OO Concepts

Background

- Software is hard to write
  - Large software projects are especially hard.
  - Many software projects are cancelled after they have consumed millions of money.
- Want techniques that will help people write large programs successfully.
  - OO is a successful process.
    - » (Software is still hard to write!)

Background 2

- As software is developed, it often has to change.
  - OO software is relatively good at being changed without messing up the rest of the program
    - If it uses data encapsulation and data hiding.

OO concepts

- Data Encapsulation
  - Data hiding
- Inheritance
- Software Reuse
- Polymorphism

Data Encapsulation

- Keep all data close to the scene of the action.
  - Each object should look after itself as much as possible.

Example

- Stock program:
  - Problem
    - » Keep track of stock.
    - » Before computers, a company would have a stock warehouse
      - When fetching an item from stock, if the quantity left was below the reorder level, they would order the "reorder quantity"
      - Noone would know how much stock was in the warehouse

Stock class

```
public class Stock
{
   String code;
   int quantity;

   public Stock(String c, int q)
   {
      code = c;
      quantity = q;
   }
}
```

Stock array

```
Stock inventory = {
        new Stock("100100", 10),
        new Stock("100101", 100),
        new Stock("100102", 5),
};

// Check how much inventory of an item: "100101"

public int getQuantity(String code)
{
   for(int i = 0; i < inventory.length; i++)
      if(code == inventory[i].code)
         return inventory[i].quantity;
   return -1; // code doesn't exist
}
```

Code has to change

```
// Some items are perishable => need to store
//  the date they are entered into the system

// Where to store the date

// In the Stock Object class

// Where to calculate if it is in date or not

// In the object class!
```

An objects internals are private

- The object can change internally without the user of the object having to change code.

Stock class

```
public class Stock
{
   String code;
   int quantity;
   Date expiry;

   public Stock(String c, int q, Date exp)
   {
      code = c;
      quantity = q;
      expiry = exp;
   }
}
```

Where to do the work

- All the things related to the object should happen in the object
- E.g. Stock should have all the information related to its use.
- This will help decide what methods the Stock item would have

Data Encapsulation

- The Stock class should have a public interface:
  - Constructor:
    - » public Stock(String code, int quantity, Date expiry)
  - int getQuantity()
  - void addQuantity()
  - int numDaysLeft() // before expiry
- The numDaysLeft() method will calculate how many days before expiry

Data Encapsulation 2

- Anything not part of the public interface should be made private.
  - Private fields and methods are used to make the object work internally. They can only be accessed by the class itself.

Software and Changes

- Changing Software often breaks it
  - Good OO design can make it easier to change the software
- E.g. When designing the system initially, the stock is stored without regard to expiry dates.
  - If you change the stock object, it may break exisitng software.
  - Use inheritance

Stock class gives rise to Expires

```
public class Expires extends Stock
{
   Date expiry; // Expires inherits code etc from Stock

   public Expires(String c, int q, Date exp)
   {
      super(c, q); // Use the parent constructor
      expiry = exp;
   }

   public int getDaysLeft(Date today)
     // Not implemented, just expiry - today
}
```

Inheritance

- Now, you can still use the original Stock class while leaving the original code unchanged
  - This supports software reuse
- The new Expires object inherits the code and quantity fields from Stock. It also inherits and of the Stock methods.

Expires

```
// create a new Expires object like any other object

Expires item = new Expires("101", 99,
      new Date(10, 12, 2017));

// As before you can create an array of Expires items
```

Inheritance 2

- Inheritance terminology:
  - parent class, child class, subclass
- Can have multiple levels of inheritance
- Can override methods
  - A class inherits methods. If it wants, it can change some of the methods to do what it wants
    - » Override the parent's method.

Polymorphism

- Polymorphism can be difficult to explain. The best way is to see an example.

Examples

- Graphics Program
  - circles, rectangles, triangles are all objects
    - » They all know how to draw themselves, move themselves, expand.
    - » Easy to create an object oriented graphics program.

Shop Products

- Different products may differ in:
  - taxes, promotions, packaging requirements, lifetime (e.g. expiry)
  - Very easy to manage if the data is encapsulated with the product.
    - » Polymorphism means that the different products can still be treated as a group

Object Oriented concepts

- Data Encapsulation
  - Data hiding
- Inheritance
- Software Reuse
- Polymorphism