CS 535 Object-Oriented Programming & Design
Fall Semester, 2000
Doc 2 Classes & Objects
Contents

Reading

Object-Oriented Design Heuristics, Riel, Chapters 1 & 2.

Designing Object-Oriented Software, Wirfs-Brock, Chapters 1 & 2

Class & Object

Class

Encapsulates a single abstraction

Uses information hiding to insure only the relevant parts of the abstraction are visible

Abstraction contains:

Data

Operation on the data

Object

An instance of a class

Represents a particular instance of the abstraction

The Main Point in OO

A class contains:

Data and
Operations on the data

If this is not the case you have a problem!

Java Example

```
class Stack  {

   private float[] elements;
   private int topOfStack = -1;

   public Stack( int stackSize )  {
      elements = new float[ stackSize ];
   }

   public void push( float item )  {
      elements[ ++topOfStack ] = item;
   }

   public float pop()  {
      return elements[ topOfStack-- ];
   }

   public boolean isEmpty()  {
      if ( topOfStack < 0 ) return true;
      else             return false;
   }

   public boolean isFull()  {
      if ( topOfStack >= elements.length )       return true;
      else                                  return false;
   }
}
```

Objects

```
Stack me = new Stack( 20 );
Stack you = new Stack( 200 );
me.push( 5 );
you.push( 12 );
System.out.println( me.pop() );
```

C++ Version

```
class Stack {
public:

   Stack();
   int isEmpty();
   int isFull();
   void push( int item );
   float pop();

private:
   float stackElements[ 100 ];
   int topOfStack;
};

Stack :: Stack()        {
   topOfStack = 0;
}

int Stack :: isEmpty() {
   if ( topOfStack == 0 ) return 1;
   else return 0;
}

int Stack :: isFull()   {
   if ( topOfStack == 100 ) return 1;
   else return 0;}

void Stack :: push( int item ) {
   stackElements[ topOfStack++ ] = item;
   }

float Stack :: pop(){
   return stackElements[ --topOfStack ];
   }
```

Using the Stack

```
int main()
{
   int X;          // No op statement at runtime

   Stack TreeLinks;          // calls Stack :: Stack() on TreeLinks

   TreeLinks.push( 5.0 );

   Stack Nodes;       // calls Stack :: Stack() on Nodes

   Nodes.push( 3.3 );

   TreeLinks.push( 9.9 );

   cout << TreeLinks.pop() << endl;

   return 0;
}
```

Smalltalk Example

```
Object subclass: #Stack
   instanceVariableNames: 'elements '
   classVariableNames: ''
   poolDictionaries: ''
   category: 'Whitney-Courses'

isEmpty
   ^elements isEmpty

isFull
   ^false

pop
   ^elements removeLast

push: anObject
   elements add: anObject

initialize
   elements := OrderedCollection new.

Stack class methodsFor: 'instance creation''
new
   ^super new initialize
```

Using the Stack

```
| stack result |
stack := Stack new.

stack
   push: 3;
   push: 'Hi mom';
   push: 4.

result := stack pop.
```

Some Beginner Errors
Direct Access to Data

```
class Stack  {

   public float[] elements;
   public int topOfStack = -1;

   public Stack( int stackSize )  {
      elements = new float[ stackSize ];
   }

   public void push( float item )  {
      elements[ ++topOfStack ] = item;
   }

   public float pop()  {
      return elements[ topOfStack-- ];
   }

   etc.
}
```

Some students did this once in an assignment. They realized they often performed pop twice in a row then did a push. To save time they accessed the array of element directly. But they messed up the array and top of pointer. It took them many hours to debug their program. Many had to come to me for help. All this to save runtime on a program that was already 100 times faster than it needed to be!

Heuristic 2.1

All data should be hidden within its class

Public data affects

Decomposability
Understandability
Continuity
Protection
Coupling

OK All the Data is Hidden

```
class StackData {
   private  float[]  elements = new float[100];
   private  int        topOfStack         = -1;

   public int getTopOfStack() {
      return topOfStack;
   }

   public void setTopOfStack( int newTop ) {
      topOfStack = newTop;
   }

   public float getElement( int elementIndex ) {
      return elements[ elementIndex ];
   }

   public void setElement( int elementIndex, float element ) {
      elements[ elementIndex ] = element;
   }
}
```

Information Hiding - Physical and Logical
Physical Information Hiding

Physical information hiding is when a class has a field and there are accessor methods, getX and setX, setting and getting the value of the field. It is clear to everyone that there is a field named X in the class. The goal is just to prevent any direct access to X from the outside. The extreme example is a struct converted to a class by adding accessor methods. Physical information hiding provides little or no help in isolating the effects of changes. If the hidden field changes type than one usually ends up changing the accessor methods to reflect the change in type.

Logical Information Hiding

Logical information hiding occurs when the class represents some abstraction. This abstraction can be manipulated independent of its underlying representation. Details are being hidden from the out side world. Examples are integers and stacks. We use integers all the time without knowing any detail on their implementation. Similarly we can use the operations pop and push without knowing how the stack is implemented.

More Heuristics

2.9 Keep related data and behavior in one place

3.3 Beware of classes that have many accessor methods defined in their public interfaces. Having many implies that related data and behavior are not being kept in one place

2.8 A class should capture one and only one key abstraction

Which is Better?

```
class StudentA {
   public String     name;
   public String  address;
   public String  phone;
}

class StudentB {
   public String  name;
   public String  address;
   public String  phone;

   public void setName( String newName ) {
      name = newName;
   }

   public String getName( ) {
      return name;
   }

   public void setAddress( String newAddress ) {
      address= newAddress ;
   }

   public String getAddress( ) {
      return address;
   }

etc.
}
```