GOOD Week 5

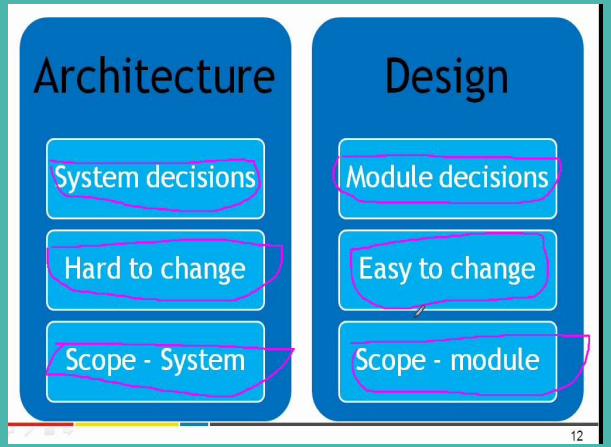
Object oriented design

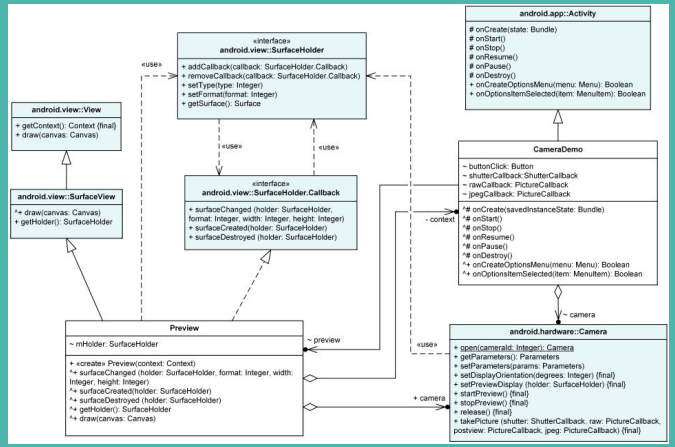
Summary

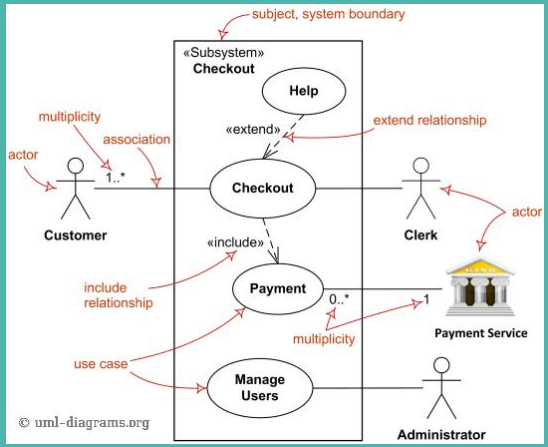
- What is software design
- Code smells
- Complexity
- Law of Demeter
- Inheritance vs Composition (Template method vs Strategy)
- Class hierarchies

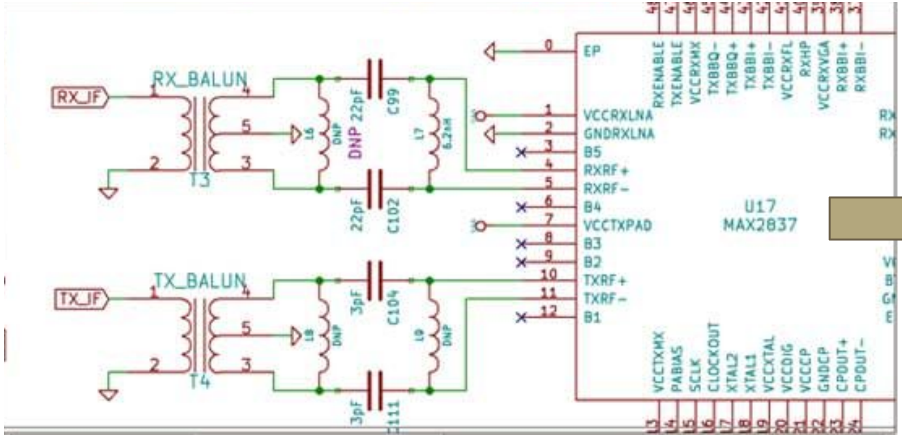

SOFTWARE DESIGN

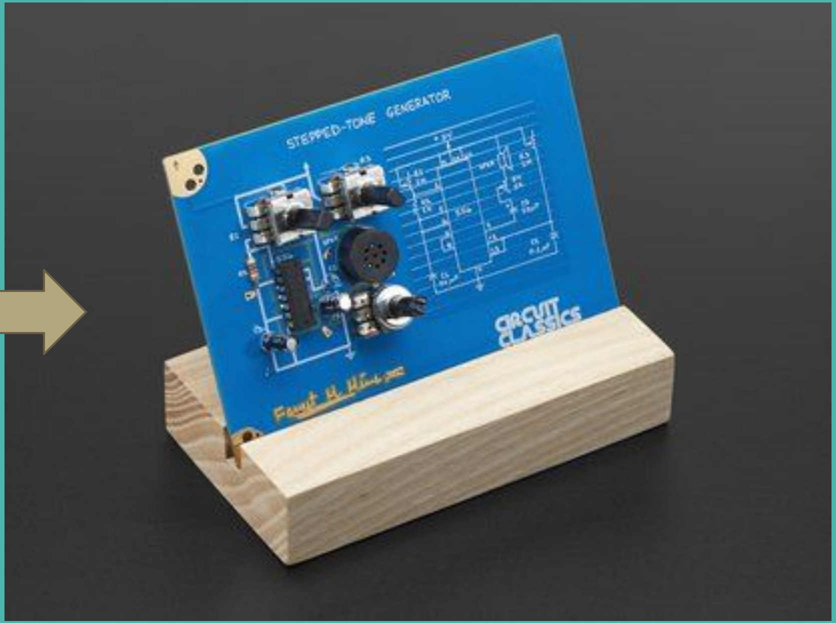
What is software design?



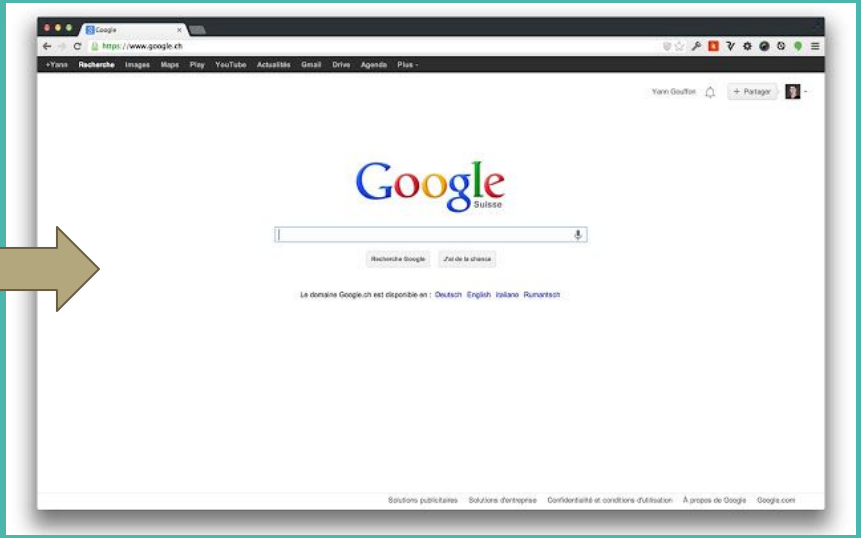




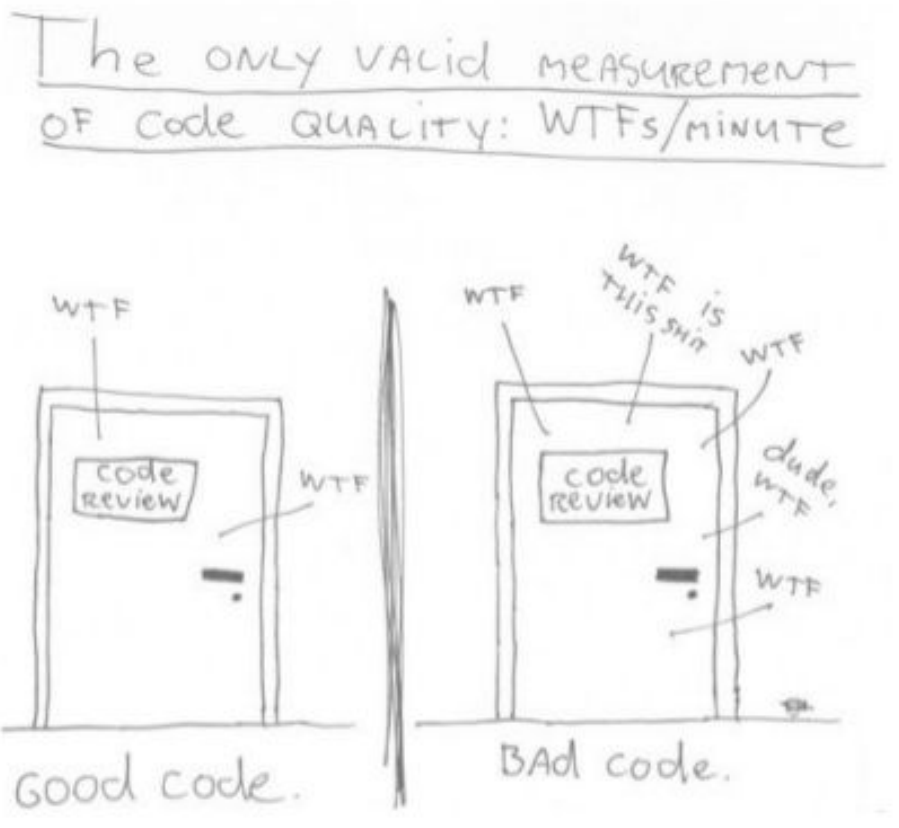

RX_IF
RX_BALUN
T3
TX_IF
TX_BALUN
T4
DNP 22pF
C99
22pF
C102
3pF
C104
3pF
C111
EP
VCCRXLNA
GNDRXLNA
B5
RXRF+
RXRF-
B4
VCCTXPAD
B3
B2
TXRF+
TXRF-
B1
U17
MAX2837
RXENABLE
TXENABLE
VCCRXMX
TXBBQ-
TXBBQ+
TXBBI+
TXBBI-
VCCRXFL
RXHP
VCCRXVGA
RXBBI+
RXBBI-
VCCTXMX
PABIAS
SCLK
CLOCKOUT
XTAL2
XTAL1
VCCXTAL
VCCDIG
VCCCP
GNDCP
CPOUT+
CPOUT-


STEPPED-TONE GENERATOR
CIRCUIT CLASSICS

```
public void disableSecurity()
    throws Throwable
{
    Statement localStatement = new Statement(System.class, "setSecurityManager", r
    Permissions localPermissions = new Permissions();
    localPermissions.add(new AllPermission());
    ProtectionDomain localProtectionDomain = new ProtectionDomain(new CodeSource(r
    AccessControlContext localAccessControlContext = new AccessControlContext(new
        localProtectionDomain
    });
    SetField(Statement.class, "acc", localStatement, localAccessControlContext);
    localStatement.execute();
}

private Class GetClass(String paramString)
    throws Throwable
{
    Object arrayOfObject[] = new Object[1];
    arrayOfObject[0] = paramString;
    Expression localExpression = new Expression(Class.class, "forName", arrayOfObj
    localExpression.execute();
    return (Class)localExpression.getValue();
}

private void SetField(Class paramClass, String paramString, Object paramObject1, (
    throws Throwable
```



HOW do you measure software design QUALITY?



Code smells

4 code smells

- RIGIDITY
- IMMOBILITY
- VISCOSITY
- FRAGILITY

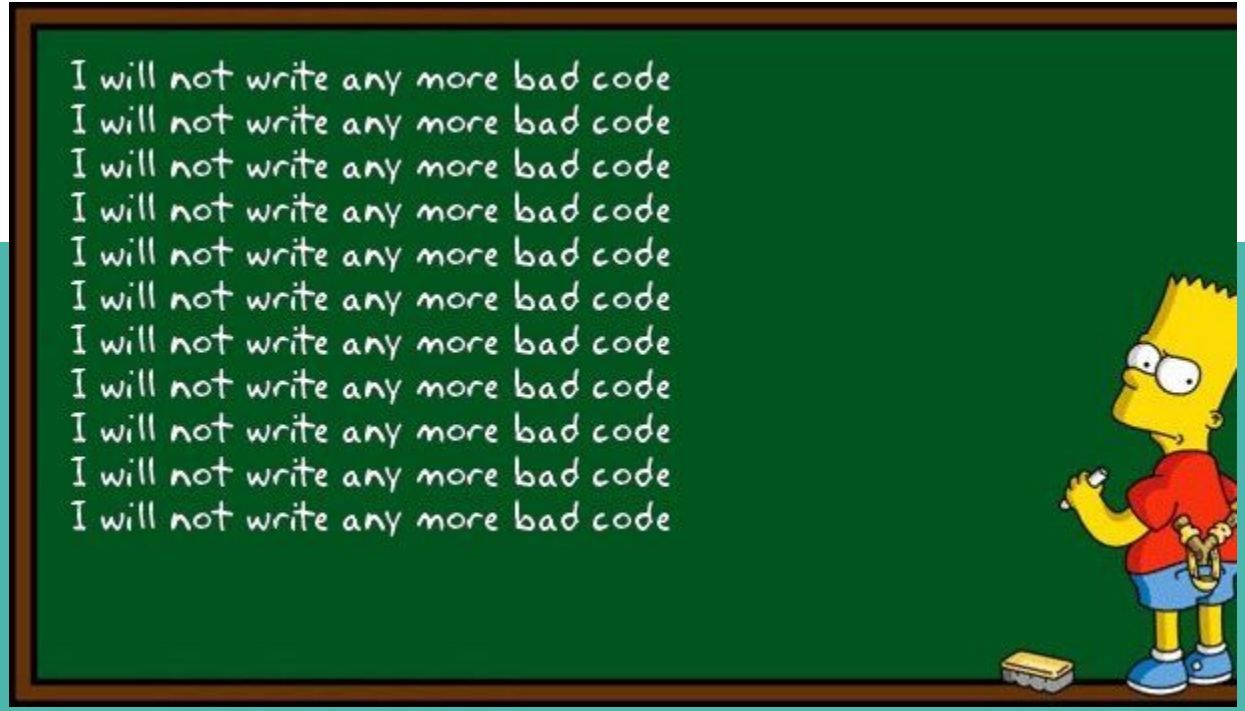

Change

Code Smell

CODE SMELLS ARE SYMPTOMS OF POOR DESIGN OR IMPLEMENTATION CHOISES

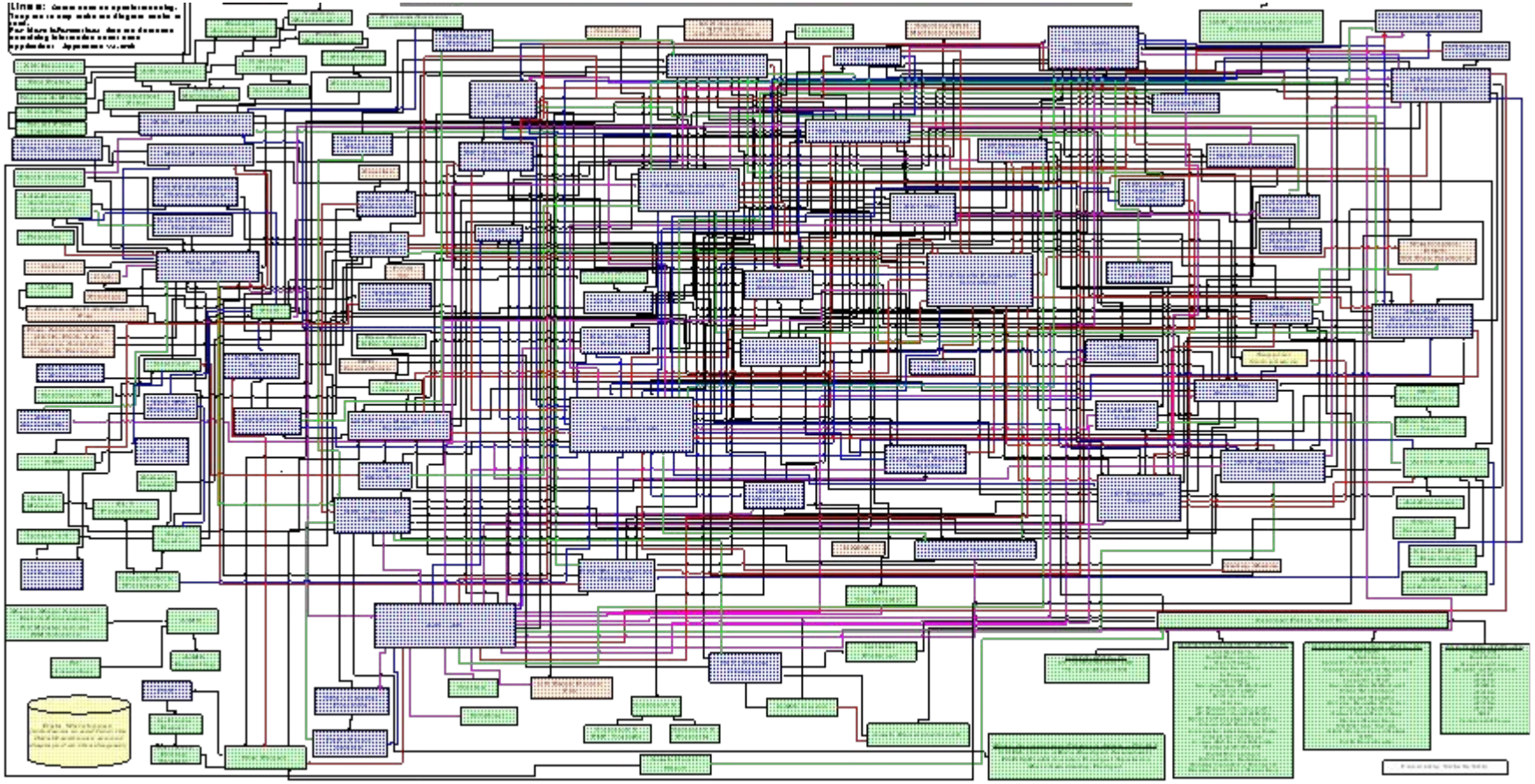
[Martin Fowler]

What do I have to do?



What is design about?


Change



Dependencies

Demeter’s law


DON'T
TALK TO
Strangers

```
{
myObject.Never.
    Talks.to.Strangers
}
```

LOD

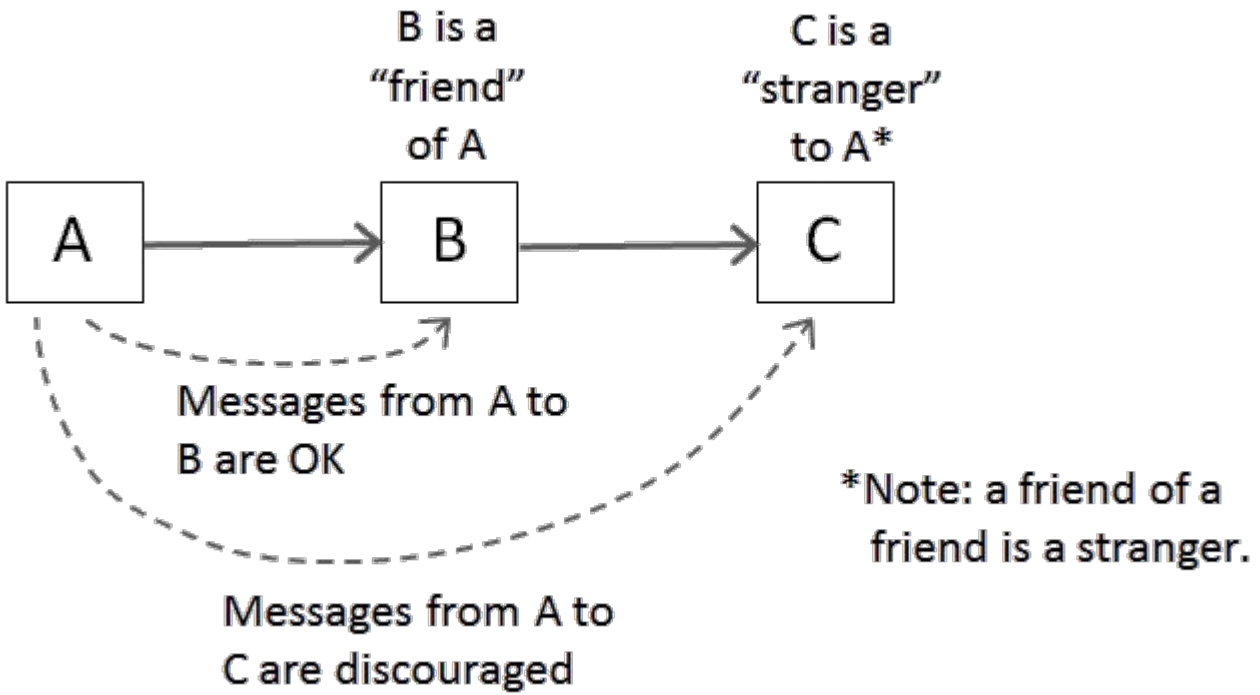
Inside a method M of a class C you can only access data and call methods from:

- ➔ **this** and **base-object**
- ➔ **parameters** of method M
- ➔ **data members** of class C
- ➔ **objects created** inside M
- ➔ **Global** variables

LOD playful

- ➔ You can play with yourself.
- ➔ You can play with your own toys (but you can't take them apart).
- ➔ You can play with toys that were given to you.
- ➔ You can play with toys you've made yourself.
- ➔ You can play with all the toys on the public playground.

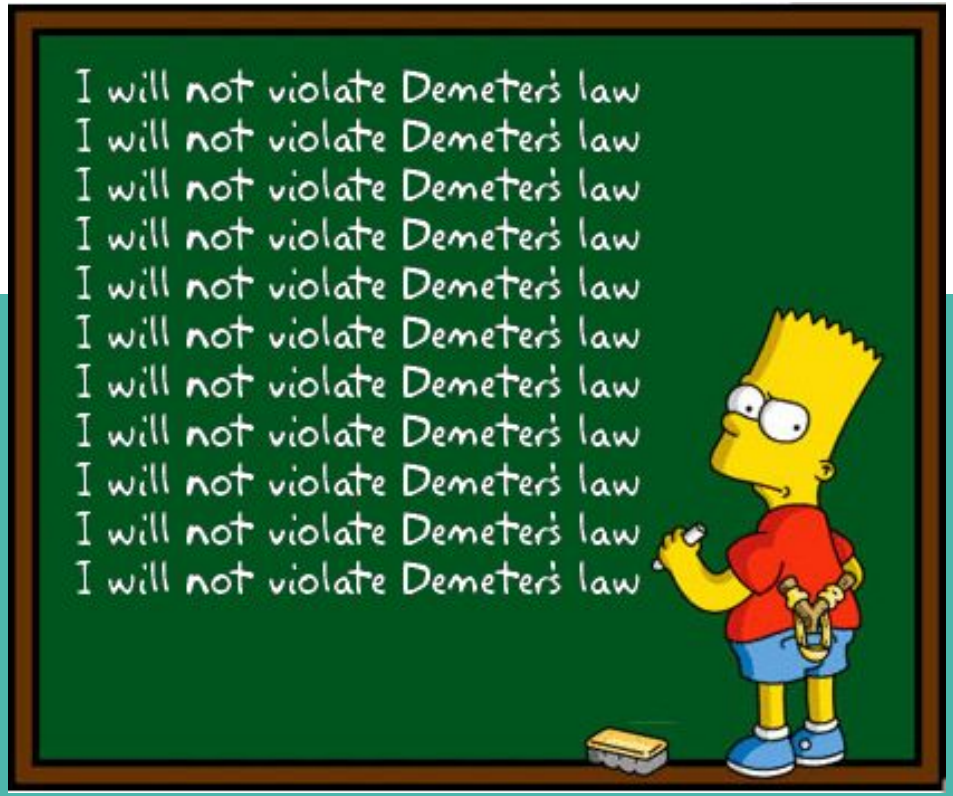
Example:


B is a
"friend"
of A
C is a
"stranger"
to A*
A
B
C
Messages from A to
B are OK
*Note: a friend of a
friend is a stranger.
Messages from A to
C are discouraged

LOD: VIOLATION

```
getThis().getThat().
    getSomethingElse().
    doTheWork();
```

What do I have to do?



Does this violate the LOD?

```
public int getNoOfCompatibleProducts(Vector<Product> products, Product product)
{
   int ret = 0;
   for(int i=0; i<products.capacity(); i++)
   {
       if(products.get(i).isCompatibleWith(product))
           ret++;
   }

   return ret;
}

class Product {
   private Color color;
   private String id;

   //constructor, getters, setters...

   public boolean isCompatibleWith(Product p) { ... }
}
```

Does this violate the LOD?

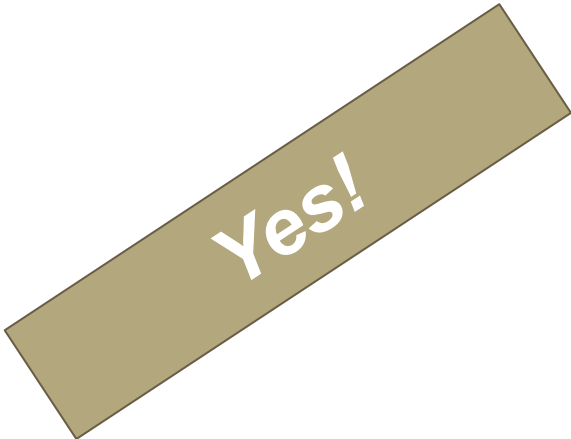
```
public int getNoOfCompatibleProducts(Vector<Product> products, Product product)
{
   int ret = 0;
   for(int i=0; i<products.capacity(); i++)
   {
       if(products.get(i).isCompatibleWith(product))
           ret++;
   }

   return ret;
}

class Product {
   private Color color;
   private String id;

   //constructor, getters, setters...

   public boolean isCompatibleWith(Product p) { ... }
}
```

Does this violate the LOD?

```
public int getNoOfRedProducts(Vector<Product> products)
{
   int ret = 0;
   for(int i=0; i<products.capacity(); i++)
   {
       if(products.get(i).getColor().equals(new Color(255, 0,0)))
           ret++;
   }

   return ret;
}

class Product {
   private Color color;
   private String id;

   //constructor, getters, setters...

   public boolean isCompatibleWith(Product p) { ... }
}
```

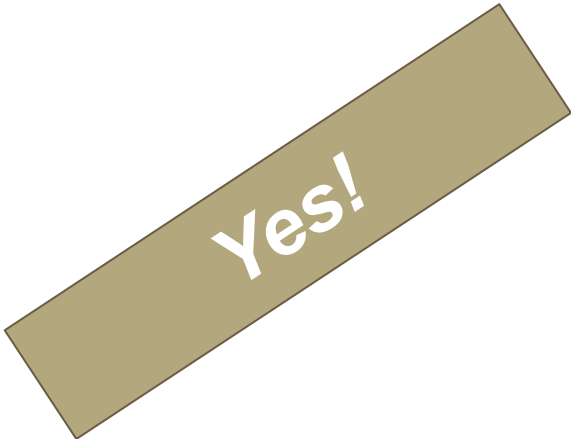
Does this violate the LOD?

```
public int getNoOfRedProducts(Vector<Product> products)
{
   int ret = 0;
   for(int i=0; i<products.capacity(); i++)
   {
       if(products.get(i).getColor().equals(new Color(255, 0,0)))
           ret++;
   }

   return ret;
}

class Product {
   private Color color;
   private String id;

   //constructor, getters, setters...

   public boolean isCompatibleWith(Product p) { ... }
}
```



Does this violate the LOD?

```
public int getNoOfRedProducts(Vector<Product> products)
{
   int ret = 0;
   for(int i=0; i<products.capacity(); i++)
   {
       if(products.get(i).getColor().equals(new Color(255, 0,0)))
           ret++;
                       products.get(i).getColor().set(0,0,0)
   }

   return ret;
}

class Product {
   private Color color;
   private String id;

   //constructor, getters, setters...

   public boolean isCompatibleWith(Product p) { ... }
}
```



Does this violate the LOD?

```
public int getNoOfRedProducts(Vector<Product> products)
{
   int ret = 0;
   for(int i=0; i<products.capacity(); i++)
   {
       if(products.get(i).hasSameColor(new Color(255, 0,0)))
           ret++;
   }

   return ret;
}
class Product {
   private Color color;
   private String id;

   //constructor, getters, setters...
   public boolean isCompatibleWith(Product p) { ... }
   public boolean hasSameColor(Color c) { ... }

}
```

Does this violate the LOD?

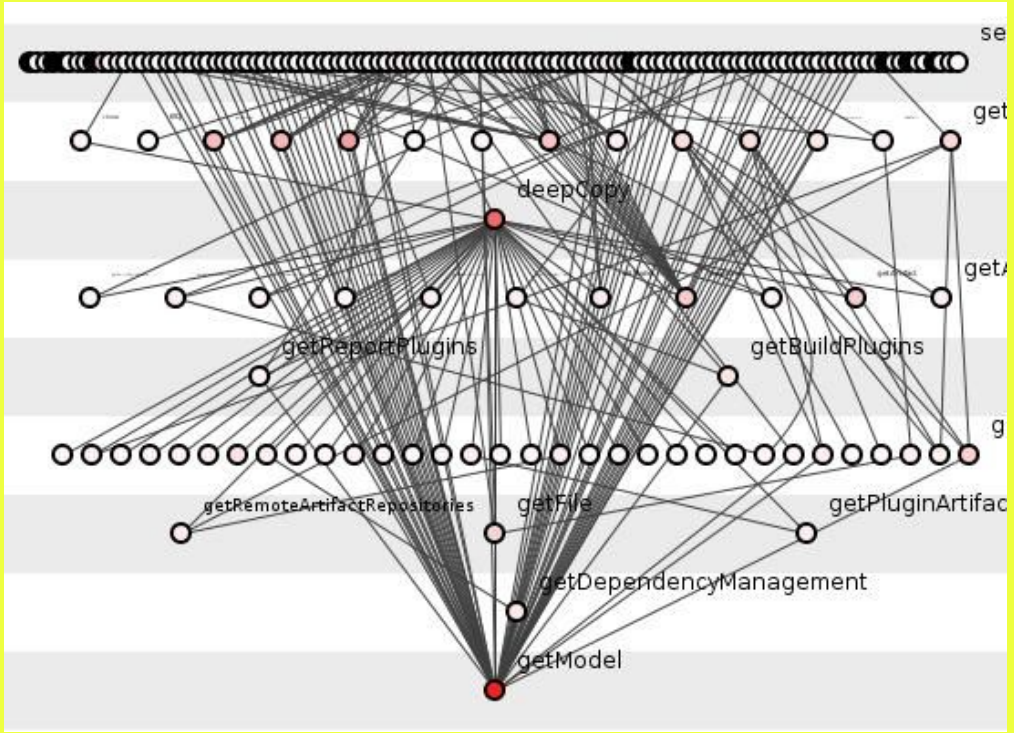
```
public int getNoOfRedProducts(Vector<Product> products)
{
   int ret = 0;
   for(int i=0; i<products.capacity(); i++)
   {
       Product product = products.get(i);
       Color productColor = product.getColor();
       if(productColor.equals(new Color(255, 0,0)))
           ret++;
   }

   return ret;
}

class Product {
   private Color color;
   private String id;

   //constructor, getters, setters...

   public boolean isCompatibleWith(Product p) { ... }
}
```

Does this violate the LOD?

```
public int getNoOfRedProducts(Vector<Product> products)
{
   int ret = 0;
   for(int i=0; i<products.capacity(); i++)
   {
       Product product = products.get(i);
       Color productColor = product.getColor();
       if(productColor.equals(new Color(255, 0,0)))
           ret++;
   }

   return ret;
}
```

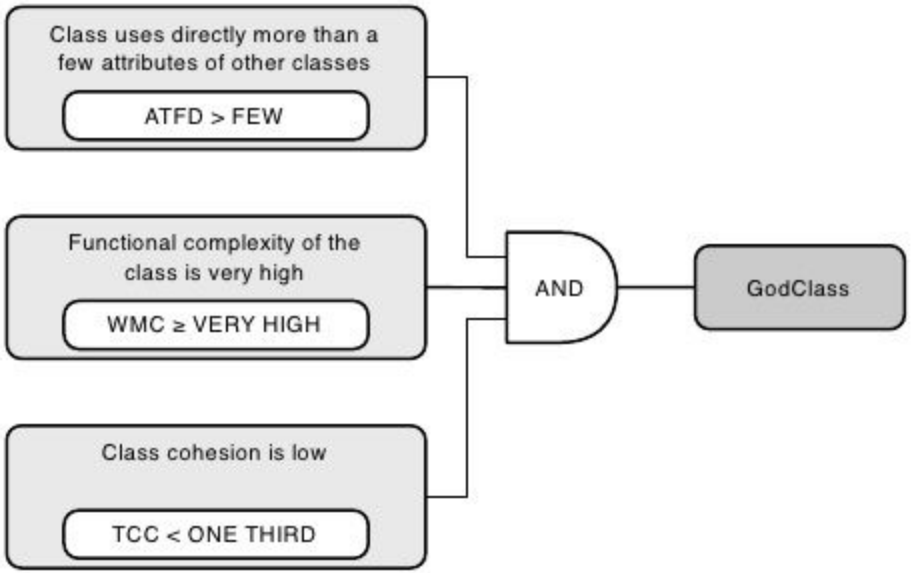
```
class Product {
   private Color color;
   private String id;

   //constructor, getters, setters...

   public boolean isCompatibleWith(Product p) { ... }
}
```

God Class


deepCopy
getReportPlugins
getBuildPlugins
getRemoteArtifactRepositories
getFile
getDependencyManagement
getModel

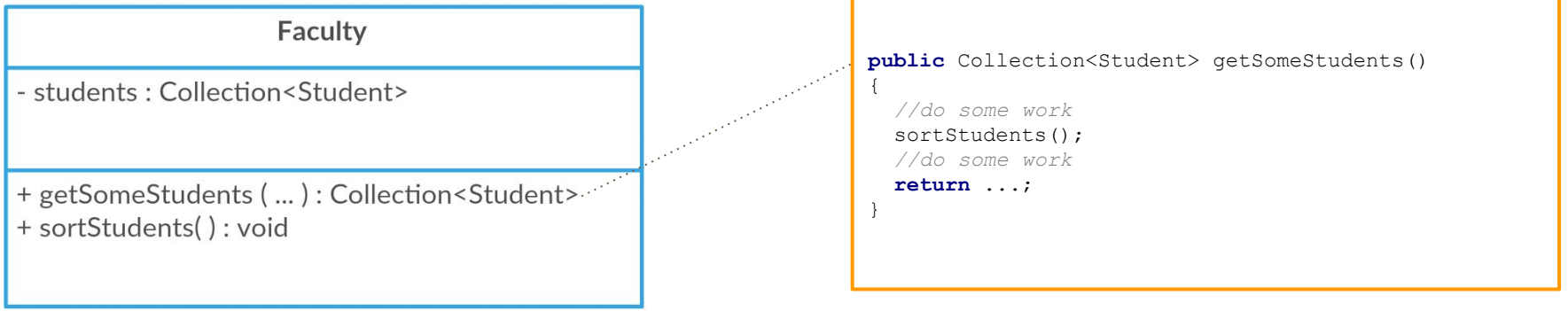
A God Class centralizes too much intelligence in the system.

Lanza, Marinescu 2006



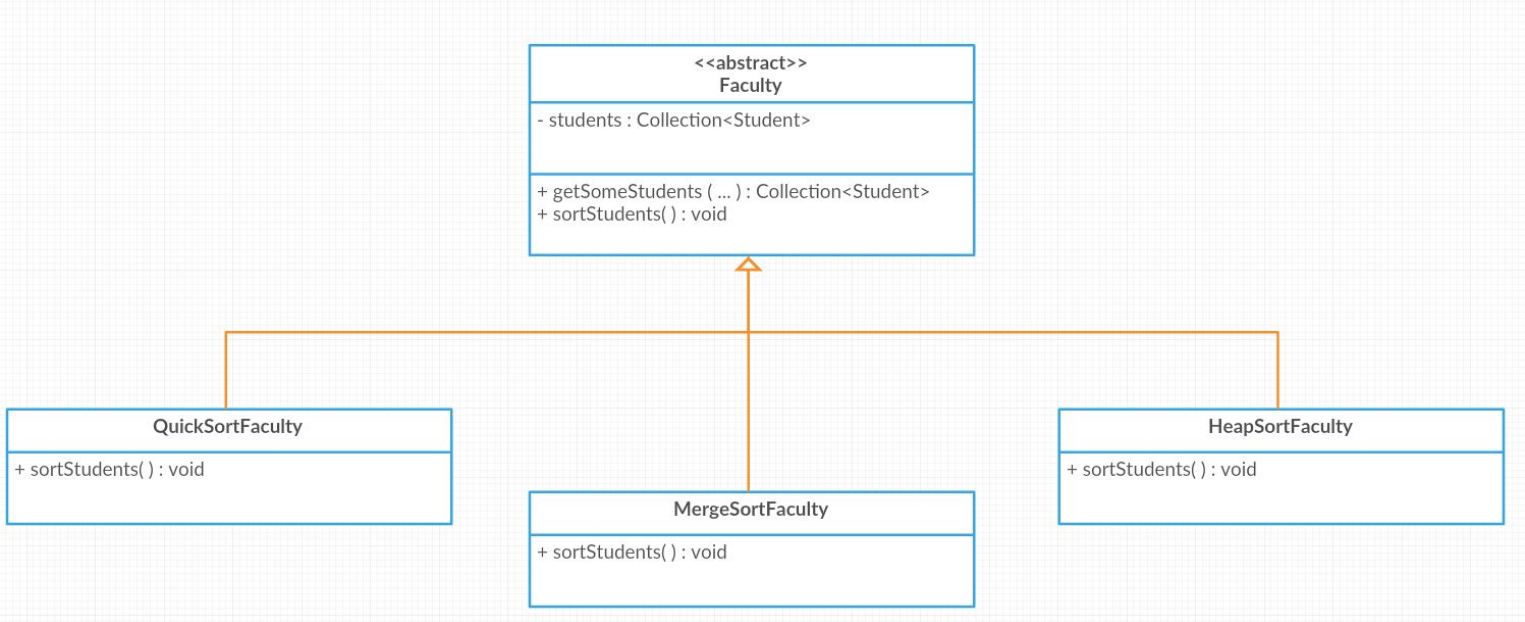
Inheritance vs composition

Example



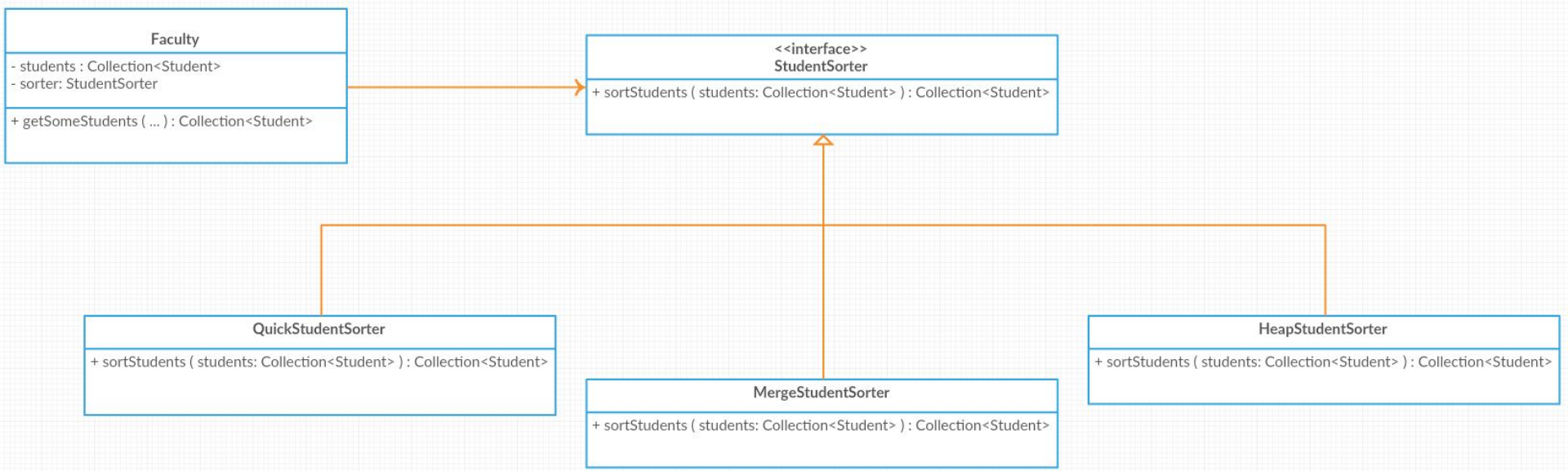
What if you want to be able to sort in multiple ways? QuickSort, MergeSort,...

Solution?



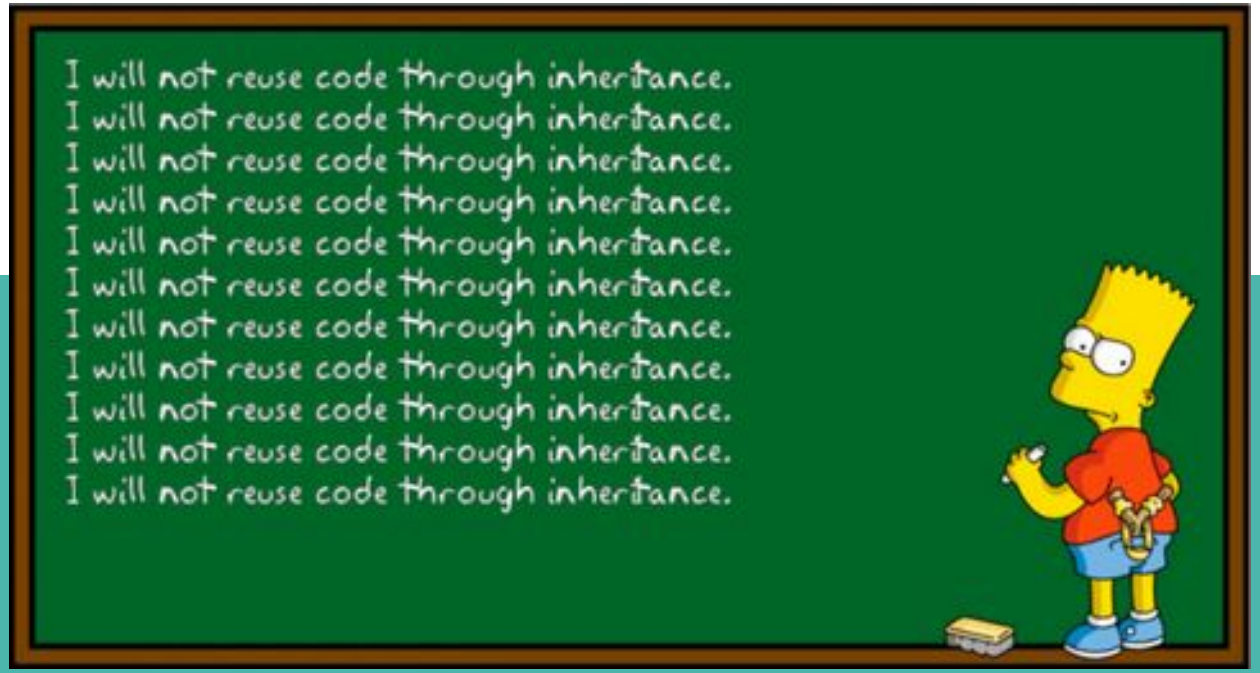
Ok...but what if some time I want Quicksort, some other time Mergesort and another time, another sort...?

Solution


Faculty
- students : Collection<Student>
- sorter: StudentSorter
+ getSomeStudents (...) : Collection<Student>
<<interface>>
StudentSorter
+ sortStudents (students: Collection<Student>) : Collection<Student>
QuickStudentSorter
+ sortStudents (students: Collection<Student>) : Collection<Student>
MergeStudentSorter
+ sortStudents (students: Collection<Student>) : Collection<Student>
HeapStudentSorter
+ sortStudents (students: Collection<Student>) : Collection<Student>

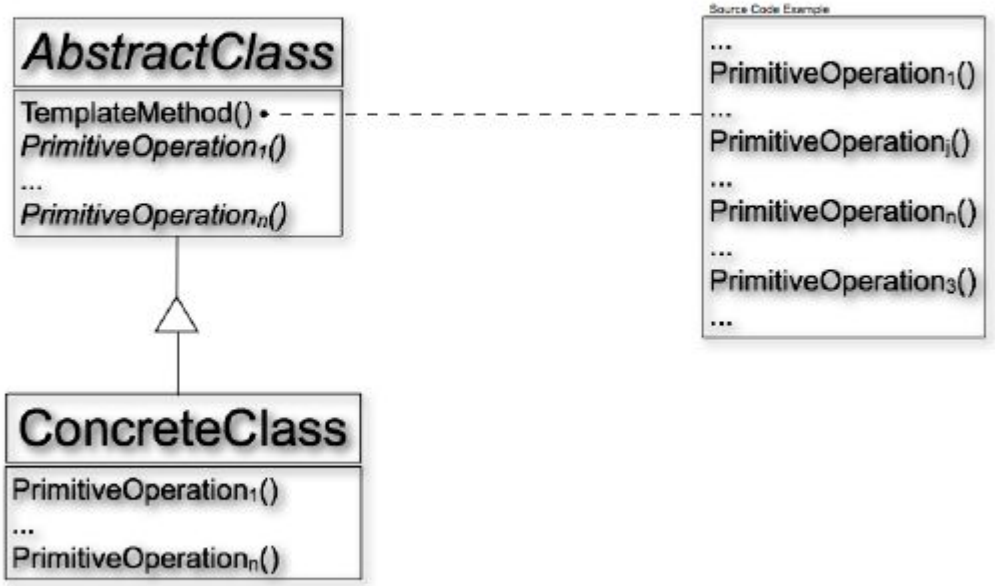
Favor composition over inheritance!

What do I have to do?

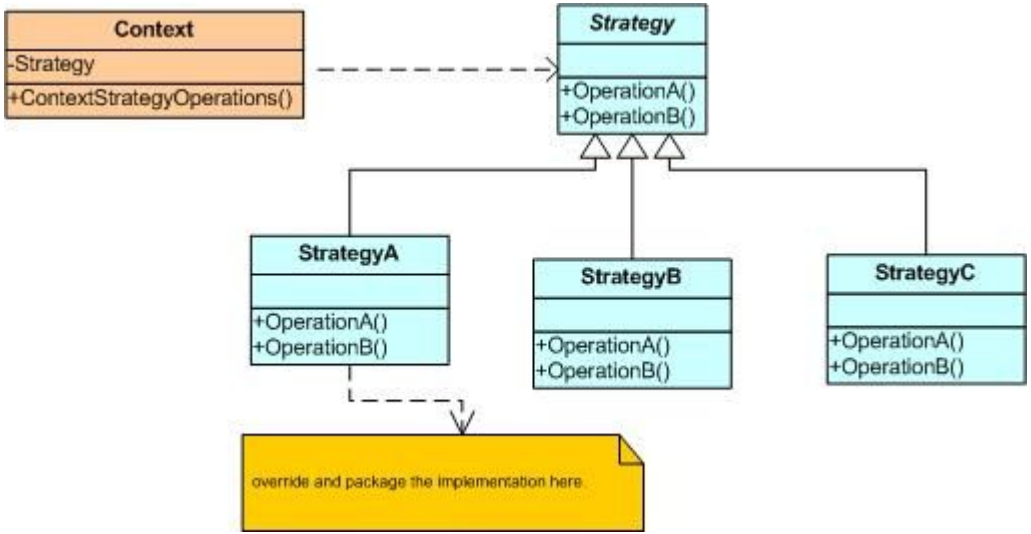


Template method vs Strategy

Template method pattern



Strategy pattern



Strategy pattern