CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2015
Doc 5 Assignment 1 Comments
Feb 10, 2015

Topics

Class = data + operations
Abstraction
Information Hiding

Code Reuse
Modifiability
Safety

```
public class LinkedList {

    public void getKthElement(int k) {
        blah
        blah
        System.out.println( blah );
```

void - not returning a value

getXXX - returning a value

Can not test using JUnit

Program can’t access elements of list

No use

No code reuse

```
public class LinkedList {

    public void displayProbationStudents() {
        blah
        blah
        System.out.println( blah );
        blah
        System.out.println( blah );
```

Can not test using JUnit

Program can’t access elements of list

Can’t use in

- Web app
- Desktop app
- Mobile App
- Batch processing
- Server side computing
- Enterprise computing

```
public class LinkedList {

    public ArrayList getProbationStudents() {
        ArrayList<Students> probation = new ArrayList<Student>():
        blah
        blah
        return probation;
    }
```

```
Somewhere else

    LinkedList students = blah;
    blah
    ArrayList<Student> probation = students.getProbationStudents();
    System.out.println(probation);
```

```
public class LinkedList {

    public ArrayList getProbationStudents() {
        ArrayList<Students> probation = new ArrayList<Student>():
        blah
        blah
        return probation;
    }
```

Can test using JUnit

Program can access elements of list

Can use in

- Web app
- Desktop app
- Mobile App
- Batch processing
- Server side computing
- Enterprise computing

Keep display code separate from computation

Makes computation resuable

System.out.println

Part of view

Rarely use standard out for view

Use for debugging

What Operations belong in LinkedList?

add element

retrieve element

remove element

remove all elements

test if element is in list

find all students on probation

find all students with 4.0 GPA

iterate over all elements in list

```
public class LinkedList {

    public Iterator iterator() { blah }
```

Somewhere else

```
    LinkedList students = blah;
    blah
    iterator allStudents = students.iterator();
    ArrayList<Student> probation = new ArrayList<Student>()
    while (allStudents.hasNext()) {
        Student current = allStudents.next();
        if (current.onProbation() )
            probation.add(current);
    }
```

```
public class LinkedList {

    public Iterator iterator() { blah }
```

Somewhere else

```
    LinkedList students = blah;
    blah
    ArrayList<Student> probation = students.stream()
                                        .filter(each -> each.onProbation())
                                        .collect(Collector.toList());
```

```
public class LinkedList {
    private Node head;

    public Node getKthElement(int k) {
        Node current = head;
        blah
        blah
        return current;
    }
```

Violates information hiding

Not safe

- Everyone has access to node
- Can change linked list directly

Shotgun surgery

- Change to node class can require change to all users of list

Clients have to

- Know about Node
- Repeatedly pull data out of node

```
public class LinkedList {
    private Node head;

    public Node getHead() {
        return head;
    }

    public void add(Node element) {
        blah
    }
```

Violates information hiding

```
class Node {
    private Student data;
    private Node next;
    private Node previous;
    etc,
}

class LinkedList {
    private Node head;
```

Code Reuse

How often do we need list of students?

```
class Node {
    private Object data;
    private Node next;
    private Node previous;
    etc,
}

class LinkedList {
    private Node head;
```

Code Reuse

Now get list of objects

Used all the time

```
class Node<E> {
    private E data;
    private Node next;
    private Node previous;
    etc,
}

class LinkedList<E> {
    private Node<E> head;
```

Code Reuse

Now get list of objects

Used all the time

Now have type checking

```
class Node {
    private String name;
    private String redId;
    private float gpa;
    private Node next;
    private Node previous;
    etc,
}

class LinkedList {
    private Node head;
```

What is it?

Node?
Student?

```
class LinkedList<E> {
    private static Node<E> head;

    public static void add(Object item) {
        blah
    }
```

```
LinkedList<Student> cs635 = new LinkedList<>();
LinkedList<Student> cs646 = new LinkedList<>();
cs635.add(joe);
cs646.add(pete);
```

Both list have both students

Really just one list

```
public class OutOfBoundsException extends Exception {

    public OutOfBoundsException(String message) {
        super("OutOfBoundsException " + message);
    }
}

public class LinkedList {

    public Student get(int index) throws OutOfBoundsException {
```

But Java has IndexOutOfBoundsException

Now we have to know about both and handle both

```
public class LinkedList {

    public Student get(int index) throws IndexOutOfBoundsException {
```

Now we just have to know about and handle one

How does the caller know an exception occured?

```
public class LinkedList {

    public Student get(int index) {
        Student s = null;
        try {
            if (index >= size)
                throw IndexOutOfBoundsException("Bad index" + index);

            now go find the right student

        } catch (IndexOutOfBoundsException e) {
            System.err.println(e.getMessage());
        }
        return s;
    }
```

```
public class LinkedList {

    public Student get(int index) {
        if (index >= size) {
            System.err.println(e.getMessage());
            return null;
        Student s;
        now go find the right student

        return s;
    }
```

Does the same thing

Simpler

LinkedList can not know what application should do when index is out of bounds

```
public class LinkedList {

    public Student get(int index) throws IndexOutOfBoundsException {
        Student s = null;
        if (index >= size)
            throw IndexOutOfBoundsException("Bad index" + index);

        now go find the right student

        return s;
    }
```

```
struct Node {
    Object data;
    Node next;
    Node previous;
}
```

```
class Node {
    public Object data;
    public Node next;
    public Node previous;
}
```

```
class Node {
    private Object data;
    private Node next;
    private Node previous;

    public Object getData() {return data;}
    public Node getNext() {return next;}
    public Node getPrevious() {
        return next;
    }
    public void setData(Object item) {
        data = item;
    }
    public void setNext(Node newNext) {
        next = newNext;
    }
    public void set Previous(Node n) {
        next = n;
    }
}
```

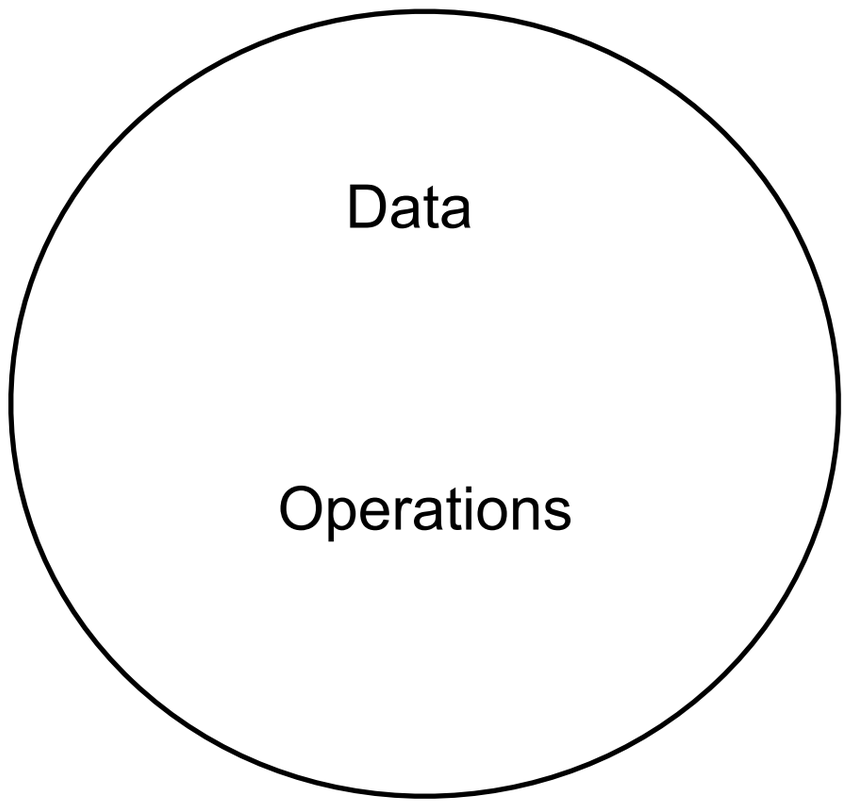
Where are the operations?

Class

Represents an abstraction

Encapsulates data and operations of the abstraction

Hide design decisions/details



Not so much a definition of a class as a goal how we should use a class.

Heuristics

2.1 All data should be hidden within it class

2.8 A class should capture one and only one key abstraction

2.9 Keep related data and behavior in one place

Numbers of the heuristics are from the text Object-Oriented Design Heuristics by Riel

```
struct Node {
    Object data;
    Node next;
    Node previous;
}
```

```
class Node {
    public Object data;
    public Node next;
    public Node previous;
}
```

Where are the operations?

Why are you writing 1/2 a class?

In an app using Linked List

```
public class LinkedList {
    private Node head;

    public Node getKthElement(int k) {}
```

There will be many uses of the list

```
public void foo(LinkedList x) {
    blah
    blah
    z = x.getKthElement(3);
```

What happens if we decide using an ArrayList would be better?

Shotgun surgery

- Have to find all uses of “getKthElement” and replace with “get”
- Find all uses of linked list methods and replace with ArrayList methods
- Have to find all occurrences of “LinkedList” and replace with “ArrayList”
- Replace “new LinkedList” with “new ArrayList”

```
public class LinkedList {
    private Node head;

    public Node getKthElement(int k) {}
```

```
public class LinkedList<E> {
    private Node<E> head;

    public <E> get(int k) {}
```

In your classes use the names that your library uses for similar purposes

Now what happens if we decide using an ArrayList would be better?

Shotgun surgery

- Have to find all occurrences of “LinkedList” and replace with “ArrayList”
- Replace “new LinkedList” with “new ArrayList”

java.util.List

Interface for ordered collections

Defines the methods in ordered collection classes

ArrayList,
AttributeList,
CopyOnWriteArrayList,
LinkedList,
RoleList,
RoleUnresolvedList,
Stack,
Vector

```
public class LinkedList<E> {
    private Node<E> head;

    public <E> get(int k) {}
```

→

```
public class LinkedList<E> implements List {
    private Node<E> head;

    public <E> get(int k) {}
```

```
LinkedList<Students> students = new LinkedList<>();
```

↘

```
List<Students> students = new LinkedList<>();
```

```
public void foo(LinkedList x) {
    blah
    blah
```

↘

```
public void foo(List x) {
    blah
    blah
```

Now what happens if we decide using an ArrayList would be better?

Just replace “new LinkedList” with “new ArrayList”

That is why you points for

getSize()
getKthElement
getStudent
addStudent
insert

```
class LinkedList {
    private Node head;
    private Node current;

    public Student get(int n) throws IndexOutOfBoundsException {
        if (root == null) {
            throw new IndexOutOfBoundsException(“ list is empty”);
        } else if (getSize() <= n) {
            throw new IndexOutOfBoundsException(“ index ” + n + “out of bounds”);
        }

        current = root;
        Student s = null;
        int i = 0;  //variable that keeps track of where we are
        blah
}
```