COMP 110
Information hiding and encapsulation

Darrell Bethea
May 24, 2011
Announcements

- Midterm Thursday
  - SN014

- Program 3 due 5/31
  - Don’t forget about JarChecker.java
Questions?
Today in COMP 110

- More about methods
- public/private
- Information hiding and encapsulation
Methods with parameters

- Parameters are used to hold the value that you pass to the method
- Parameters can be used as (local) variables inside the method

```java
public int square(int number)
{
    return number * number;
}
```

Parameters go inside parentheses of method header
Methods with multiple parameters

- Multiple parameters separated by commas

```java
public double getTotal(double price, double tax) {
    return price + price * tax;
}
```
Method parameters and arguments

public class SalesComputer
{
    public double getTotal(double price, double tax)
    {
        return price + price * tax;
    }
    // ...

    SalesComputer sc = new SalesComputer();
    double total = sc.getTotal(“19.99”, Color.RED);
    double total = sc.getTotal(19.99);
    double total = sc.getTotal(19.99, 0.065);
    int price = 50;
    total = sc.getTotal(price, 0.065); // Automatic typecasting
A method body can call another method
  ◦ Done the same way:
    objectName.method();

If calling a method in the same class, do not need objectName.:
  ◦ method();

Alternatively, use the this keyword
  ◦ this.method();
Within a class definition, \textit{this} is a name for the receiving object
\begin{itemize}
  \item \texttt{this.age}
  \item \texttt{this.major}
  \item \texttt{this.getAge()}
\end{itemize}

Frequently omitted, but understood to be there

See book for details
public/private modifier

- public void setMajor()
- public int classYear;

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

- private void setMajor()
- private int classYear;

private: cannot directly use the method or instance variable’s name outside the class
```java
public class Student {
    public int classYear;
    private String major;
}

Student jack = new Student();
jack.classYear = 1;
jack.major = "Computer Science";
```

**Example**

Student jack = new Student();
jack.classYear = 1;  // OK, classYear is public
jack.major = "Computer Science";  // 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

- 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;
    }
}
```

```java
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!
```
Accessors and 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 void setName(String studentName) {
        name = studentName;
    }

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

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}
```
Okay, but why make methods \textit{private}? 

- Helper methods that will only be used from inside a class should be \textit{private} 
  - External users have no need to call these methods

- Encapsulation
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
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
Example: two implementations of Rectangle

```java
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;
    }
}
```

```java
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
Class Definition

Implementation:
- Private instance variables
- Private constants
- Private Methods
- Bodies of all methods
- Method definitions

Interface:
- Comments
- Headings of public methods
- Public defined constants

Well encapsulated
Imagine a wall between interface and implementation
Comments

- **Precondition** – everything that needs to be true before method

- **Postcondition** – describes effect of method call
When to use Pre and Post

- You can omit for obvious methods
  - get (accessor), set (mutator) …
- All other methods need pre and post conditions
- If you are unsure, write pre and post!
Guidelines

- Comments before class definition (this is your header)
- Instance variables are private
- Provide public accessor and mutator methods
- Pre and post comments before methods
- Make helping methods private
- /**/ for user–interface comments and // for implementation comments
Variables are local to methods

Instance variables are Global for all methods in a class