Object Oriented Programming

by

Vivek Dutta Mishra

Slum House vs Architected House

- Slum Houses
- Have
  - Have Rooms
  - Have Electricity
  - People Leave in there
- Who Designs Them?
- How Much time it takes to build them?
- What's the cost?
- Ask same question for Architected House.

What would you chose - Slum House vs Architected House?

- Why?
  - Cost
  - Time to Build
- Really?
- Why do you never chose Slum?
  - Better Quality
  - Not used to living in slums?

How come we are not ready to live in slum but more than ready to write slum code?

quick dirty code

Case Study 1 – Is This Object Oriented Design

- Two objects Employee and File are Interacting
- File is writing the Employee details.

Isn't this violation of Encapsulation ???

```
void main(){
        Employee emp=new Employee(...);
        File file=new File ("c:/db/emp.db");

        file.Write("%s, %s, %s\n", emp.name,
                                   emp.id,
                                   emp.password);

}
```

Is File writing Employee or String???

We can use OO and still create a bad design.

Case Study 1 – Is This Object Oriented Design

- Two objects Employee and File are Interacting
- File is writing the Employee details.

```
void main(){
        Employee emp=new Employee(...);
        File file=new File ("c:/db/emp.db");

    //file.Write("%s, %s, %s\n", emp.name,
                                emp.id,
                                emp.password);

     file.Write(emp);

}
```

Lets change the code

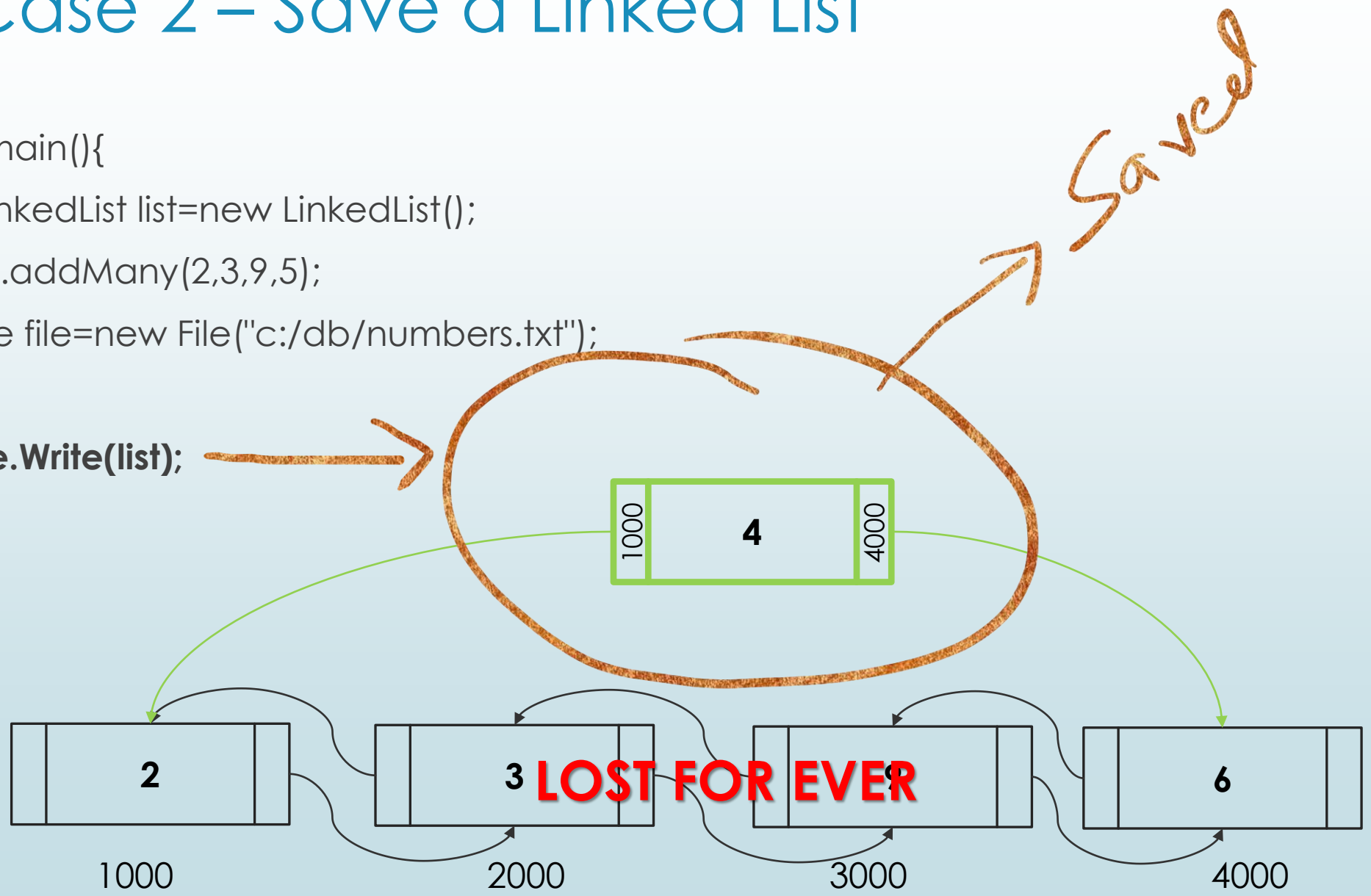
But is this a universal solution???

Now file is writing the entire Employee and not breaking Encapsulation

Case 2 – Save a Linked List

```
void main(){
    LinkedList list=new LinkedList();
    list.addMany(2,3,9,5);
    File file=new File("c:/db/numbers.txt");

    file.Write(list);
}
```



What have be Observed?

```
void main(){
        Employee emp=new Employee(...);
        File file=new File ("c:/db/emp.db");

    //file.Write("%s, %s, %s\n", emp.name,
                                   emp.id,
                                   emp.password);

     file.Write(emp);

}
```

NOT OO

Works Here

What is the real Defect???

```
void main(){
        LinkedList list=new LinkedList();
       list.addMany(2,3,9,5);
       File file=new File("c:/db/numbers.txt");

       file.Write(list);
}
```

But Not Here

What do we do?

1. Find Next Solution That may work
   - No assurance that the new solution work in next case.
   - This is patch work
   - Avoid
2. Find The Actual Problem
   - Often finding problem is the solution
   - Avoid "You should be doing…" approach.
   - Beawre "Often you may be offering your suggestion as problem statement"

Find the problem

```
void main(){
        Employee emp=new Employee(...);
        File file=new File ("c:/db/emp.db");

    //file.Write("%s, %s, %s\n", emp.name,
                                   emp.id,
                                   emp.password);

     file.Write(emp);

}
```

```
void main(){
        LinkedList list=new LinkedList();
       list.addMany(2,3,9,5);
       File file=new File("c:/db/numbers.txt");

       file.Write(list);
}
```

Hint !!!
All 3 write has same problem

And It has nothing to do with “Write”

What's your conclusion???

- We Should Have Write Method in Employee class rather than the File Class.
- You haven't followed Problem Finding Approach.
- You have patched.
- We will revisit this solution soon!!!

The Real Problem

- File class doesn't have any knowledge about Employee or LinkedList
- How can we expect File class to handle something it doesn't know about

Responsibility Theory

- Knowledge is Ownership
  - If you know something you own it.
- Ownership is Responsibility
  - If you know something you are responsible

Lesson #
The Less you know less you need to be responsible about.

Where does it lead us?

- Since Employee Object knows its own structure Employee should be responsible for Writing Itself.

```
class Employee{
    public void WriteTo(File file){
        file.Write("%s, %s, %s\n", this.name,
                                   this.id,
                                   this.password);
    }
}
```

But isn't it the same code we started with???

The same code in main is bad as main doesn't know Employee details

#Lession 1
Its not about executable code. Its about responsibility.

Ask whose responsibility?

Serialization

- When we have a reusable design it makes sense to name it.
- What we used is known as Serialization

Serialization is an Objects Ability to Persist Itself. Deserialization is the reverse of Serialization.

Where does it lead us?

- Since Employee Object knows its own structure Employee should be responsible for Writing Itself.

```
class Employee{
    public void WriteTo(File file){
        file.Write("%s, %s, %s\n", this.name,
                                   this.id,
                                   this.password);
    }
}
```

But isn't the same design we reached anyway?

#Lession 2
Wrong approaches may still lead to a seemingly correct design.

Points to reconsider

- Which is More correct About:
  - File Knowing Linked List/ Employee details
    - File Doesn't know Employee Details or
    - File Shouldn't know Employee Details
  - Responsibility of writing
    - Employee should know how to write itself or
    - File shouldn't know How to write Employee?

File Can Know Employee and LinkedList

```
class File{
    public void  Write(Employee emp){
        //specific logic for Employee
    }

    public void  Write(LinkedList list){
        //specific logic for Employee
    }
}
```

Code works

But File has too many responsibility

How many more object would it need to know about?

If it should always be Object to write about itself

- It can
- But what about
  - Writing to Different Targets : File, Database , Cloud etc?
  - Writing in different format : CSV, JSON, XML?

Takeaway

- File **shouldn't** know the details of Employee and LinkedList
  - File Shouldn't be responsible for writing other Objects
  - It should still exist to implement Generic File Operation
- There are two choices for Writing an Object to File.
  - Object Itself (We already Discussed This)
  - Another Object who is specifically responsible for Writing Object in specific format

One Object May be Responsible to Write Another Object

```
class EmployeeXmlWriter {
    public void  Write(File file, Employee emp){
        //specific logic for Employee
    }
}
class ListXmlWriter{
    public void  Write(File file, LinkedList list){
        //specific logic for Employee
    }
}
```

What if **File** object can read other objects details

- In such cases File can write any Object
  - Remember: Knowledge is responsibility
- Modern Language have features like reflection that can Help an object understand another object dynamically
- Still a better idea would be

```
class ObjectXmlWriter
{
  public void Write(File file ,Object anyObject) {…}
}
```

Lets Redefine Serialization

- ***Serialization is the process of Persisting an Object. Deserialization is the reverse of Serialization.***
- ***What changed from Original Definition?***
- ***Serialization is an Objects Ability to Persist Itself. Deserialization is the reverse of Serialization.***

Summary

- Just using Object Oriented Construct Doesn't Make an **Object Oriented Design.**
  - We can still write bad code
- Even Bad codes work. They may give desired result.
- A good design is a responsible design.
- Knowledge is Ownership → Ownership is responsibility
- Responsibility may not be vested in object having the property
  - It may be in any object that has the knowledge.
- To Reduce Responsibility you should reduce knowledge.