

Encapsulation in Java



EECS2030: Advanced
Object Oriented Programming
Fall 2017

CHEN-WEI WANG

Encapsulation (1.1)

Consider the following problem:

- A person has a name, a *weight*, and a *height*.
- A person's weight may be in *kilograms* or *pounds*.
- A person's height may be in *meters* or *inches*.
- A person's BMI is calculated using their height in *meters* and weight in *kilograms*.

Consider a first solution:

```
class Person {  
    public String name;  
    public double weight; /* in kilograms */  
    public double height; /* in meters */  
    public double getBMI() { return weight / (height * height); }  
}
```

- Since both attributes `height` and `weight` are declared as *public*, we do not need the setter methods for them.

Encapsulation (1.2)

Say an application of the `Person` class *mistakenly* thinks that the height in inches and weight in pounds should be set:

```
1 class BMICalculator {
2     public static void main(String args[]) {
3         Person jim = new Person();
4         /* Jim's height and weight are 1.78 m and 85 kg */
5         jim.weight = 85 * 2.2;
6         jim.height = 1.78 * 39;
7         System.out.println(jim.getBMI());
8     } }
```

- **Line 7:** $\frac{85 \times 2.2}{(1.78 \times 39)^2} = 0.038$, rather than $\frac{85}{1.78^2} = 26.827!!!$
- **Solution:**
 - Disallow any application class of `Person` to directly assign to `weight` and `height`.
 - Provide proper setter methods as the only means for assigning these two attributes.

Encapsulation (2.1)

Now consider a better solution:

```
class Person {  
    public String name;  
    private double weight; /* in kilograms */  
    private double height; /* in meters */  
    public void setWeightInKilograms(double k) { weight = k; }  
    public void setWeightInPounds(double p) { weight = p / 2.2; }  
    public void setHeightInMeters(double m) { height = m; }  
    public void setHeightInInches(double i) { height = i / 39; }  
    public double getBMI() { return weight / (height * height); }  
}
```

Exercise: Modify the `Person` class so that `weight` is measured in pounds and `height` is measured in inches.

Encapsulation (2.2)

Now an application of the `Person` class may only set the `weight` and `height` of a person by calling the appropriate methods:

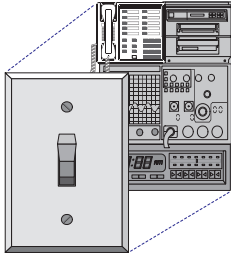
```
1 class BMICalculator {
2   public static void main(String args[]) {
3     Person jim = new Person();
4     /* Jim's height and weight are 1.78 m and 85 kg */
5     jim.setWeightInPounds(85 * 2.2);
6     jim.setHeightInInches(1.78 * 39);
7     System.out.println(jim.getBMI());
8   }
```

- Since both attributes `weight` and `height` in class `Person` are declared as `private`, it is disallowed in any other class (e.g., `BMICalculator`) to access them (e.g., `jim.weight`).
- **Line 7** now should return the correct BMI value.

Encapsulation (3.1)

- **Question**: What if in the `Person` class, we want the `weight` attribute to mean pounds and `height` to mean inches?
- **Hint**: Which classes will you have to change? `Person`? `BMICalculator`? Both?
- Modify the setter methods in `Person` accordingly. [Exercise!]
- No change is needed in the `BMICalculator`!
 - Since class `BMICalculator` was disallowed to access `weight` and `height`, as soon as the setter definitions are modified in `Person`, the calculation will still work!
- What we have achieved:
 - Implementation details in `Person` (i.e., `weight` and `height`) are *hidden* from all potential applications (e.g., `BMICalculator`).
 - When these implementation details are *changed* in `Person` (e.g., `weight` interpreted in pounds rather than in kilograms):
 - Only the `Person` class has to be *changed*.
 - All existing application classes can remain *unchanged*.

Encapsulation (3.2)



- A software component hides the internal details of its implementation, so that:
 - It has a *stable* interface;
 - Programmers of other components can *only depend on its public interface*, rather than writing code that depends on those *implementation decisions* ;
 - The component developer may change the implementation *without affecting* the code of any other components.
- In Java, we achieve this by
 - declaring attributes or helper methods as *private*;
 - providing *public* accessors or mutators.

Encapsulation (3.3)

- Follow this tutorial video:

`https:`

`//www.youtube.com/watch?v=d2Q-uasRmAU&index=1&list=PL5dxAmCmjv_492h1b0yizSyhC3ImEetLV`

- For complete details about controlling the access for attributes, refer to:

`https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html`

Index (1)

Encapsulation (1.1)

Encapsulation (1.2)

Encapsulation (2.1)

Encapsulation (2.2)

Encapsulation (3.1)

Encapsulation (3.2)

Encapsulation (3.3)