Information Hiding & Encapsulation

COMP 110
Summer II 2012

Peter Lincoln
7/10/2012
Outline

- Public/Private
- Documentation++
Classes Review

- **Class**: A definition of a kind of object
  - Specifies kinds of data and methods

- **Object**: An instance of a class
  - Contains data (stored in instance variables)

- **Methods**: An operation
  - Perform actions and return a value (or not)
public/private modifier

- Examples
  - public void setMajor(...)
  - public String major;

- public: there is no restriction how you can use the method or instance variable
public/private modifier

Examples

- `private void setMajor(...)`
- `private String major;`

`private:` can not directly use the method or instance variable name outside the class

- You can use it anywhere inside the class
Example: public/private

Class
- **public class** Student
  ```java
  {
      public int classYear;
      private String major;
  }
  ```

Code Elsewhere
- 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
  - External users cannot access private class members directly

- This is Information hiding
Instance variables should be private

- 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

```java
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!

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;
    }
}

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

// Output: 50

box.width = 6; // Error!
Accessors & Mutators

- How do you access private instance variables?

- Accessor methods (a.k.a. get methods, getters)
  - Allow you to look at data in private instance variables

- Mutator methods (a.k.a. set methods, setters)
  - Allow you to change data in private instance variables
```java
public class Student {
    private String name;
    private int age;

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

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

    public void setAge(int studentAge) {
        age = studentAge;
    }
}
```
private Methods?

- Helper methods that will only be used from inside a class should be private
  - External users have no need to call these methods
  - Example: swap(...) or interchange(...) 

- This is Encapsulation
Encapsulation

- The interface to the user is the same
- The underlying implementation may be different
Encapsulation in Classes

- A **class interface** tells programmers all they need to know to use the class in a program.

- The **implementation** of a class consists of the private elements of the class definition:
  - private instance variables and constants
  - private methods
  - bodies of public methods
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
Imagine a wall between interface and implementation

**Implementation:**
- Private instance variables
- Private constants
- Private methods
- Bodies of public methods

**Interface:**
- Comments
- Headings of public methods
- Public named constants

Programmer who uses the class
Declaration Syntax Summary

- **Instance Variable**
  - `AccessModifier Type variableName;`

- **Method Heading**
  - `AccessModifier TypeOrVoid methodName(Parameter_List)`
  - Where `Parameter_List` is
    - Empty
    - `Type1 paramName1, Type2 paramName2, ...`

- Where `Access_Modifier` is
  - `public`: can be used anywhere
  - `private`: can be used only inside the class
    - Recommended for all instance variables
Why Use Private? Summary

- Protect against unexpected changes in meaning
  - E.g. Rectangle (width, height, area)

- Enable changing the internals of the class while keeping the public interface the same
Comments

- From the previous lecture:

```java
/**
 * Represents the name of this student.
 */
public String name;

/**
 * Returns the name of this student.
 * @return the name of this student
 */
public String getName()
{
    return name;
}
```
Comments: Pre-/Postcondition

- **Precondition**
  - Everything that needs to be true before calling the method

- **Postcondition**
  - Describes the effect of the method call
  - Assumes that the precondition was true
Using Pre- and Postcondition

- Can omit for obvious methods
  - get... (accessor), set...(mutator)

- All other methods need pre- and postconditions

- If unsure, then write pre and post!
Example: Pre- and Postconditions

/**
 * Writes this student's data to the screen.
 *
 * Precondition: The instance variables of this Student have values
 *
 * Postcondition: The data stored in this Student have been written to the screen.
 */

public void writeOutput()
Example: Pre- and Postconditions

/**
* Adds a class grade to the the students overall grade
* Precondition: This Student's credit and gpa sums have valid values.
* Postcondition: This Student's credit and gpa sums include the specified credits and gpa values.
* @param credits the number of credits the class was worth
* @param gpa the grade point value earned for the class
*/
public void addGrade(int credits, double gpa)
General Guidelines

- **Comments**
  - Put comments before the class definition (this is your header)
  - Use /** ... */ comments for class interface comments
    - Include pre- and postconditions
  - Use // and /* ... */ comments for implementation comments

- **public/private**
  - Make instance variables private
  - Make helper methods private
  - Provide public accessor and mutator methods
Questions?
Logistics

Next:

- In-Class Exercise