CSCI-2320
Object-Oriented Programming (OOP)

Mohammad T. Irfan

Imperative vs. object-oriented paradigms
Imperative vs. object-oriented

- Imperative
  - Procedural decomposition
  - Procedures are all powerful
  - Data is helpless, at the mercy of procedures

- Object-oriented (OO)
  - Data-centric: data governs the decomposition
  - Classes - templates/patterns, abstracts away data
  - Objects - actual things/instantiations of classes

- Advantages of OO paradigm
  - Collaboration
  - Debugging
  - Reuse

Roadmap

- OOP principles (current slides)
- OOP paradigm: Ruby (next topic after this)
  - Learn the basics of Ruby
  - Investigate Ruby’s object-oriented design principles
  - Ruby on Rails for web programming
OOP Principles
Examples: Java

OOP principles

1. Encapsulation
2. Inheritance
3. Polymorphism
4. Abstraction

These principles typically interact with one another.
1. Encapsulation

- “to enclose in or as if in a capsule” (Mish, 1988)
- Data abstraction
  - “List” - hides how it’s implemented; how to append to a list, etc.
  - Hide internal representation of data by providing methods (e.g., getter, setter, and other methods that manipulate the data)
  - Allow for multiple levels of visibility: from not hiding at all (public) to hiding (private)
- Benefits: decoupling data and functionality, protection of data, hiding unnecessary details of representation

Encapsulation example: Book class

- Want a class for representing certain information about a book
  - Note: each object is one single book
  - Multiple objects → many books
1. What are the attributes or properties of a book?
2. What are the actions or behaviors that you can apply on book data?

- Encapsulation: bind the above two together
public class Book {
    private String title;
    private int pubYear; // publication year
    static int count; // how many book objects?

    Book(String _title, int _pubYear) // constructor
    {
        title = _title;
        pubYear = _pubYear;
        ++count;
    }

    Book(String title) // another constructor: overloading
    {
        setTitle(title);
        setPubYear(-1);
        ++count;
    }

    String getTitle()
    {
        return title;
    }
}

int getPubYear()
{
    return pubYear;
}

void setTitle(String title)
{
    // this.title is instance var, title is parameter
    this.title = title;
}

void setPubYear(int pubYear)
{
    // this.pubYear is instance var, pubYear is parameter
    this.pubYear = pubYear;
}

static void showCount()
{
    System.out.println("# of objects = " + count);
}
public static void main(String[] args) {
    Book book1 = new Book("A Tale of Two Cities", 1859);
    Book book2 = new Book("Moby Dick", 1851);
    Book book3 = new Book("Unknown");
    System.out.println(book1.getTitle()); // Output: A Tale of Two Cities
    System.out.println(book2.getTitle()); // Output: Moby Dick
    System.out.println(book3.getPublishYear()); // Output: -1
    Book.showCount(); // Output: count = 3
}

Encapsulation question

- Build on the Book class to include author names. How would you represent multiple authors?

2. Inheritance

- Allows one class to "inherit" the methods (functionalities) and attributes (variables) of another class
- **Subclass** (AKA derived class)
  - **extends** (or inherits)
- **Superclass** (AKA base class)
- Java's keyword: **extends**
Inheritance in picture

- Hierarchical organization

Subclass of subclass variables and methods

Subclass variables and methods

Superclass variables and methods

Inheritance in Java

Example of multiple superclasses:

Subclass
Multiple superclasses

Example of multiple subclasses:

Multiple subclasses
Superclass

Chain of inheritance
```java
// Ref: Java the Complete Reference
// Superclass
public class PlainBox {

    private double width;
    private double height;
    private double depth;

    // constructor
    PlainBox(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }

    // compute and return volume
    double getVolume() {
        return width * height * depth;
    }
}

// Here, PlainBox is extended to include weight.
class WeightedBox extends PlainBox {

    private double weight; // weight of box

    // constructor for WeightedBox
    WeightedBox(double w, double h, double d, double m) {
        //super(...) must be the first line to call Superclass constructor
        //unless superclass has a "default constructor" (no parameter)
        super(w, h, d);
        weight = m;
    }

    double getWeight() {
        return weight;
    }
}
```
Chain of inheritance  
(Multilevel inheritance)  

- New class for shipping a box, inherits WeightedBox

```java
38     class Shipping extends WeightedBox {
39         private double unitCost;
40         Shipping(double w, double h, double d, double m, double c) {
41             super(w, h, d, m);
42             unitCost = c;
43         }
44         double getTotalCost() {
45             return getVolume() * getWeight() * unitCost;
46         }
47     }
```

Demo

```java
51     class Demo {
52         public static void main(String args[]) {
53             // Super class object (not mandatory, just for demo)
54             PlainBox mybox1 = new PlainBox(w: 10, h: 20, d: 15);
55             // Subclass object of the previous subclass
56             WeightedBox mybox2 = new WeightedBox(w: 2, h: 3, d: 4, m: 5.5);
57             // Subclass object of the previous subclass
58             Shipping parcel = new Shipping(w: 5, h: 10, d: 20, m: 15, c: 0.01);
59             System.out.println("Volume of mybox1 is " + mybox1.getVolume());
60             System.out.println("Volume of mybox2 is " + mybox2.getVolume());
61             System.out.println("Weight of mybox2 is " + mybox2.getWeight());
62             System.out.println("Total shipping cost is $" + parcel.getTotalCost());
63         }
```
3. Polymorphism

- The ability of an object to take many forms
  - Compare with method overloading
- This is achieved via inheritance
  - Note the inter-relation
- Superclass can refer to subclass object
- Concept: method overriding

Method overriding

- Subclass re-defines a superclass method with the same method signature
- Next few slides
  - PlainBox class models a simple 3D box
  - WeightedBox class extends PlainBox class
    - Adds weight variable
    - Overrides the multiply method
  - BoxDemo class gives a demo
Superclass' multiply method is hidden from the subclass unless the subclass explicitly calls it using super.

```
// Class for a simple box
class PlainBox {
    private double width;
    private double height;
    private double depth;

    // constructor
    PlainBox(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }

    // multiply all dimensions (= volume)
    double multiply() {
        return width * height * depth;
    }
}

// Here, PlainBox is extended to include weight.
class WeightedBox extends PlainBox {
    private double weight; // weight of box

    // constructor for WeightedBox
    WeightedBox(double w, double h, double d, double m) {
        // super(...) to call superclass constructor
        super(w,h,d);
        weight = m;
    }

    // Method overriding
    double multiply() // multiply all dimensions & weight
    {
        return super.multiply() * weight; // new use of super
    }
}
```
4. Abstraction

- Generalization
  - “Process abstraction”

- Examples
  - C (not OOP): qsort works with different data types
  - C++: STL
  - Java: Abstract class allows generalization by hiding implementation details
  - Daily example: driving a car (hiding implementation details of pressing on gas pedal)
Abstract class in Java

- Give high-level ideas while hiding implementation details
- Use: manage complexity
- Next few slides
  - Abstract class Shape outlines a geometric shape
    - Which shape?
    - getArea(): Area depends on shape!
  - Subclasses of Shape: defines the getArea() method
    - Rectangle
    - Triangle

Cannot create any object of abstract class!

//Basic geometric shape class
abstract class Shape
{
    String name; //name of the shape
double[] dims;

    //Constructor will only be used by subclasses
    Shape(String name, double[] dims)
    {
        this.name = name;
        this.dims = dims;
    }

    String getName()
    {
        return name;
    }

    abstract double getArea(); //not defined here
```java
// Simple rectangle

class Rectangle extends Shape {

    Rectangle(double w, double h) {
        super(name: "Rectangle", new double[]{w, h});
    }

    double getArea() {
        return dims[0]*dims[1];
    }
}
```

```java
class Triangle extends Shape {

    Triangle(double s1, double s2, double s3) {
        super(name: "Triangle", new double[]{s1, s2, s3});
    }

    double getArea() {
        double peri = (dims[0]+dims[1]+dims[2])/2;
        return Math.sqrt(peri * (peri-dims[0]) * (peri-dims[1]) * (peri-dims[2]));
    }
}
```
Encapsulation vs abstraction vs information hiding

- Debates on orthogonality of concepts
- Roughly—data abstraction (capsule) vs process abstraction (generalization)
- [http://www.tonymarston.co.uk/php-mysql/abstraction.txt](http://www.tonymarston.co.uk/php-mysql/abstraction.txt)
Snapshot of debate

“Encapsulation or equivalently information hiding refers to the practice of including within an object everything it needs, and furthermore doing this in such a way that no other object need ever be aware of this internal structure.”
-- [Ian Graham, 1991]

“If encapsulation was “the same thing as information hiding,” then one might make the argument that “everything that was encapsulated was also hidden.” This is obviously not true. ... It is indeed true that encapsulation mechanisms such as classes allow some information to be hidden. However, these same encapsulation mechanisms also allow some information to be visible. Some even allow varying degrees of visibility, e.g., C++’s public, protected, and private members.”
-- Tony Marston

Encapsulation: correct definitions?

“Encapsulation is used as a generic term for techniques which realize data abstraction. Encapsulation therefore implies the provision of mechanisms to support both modularity and information hiding. There is therefore a one to one correspondence in this case between the technique of encapsulation and the principle of data abstraction.”
-- [Blair et al, 1991]
Abstraction: correct definition?

"Abstraction is generally defined as 'the process of formulating generalised concepts by extracting common qualities from specific examples.'"

-- [Blair et al, 1991]