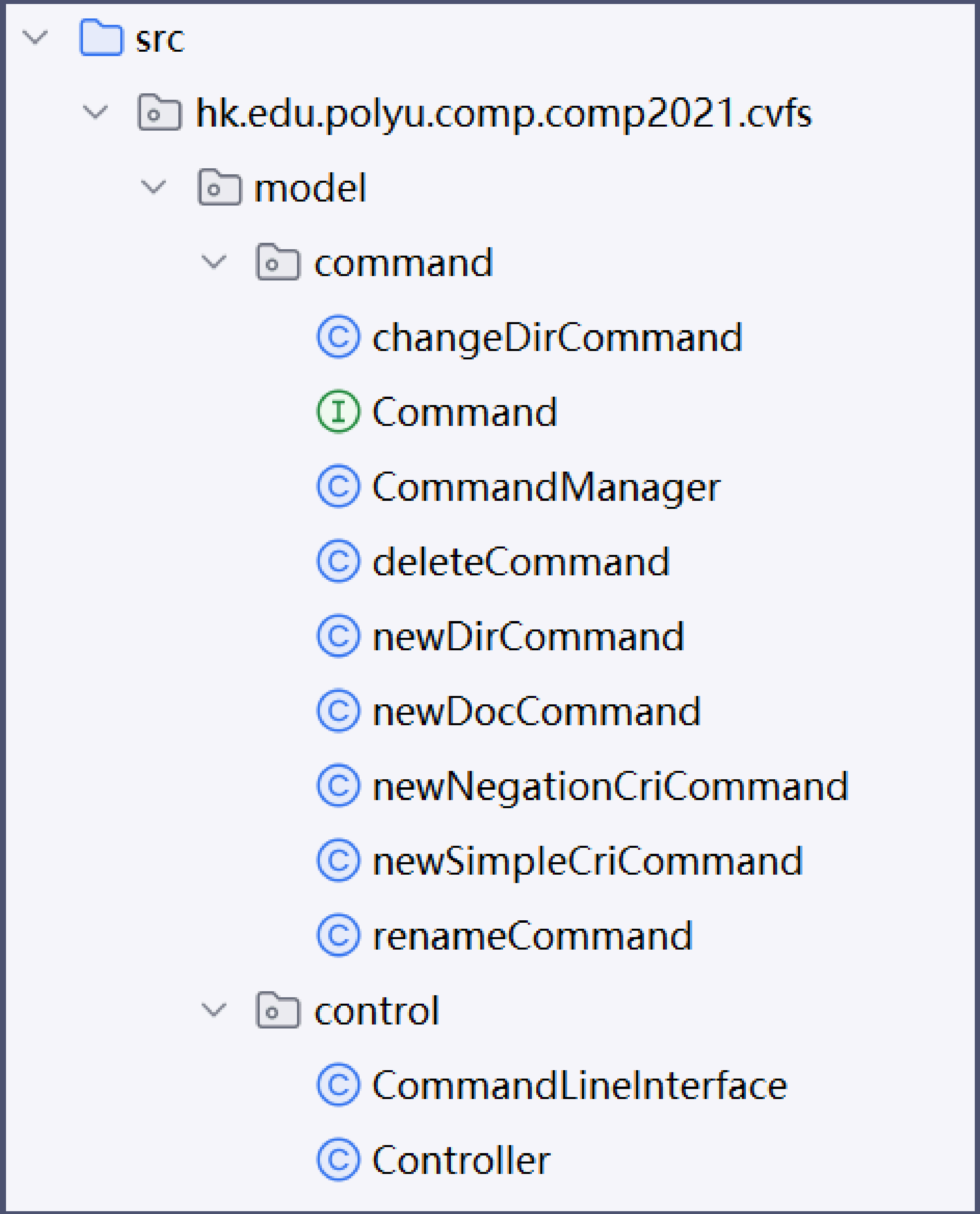
OVERVIEW

Overall Architecture

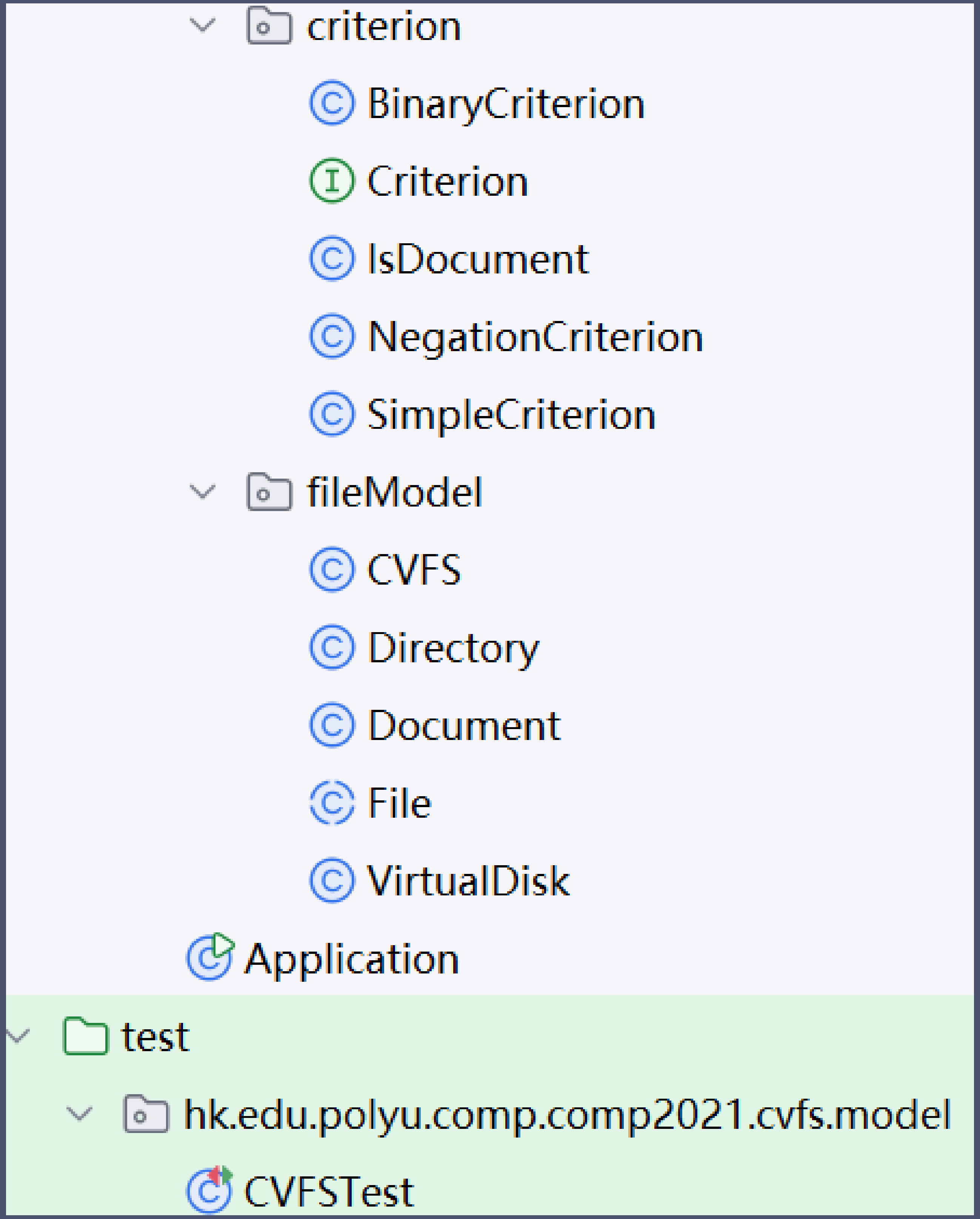
Specific Design Choice(inheritance and polymorphism)

Object-oriented Programming

Overall Architecture

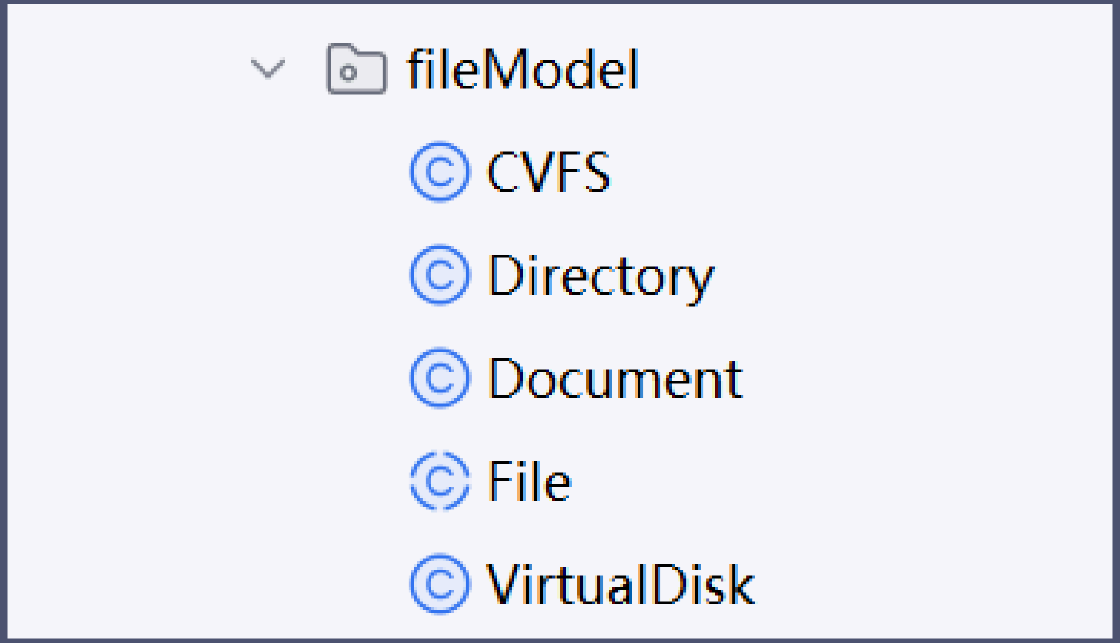


```
src
  hk.edu.polyu.comp.comp2021.cvfs
    model
      command
        changeDirCommand
        Command
        CommandManager
        deleteCommand
        newDirCommand
        newDocCommand
        newNegationCriCommand
        newSimpleCriCommand
        renameCommand
    control
      CommandLineInterface
      Controller
```



```
      criterion
        BinaryCriterion
        Criterion
        IsDocument
        NegationCriterion
        SimpleCriterion
      fileModel
        CVFS
        Directory
        Document
        File
        VirtualDisk
    Application
test
  hk.edu.polyu.comp.comp2021.cvfs.model
    CVFSTest
```

Overall Architecture



fileModel
CVFS
Directory
Document
File
VirtualDisk

package fileModel

-class CVFS: most of the requirements are completed here.

instantiate current working directory and working disk

-abstract class File: provides common setup

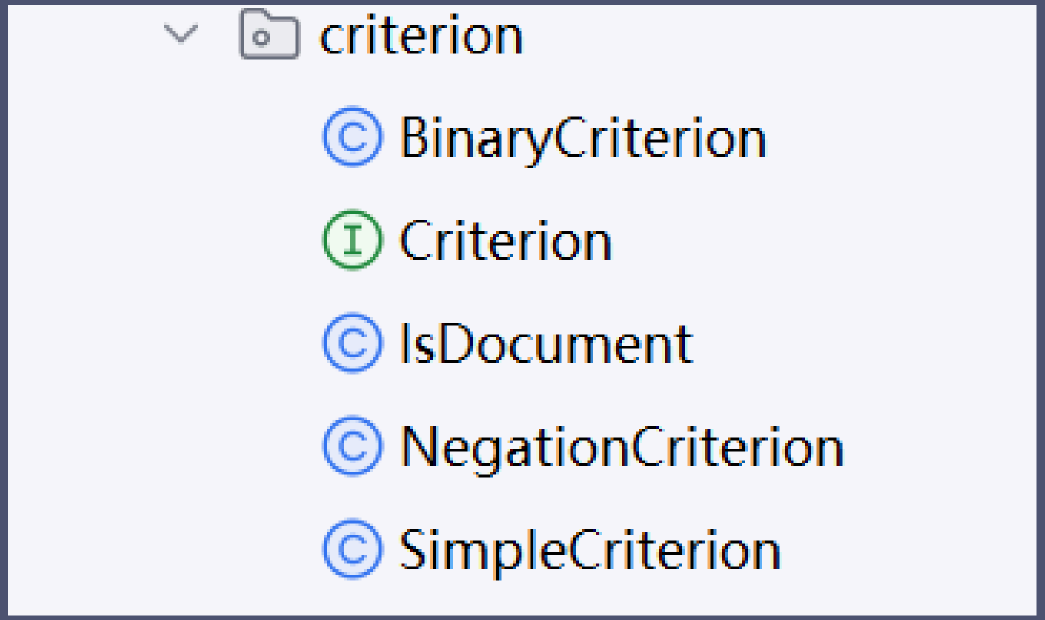
-class Document, class Directory: extends File

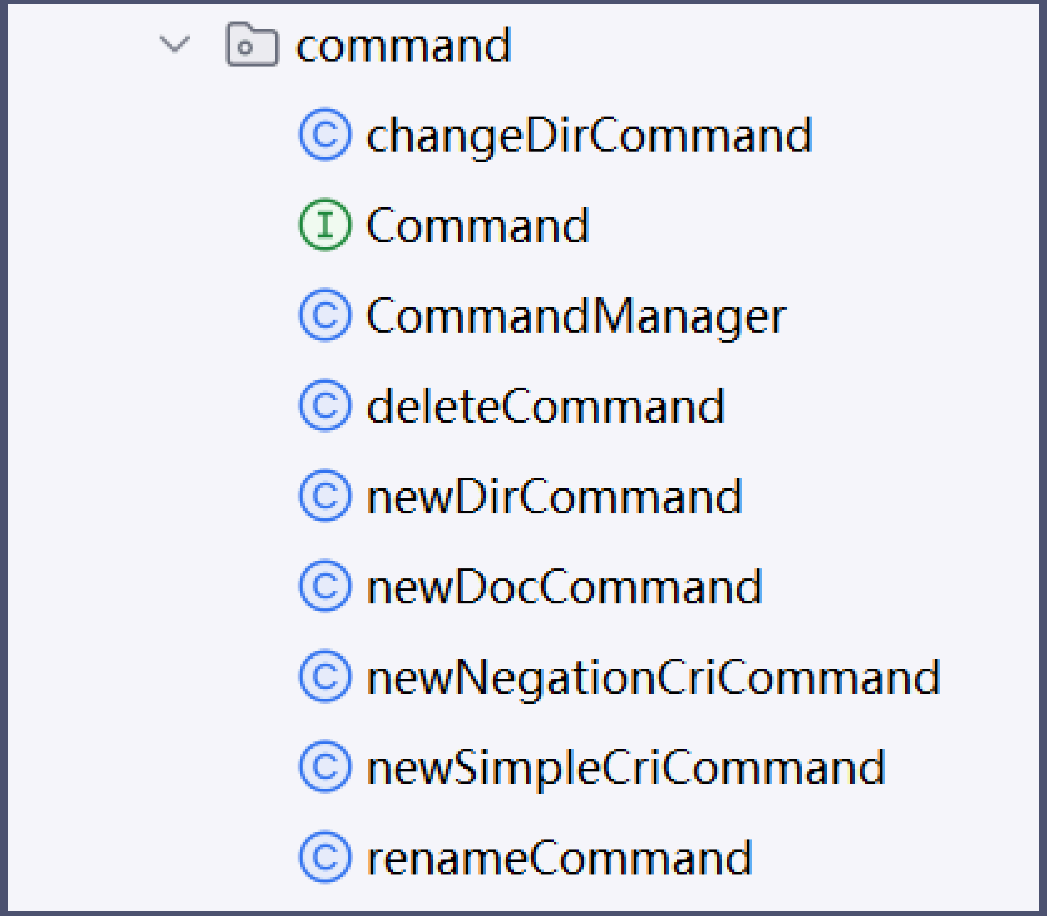
inherit common variables and methods,

override getSize(), define unique methods.

-class VirtualDisk: connect to a root directory (“/”)

Overall Architecture


criterion
BinaryCriterion
Criterion
IsDocument
NegationCriterion
SimpleCriterion


command
changeDirCommand
Command
CommandManager
deleteCommand
newDirCommand
newDocCommand
newNegationCriCommand
newSimpleCriCommand
renameCommand

package criterion

-interface Criterion: provides abstract method checkFile and toString.

-class SimpleCriterion, IsDocument, NegationCriterion and BinaryCriterion: implements Criterion and override two abstract methods.

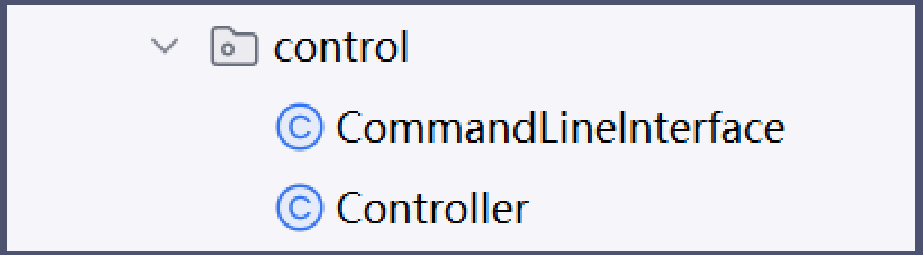
package command

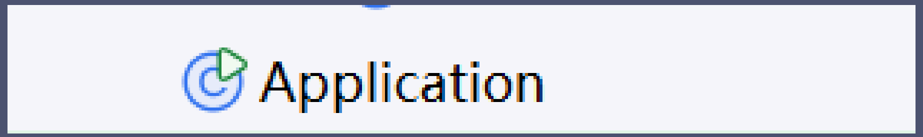
-class CommandManager: create two **stacks** to manipulate user's commands.

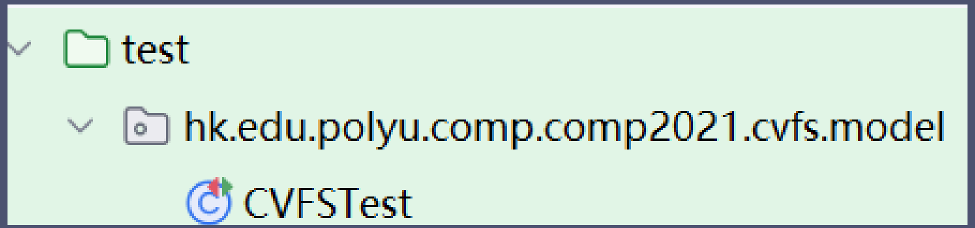
-interface Command: provides abstract method execute, undo, redo.

-6 <operation>Command classes: implements Command and override 3 abstract methods.

Overall Architecture


control
CommandLineInterface
Controller


Application


test
hk.edu.polyu.comp.comp2021.cvfs.model
CVFSTest

package control:
-class CommandLineInterface: **pass** in a cvfs instance
-class Controller: used for save and load

class Application:

instantiate CVFS
instantiate CommandLineInterface and **activate** it.

package test:

includes tests for all classes and methods
(excluding codes in CIL)
satisfactory coverage (>90%)

Specific Design Choice

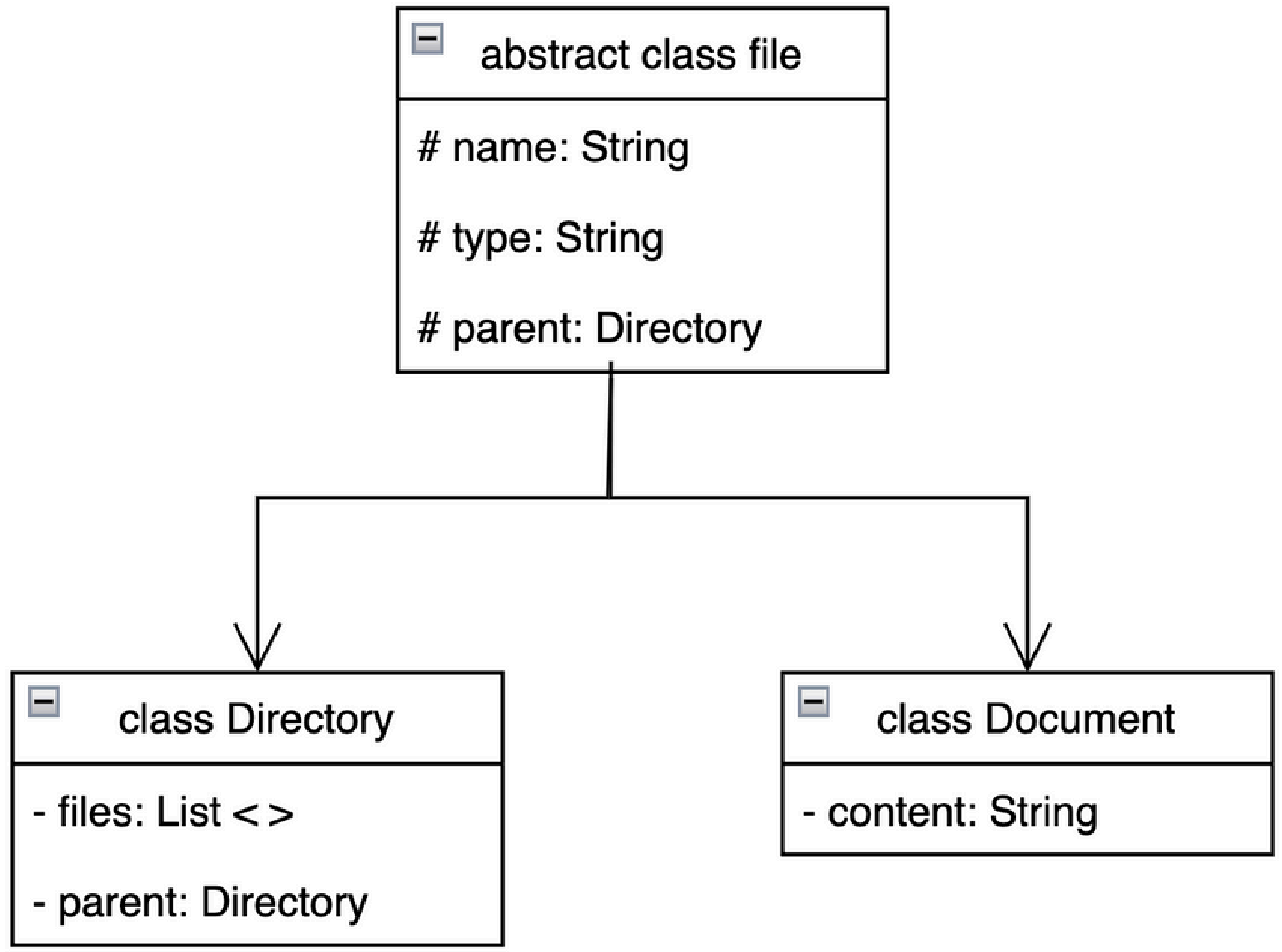
Inheritance (Hierarchical Inheritance)

Parent Class: abstract class File

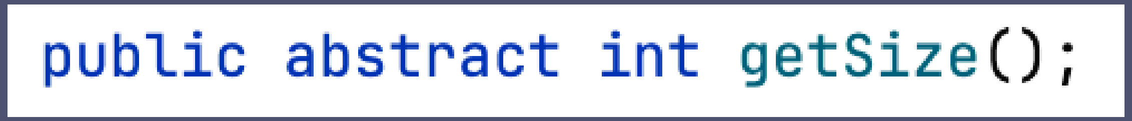
- Abstract Class: contains abstract method as a common interface

Child Class: class Document, class Directory

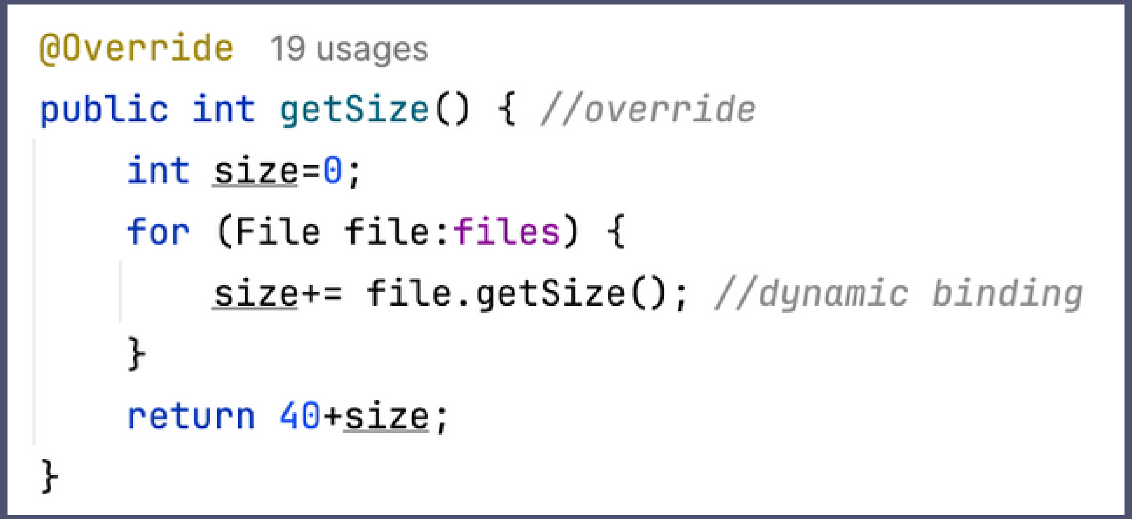
- inherit methods and field



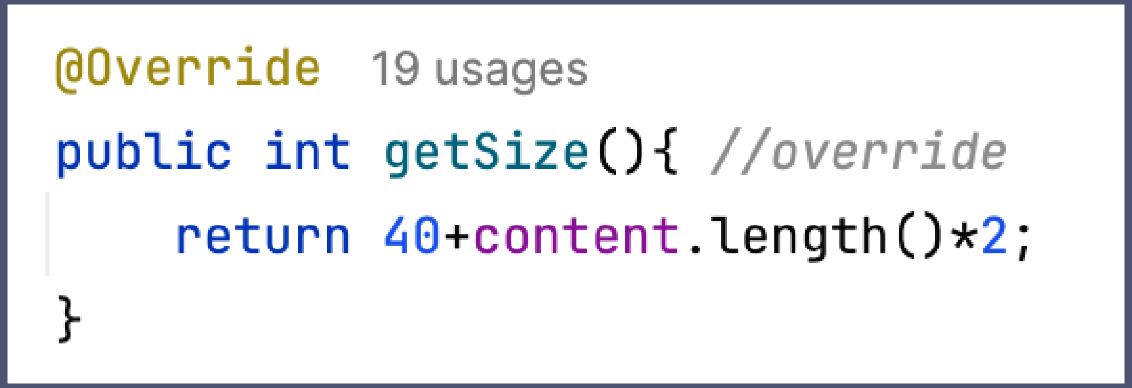
Specific Design Choice



```
public abstract int getSize();
```



```
@Override  19 usages
public int getSize() { //override
    int size=0;
    for (File file:files) {
        size+= file.getSize(); //dynamic binding
    }
    return 40+size;
}
```



```
@Override  19 usages
public int getSize(){ //override
    return 40+content.length()*2;
}
```

Run-time Polymorphism (Method Overriding)

- implement getSize( ) method provided by parent class

Parent Class: abstract class File

- return size of files

Child Class: class Directory

- return size of directory

Child Class: class Document

- return size of document

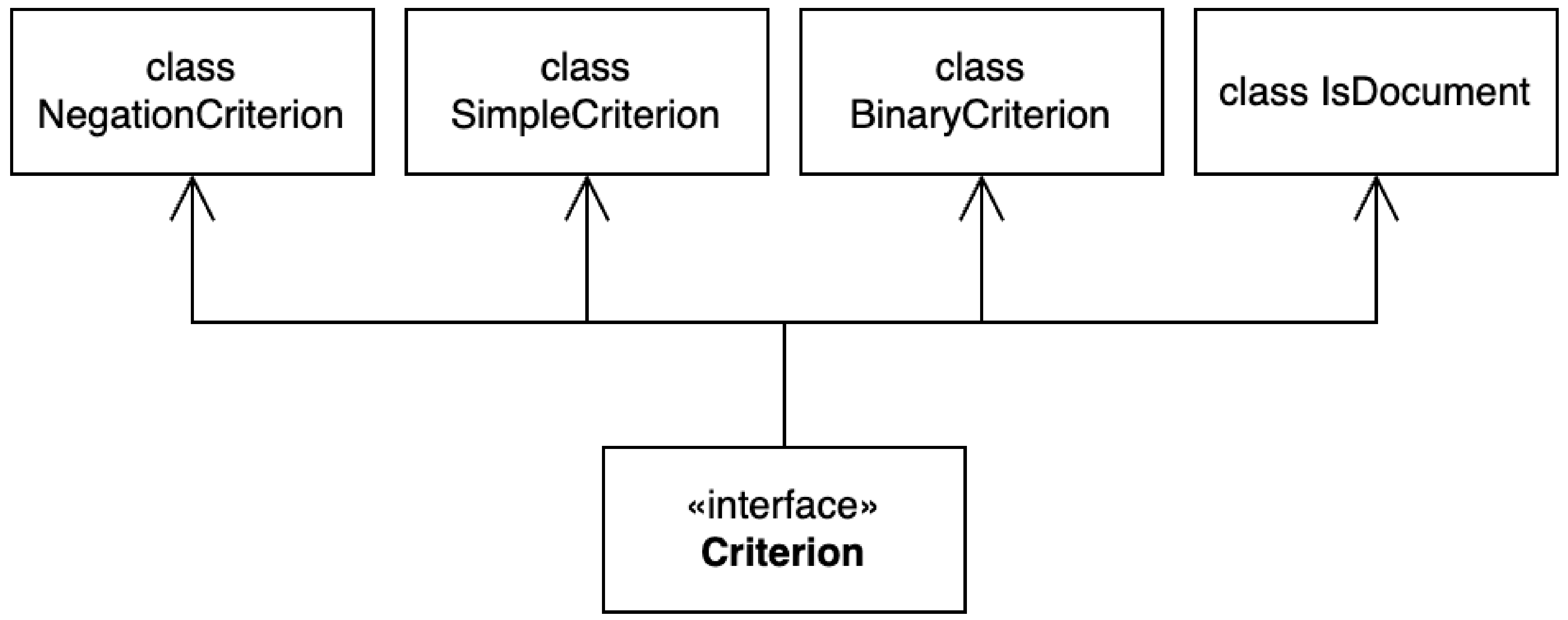
Specific Design Choice

Inheritance (Multiple Inheritance - Through Interface)

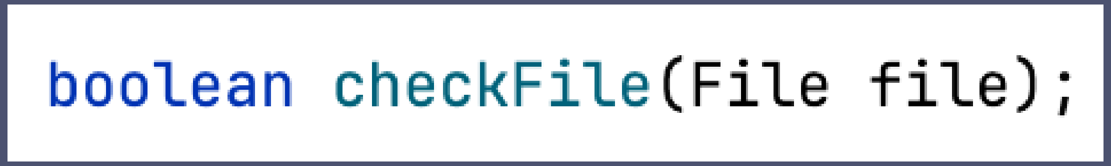
Parent Class: interface Criterion

Child Class: class NegationCriterion, class SimpleCriterion, class BinaryCriterion, Class IsDocument

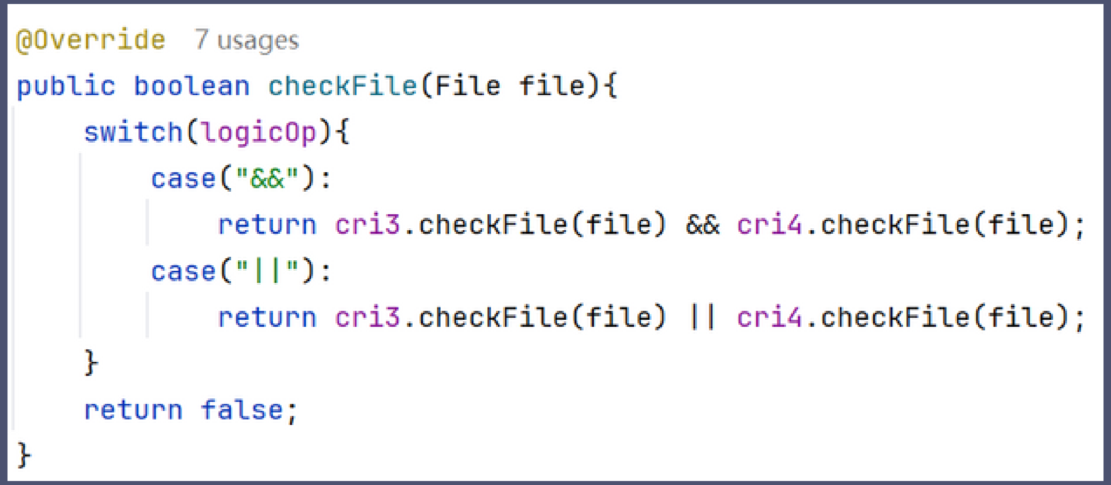
- support certain behaviour
- not need to inherit common behaviour



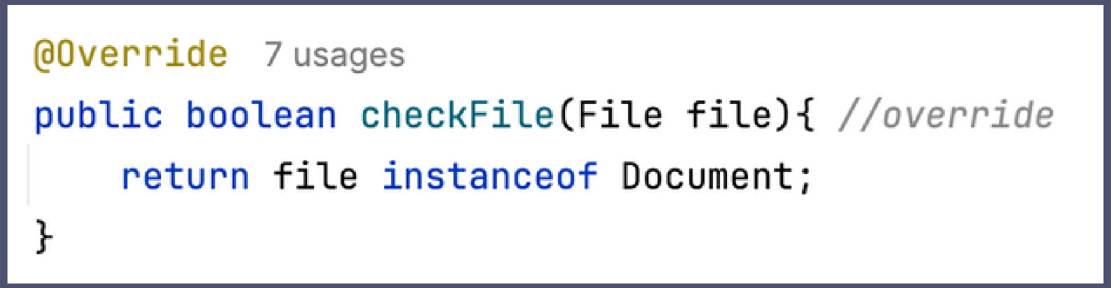
Specific Design Choice



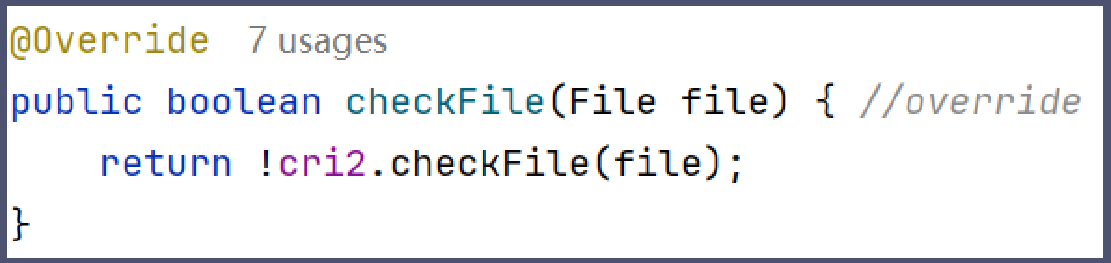
```
boolean checkFile(File file);
```



```
@Override  7 usages
public boolean checkFile(File file){
    switch(logicOp){
        case("&&"):
            return cri3.checkFile(file) && cri4.checkFile(file);
        case("||"):
            return cri3.checkFile(file) || cri4.checkFile(file);
    }
    return false;
}
```



```
@Override  7 usages
public boolean checkFile(File file){ //override
    return file instanceof Document;
}
```



```
@Override  7 usages
public boolean checkFile(File file) { //override
    return !cri2.checkFile(file);
}
```

Run-time Polymorphism (Method Overriding)

- implement checkFile( ) method provided by parent class

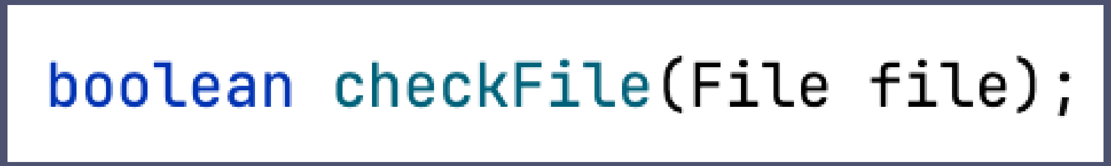
Parent Class: interface Criterion

Child Class: class BinaryCriterion

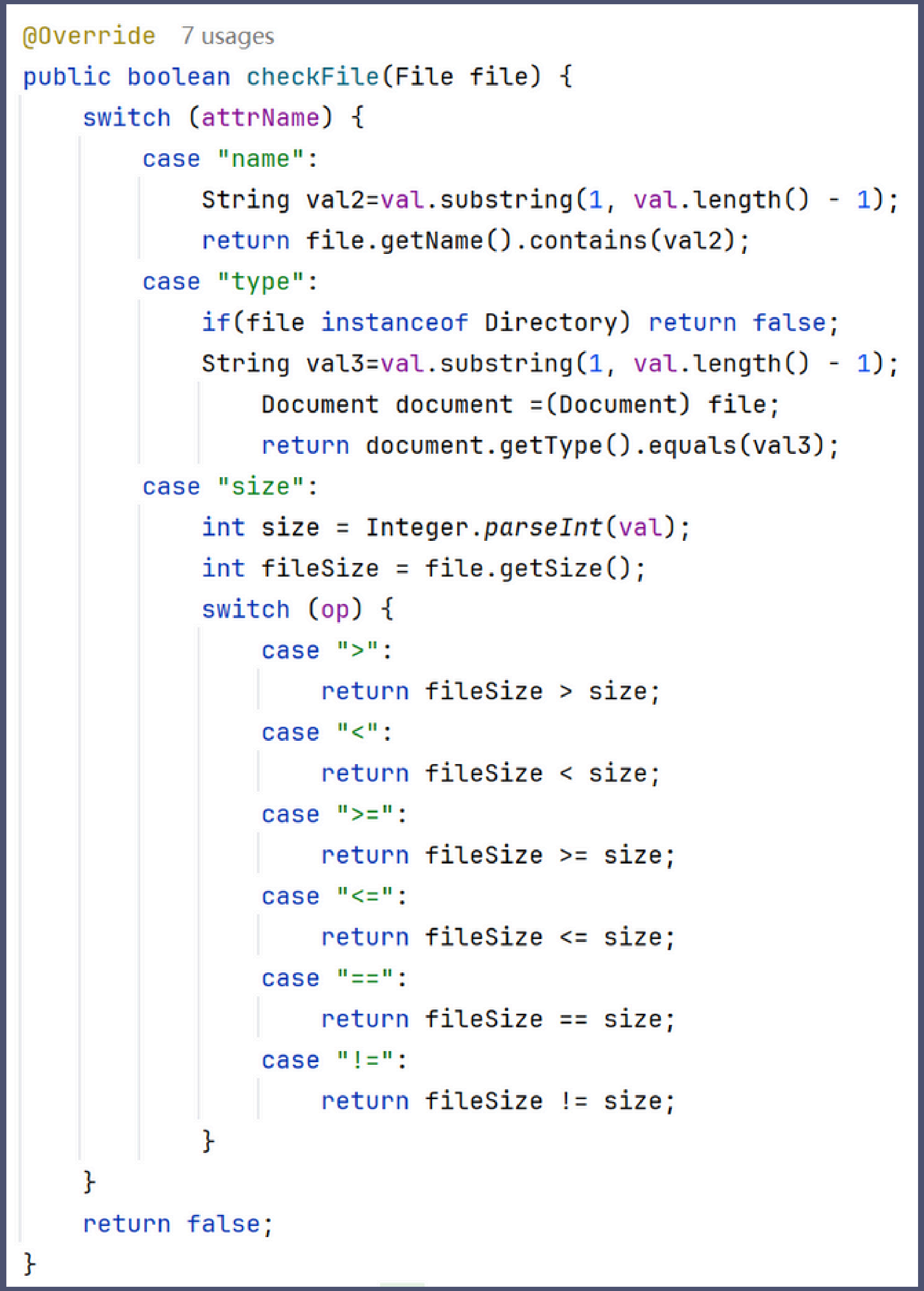
Child Class: class IsDocument

Child Class: class NegationCriterion

Specific Design Choice



```
boolean checkFile(File file);
```



```
@Override  7 usages
public boolean checkFile(File file) {
    switch (attrName) {
        case "name":
            String val2=val.substring(1, val.length() - 1);
            return file.getName().contains(val2);
        case "type":
            if(file instanceof Directory) return false;
            String val3=val.substring(1, val.length() - 1);
                Document document =(Document) file;
                return document.getType().equals(val3);
        case "size":
            int size = Integer.parseInt(val);
            int fileSize = file.getSize();
            switch (op) {
                case ">":
                    return fileSize > size;
                case "<":
                    return fileSize < size;
                case ">=":
                    return fileSize >= size;
                case "<=":
                    return fileSize <= size;
                case "==":
                    return fileSize == size;
                case "!=":
                    return fileSize != size;
            }
    }
    return false;
}
```

Run-time Polymorphism (Method Overriding)

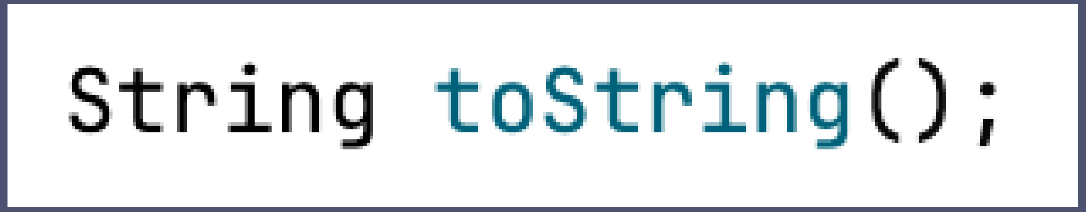
- implement checkFile( ) method provided by parent class

Parent Class: interface Criterion

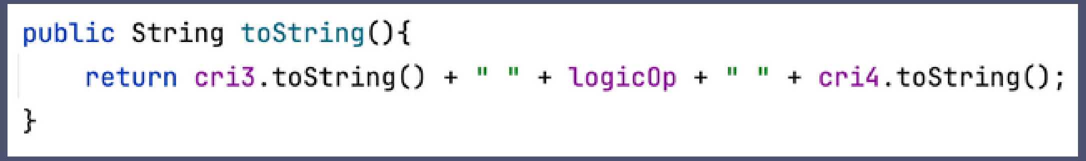
Child Class: SimpleCriterion

- valid attribute name

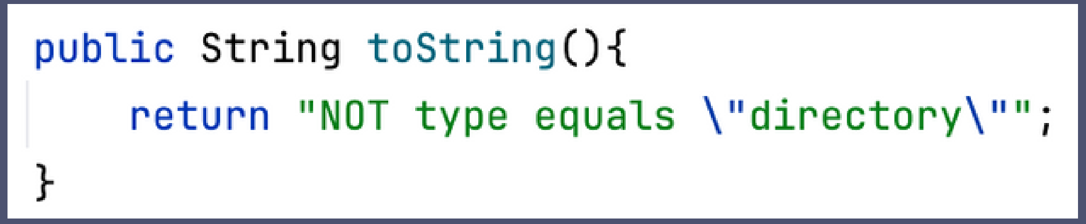
Specific Design Choice



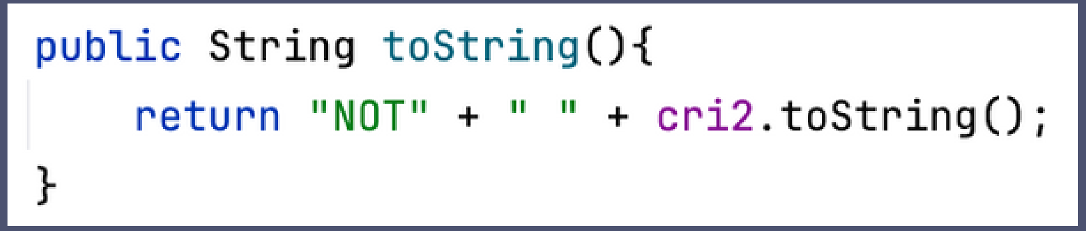
```
String toString();
```



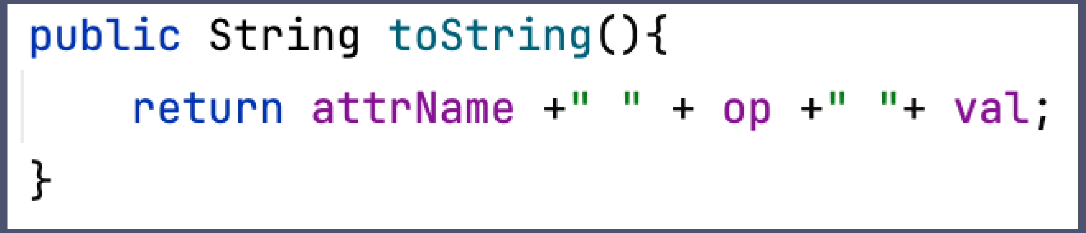
```
public String toString(){
    return cri3.toString() + " " + logicOp + " " + cri4.toString();
}
```



```
public String toString(){
    return "NOT type equals \"directory\"";
}
```



```
public String toString(){
    return "NOT" + " " + cri2.toString();
}
```



```
public String toString(){
    return attrName +" " + op +" "+ val;
}
```

Run-time Polymorphism (Method Overriding)

- implement toString( ) method provided by parent class

Parent Class: interface Criterion

Child Class: class BinaryCriterion

- return cri3, logicOP and cri4 as String

Child Class: class IsDocument

- print message

Child Class: NegationCriterion

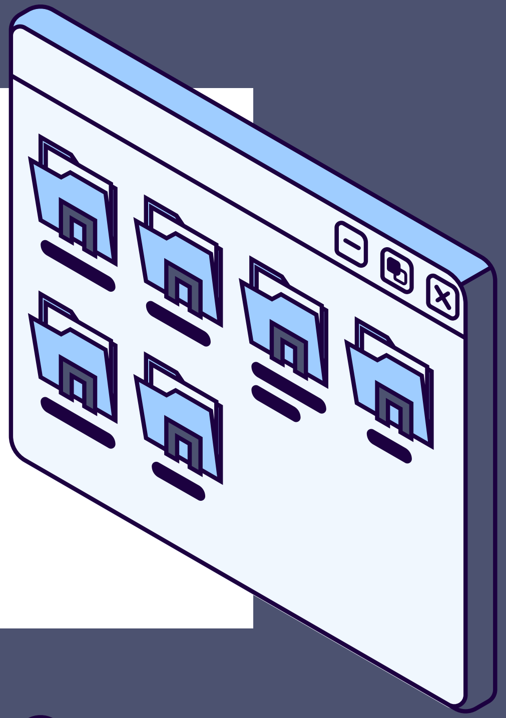
- print message and return cri2 as String

Child Class: SimpleCriterion

- return attribute name as String

OBJECT-ORIENTED PROGRAMMING

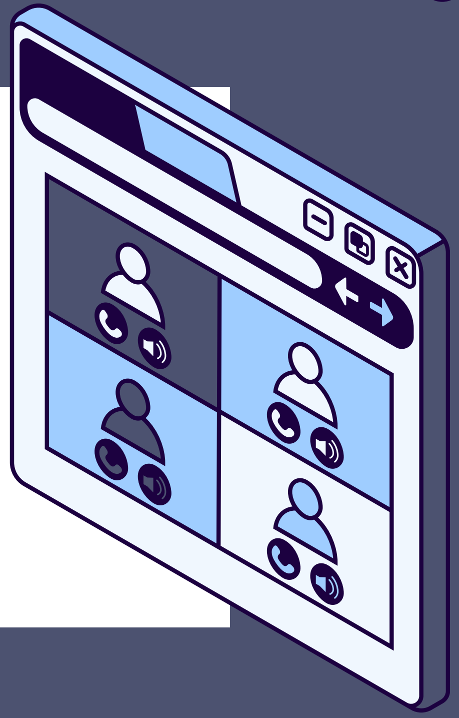
Encapsulation



Inheritance

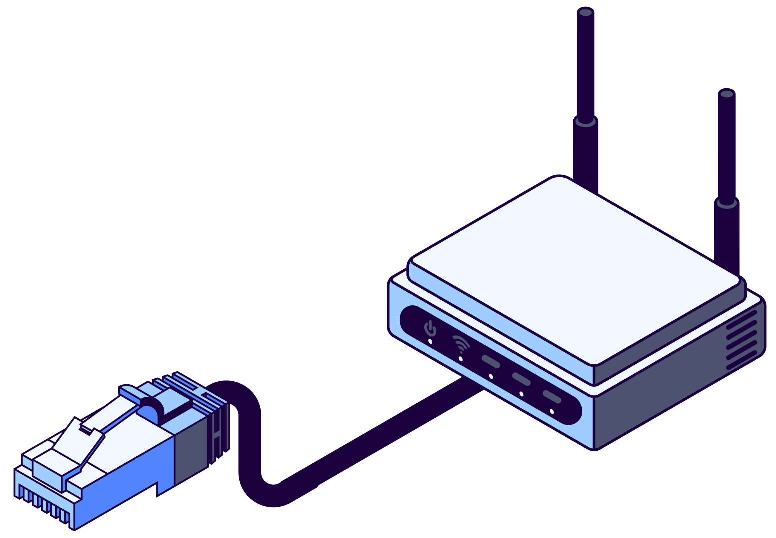
Polymorphism

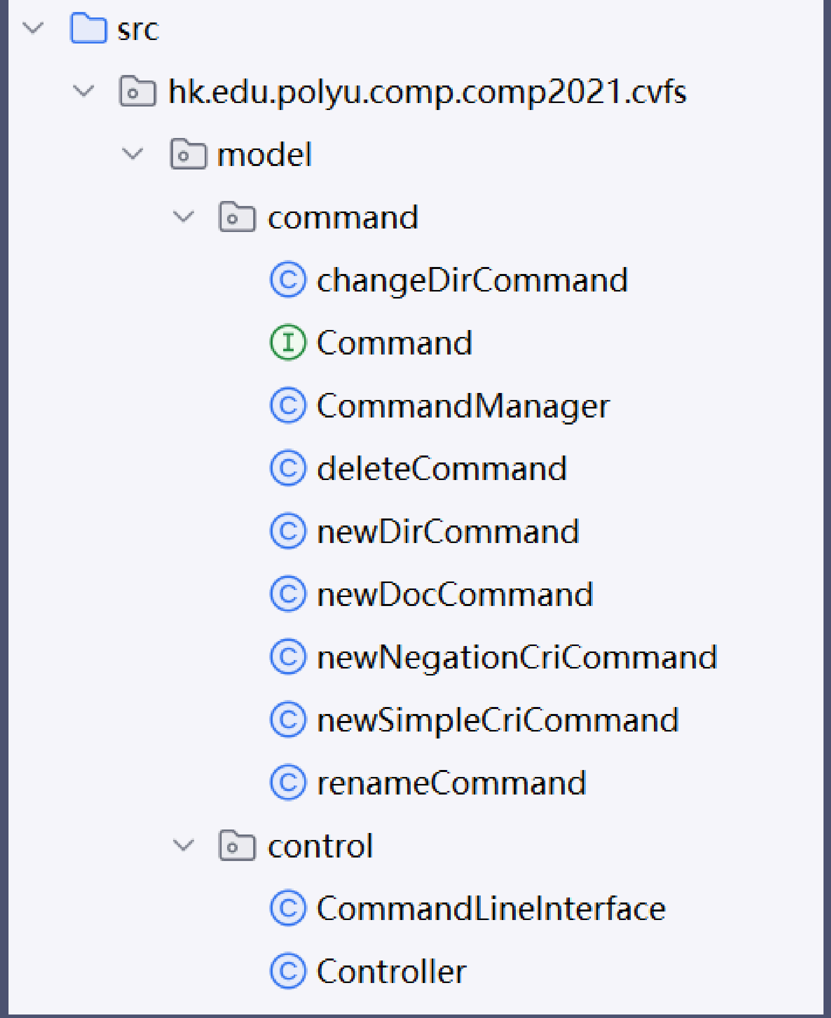
Abstraction



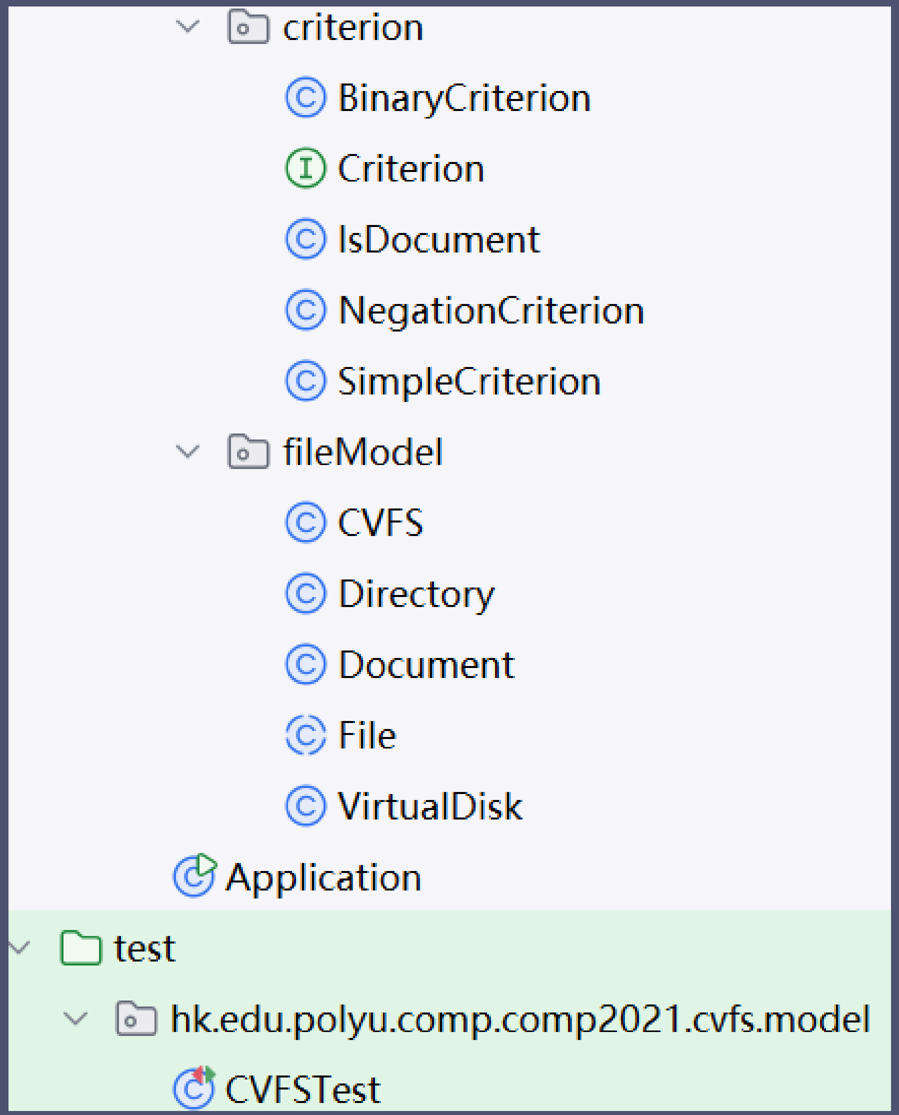
ENCAPSULATION

- Encapsulation is a fundamental concept in object-oriented programming that refers to the bundling of data and methods that operate on that data within a single unit, which is called a class.
- a way of hiding the implementation details of a class from outside access and only exposing a public interface that can be used to interact with the class.





```
src
  hk.edu.polyu.comp.comp2021.cvfs
    model
      command
        changeDirCommand
        Command
        CommandManager
        deleteCommand
        newDirCommand
        newDocCommand
        newNegationCriCommand
        newSimpleCriCommand
        renameCommand
      control
        CommandLineInterface
        Controller
```

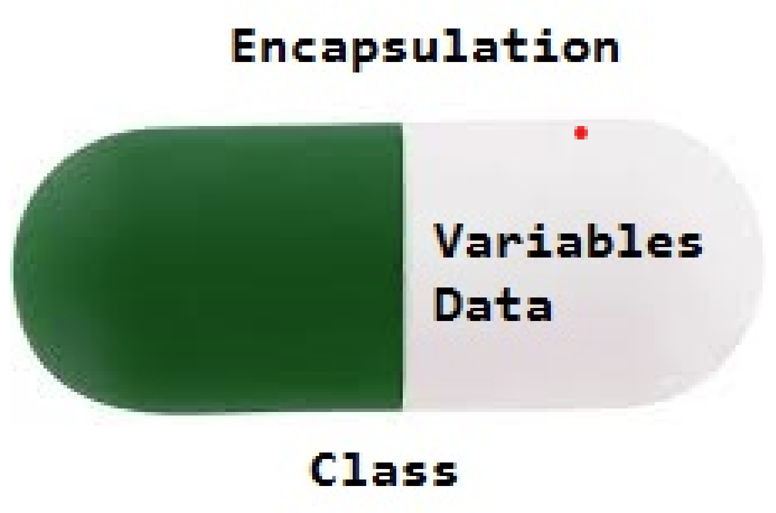


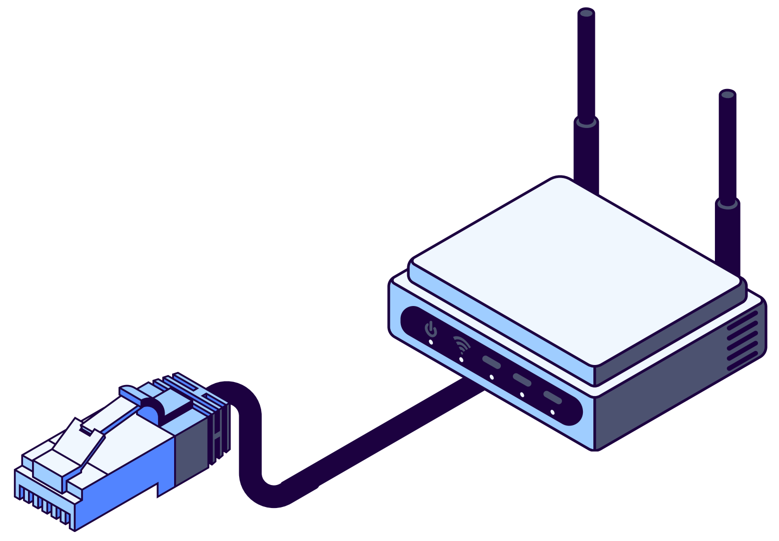
```
      criterion
        BinaryCriterion
        Criterion
        IsDocument
        NegationCriterion
        SimpleCriterion
      fileModel
        CVFS
        Directory
        Document
        File
        VirtualDisk
      Application
test
  hk.edu.polyu.comp.comp2021.cvfs.model
    CVFSTest
```

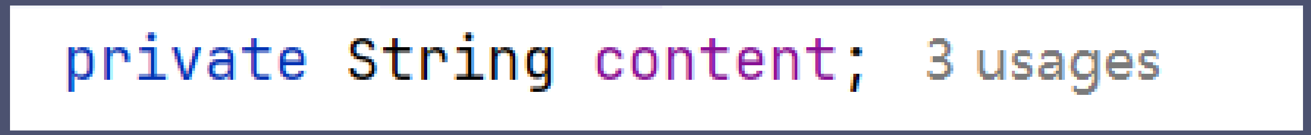
HOW?

- The data in a class is hidden from other classes using the data hiding concept——by declaring the instance variables of a class as private

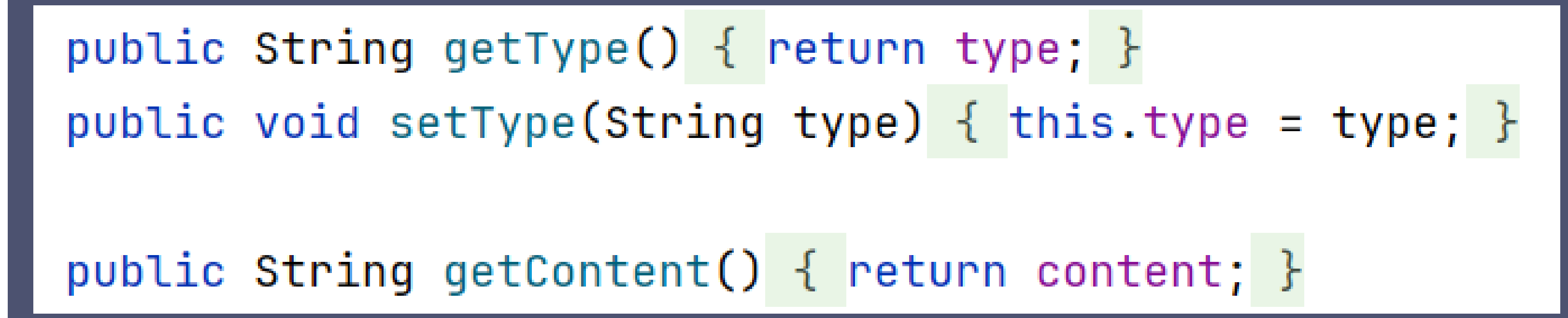
- To allow outside access to the instance variables, public methods called getters and setters are defined.







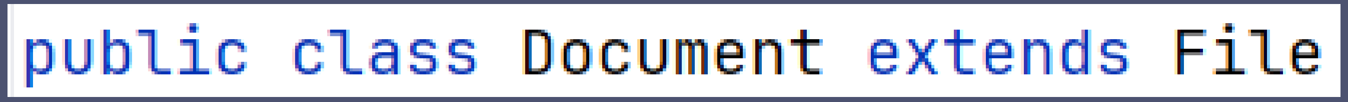
```
private String content;  3 usages
```



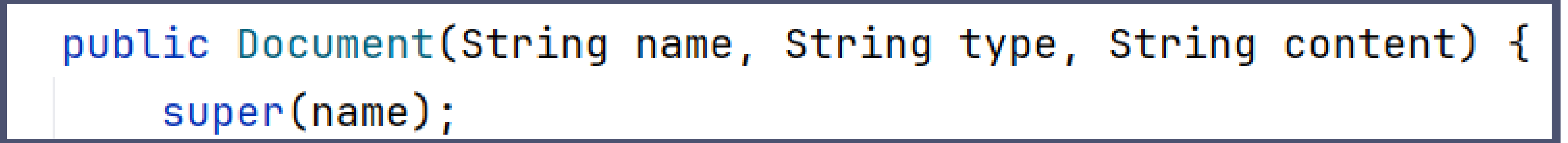
```
public String getType() { return type; }
public void setType(String type) { this.type = type; }

public String getContent() { return content; }
```

Advantages

Data Hiding	The user will have no idea about the inner implementation of the class. It will not be visible to the user how the class is storing values in the variables.
Increased Flexibility	We can make the variables of the class read-only or write-only depending on our needs.
Increased Reusability	Easy to change with new requirements.
Convenient Testing	Easy to test for unit testing.



```java
public class Document extends File
```



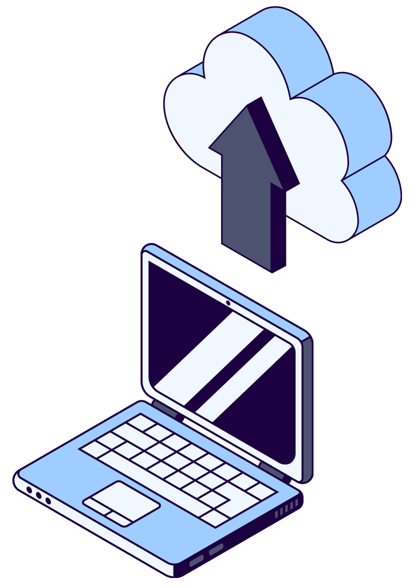
```java
public Document(String name, String type, String content) {
    super(name);
```

INHERITANCE & POLYMORPHISM

- Code Reusability: The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class code.

- Method Overriding: Method Overriding is achievable only through Inheritance.

- Inheritance lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.



ABSTRACTION

- Data abstraction is the process of hiding certain details and showing only essential information to the user.

- It reduces the complexity of viewing things.
- Avoids code duplication and increases reusability.
- Abstraction enables modularity and separation of concerns, making code more maintainable and easier to debug.

```
public abstract class File
```

```
public abstract int getSize();
```

Group 8 COMP2021 Project

THANKS FOR LISTENING!

Huang Lingru, Shen Linghui, Tan Rou Xin