CS 535 Object-Oriented Programming & Design
Fall Semester, 2000
Doc 15 Heuristics & Coupling
Contents

References

Object-Oriented Design Heuristics, Riel, Addison-Wesley, 1996, Heuristics 2.3, 2.5, 2.6, 3.7, 3.8, 3.9, 4.1, 4.2, 4.3, 4.4,

Keep it Small

2.3 Minimize the number of messages in the protocol of a class

2.5 Do not put implementation details such as common-code private functions into the public interface of a class.

2.6 Do not clutter the public interface of a class with things that users of the class are not able to use or are not interested in using.

3.7 Eliminate irrelevant classes from your design.

3.8 Eliminate classes that are outside the system.

Common Minimal Public Interface

2.4 Implement a minimal public interface that all classes understand

This can be applied a global Object class, but also applies to classes in a project or domain

Java's Object

clone()
Creates and returns a copy of this object.
equals(Object obj)
Indicates if another object is "equal to" this one.
finalize()
Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
getClass()
Returns the runtime class of an object.
hashCode()
Returns a hash code value for the object.
toString()
Returns a string representation of the object.

Thread Related Methods

notify()	wait(long timeout)
notifyAll()	wait(long timeout, int nanos)
wait()	

Squeak 2.8 (Smalltalk from Disney)

The Object class has 279 instance methods

Operations, Classes, Methods

3.9 Do not turn an operation into a class

Does it make sense to have:

DisplayText Class
ParseNameValuePairs Class

Relationships between Objects

Type of Relations:

Type	Relation between
Uses	(Object)
Containment	(Object)
Inheritance	(Class)
Association	(Object)

Uses

Object A **uses** object B if A sends a message to B

Assume that A and B objects of different classes

A is the sender, B is the receiver

Containment

Class A contains class B when A has a field of type B

That is an object of type A will have an object of type B inside it

Six different Ways to Implement Uses

How does the sender access the receiver ?

1. **Containment**

The receiver is a field in the sender

```
class Sender {
  Receiver here;

  public void method() {
     here.sendAMessage();
  }
}
```

2. **Argument of a method**

The receiver is an argument in one of the senders methods

```
class Sender {
  public void method(Receiver here) {
     here.sendAMessage();
  }
}
```

3. **Ask someone else**

The sender asks someone else to give them the receiver

How does receiver access the someone else?

```
class Sender {
  public void method() {
     Receiver here = someoneElse.getReceiver();
     here.sendAMessage();
  }
}
```

4. **Creation**

The receiver is created by the sender

```
class Sender {
  public void method() {
     Receiver here = new Receiver();
     here.sendAMessage();
  }
}
```

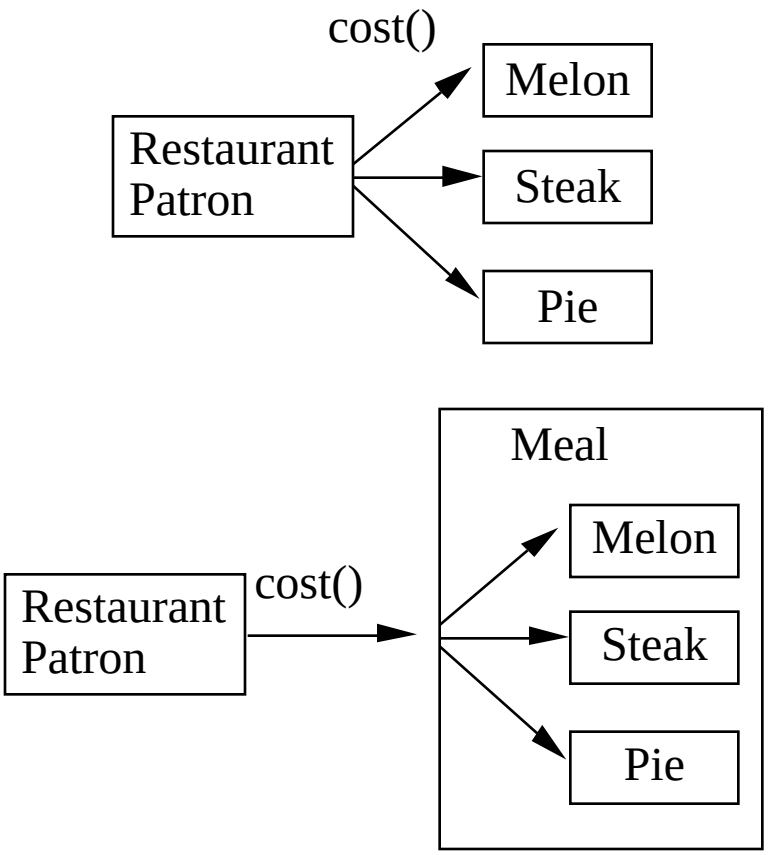
5 Referential Attribute (Covered later)

6. **Global**

The receiver is global to the sender

Heuristics for the Uses Relationship

4.1 Minimize the number of classes with another class collaborates

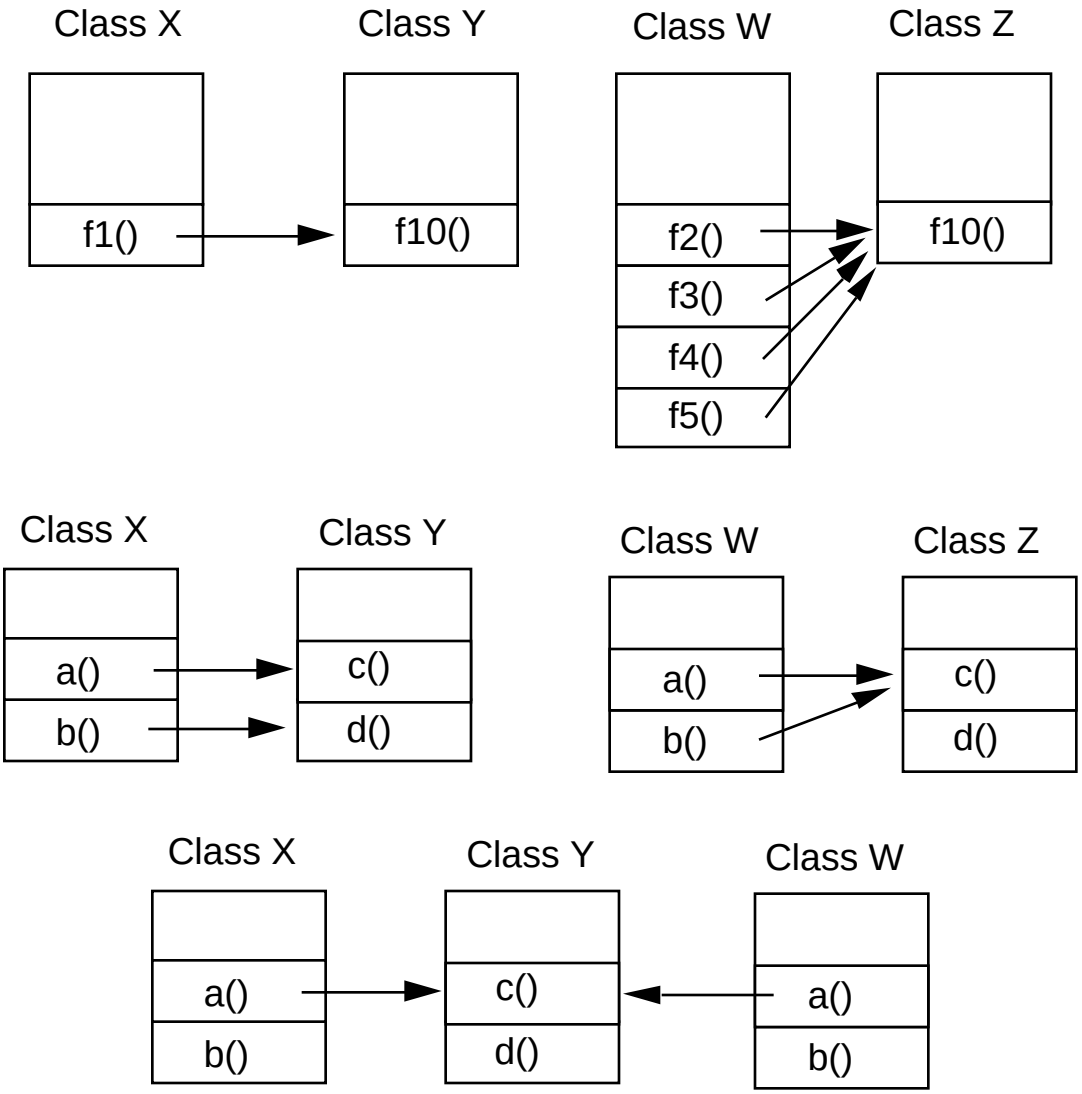


4.2 Minimize the number of messages sends between a class and its collaborator

4.3 Minimize the number of different messages a class sends to another class.

4.4 Minimize the product of the number of methods in a class and the number of different messages they send.

Which is more complex?



Containment Relationship

4.5 If class A contains objects of class B, then A should be sending messages to its fields of type B.

This heuristic prohibits:
- Orphaned fields (ones that are never used)
- Fields that are only accessed in get/set methods
  - The one exception to 4.5 is container classes

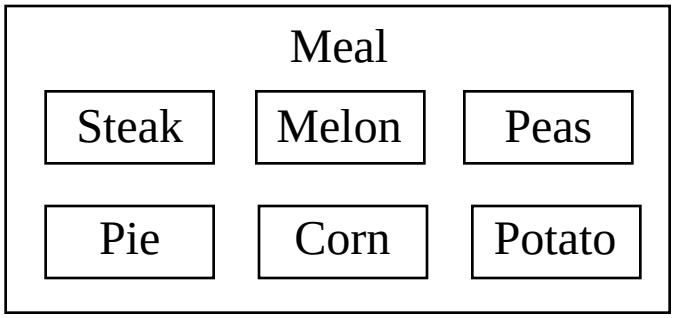
```
class Foo {
   Bar data;

   public Bar getData()  {
      return data;
   }

   public void setData( Bar newData) {
      data = newData;
   }
}
```

4.6 Most of the methods defined on a class should be using most of the fields in the class most of the time

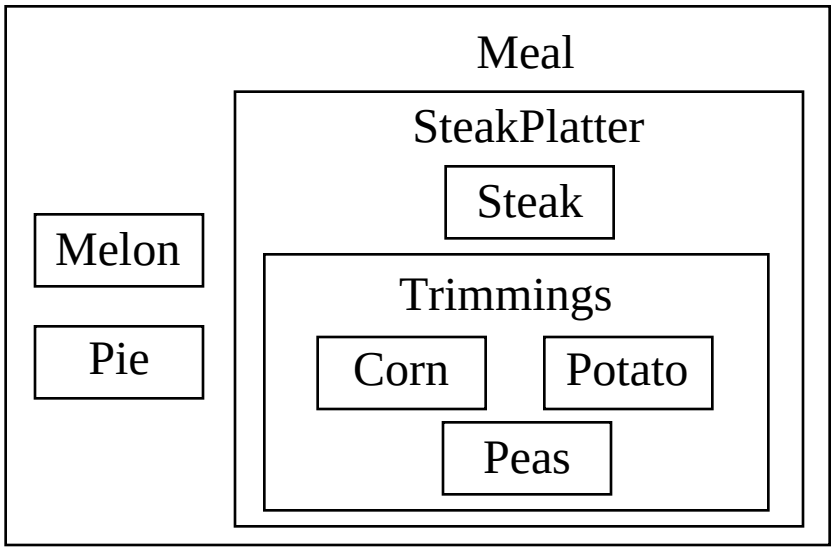
4.7 Classes should not contain more objects than a developer can fit in his or her short-term memory. A common value for this number is 6

Narrow and Deep Containment Hierarchies

Combining fields into new classes can reduce the number of fields in a class



A broad and shallow Containment Hierarchy

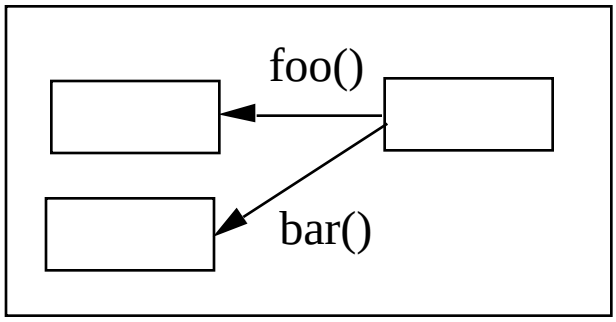


A narrow and deep Containment Hierarchy

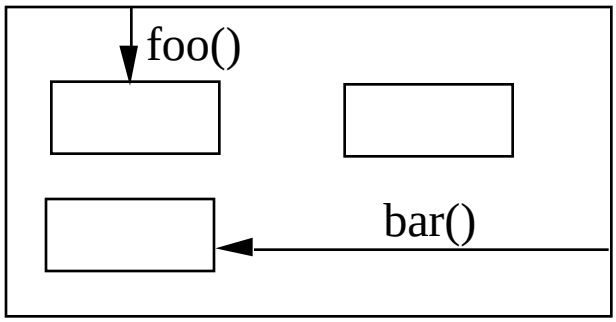
4.8 Distribute system intelligence vertically down narrow and deep containment hierarchies.

No Talking between Fields

4.14 Objects that are contained in the same containing class should not have a uses relationships between them.



Contained Objects with uses relationships



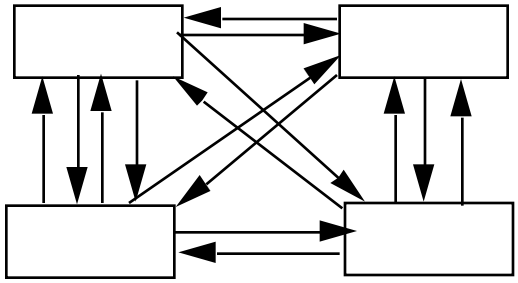
The containing class should send messages to the contained objects

4.13 A class must know what it contains, but it should not know who contains it.

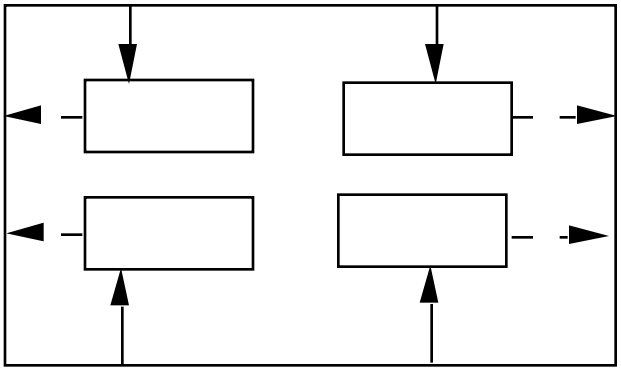
If a number of classes depend on each other in a complex way, you can violate 4.13 to reduce the number of classes they interact with.

Wrap the classes in a containing class.

Each contained class sends a message to the containing class, which broadcasts the message to the proper contained objects



Complex interactions



Replacing complex interaction with containing class