Object-Oriented Design

Approaches to object-oriented design

Some principles for good design

Benefits and Drawbacks of the Object-Oriented Approach

A Model For Creating Computer Software

Specify the problem
- i.e., Determine what problem do we wish to solve

Determine how to solve the problem
- Designing a system to solve the problem (plan)

Implement the solution
- Writing up a solution in the chosen application language

Maintenance of the solution
- Fixing bugs, implementing new features

A Model For Creating Computer Software

Specify the problem

- i.e., Determine what problem do we wish to solve

Determine how to solve the problem

- Designing a system to solve the problem (plan)

Implement the solution

- Writing up a solution in the chosen application language

Maintenance of the solution

- Fixing bugs, implementing new features

Approaches To Object-Oriented Design

Traditional techniques

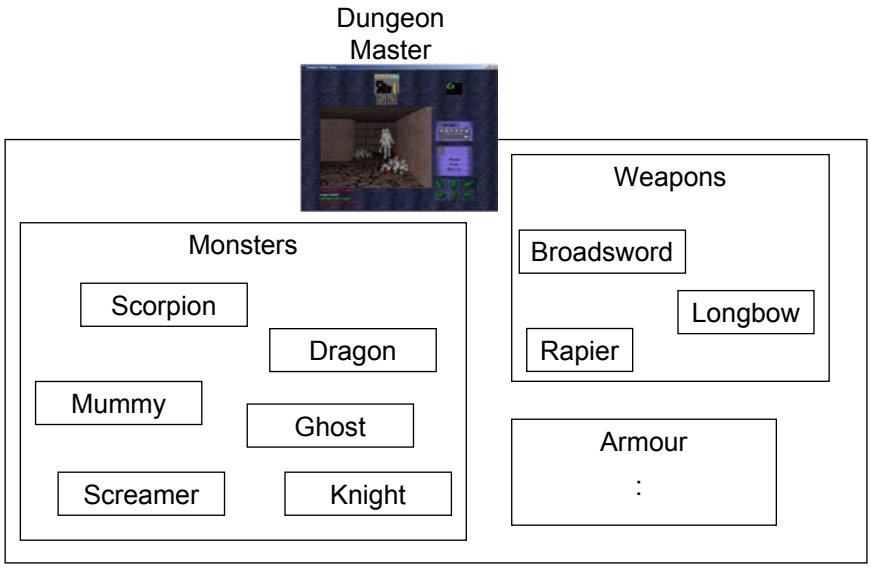
- Finding the nouns
- Using CRC cards

Currently used techniques

- Apply object decomposition
- Examine similar systems

This list is by no means complete but provides a starting point for students who are new to the Object-Oriented approach

Finding The Nouns



Finding The Nouns:Approach

1. Obtain a description of the problem to be solved
2. Identify the nouns in the document in order to look for candidate classes
3. Identity the verbs in the document in order look for potential methods

Finding Nouns: Discussion

Advantage

- Requires no training, easy to apply

Drawback

- There is not always a direct mapping between nouns and classes

Using CRC (Class-Responsibility-Collaboration) Cards

CollectionManager	
Responsibilities	Collaborations
Display collection	Book
Add a new book	
Remove a book	
Update book info.	

CollectionManager
List of books in the collection

Using CRC Cards: Approach

1. Obtain a description of the system to be modeled
2. On the front of the CRC card (cardboard or sticky note) state the name of the class and list it's responsibilities and collaborations
3. On the back of the card list the attributes of the class
4. Pass the completed deck of cards to experienced developers to look for holes in the deck

Using CRC Cards: Discussion

Advantage:

- Relatively simply to learn and apply
- Having the cards examined by experts may result in fewer gaps than just finding the nouns

Disadvantage:

- More useful for defining rather than identifying classes (should already have a list of classes)

Combining Techniques

1) Use the “find the nouns” approach to derive a list of candidate classes

→

2) Analyze this list by employing the CRC technique

Applying Object Decomposition

Dungeon Master

Dungeon

Dungeon level

Dungeon walls

Applying Object Decomposition: Approach

1. Obtain a description of the system to be modeled
2. Find the aggregate categories of classes
3. Decompose the categories in order to identify their components
4. Continue the process of decomposition until you reach the bottom level classes

Applying Object Decomposition: Discussion

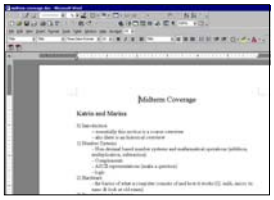
Advantages

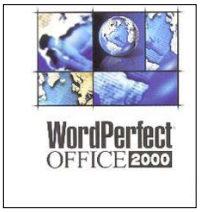
- A natural approach for dealing with aggregates

Disadvantages

- Not all classes are components of an aggregate
- More useful for describing how classes are related rather than indicating how to derive a list of classes

Examining Similar Systems





Examining Similar Systems: Approach

It can be done in a number of ways:

1. Look at how similar systems were implemented (whole systems)
2. Examine existing code libraries for reusable code (portions)
3. Using personal experience

Some Principles Of Good Design

Avoid going "method mad"
Keep an eye on your parameter lists
Avoid real values when an integer will do
Minimize modifying immutable objects
Be cautious in the use of references
Consider where you declare local variables

This list was partially derived from "Effective Java" by Joshua Bloch and is by no means complete. It is meant only as a starting point to get students thinking more about why a practice may be regarded as "good" or "bad" style.

Avoid Going Method Mad

There should be a reason for each method
Creating too many methods makes a class difficult to understand, use and maintain
A good approach is to check for redundancies

Keep An Eye On Your Parameter Lists

Avoid long parameter lists

•Rule of thumb: Three is the maximum

Avoid distinguishing overloaded methods solely by the order of the parameters

Avoid Real Values When An Integer Will Do

```
double db = 1.03 - 0.42;
if (db == 0.61)
   System.out.println("Sixty one cents");
System.out.println(db);
```

Minimize Modifying Immutable Objects

Immutable objects

Once instantiated they cannot change

e.g., String s = "hello";

```
s = s + " there";
```


Minimize Modifying Immutable Objects (2)

Substitute Immutable objects with mutable ones

e.g.,

```
Class StringBuffer
{
  public StringBuffer (String str);
  public StringBuffer append (String str);
    :       :          :       :

}
```

For more information about this class

http://java.sun.com/j2se/1.4/docs/api/java/lang/StringBuffer.html

Minimize Modifying Immutable Objects (3)

```
class StringExample
{
    public static void main (String [] argv)
    {
       String s = "0";
        for (int i = 1; i < 10000; i++)
          s = s + i;
    }
}
```

```
class StringBufferExample
{
    public static void main (String [] argv)
    {
       StringBuffer s = new StringBuffer("0");
       for (int i = 1; i < 10000 i++)
          s = s.append(i);;
    }
}
```


Be Cautious In The Use Of References

Similar to global variables:

```
program globalExample (output);
var
  i : integer;

procedure proc;
begin
  for i:= 1 to 100 do;
end;

begin
  i := 10;
  proc;
end.
```

Be Cautious In The Use Of References (2)

```
class Foo
{
   private int num;
   public int getNum () {return num;}
   public void setNum (int no) {num = no;}
}
```


Be Cautious In The Use Of References (3)

```
class Driver
{
   public static void main (String [] argv)
   {
      Foo f1, f2;
      f1 = new Foo ();
      f1.setNum(1);

      f2 = f1;
      f2.setNum(2);

      System.out.println(f1.getNum());
      System.out.println(f2.getNum());

   }
}
```

Consider Where You Declare Local Variables

First Approach: Declare all local variables at the beginning of a method:

```
void methodName (..)
{
      int num;
      char ch;
        :

}
```

Advantage:

- Putting all variable declarations in one place makes them easy to find

Consider Where You Declare Local Variables (2)

Second Approach: declare local variables only as they are needed

```
void methodName (..)
{
      int num;
      num = 10;
        :
      char ch;
      ch = 'a';

}
```

Advantage:

- For long methods it can be hard to remember the declaration if all variables are declared at the beginning
- Reducing the scope of a variable may reduce logic errors

Common Reasons Given For Adopting The Object-Oriented Approach

It's the latest thing, isn't it?

I read about it in Business Week

My boss's boss read about it in Business Week

I think our competitors are using it

Three of us went to a talk by Bertrand Meyer and we're really enthused

Structured techniques don't work

Extracted from: http://www.elj.com/elj/v1/n2/mpj/

Benefits Of Object-Orientation

May be more intuitive for some types of applications

Provides mechanisms for more robust programs

- Fewer bugs
- Allow for more code reuse

Easier to maintain and modify programs

Drawbacks Of Object-Orientation

Programs tend to be larger

Typically a steeper learning curve than with procedural languages

Summary

What are some traditional and modern approaches to Object-Oriented design

- Finding the nouns
- Using CRC cards
- Applying object decomposition
- Examining similar systems

A sample list of good design principles in Java

- Avoid going "method mad"
- Keep an eye on your parameter lists
- Avoid real values when an integer will do
- Minimize modifying immutable objects
- Be cautious in the use of references
- Consider where you declare local variables

What are some of the benefits and drawbacks of the Object-Oriented approach