COMP 110-001
Information Hiding and Encapsulation

Yi Hong

June 03, 2015

Review of Pass-By-Value

- What is the output?

```
public void swap(Student s1, Student s2) {
    Student s3 = s1;
    s1 = s2;
    s2 = s3;
}

Student berkeley = new Student();      berkeley.setYear(2);
Student brett = new Student();         brett.setYear(3);

swap(berkeley, brett);
System.out.println(berkeley.year);
```

Review of Pass-By-Value

- What is the output?

```
public void swapYear(Student s1, Student s2) {
        int year = s1.year;
        s1.year = s2.year;
        s2.year = year;
}

Student berkeley = new Student();        berkeley.setYear(2);
Student brett = new Student();           brett.setYear(3);

swapYear(berkeley, brett);
System.out.println(berkeley.year);
```

Today

- Public / private
- Information hiding and encapsulation

public/private Modifier

- public void setMajor()
- public int classYear;

- public: there is no restriction on how you can use the method or instance variable
- Any class can use a public class, method, or instance variable

public/private Modifier

- private void setMajor()
- private int classYear;

- private: can not directly use the method or instance variable's name outside the class

Example

```
public class Student
{
    public int classYear;
    private String major;
}

Student jack = new Student();

jack.classYear = 1;

jack.major = “Computer Science”;
```

OK,
classYear is *public*

Error!!!
major is *private*

More About private

- Hides instance variables and methods inside the class/object.
- The private variables and methods are still there, holding data for the object.
- Invisible to external users of the class
  - Users cannot access private class members directly
- **Information hiding**

Instance Variables Should Be private

- Private instance variables are accessible by name only within their own class
- Force users of the class to access instance variables only through methods
  - Gives you control of how programmers use your class
- Why is this important?

Example: Rectangle

```
public class Rectangle
{
    public int width;
    public int height;
    public int area;

    public void setDimensions(
        int newWidth, int newHeight)
    {
        width = newWidth;
        height = newHeight;
        area = width * height;
    }

    public int getArea()
    {
        return area;
    }
}
```

```
Rectangle box = new Rectangle();
box.setDimensions(10, 5);
System.out.println(box.getArea());

// Output: 50

box.width = 6;
System.out.println(box.getArea());

// Output: 50, but wrong answer!
```

Instance Variables Should Be Private

- Public instance variables can lead to the corruption of an object's data, inconsistent data within an object
- Private instance variables enable the class to restrict how they are accessed or changed
- Always make instance variables private

Accessors and Mutators

- How do you access private instance variables?
- Accessor methods (a.k.a. get methods, getters)
  - A public method that allows you to look at data in an instance variable
  - Typically begin with *get*
- Mutator methods (a.k.a. set methods, setters)
  - A public method that allows you to change data in an instance variable
  - Typically begin with *set*

Example: Student

```
public class Student
{
    private String name;
    private int age;

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

    public void setAge(int studentAge)
    {
        age = studentAge;
    }

    public String getName()
    {
        return name;
    }

    public int getAge()
    {
        return age;
    }
}
```

Mutators

Accessors

Okay, But Why Making Methods private?

- Helping methods that will only be used from inside a class should be private
  - External users have no need to call these methods

- **Encapsulation**
  - **Groups instance variables and methods into a class**
  - **Hides implementation details, and separates the what from the how**

Example: Driving a Car

- Accelerate with the accelerator pedal
- Decelerate with the brake pedal
- Steer with the steering wheel
- Does not matter if:
  - You are driving a gasoline engine car or a hybrid engine car
  - You have a 4-cylinder engine or a 6-cylinder engine
- You still drive the same way

Encapsulation

- The *interface* is the same
- The underlying *implementation* may be different
- A programmer who uses a method (interface) should need only know what the method does, not how it does it

Encapsulation in Classes

- A *class interface* tells programmers all they need to know to use the class in a program
  - A class interface describes the class's public view
- The *implementation* of a class consists of the private elements of the class definition, hidden from public view
  - private instance variables and constants
  - private methods
  - bodies of public methods

Example: Two Implementations of Rectangle

```
public class Rectangle
{
    private int width;
    private int height;
    private int area;

    public void setDimensions(
        int newWidth,
        int newHeight)
    {
        width = newWidth;
        height = newHeight;
        area = width * height;
    }

    public int getArea()
    {
        return area;
    }
}
```

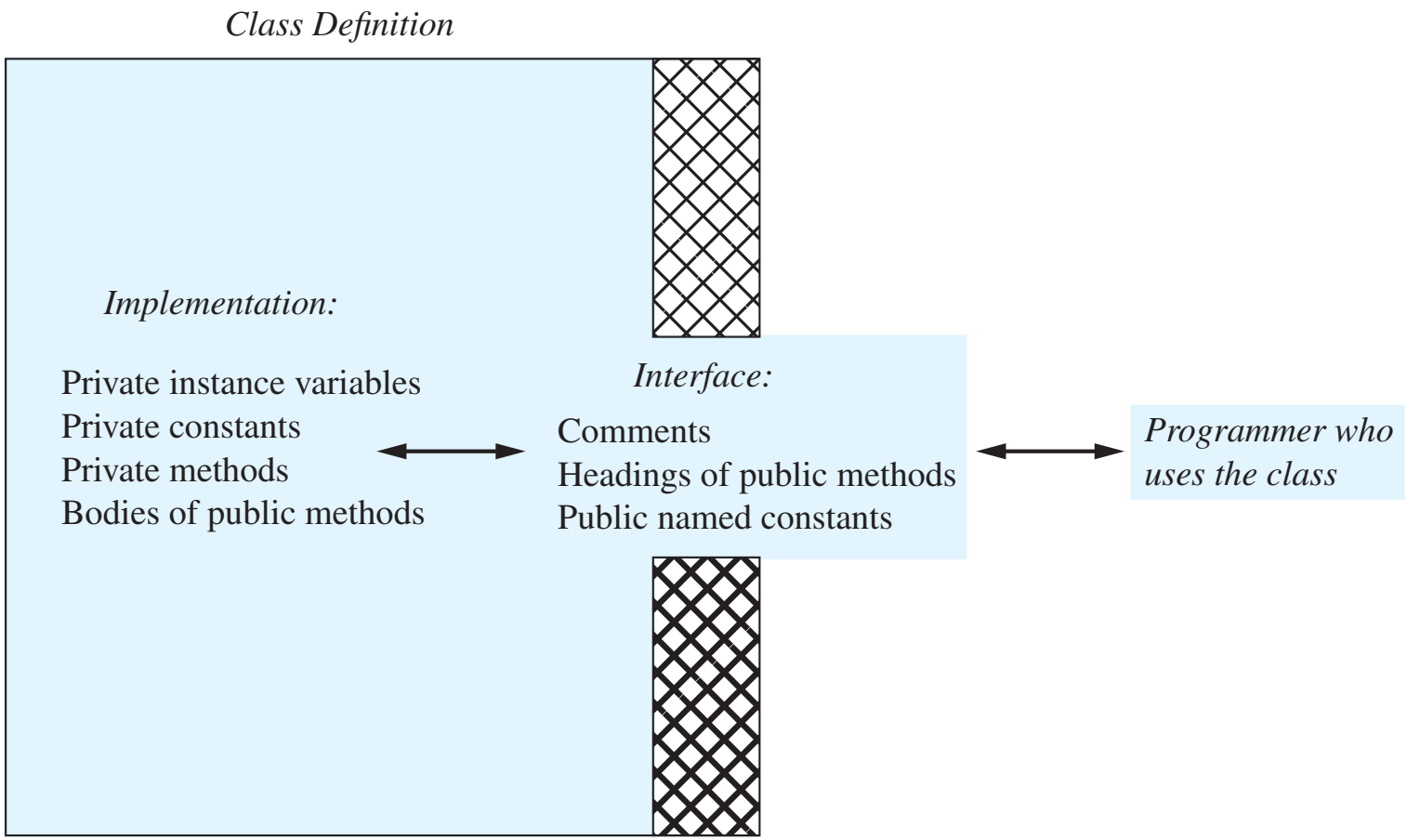
```
public class Rectangle
{
    private int width;
    private int height;

    public void setDimensions(
        int newWidth,
        int newHeight)
    {
        width = newWidth;
        height = newHeight;
    }

    public int getArea()
    {
        return width * height;
    }
}
```

Encapsulation

- Implementation should not affect behavior described by interface
  - Two classes can have the same behavior but different implementations

A Well-Encapsulated Class Definition

- Imagine a wall between interface and implementation



Comments Before Method's Definition

- Precondition: states a method's requirements
  - Everything that needs to be true before the method is invoked
- Postcondition: states a method's effect
  - Tells what will be true after the method is executed in a situation in which the precondition holds
  - For a method that returns a value, the postcondition will include a description of the value returned by the method

Encapsulation Guidelines

- Comments before class definition that describes how the programmer should think about the class data and methods
- Instance variables are *private*
- Provide *public* accessor and mutator methods
- Pre and post comments before methods
- Make any helping methods *private*
- Write comments within the class definition to describe implementation details
  - A good rule: /* * */ style for class interface comments, and the // style for implementation comments

Summary of Encapsulation

- The process of hiding all the details of how a piece of software works and describing only enough about the software to enable a programmer to use it
- Data and actions are combined into a single item, a class object that hides the details of the implementation

Next Class

- Constructors and static methods