無法在測試控制工具中執行方法
&
修改時應當測試哪些方法?

•••

Joe Hsu

04.15.2020

關於我

- backend engineer
- Java, Golang
- AWS ECS, ELK ...

大綱

- 在遇到難以測試時的方法要如何解決?
- 要如何找出哪些方法是需要被測試的?

遇到難以測試時的方法要如何解決?

- The Case of Hidden Method
- The Case of the “Helpful” Language Feature
- The Case of the Undetectable Side Effect

The Case of Hidden Method

The Case of Hidden Method

picture source

Why would making a private method public bother us?

- The method is just a utility; it isn't something clients would care about.

- If clients use the method, they could adversely affect results from other methods on the class.

The Case of Hidden Method

```
1  class CCAImage
2  {
3  private:
4   void setSnapRegion(int x, int y, int dx, int dy);
5   ...
6  public:
7   void snap();
8   ...
9  };
10
```

Solution

- private -> public
- refactor it

The Case of Hidden Method

```
1  class CCAImage
2  {
3  private:
4   void setSnapRegion(int x, int y, int dx, int dy);
5   ...
6  public:
7   void snap();
8   ...
9  };
10
```

```
1  class CCAImage
2  {
3  protected:
4   void setSnapRegion(int x, int y, int dx, int dy);
5   ...
6  public:
7   void snap();
8   ...
9  };
10
```

The Case of Hidden Method

```
1  class TestingCCAImage : public CCAImage
2  {
3  public:
4   void setSnapRegion(int x, int y, int dx, int dy)
5   {
6   // call the setSnapRegion of the superclass
7   CCAImage::setSnapRegion(x, y, dx, dy);
8   }
9  };
```

violate encapsulation

The Case of the “Helpful” Language Feature

The Case of the “Helpful” Language Feature

```
public void IList getKSRStreams(HttpFileCollection files) {
    ArrayList list = new ArrayList();

    foreach(string name in files) {
        HttpPostedFile file = files[name];
        if (file.FileName.EndsWith(".ksr") ||
        (file.FileName.EndsWith(".txt")
        && file.ContentLength > MIN_LEN)) {
        ...
        list.Add(file.InputStream);
        }
    }
    return list;
}
```

Adapt Parameter method (Chapter 25.1)

Use Adapt Parameter when you can’t use Extract Interface (362) on a parameter’s class or when a parameter is difficult to fake.

Refactor

建立 OurHttpFileCollection 繼承 NameObjectCollectionBase

```
public void LList getKSRStreams(OurHttpFileCollection files) {
    ArrayList list = new ArrayList();
    foreach(string name in files) {
        HttpPostedFile file = files[name];
        if (file.FileName.EndsWith(".ksr") ||
        (file.FileName.EndsWith(".txt")
        && file.ContentLength > MAX_LEN)) {
            ...
            list.Add(file.InputStream);
        }
    }
    return list;
}
```

Refactor - adapt parameter method

1. 建立 IHttpPostedFile interface 來替換 HttpPostedFile
2. 建立一個 Wrapper class 來包覆 HttpPostedFile

```
interface IHttpPostedFile {

    int ContentLength {get;}
}

public class HttpPostedFileWrapper : IHttpPostedFile
{
    public HttpPostedFileWrapper(HttpPostedFile file) {
        this.file = file;
    }
    public int ContentLength {
        get { return file.ContentLength; }
    }
    ...
}
```

Refactor

建立 Fake Object 進行測試

```
public class FakeHttpPostedFile : IHttpPostedFile
{
    public FakeHttpPostedFile(int length, Stream stream, ...) { ... }
    public int ContentLength {
        get { return length; }
    }
}
```

Refactor

重構成可測試化的程式碼

```
public IList getKSRStreams(OurHttpFileCollection files) {
    ArrayList list = new ArrayList();
    foreach(string name in files) {
        IHttpPostedFile file = files[name];
        if (file.FileName.EndsWith(".ksr") ||
        (file.FileName.EndsWith(".txt"))
        && file.ContentLength > MAX_LEN)) {
            ...
            list.Add(file.InputStream);
        }
    }
    return list;
}
```

- 優點:
  - 程式變得可測試化
- 缺點:
  - 程式碼中所有用到 HttpFileCollection 要改成 OurHttpFileCollection
  - 程式碼中所有用到 HttpPostedFile 要先包成 HttpPostedFileWrapper 再放到 OurHttpFileCollection

The Case of the Undetectable Side Effect

The Case of the Undetectable Side Effect

當遇到一個 method 裡面呼叫許多其他物件的時候，讓測試的結果無法預測

來源

The Case of the Undetectable Side Effect

```
public class AccountDetailFrame extends Frame implements ActionListener, WindowListener {

    private TextField display = new TextField(10);
    ...
    public AccountDetailFrame(...) { ... }

    public void actionPerformed(ActionEvent event) {
        String source = (String)event.getActionCommand();

        if (source.equals("project activity")) {
            detailDisplay = new DetailFrame();
            detailDisplay.setDescription(
                getDetailText() + " " + getProjectionText());
            detailDisplay.show();
            String accountDescription = detailDisplay.getAccountSymbol();
            accountDescription += ": ";
            ...
            display.setText(accountDescription);
            ...
        }
    }
    ...
}
```

Refactor

1. 方法提取 extract method
    - 把大方法拆分成多個小方法

2. 命令/查詢 分離 Command/Query Separation

3. 子類別化並覆寫方法 Subclass and Override method
    - 繼承要測試的類別, 把 method 設定想要的回傳結果

Command Query Separation (CQS)

由 Bertrand Meyer 在 1988 於 "Object Oriented Software Construction" 這本書中提出的軟體架構概念,

- method 設計應該分成兩類, 擇一設計:
  - **command**: void setMethod(int a): 會改變物件狀態, 不返回值
  - **query**: int getMethod(): 不會改變物件狀態, 會返回值
- 優點:
  - command 和 query 分工明確, 可分別進行效能調整及最佳化
  - 將業務邏輯上的 command 和 query 職責分離, 提高系統的性能與可擴展性
  - 簡化程式邏輯, 可以從事件過程中看出系統中的行為或者操作造成導致了系統的狀態變化

Refactor - extract method

```
public class AccountDetailFrame extends Frame implements ActionListener, WindowListener {

    // extract method
    public void actionPerformed(ActionEvent event) {
        String source =(String)event.getActionCommand();
        performCommand(source);
    }

    // extract method
    public void performCommand(String source) {
        if (source.equals("project activity")) {
            setDescription(getDetailText() + " " + getProjectionText());
            ...
            String accountDescription = detailDisplay.getAccountSymbol();
            accountDescription += ": ";
            ...
            setDisplayText(accountDescription);
            ...
        }
    }
```

Refactor - command/query separation

```
    // Command Separation
    void setDescription(String description) {
        detailDisplay = new DetailFrame();
        detailDisplay.setDescription(description);
        detailDisplay.show();
    }

    // Query Separation
    String getAccountSymbol() {
        return detailDisplay.getAccountSymbol();
    }

    // Command Separation
    void setDisplayText(String description) {
        display.setText(description);
    }
    ...
}
```

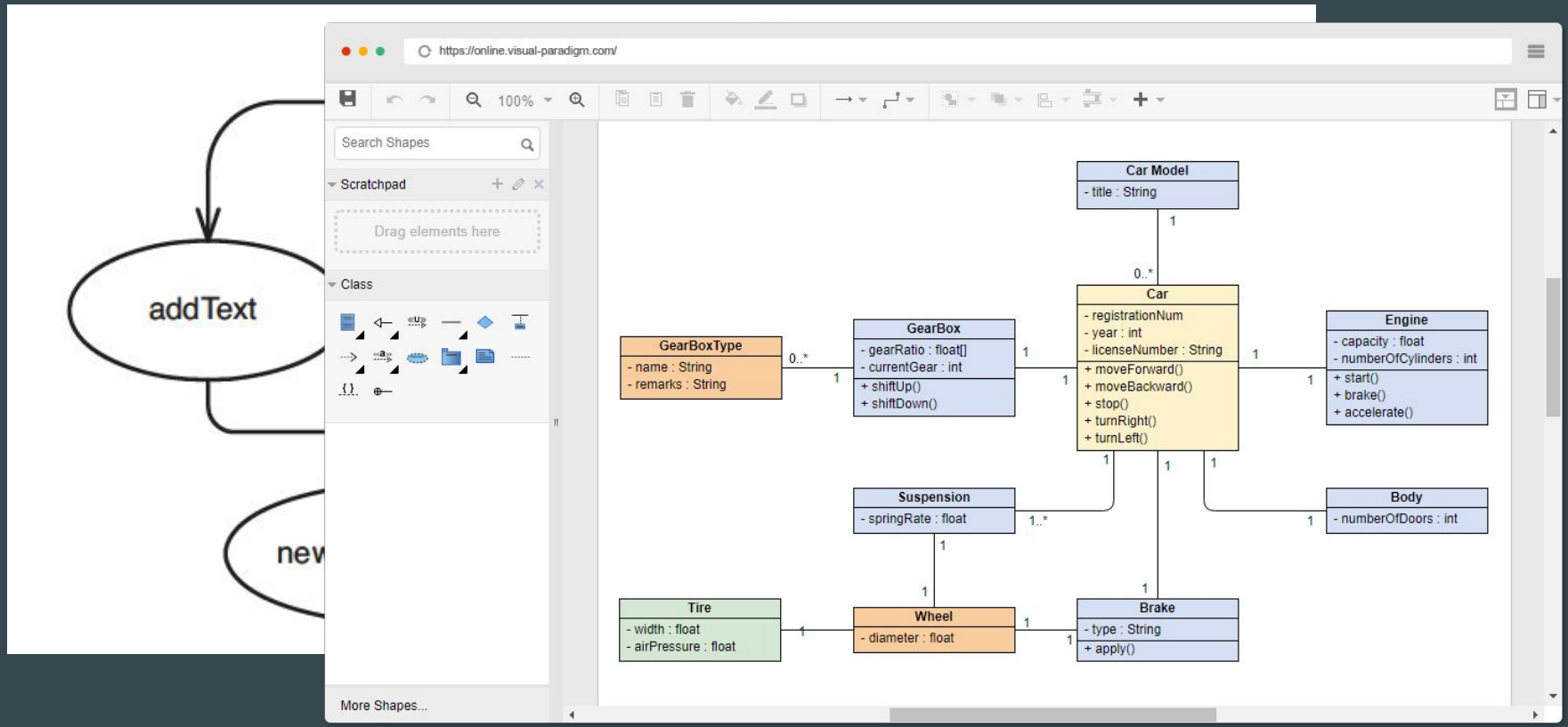
Refactor - subclass and override method

```
// Subclass and Override Method
public class TestingAccountDetailFrame extends AccountDetailFrame {
    String displayText = "";
    String accountSymbol = "";

    void setDescription(String description) {
    }

    String getAccountSymbol() {
        return accountSymbol;
    }

    void setDisplayText(String text) {
        displayText = text;
    }
}
```

```
public void testPerformCommand() {
    TestingAccountDetailFrame frame = new TestingAccountDetailFrame();
    frame.accountSymbol = "SYM";
    frame.performCommand("project activity");
    assertEquals("SYM: basic account", frame.displayText);
}
```

要如何找出哪些方法是需要被測試的?

Effect sketches 影響草圖



```
public class CppClass {
    private String name;
    private List declarations;

    public CppClass(String name, List declarations) {
        this.name = name;
        this.declarations = declarations;
    }

    public int getDeclarationCount() {
        return declarations.size();
    }

    public String getName() {
        return name;
    }

    public Declaration getDeclaration(int index) {
        return ((Declaration)declarations.get(index));
    }

    public String getInterface(String interfaceName, int [] indices) {
        String result = "class " + interfaceName + " {\npublic:\n";

        for (int n = 0; n < indices.length; n++) {
            Declaration virtualFunction = (Declaration)(declarations.get(indices[n]));
            result += "\t" + virtualFunction.asAbstract() + "\n";
        }
        result += "};\n";
        return result;
    }
}
```

```
public class CppClass {
    private String name;
    private List declarations;

    public CppClass(String name, List declarations) {
        this.name = name;
        this.declarations = declarations;
    }

    public int getDeclarationCount() {
        return declarations.size();
    }

    public String getName() {
        return name;
    }

    public Declarat
        return ((De
    }

    public String g
        String resu

        for (int n
            Declara
            result
        }
        result += "
        return resu
    }
}
                                                    ]));
```

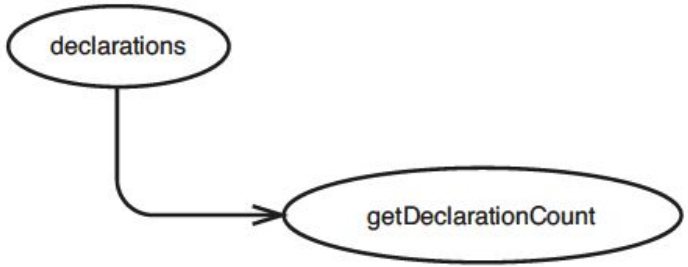


Figure 11.1 declarations *impacts* getDeclarationCount.

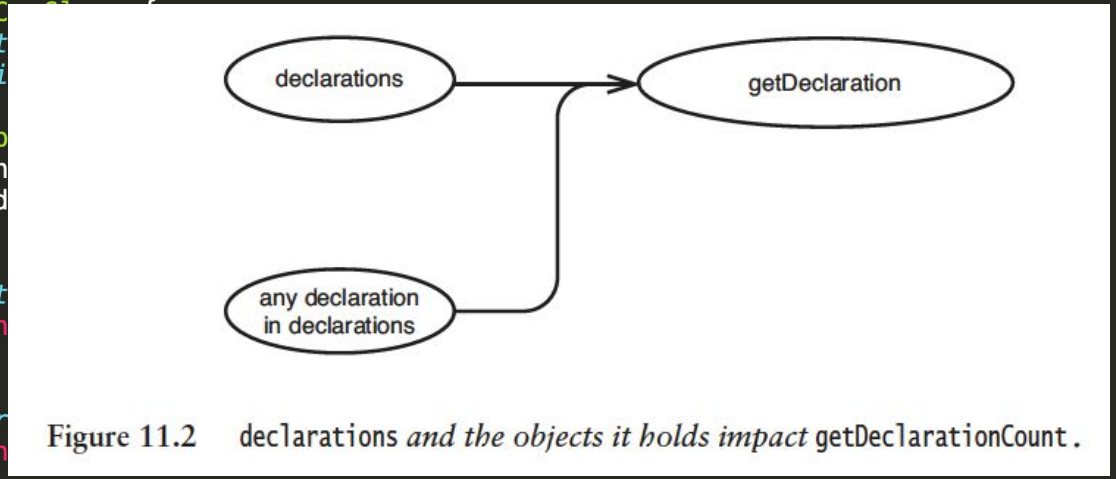


Figure 11.2 declarations *and the objects it holds impact* getDeclarationCount.

```
public class C
    private St
    private Li

    public Cpp
        this.n
        this.d
    }

    public int
        return
    }

    public Str
        return
    }

    public Declaration getDeclaration(int index) {
        return ((Declaration)declarations.get(index));
    }

    public String getInterface(String interfaceName, int [] indices) {
        String result = "class " + interfaceName + " {\npublic:\n";

        for (int n = 0; n < indices.length; n++) {
            Declaration virtualFunction = (Declaration)(declarations.get(indices[n]));
            result += "\t" + virtualFunction.asAbstract() + "\n";
        }
        result += "};\n";
        return result;
    }
}
```

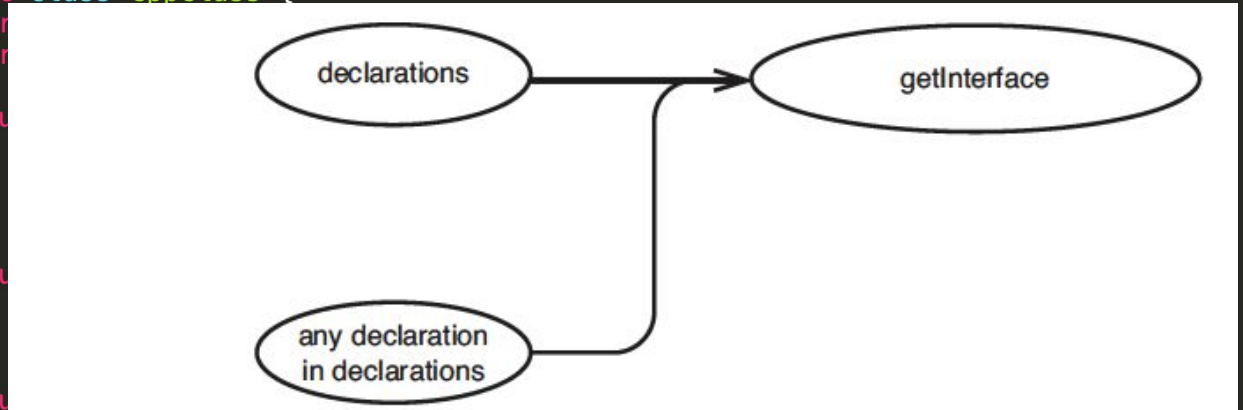


Figure 11.3 *Things that affect* getInterface.

```
public class CppClass {
    pr
    pr

    pu

    }

    pu

    }

    pu

    }

    pu
        return ((Declaration)declarations.get(index));
    }

    public String getInterface(String interfaceName, int [] indices) {
        String result = "class " + interfaceName + " {\npublic:\n";

        for (int n = 0; n < indices.length; n++) {
            Declaration virtualFunction = (Declaration)(declarations.get(indices[n]));
            result += "\t" + virtualFunction.asAbstract() + "\n";
        }
        result += "};\n";
        return result;
    }
}
```

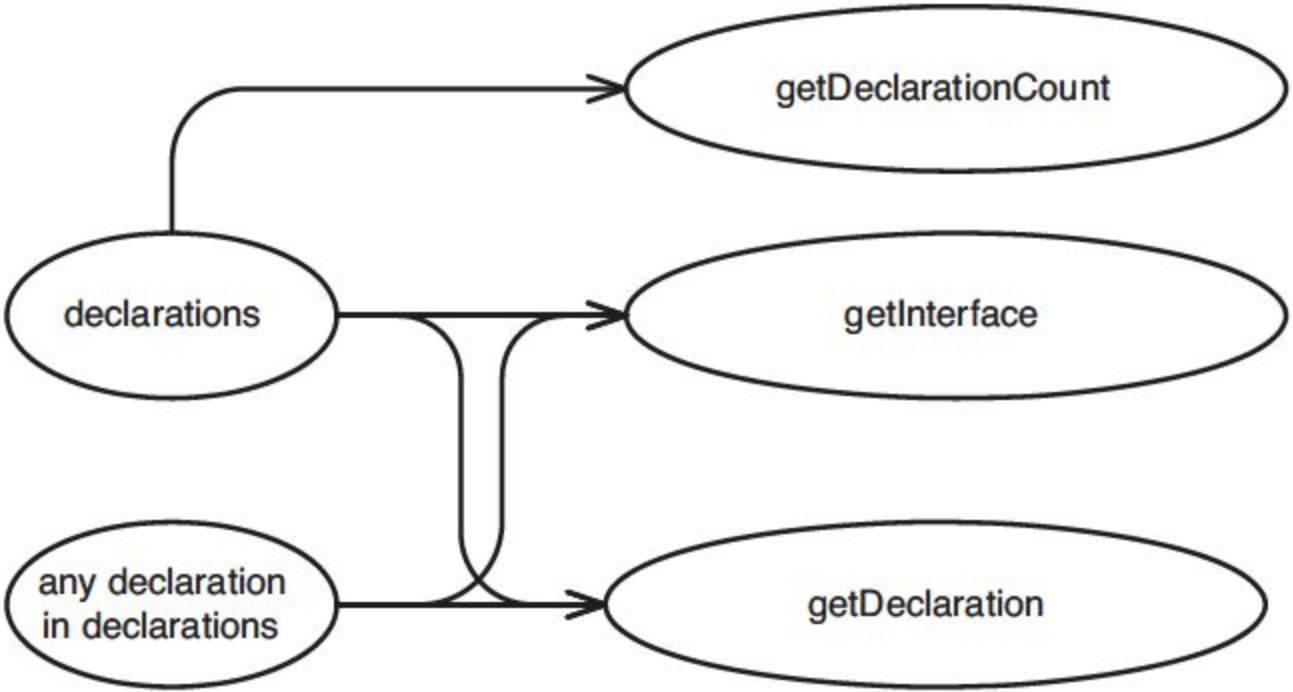


Figure 11.4 *Combined effect sketch.*

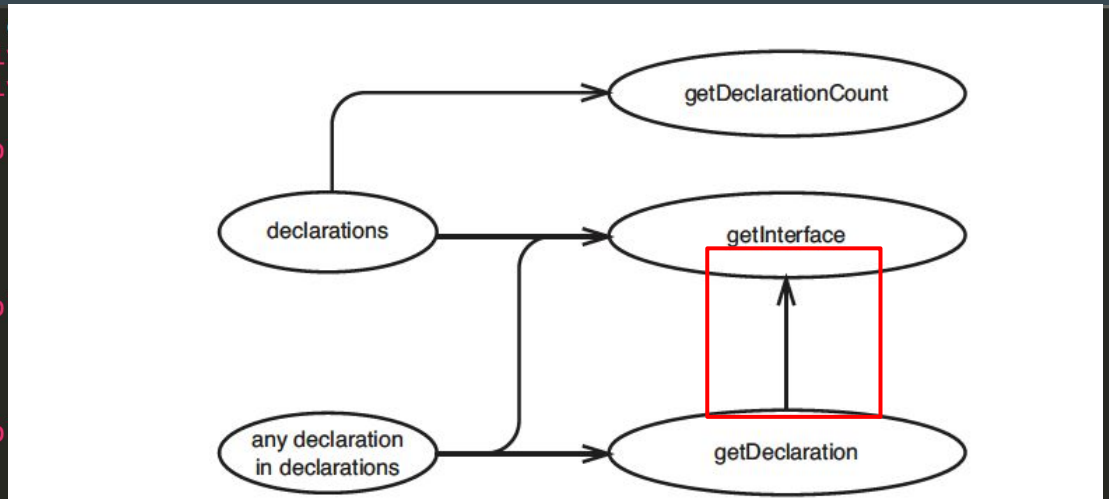
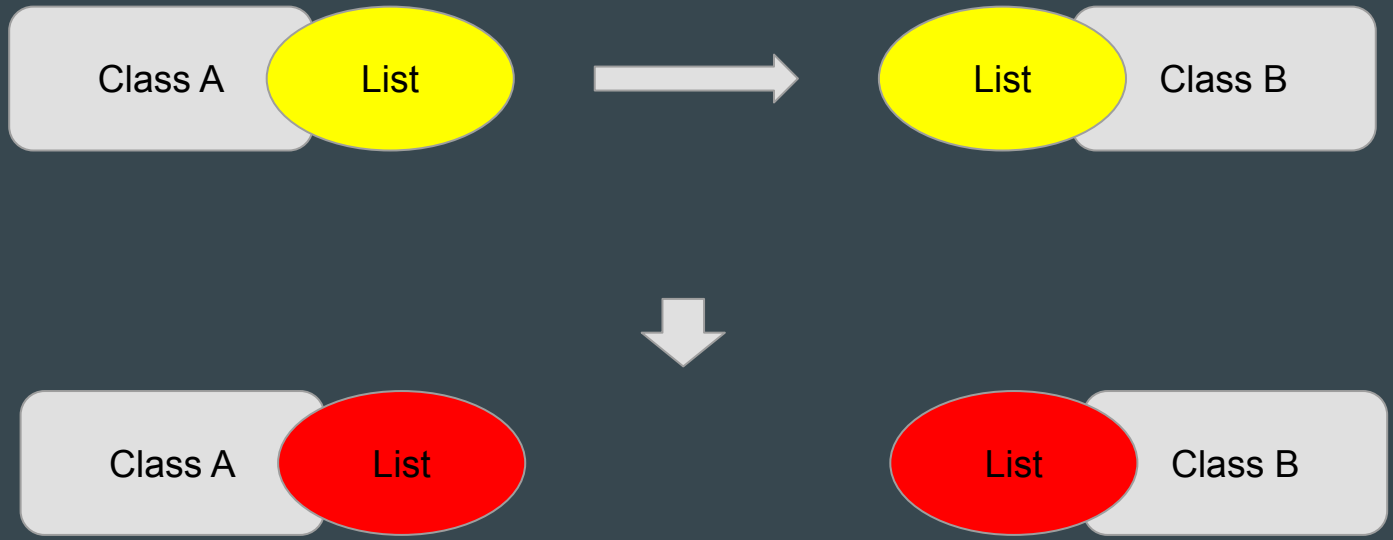


Figure 11.13 *Effect sketch for Changed* CppClass.

```
public String getInterface(String interfaceName, int [] indices) {

    String result = "class " + interfaceName + " {\npublic:\n";
    for (int n = 0; n < indices.length; n++) {
        Declaration virtualFunction = getDeclaration(indices[n]);
        result += "\t" + virtualFunction.asAbstract() + "\n";
    }
    result += "};\n";
    return result;
}
}
```

Effect propagation

程式碼的修改所產生的影響有時候會以不易察覺的方式傳播.



Conclusion

- 介紹三個在寫測試中遇到棘手的情況時候要如何解決
  - **hidden method** -> protect
  - **無法使用介面抽取的方式替換輸入的參數** -> adapt parameter method
  - **method 當中有許多物件交互影響** -> extract method, CQS, subclass and override method

- 如何從程式碼找出在物件建立後受影響的參數與函式, 進而找出程式潛在的問題, 並且找出寫測試的點