Encapsulation

- Encapsulation
  - A Class/Bounded Context/Component/Library should be as shy as possible

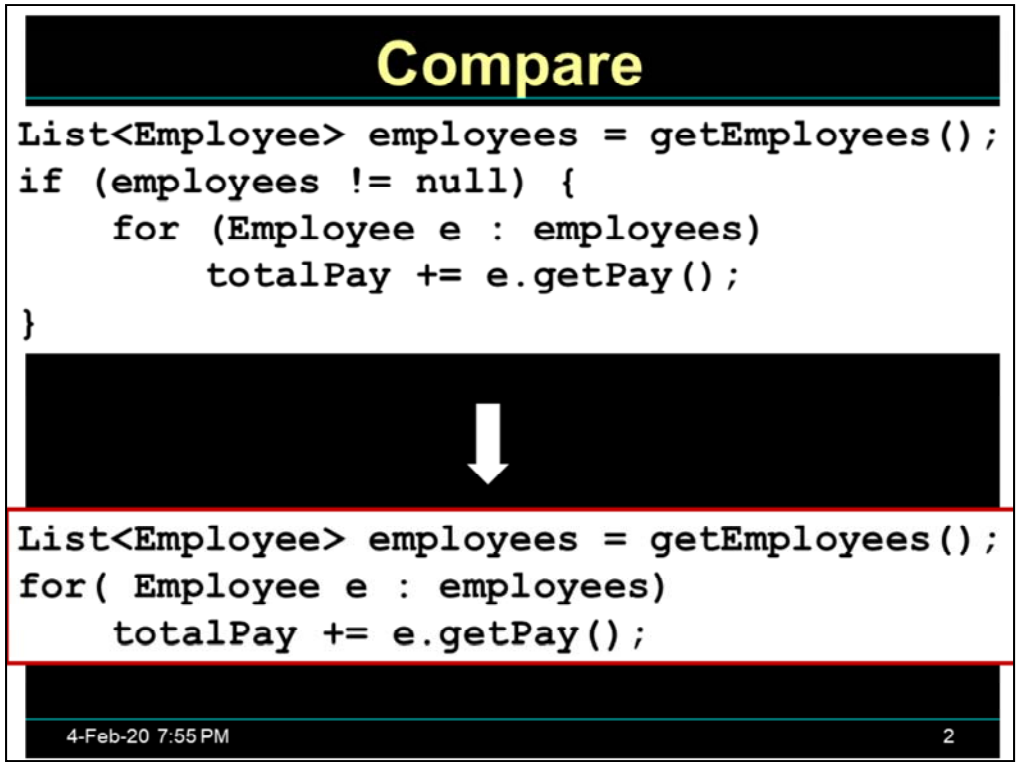
4-Feb-20 7:55 PM 1

It is about managing complexity

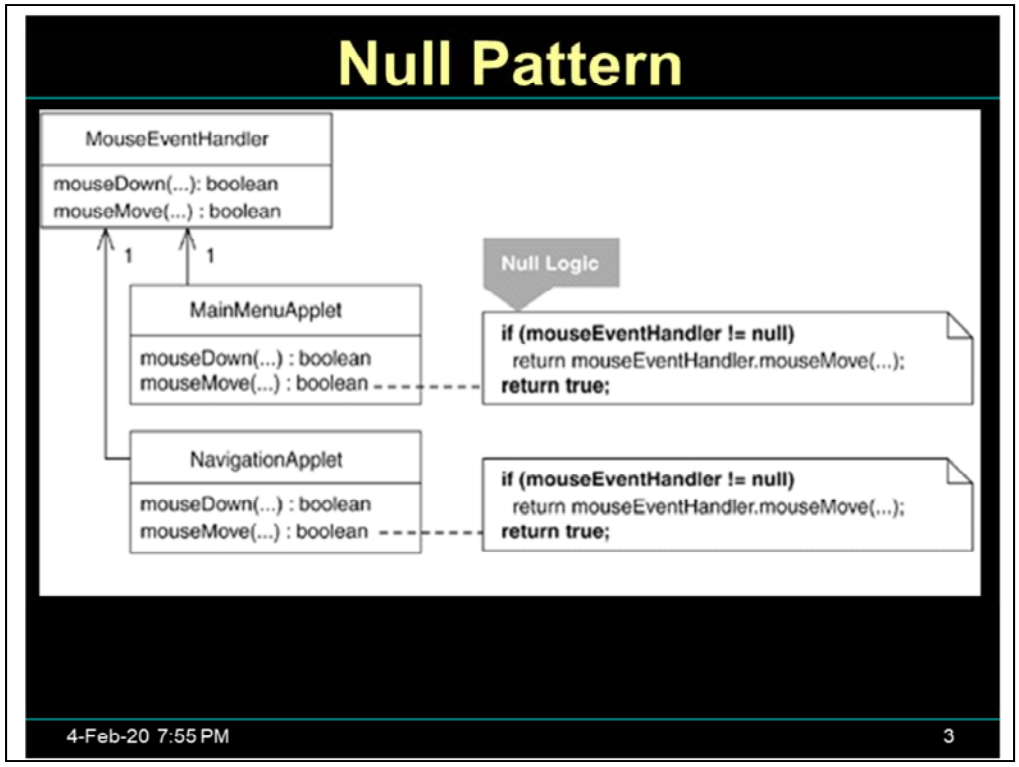
Put state information in the class that works on it. Collaborating classes are unaware of the internal state of this object.

Keep a variable/function private, if possible. Use minimum visibility.

Every module should have a secret. If it does not have a secret, why does it exist?



Java has Collections.emptyList() for this. It is immutable.



Null pattern

Avoid NullPointerException in code

Return “” instead of null for String class

Return an array with zero elements instead of null.

==========

Benefits and Liabilities

\+ Prevents null errors without duplicating null logic.

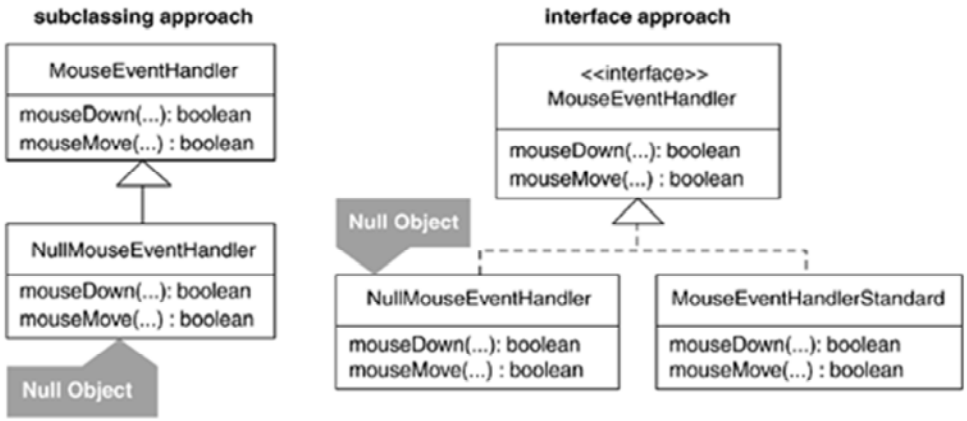
\+ Simplifies code by minimizing null tests.

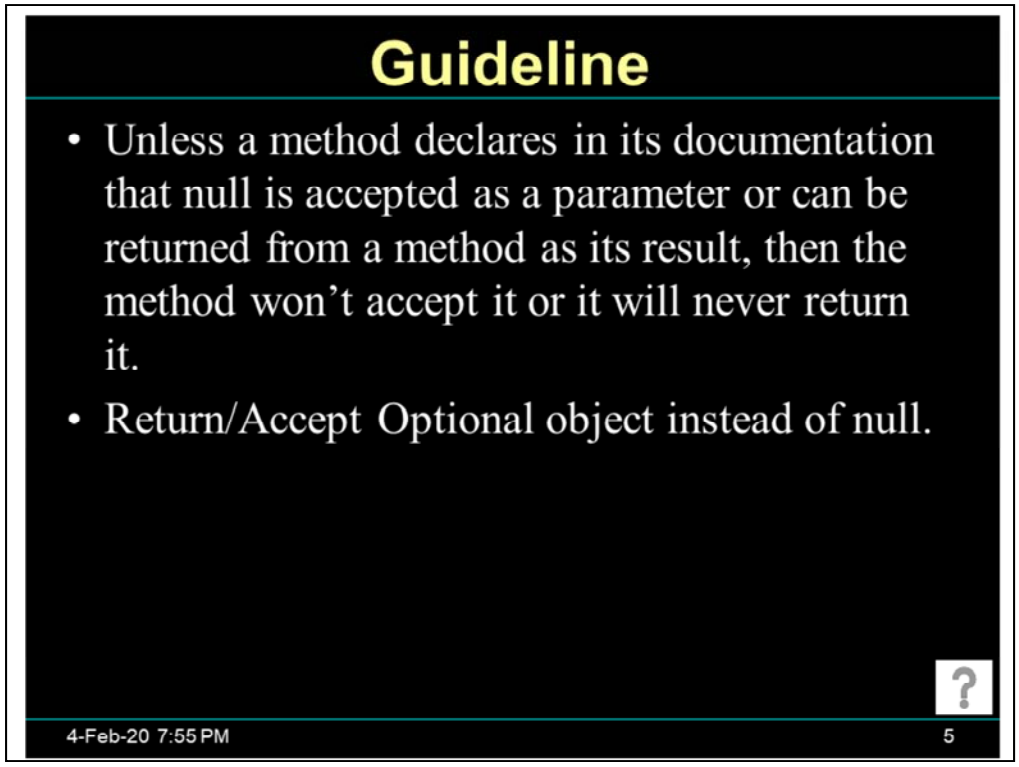
– Complicates a design when a system needs few null tests.

– Can yield redundant null tests if programmers are unaware of a Null Object implementation.

– Complicates maintenance. Null Objects that have a superclass must override all newly inherited public methods.

Null Object by Interface



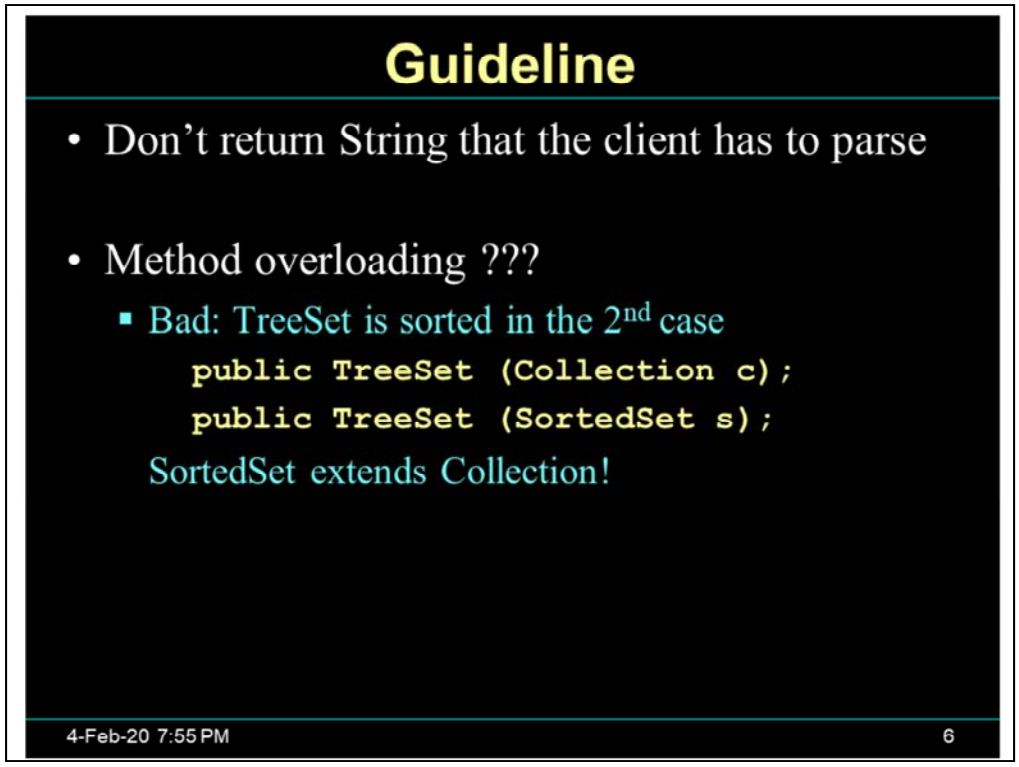


Q28Artists

Optional class is present in Java 8. If you are using older versions of Java, then it is also present in Guava library.

Optional has been coded for C# and stored in same directory other examples. It is also present is an open source library https://github.com/nlkl/Optional

Optional is present in C++17, not in older versions



Return of sting with multiple values is bad because

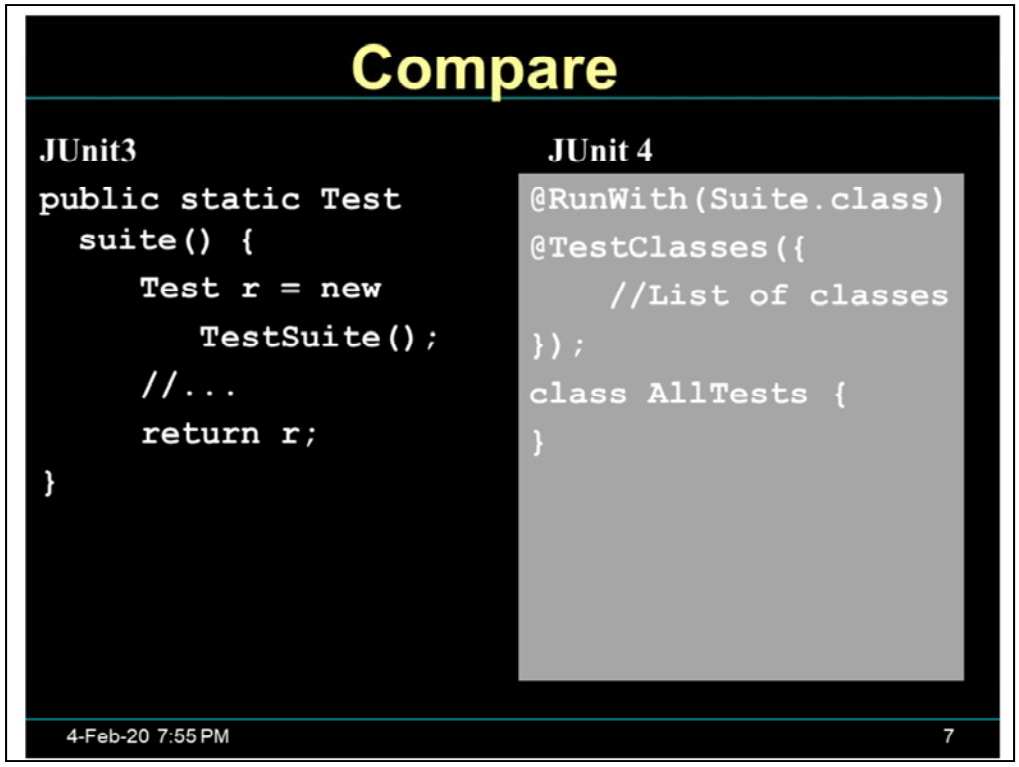
Future modifications will become difficult

Bad: In Java, printStackTrace of Throwable class

==========

Overloading - Avoid same name for multiple methods with same number of arguments

In C# and C++, operator overloading should be used only when it is clear. It should not be very frequent.



The code on the right

•Is flexible because any runner can be used

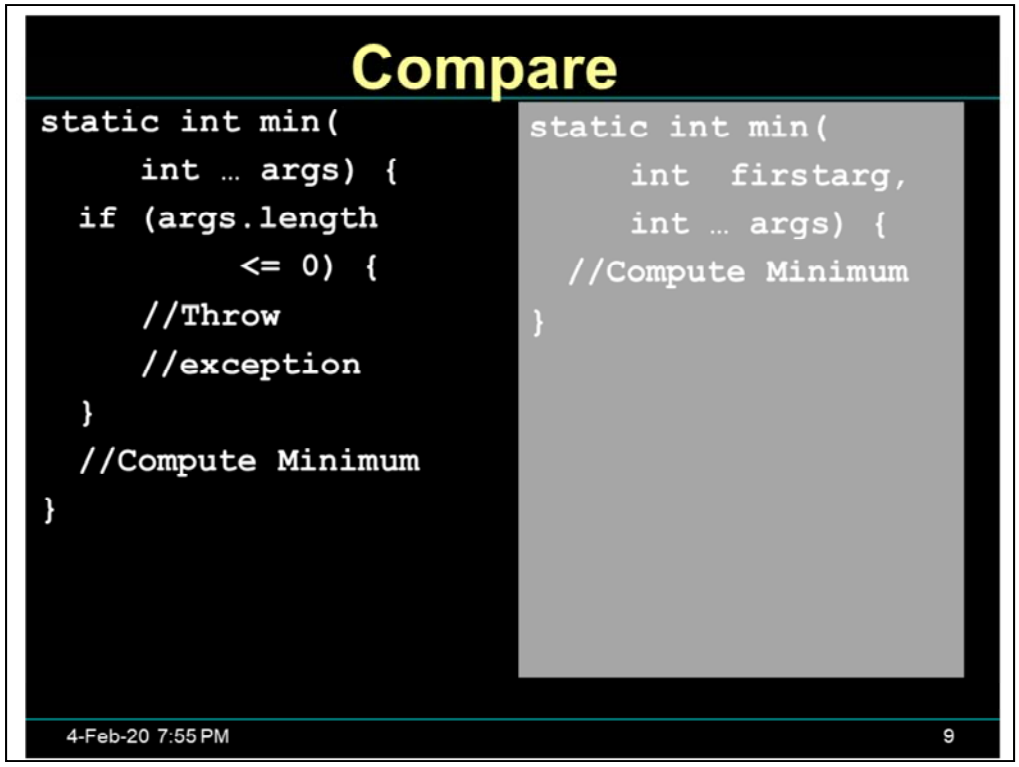
•Less prone to error, because the caller does not have to worry about exceptions

Improve

```
Collection<String> keys =
      new ArrayList<String>();
keys.add(key1);
keys.add(key2);
object.index(keys);
```

- Use varargs

```
object.index(key1, key2);
```



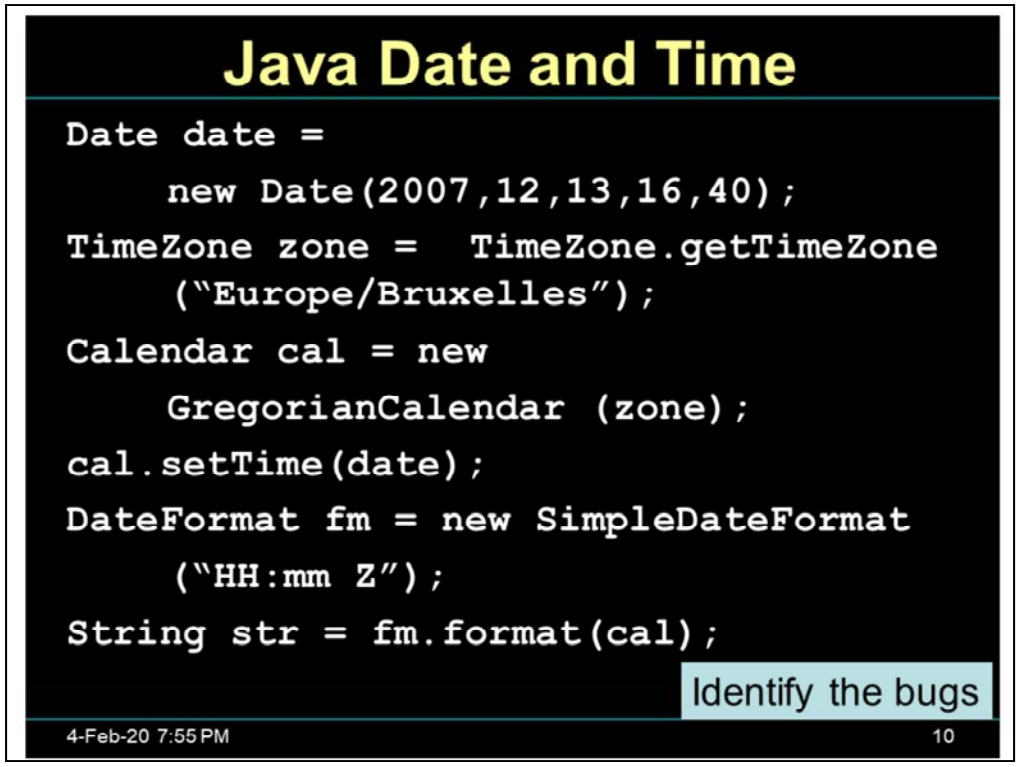
The code on the right

•Is flexible because any runner can be used

•Less prone to error, because the caller does not have to worry about exceptions

In C#, the syntax is

static int min(int firstarg, params int[] args)



Bug 1: Year should be 107, since base is 1900.

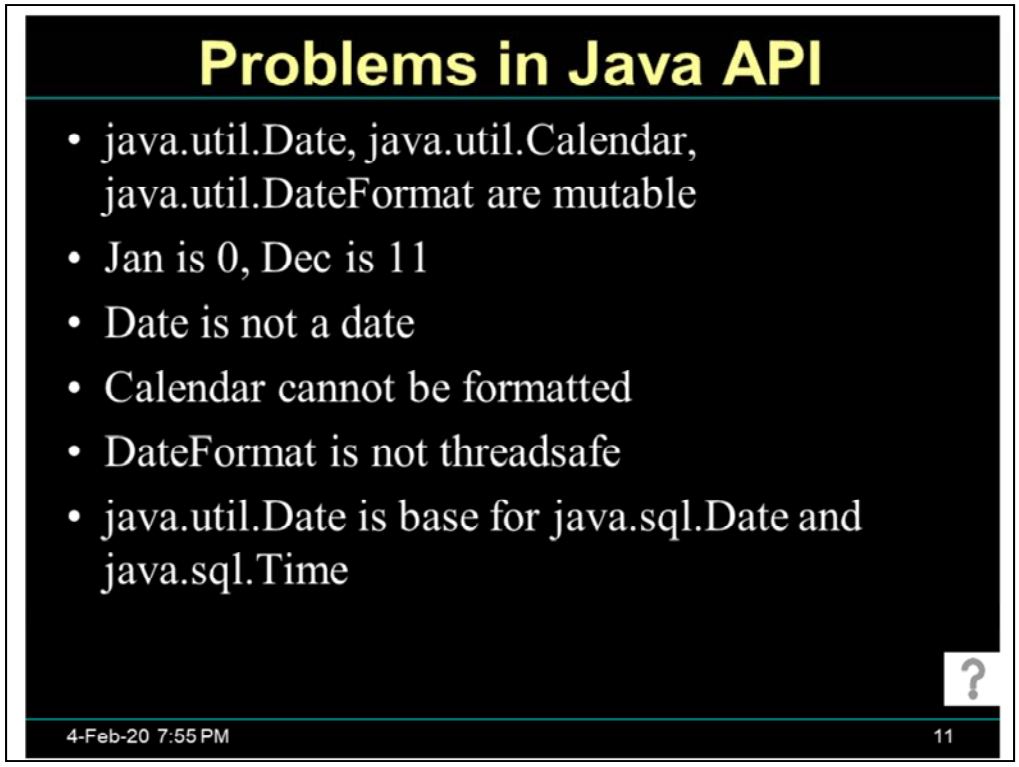
Bug 2: Month should be 11 instead of 12. Since Jan is 0.

Bug 3: “Europe/Bruxelles”. Bruzelles is capital of Brussels. It is capital of Belgium. Different people pronounce it different ways. Java returns GMT.

Bug 4: We are creating cal object with invalid or wrong value of date.

Not sure - Bug 5: fm.format gives runtime exception because it cannot format calendar. It can format only dates. So we need to call get “cal.getTime()” and pass the returned date to “fm.format”

Not sure - Bug 6: We have not set the timezone in DateFormat. It needs to be set before calling format.



Q53 – Piano

Q73 – Student, Teacher

Date is not a date because: It has time & It uses from 1900

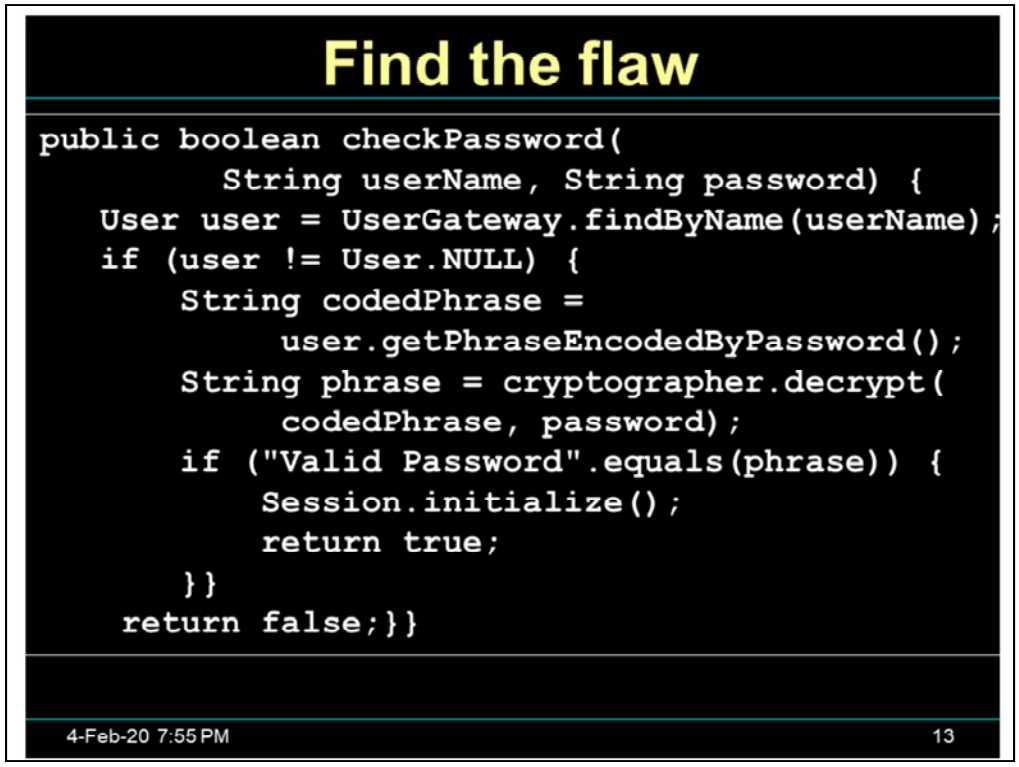
java.util.Date should not base class for java.sql.Date and Time because getYear on java.sql.Time throws an illegal argument exception.

Principle of Least Astonishment

- User of API should not be surprised by behavior
  - interrupted method in Thread class clears interrupted flag!

4-Feb-20 7:55 PM 12

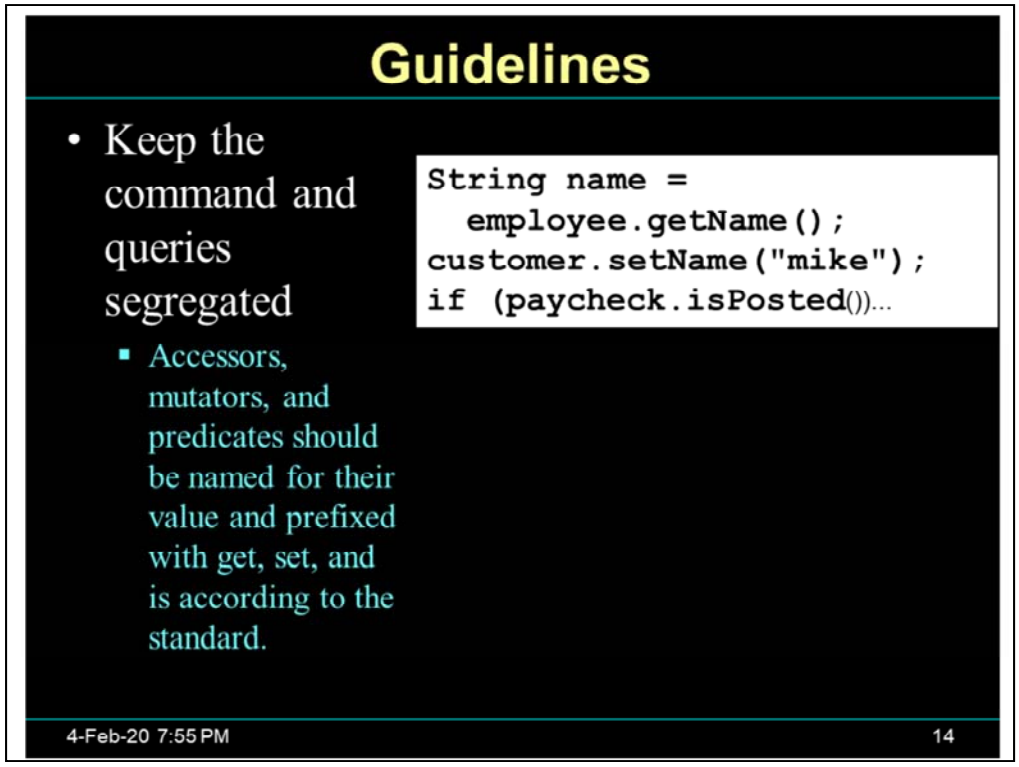
It should have been called clearInterruptedFlag



This function uses a standard algorithm to match a userName to a password. It returns true if they match and false if anything goes wrong. But it also has a side effect. Can you spot it?

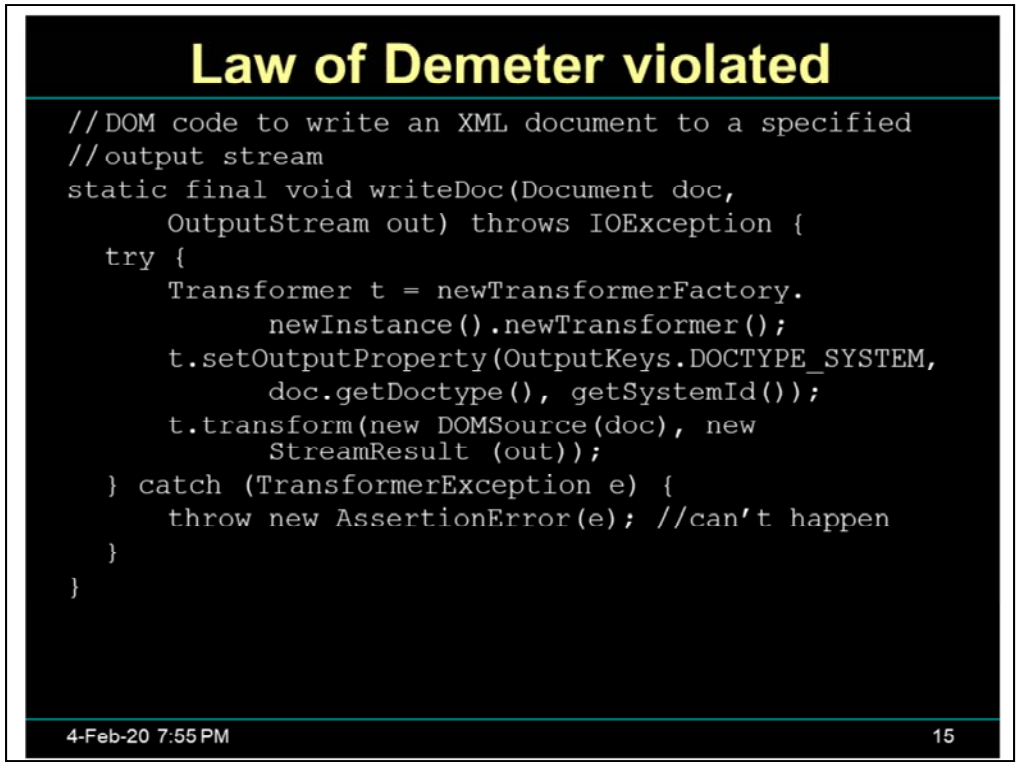
The side effect is the call to Session.initialize(), of course. The checkPassword function, by its name, says that it checks the password. The name does not imply that it initializes the session. So a caller who believes what the name of the function says runs the risk of erasing the existing session data when he or she decides to check the validity of the user.

This side effect creates a temporal coupling. That is, checkPassword can only be called at certain times (in other words, when it is safe to initialize the session). If it is called out of order, session data may be inadvertently lost. Temporal couplings are confusing, especially when hidden as a side effect. If you must have a temporal coupling, you should make it clear in the name of the function. In this case we might rename the function checkPasswordAndInitializeSession, though that certainly violates "Do one thing."



Exception: DSL.

Commands : return void.



Principle of least knowledge or Law of Demeter:

Each unit should only talk to its friends; Don't talk to strangers.

Don’t make the client do anything, that the Module could do

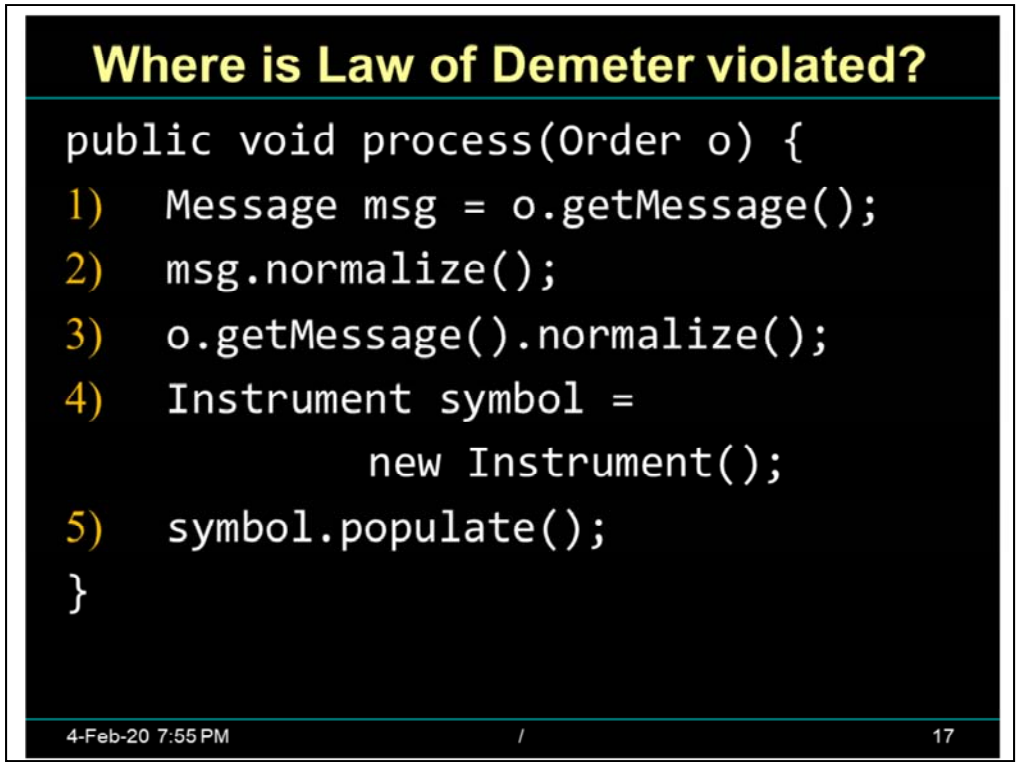
- Reduce the need for boilerplate code
- Generally done via cut-and-paste
- Ugly, Annoying and error-prone

Reduce Coupling

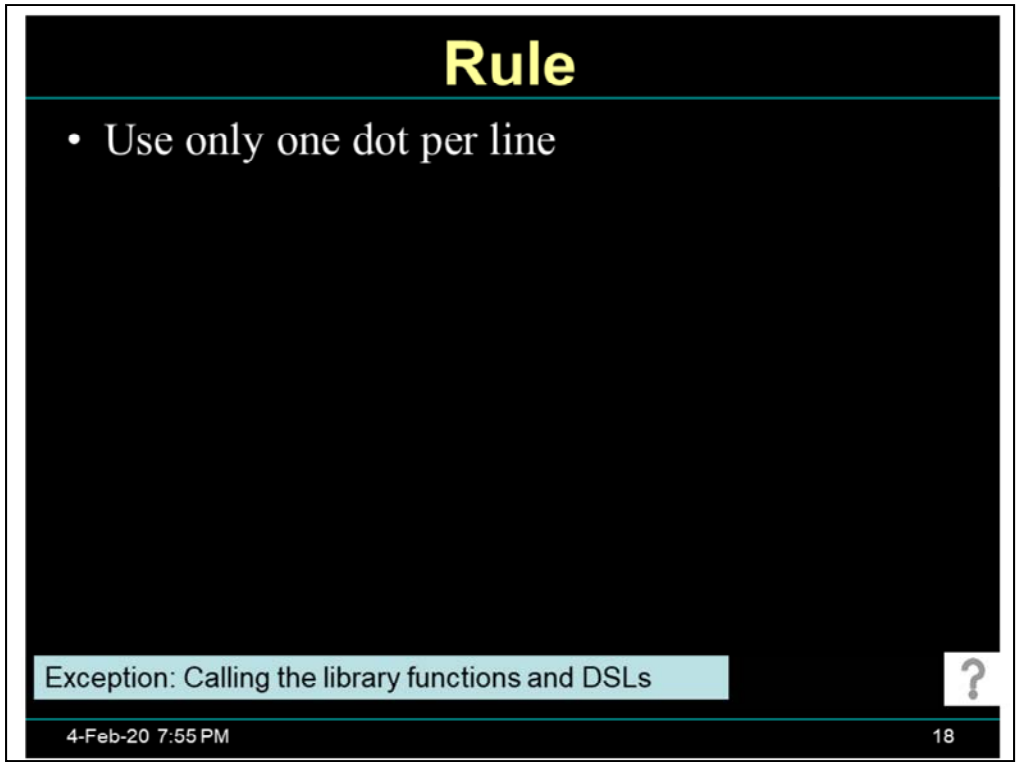
```
public float getTemp() {
  return
  station.getThermometer().getTemp();
}
```

Vs.

```
public float getTemp() {
  return station.getTemp();
}
```



Answer) lines 2 and 3


Rule
• Use only one dot per line
Exception: Calling the library functions and DSLs
?
4-Feb-20 7:55 PM
18

Q26Demeter.

Guidelines

- An interface should be designed so that it is easy to use and difficult to misuse.

API should be intuitive

- Size of String

```
myString.length(); //Java
myString.Length; //C#
length($my_string) #Perl
```

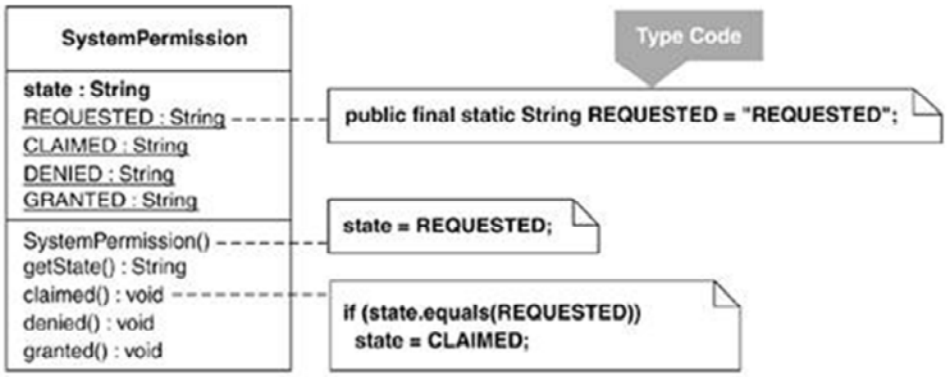
- Size of List

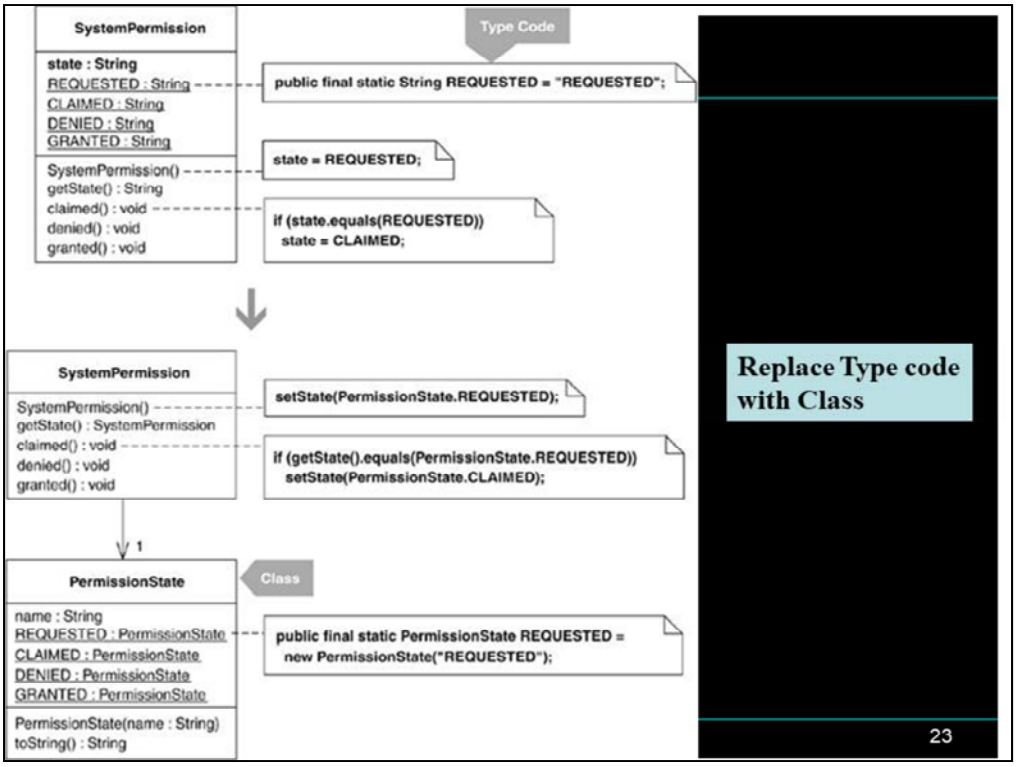
```
myList.size(); //Java
myList.Count; //C#
scalar(@my_list) #Perl
```

PHP String Library

- str_repeat
- strcmp
- str_split
- strlen
- str_word_count
- strrev

Improve Code





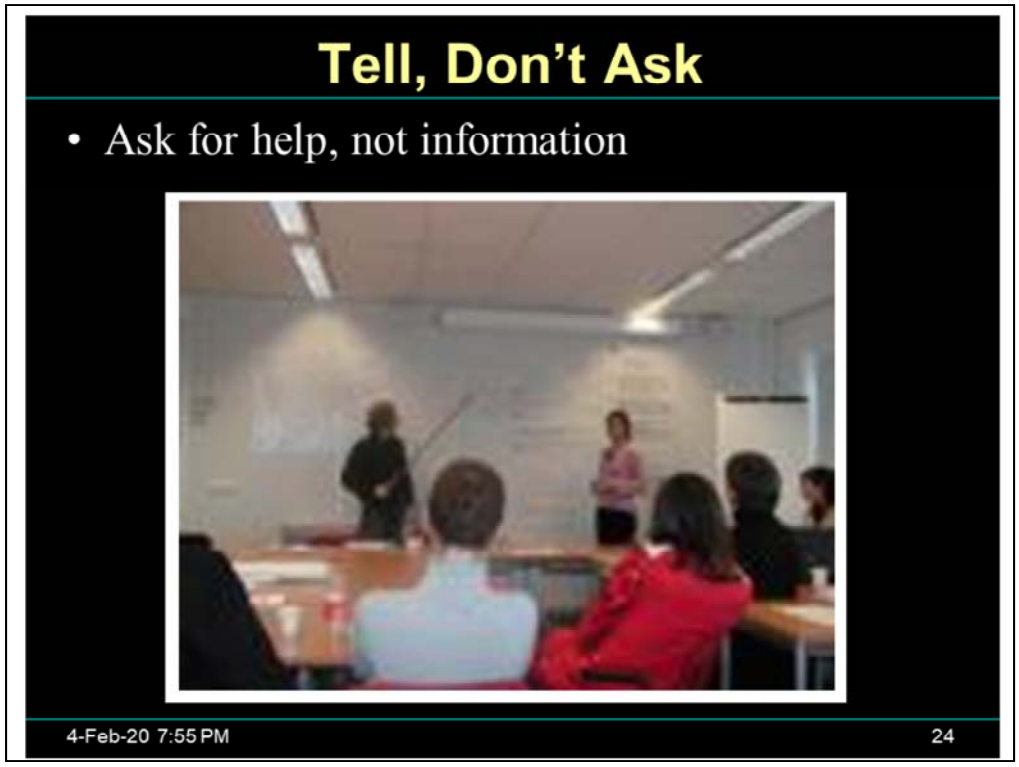
A field's type (e.g., a String or int) fails to protect it from unsafe assignments and invalid equality comparisons.

Constrain the assignments and equality comparisons by making the type of the field a class.

Benefits and Liabilities

\+ Provides better protection from invalid assignments and comparisons.

– Requires more code than using unsafe type does.



Never ask an object for information that you need to do something; rather, ask the object that has the information to do the work for you.

In other words: Don’t use any getters/setters/properties.

Avoid getters and setters

- Wrong

```
Money a, b, c;
//...
a.setValue( a.getValue() +
                b.getValue() );
```

- Right

```
Money a, b, c;
//...
a.increaseBy( b );
```

Improve

```
if (aCargo.getStatus() ==
          HandlingStatus.MISDIRECTED)
    ...

if (aCargo.isMisdirected())
    ...
```

Compare

```
Dog dog = new Dog();
dog.setBall(
      new Ball());
Ball ball =
      dog.getBAll());
```

```
Dog dog = new Dog();
dog.take(new Ball());
Ball ball =
      dog.give();
```

```
Dog dog = new Dog();
Dog.setWeight("23Kg");
```

```
Dog dog =
      new Dog("23Kg");
```

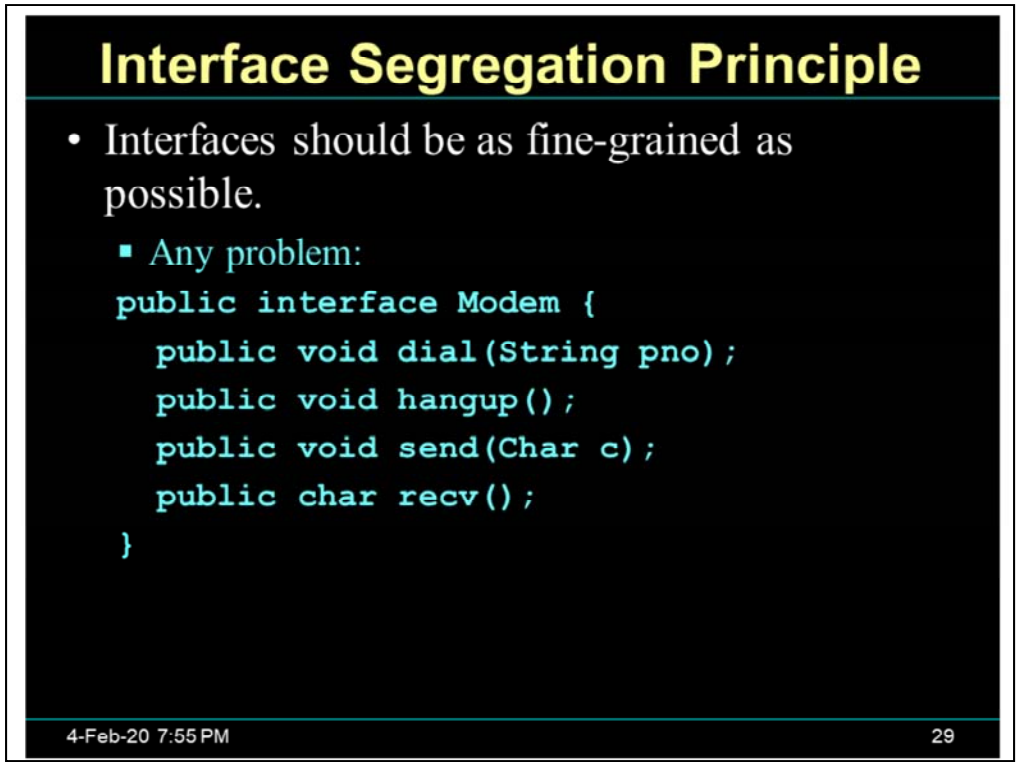
Example

Wrong:

```
MyThing[] things =
    thingManager.getThingList();
for (int i = 0; i < things.length; i++) {
    MyThing thing = things[i];
    if (thing.getName().equals(thingName))
        return thingManager.delete(thing);
}
```

Right:

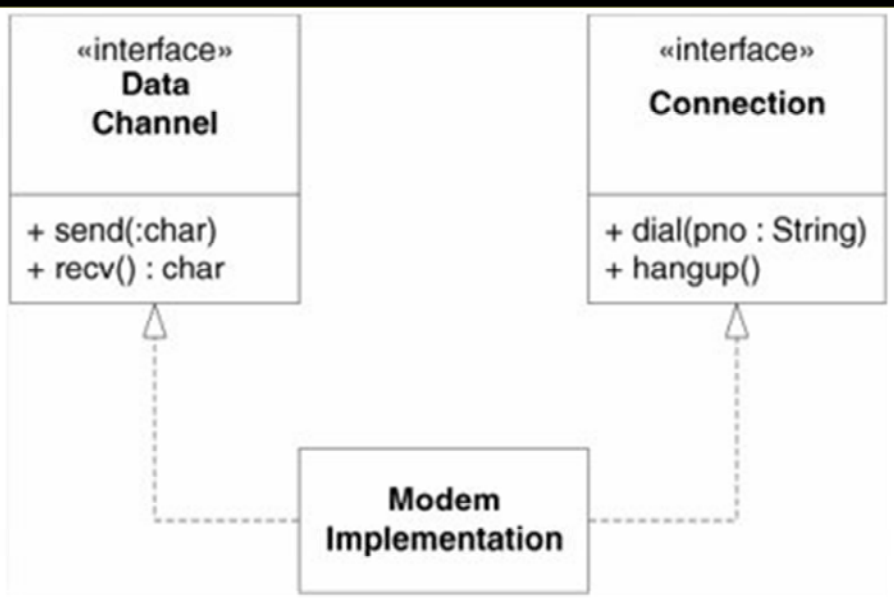
```
    return thingManager.deleteThingNamed
        (thingName);
```



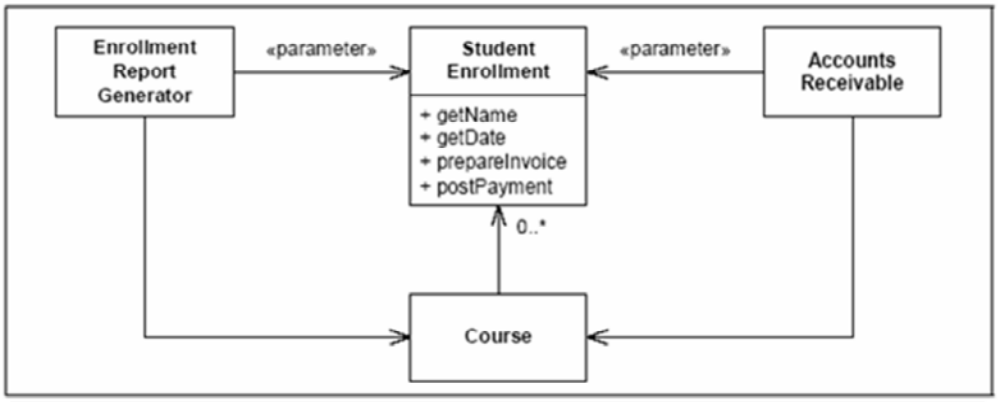
Clients should not be forced to depend upon the interfaces that they do no use. – Robert Martin

If a class implements an interface with multiple methods, but in one of the methods throws notSupportedException, then this principle is violated.

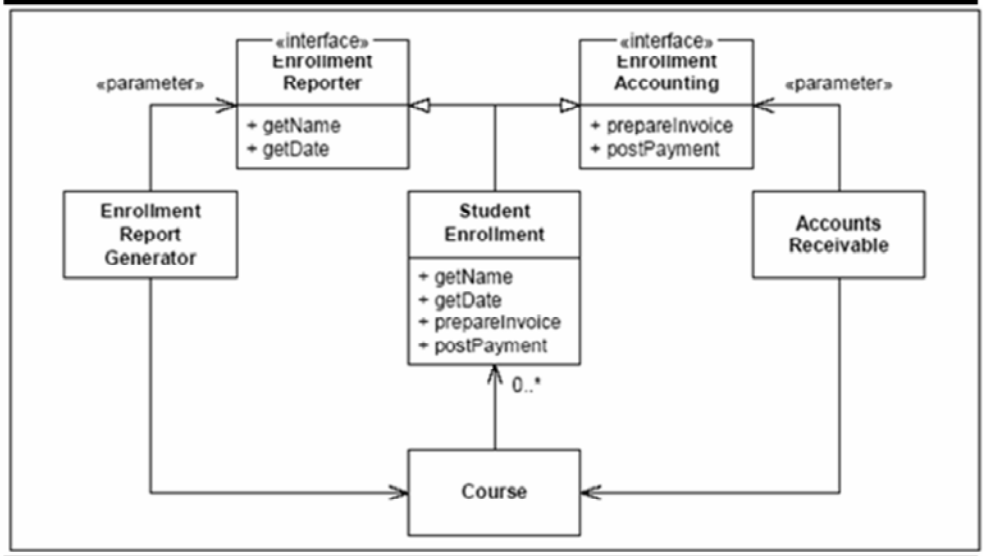
ISP implemented



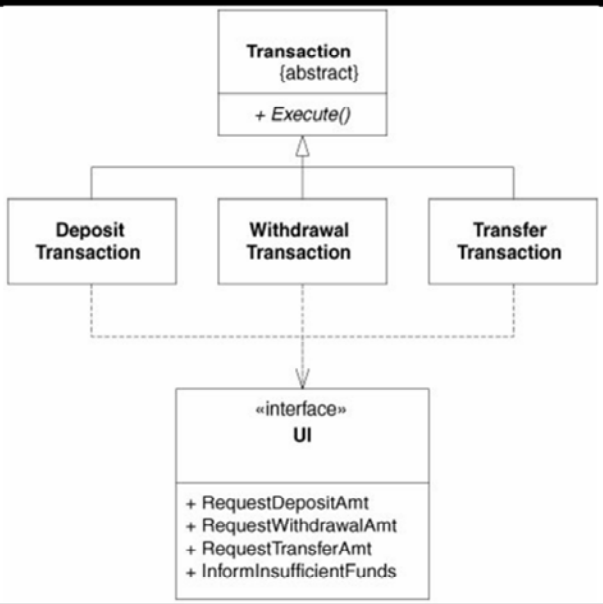
ISP violated



ISP implemented



ISP violated



ISP implemented