Object Oriented Design and Analysis
CPE 372

Lecture 2

Encapsulation and Information Hiding

Dr. Sally E. Goldin
Department of Computer Engineering
King Mongkut's University of Technology Thonburi
Bangkok, Thailand

Review – Why use OOD?

- Manage complexity
- Enhance maintainability
- Enforce modularity

With OO, modules are not optional

Break down a system into categories of objects (*classes*)

Classes interact in formally-defined ways (by calling each other's *methods*)

Two other desirable qualities

Low coupling

- Minimize dependencies between modules so changes in one module do not require changes in other modules
- Facilitate debugging: if module A does not depend on module B, bugs in A cannot be caused by events in B.

Abstraction

- Describe modules in terms of their information content and behavioral effects, not the details of their implementation
- Allows changes in implementation over time with minimal effects on other modules

Encapsulation (Information Hiding)

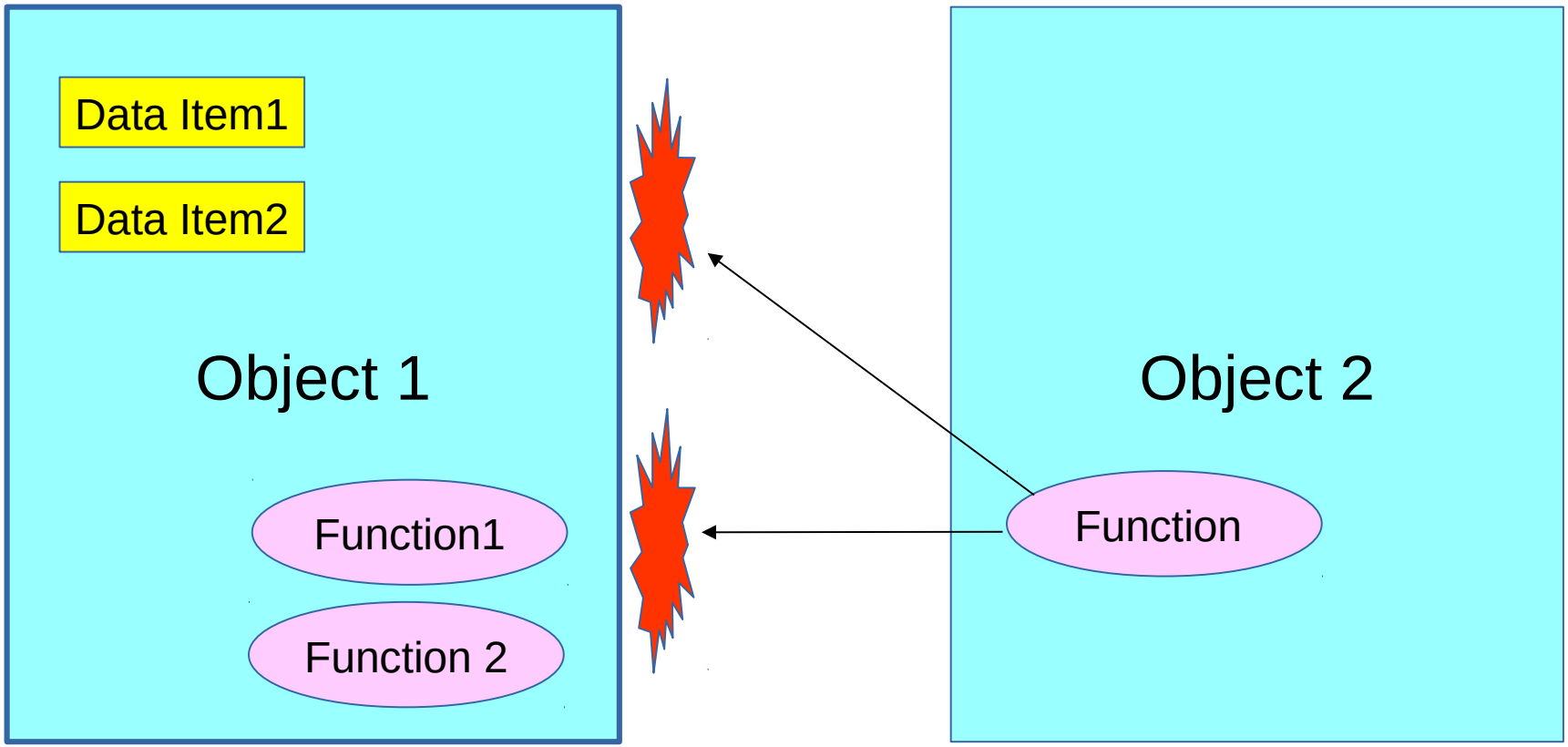
Allows OO design tools and languages to minimize coupling and maximize abstraction.

Medicine Capsules

Space Capsule

A capsule has a clearly defined boundary between “inside” and “outside”

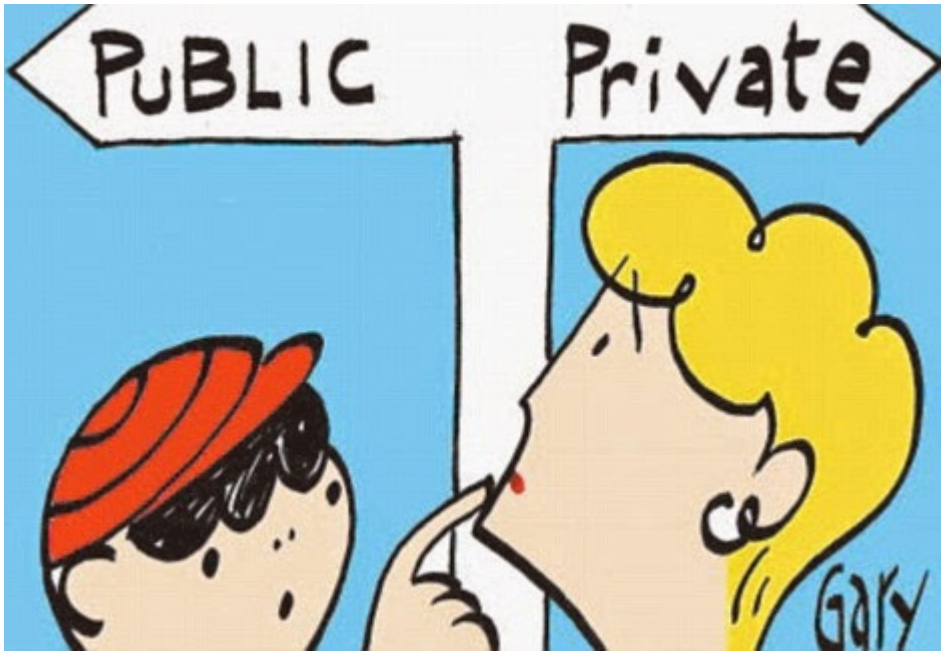
Object 2 cannot penetrate Object 1's boundary



The designer of Object 1 decides what is visible and accessible, what is not

Encapsulation in Java

Java enforces encapsulation by associating each member data item and method with a ***visibility specifier***



Visibility in Java

public members and methods are visible and accessible to any other class or instance

private members and methods are not visible or accessible outside the current instance

protected members and methods are “in between” - visible and accessible within *subclasses* of the current class or to any class in the same *package* (explicit grouping of classes)

We will talk about packages and subclasses next week.

OO Design Rule of Thumb: 1

Use the *most restrictive* visibility modifier that will allow your program to do what is necessary

OO Design Rule of Thumb: 2

Explicitly specify visibility for all members and methods. Do not use the default.

In Java, if you do not specify a visibility modifier, the data or method will be visible and accessible to all classes in the package.

This is almost never what you want!

Example: Triangle.java

Triangle.java is a simple class that defines a triangle, that is, a geometric feature defined by 3 points.

This class has the following member data:

```
/** X coordinates of three points defining the triangle */
private int xcoord[] = new int[3];

/** Y coordinates of three points defining the triangle */
private int ycoord[] = new int[3];
```

Because these two arrays are private, no other class can view or modify them.

When we try, we get a compilation error.*

**** Uncomment lines in TriangleTester.java to see this effect***

Methods in Triangle.java

Most of the methods in the Triangle class are public.

However, there is one “helper” method used in the *calcPerimeter* method:

```
/**
 * Calculate the length of one side of the triangle.
 * This is private method used by calcPerimeter and calcArea.
 * @param  which    1,2 or 3, for which side
 * @return length of side, or -1 if 'which' is out of range.
 */
private double calcLength(int which)
```

This method cannot be called outside of the class.

If you try, you will get a compile error.*

**** Uncomment lines in TriangleTester.java to see this effect***

Controlling Access to Private Data

What if another class needs to use private data?

Consider *Square.java*

```
/* A square can be defined by an upper left corner point plus
 * the length of a side. However, for drawing purposes it is
 * more convenient to have four corner points in order.
 */
/** X coordinates of four points */
private int xcoord[] = new int[4];

/** Y coordinates of four points */
private int ycoord[] = new int[4];

/** also keep the length of one side */
private int oneside = 0;
```

FigureViewer.java needs these coordinates in order to draw a square.

Getter Methods (“Getters”)

The standard solution is to provide public functions that will return the data upon request

```
/**
   * Return one of the X coordinates - 0 is upper left
   * @param  which    0,1, 2, or 3 specifying which vertex we want
   * @return vertex X coordinate, or -1 if argument is wrong
   */
  public int getX(int which)
  {
      int coordVal = -1;
      if ((which >= 0) && (which < 4))
          coordVal = xcoord[which];
      return coordVal;
  }
```

See SquareTesterGraphics.java and FigureViewer.java to see this being used.

Setter Methods (“Setters”)

What if another class needs to *change* the values of some private data? In this case, the class that owns the data should supply “setter” methods.

These methods can enforce restrictions on the data to make sure it remains valid.

```
/**
 * Set upper left X. This effectively moves the square to the left or right
 * The setter re-initializes the other coordinates in the array.
 * @param   X    New upper left X coordinate
 */
 public void setX(int upperLeftX)
 {
    xcoord[0] = upperLeftX;
    xcoord[1] = upperLeftX + oneside;  // upper right
    xcoord[2] = upperLeftX + oneside;  // lower right
    xcoord[3] = upperLeftX;
 }
```

Uncomment lines in SquareTesterGraphics.java to see this being used.

What if we make the data public?

It might seem like a lot of extra work to create getter and setter methods.

You might be tempted to just make the data be public, so other classes can set it directly.

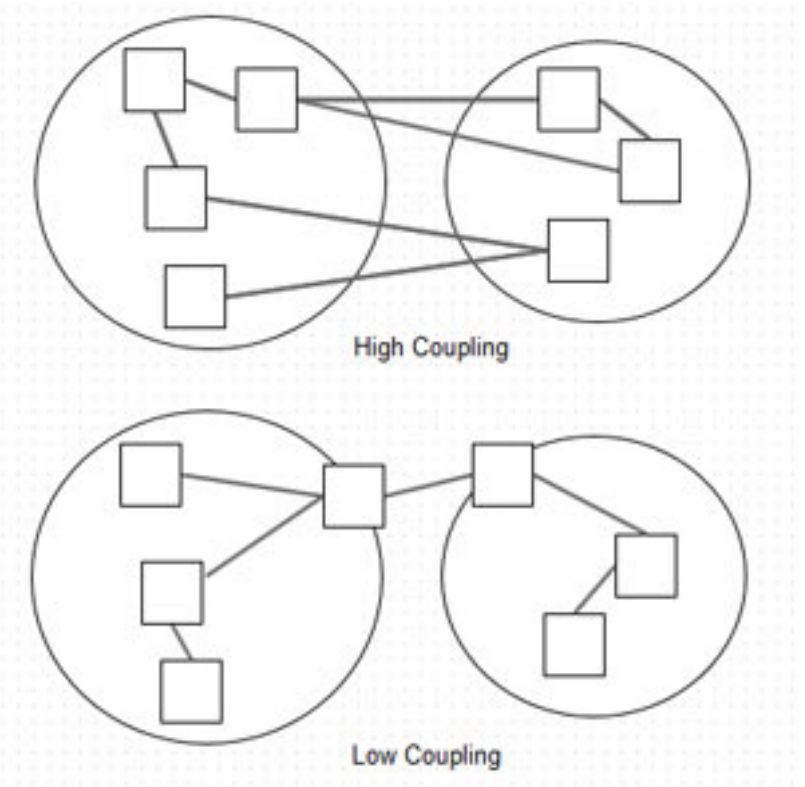
To see the bad results, check out the demo classes *BadSquare.java* and *BadSquareTesterGraphics.java*!

Too much coupling?

Even if ***Square*** keeps its data private, there are still strong dependencies between ***Square*** and ***FigureViewer***'s *drawSquare* method.

If we decided to change the way the ***Square*** class stored its data, we would need to change the getters/setters, the *drawSquare* method, or both.

It would be better if ***Square*** knew how to "draw itself" rather than relying on another class.



Static Member Data

Usually member data items apply on a per-instance basis.

Each instance of the class (for example, each ***Square***) has its own copy of the data items, which will very likely have different values.

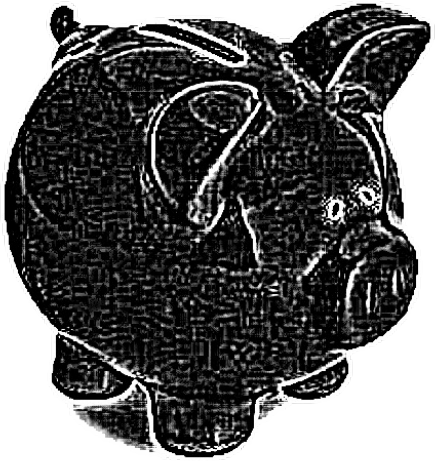
Sometimes, however, you want member variables that apply to the class as a whole; where every instance will see the same value.

In this case, you declare the member variable as ***static***.

```
public class FigureViewer extends JFrame
                                    implements ActionListener
{
   /** so we can count and label triangles */
   private static int counter = 0;
.....
```

Static versus Instance Member Data

bankCount: 3
totalMoney: 0



PiggyBank class

```
private String owner;
private Color color;
private int currentAmount = 0;

private static int bankCount = 0;
private static int totalMoney = 0;

public void insertMoney(int amount);
```

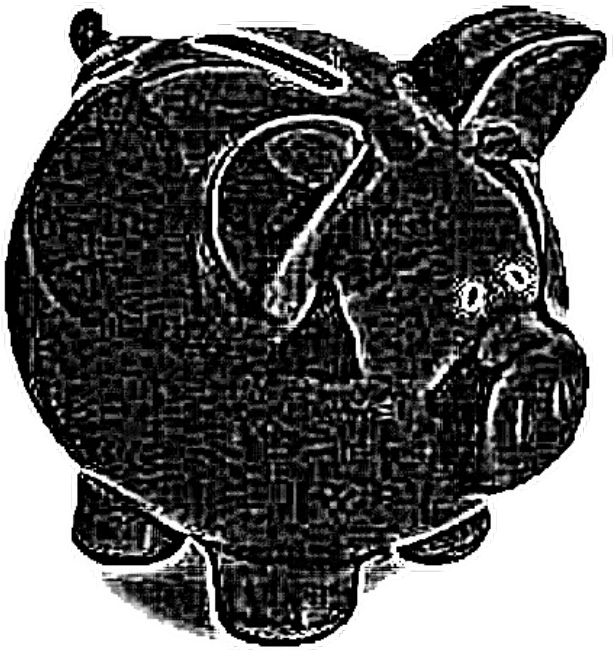
PiggyBank instances

owner: “Sally”
color: pink
currentAmount: 0

owner: “Kurt”
color: yellow
currentAmount: 0

owner: “Fred”
color: turquoise
currentAmount: 0

Updating instance and static variables

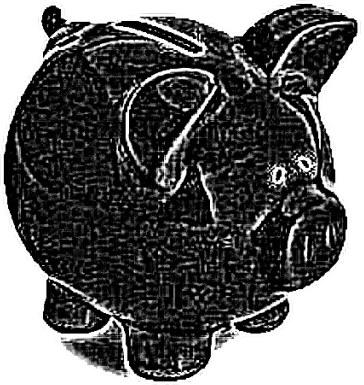


***PiggyBank* class**

```
/** add money to a particular bank.
 * Also update the grand total
 * @param amount  How much to add.
 */
public void insertMoney(int amount)
{
   currentAmount += amount;
   totalMoney += amount;
}
```

Updating instance and static variables (2)

bankCount: 3
totalMoney: 0



```
pinkBank.insertMoney(120);
```

bankCount: 3
totalMoney: **120**

owner: “Sally”
color: pink
currentAmount: **120**

```
greenBank.insertMoney(200);
```

bankCount: 3
totalMoney: **320**

owner: “Fred”
color: turquoise
currentAmount: **200**

When should you use static member data?

- Counting instances
- Combining information across instances
- Creating a collection to manage all instances
  - For example, you could save all the squares created in a static list so you could draw them all, or any one of them
- Maintaining state information
  - For example, in a graphics editor, is the user drawing, or editing?
- Storing constants (may be static and public)
  - For example, a URL that is needed in many classes

(These are just some common cases)

Static Methods

The `insertMoney` method is an instance method.

To call it, you need a variable that represents a specific, particular ***PiggyBank***.

You can create static methods also:

```
public static void summarizeBanks()
{
    System.out.println("Currently there are " + bankCount +
        " piggy banks");
    System.out.println("Grand total in all banks is " +
        totalMoney + " baht");
}
```

You call this method using the class name, rather than an instance variable:

```
PiggyBank.summarizeBanks();
```

The Big Question

What's the minimum information

class A needs to know about class B?

Assignments

1. Do Exercise 2. This is due Monday at 17:00.
2. By next Thursday (31 Jan), choose a topic for the term project.

   You can find a list of suggested topics here:

   http://windu.cpe.kmutt.ac.th/cpe372/ProjectTopics2019.html

   Your team can also create your own topic (but I must approve it).

   Fill in your team name next to your chosen topic on the following sheet. One team per topic!

https://docs.google.com/spreadsheets/d/1vBwEylgyHOfr3iuN1mlbEHcFT1LBiudbV7rwrOwl7js/edit?usp=sharing