Encapsulation

Hsuan-Tien Lin

Deptartment of CSIE, NTU

OOP Class, March 18, 2013

Encapsulation: Key Point

- **separate implementation and use**: you implement the `Record` class, and other people (possibly you after two years) use it
- **don't trust other people**: silly mistakes can happen —force your user to use your code correctly
- **hide unnecessary details** (a.k.a. instance variables)
- **think about possible correct/incorrect use of your class**: check them in the methods

> as a designer, you should avoid giving the users of your code too much freedom to do bad and/or make bugs

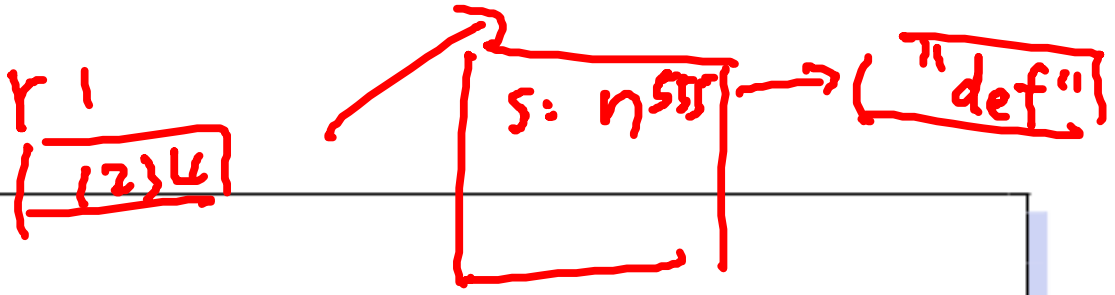
Hiding Variables from All Classes (1/3)

```
1  class Record{
2    private String encoded_password;
3  }
```

- `private`: hiding from all classes (except myself, of course)

```
1  class Record{
2    private String encoded_password;
3    public boolean compare_password(Record another_record){
4      return (
5        this.encoded_password ==
6        another_record.encoded_password
7      );//okay?
8    }
9  }
```

Hiding Variables from All Classes (2/3)




```
1  class Record{
2    private String name;
3    public String get_name(){
4      return name;
5    }
6    public String get_copied_name(){
7      return new String(name);
8    }
9  }
```

copy constructor

X r1.name.append("a");

✓ r1.get_name().append("a");

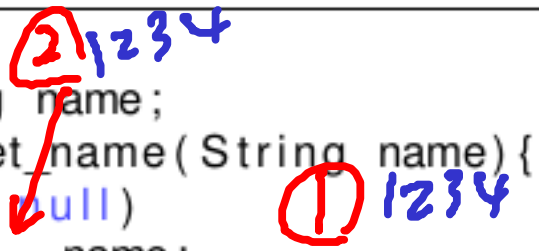
privacy leak

- `public`: accessible by all classes
- **accessor**: get the content of the instance

but, string no leak.

Hiding Variables from All Classes (3/3)

```
1  class Record{
2    private String name;
3    public void set_name(String name){
4      if (name != null)
5        this.name = name;
6    }
7  }
```



- **mutator**: check and set the instance variable to a value

Hiding Variables from All Classes: Key Point

private instance variables
public accessor/mutator instance methods

More on Hiding Details (1/3)

```
1  class Date{//implemented for your desktop
2    private int month, day;
3    public int get_month(){ return month; }
4    public int get_day(){ return day; }
5  }
6  class Date_TWO{//implemented on a small-memory machine
7    private short encoded_month_and_day;
8    public int get_month(){
9      return encoded_month_and_day / 100;
10   }
11   public int get_day(){
12     return encoded_month_and_day % 100;
13   }
14 }
```

- two implementations, same behavior—easy for users to switch on different machines
- trade-offs: memory usage, computation, etc.

More on Hiding Details (2/3)

```
1  class Distance{
2    private double mile;
3    public double get_mile(){
4      return mile;
5    }
6    public double get_km(){
7      return mile * 1.6;
8    }
9    public void set_by_km(double km){
10     this.mile = km / 1.6;
11   }
12   public void set_by_mile(double mile){
13     this.mile = mile;
14   }
15 }
```

- one storage, different information from different mutator/accessor

More on Hiding Details (3/3)

Some rules of thumb:

- make all instance variables private
- use mutators/accessors for safely manipulate the variables

Cons:

- accessing requires method calls, slower

Pros:

- less chance of misuse by other users
- decouple implementation and use
- flexibility

Design by Contract

- **interface** to others: methods opened as `public`, with usage shown in internal and external documents
- **invariant** behaviors: integrity maintained by making variables `private`, with proper constructors and methods

More on Hiding Details (Yet Another Case)

```
1  class Solver{
2    public void read_in_case(){}
3    public void compute_solution(){}
4    public void output_solution(){}
5    public void solve(){
6      read_in_case();
7      compute_solution();
8      output_solution();
9    }
10 }
```

- should the three utility functions be public?

More on Hiding Details: Key Point

hiding details: don't directly access internal stuff to gain flexibility and avoid misuse

Java Member Encapsulation (1/2)

```
1  class Distance{
2    private double mile;
3    public double get_mile(){
4      return mile;
5    }
6    private double get_ratio(){
7      return 1.6;
8    }
9    public double get_km(){
10     return mile * ratio;
11   }
12 }
```

- private: hidden, on variables and methods that you do not want anyone to see
- public: on variables and methods that you want everyone to see

Java Member Encapsulation (2/2)

```
1  class Demo{
2    private double mile;
3    default int a; //imagine, but not correct grammar
4    int b;
5    public double get_mile(){
6      return mile;
7    }
8    private double get_ratio(){
9      return 1.6;
10   }
11   double get_km(){
12     return mile * ratio;
13   }
14 }
15 class Another{
16   void lalala(){ int lulu = new Demo().b + 1; }
17 }
```

- default: classes in the same source file (et al.) can access it
- a “gray-area” usage

Java Member Encapsulation: Key Point

- public/private:
  the more common pair for OO programmers
- default:
  for laziness of beginners, or real-advanced use (later)