Encapsulation

Hsuan-Tien Lin

Deptartment of CSIE, NTU

OOP Class, March 8-9, 2010

Encapsulation (1/5)

```java
class Record{
  String name;
  String password;
}

public class RecordDemo{
  public static void main(String[] argv){
    Record r;
    String s, p;
    s = getLoginNameFromUser();
    r = getRecordFromFile(s);
    System.out.println(r.password);
    p = getPasswordFromUser();
    if (p.equals(r.password)){
      // ...
    }
  }
}
```

- if password not encoded, the SYSOP might easily get your password by `getRecordFromFile`

Encapsulation (2/5)

```java
class Record{
  String name;
  String encoded_password;
}

public class RecordDemo{
  public static void main(String[] argv){
    Record r;
    String s, p;
    s = getLoginNameFromUser();
    r = getRecordFromFile(s);
    p = getPasswordFromUser();
    if (YOUR_ENODING(p).equals(r.encoded_password)){
      // ...
    }
    //A new and careless programmer adds this line
    r.encoded_password = null;
  }
}
```

- even when password encoded, a careless programmer may make stupid bugs

Encapsulation (3/5)

```java
class Record{
  private String encoded_password;
  public String get_encoded_password(){
    return encoded_password;
  }
}
public class RecordDemo{
  public static void main(String[] argv){
    Record r;
    String s, p;
    s = getLoginNameFromUser();
    r = getRecordFromFile(s);
    p = getPasswordFromUser();
    if (YOUR_ENODING(p).equals(r.get_encoded_password())){
      // ...
    }
    //A new and careless programmer adds this line
    r.encoded_password = null; //won't work
  }
}
```

- what if you want to set a new password?

Encapsulation (4/5)

```
class Record{
  String name;
  private String encoded_password;
  public String get_encoded_password(){
    return encoded_password;
  }
  public void set_encoded_password(String raw_password){
    if (blahblah)
      encoded_password = YOUR_ENCODING(raw_password);
  }
}
```

- **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
- **hide unnecessary details** (a.k.a. instance variables)
- **think about possible correct/incorrect use of your class**: check them in the methods

Encapsulation (5/5)

```
public class RecordDemo{
  public static void main(String[] argv){
    Record r;
    String s, p;
    s = getLoginNameFromUser();
    r = getRecordFromFile(s);
    p = getPasswordFromUser();
    if (r.match_password(p)){
      //no need to show encoding to the outside
    }
    r.set_encoded_password(old_password, new_password);
    //don't want this to happen
    r.encoded_password = null;
  }
}
```

- freedom on making assignments: a potential hole to do bad and/or make bugs

Encapsulation: Key Point

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)

```java
class Record{
  private String encoded_password;
}
```

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

```java
class Record{
  private String encoded_password;
  public boolean compare_password(Record another_record){
    return (
      this.encoded_password ==
      another_record.encoded_password
    );//okay?
  }
}
```

Hiding Variables from All Classes (2/3)

```
class Record{
  private String name;
  public String get_name(){
    return name;
  }
  public String get_copied_name(){
    return new String(name);
  }
}
```

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

Hiding Variables from All Classes (3/3)

```java
class Record{
  private String name;
  public void set_name(String name){
    if (name != null)
      this.name = name;
  }
}
```

- **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)

```
class Date{ //implemented for your desktop
  private int month, day;
  public int get_month(){ return month; }
  public int get_day(){ return day; }
}
class Date_TWO{ //implemented on a small-memory machine
  private short encoded_month_and_day;
  public int get_month(){
    return encoded_month_and_day / 100;
  }
  public int get_day(){
    return encoded_month_and_day % 100;
  }
}
```

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

More on Hiding Details (2/3)

```java
class Distance{
  private double mile;
  public double get_mile(){
    return mile;
  }
  public double get_km(){
    return mile * 1.6;
  }
  public void set_by_km(double km){
    this.mile = km / 1.6;
  }
  public void set_by_mile(double mile){
    this.mile = mile;
  }
}
```

- 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
- flexibility

More on Hiding Details (Yet Another Case)

```
class Solver{
  public void read_in_case(){}
  public void compute_solution(){}
  public void output_solution(){}
  public void solve(){
    read_in_case();
    compute_solution();
    output_solution();
  }
}
```

- 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)

```
class Distance{
  private double mile;
  public double get_mile(){
    return mile;
  }
  private double get_ratio(){
    return 1.6;
  }
  public double get_km(){
    return mile * ~~ratio;~~
                  get_ratio();
  }
}
```

- 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)

```java
class Demo{
  private double mile;
  default int a; //imagine, but not correct grammar
  int b;
  public double get_mile(){
    return mile;
  }
  private double get_ratio(){
    return 1.6;
  }
  double get_km(){
    return mile * ratio;
  }
}
class Another{
  void lalala(){ int lulu = new Demo().b + 1; }
}
```

- 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)