Design Defects and Restructuring

Engr. Abdul-Rahman Mahmood

 abdulrahman@nu.edu.pk

 alphapeeler.sf.net/pubkeys/pkey.htm

 pk.linkedin.com/in/armahmood

 www.twitter.com/alphapeeler

 www.facebook.com/alphapeeler

 abdulmahmood-sss  alphasecure

 armahmood786

 http://alphapeeler.sf.net/me

 alphapeeler#9321

 *reddit.com/user/alphapeeler*

 www.flickr.com/alphapeeler

 http://alphapeeler.tumblr.com

 armahmood786@jabber.org

 alphapeeler@aim.com

 mahmood_cubix  48660186

 alphapeeler@icloud.com

 pinterest.com/alphapeeler

 www.youtube.com/user/AlphaPeeler

Refactoring-IV

Bad Smells in Code

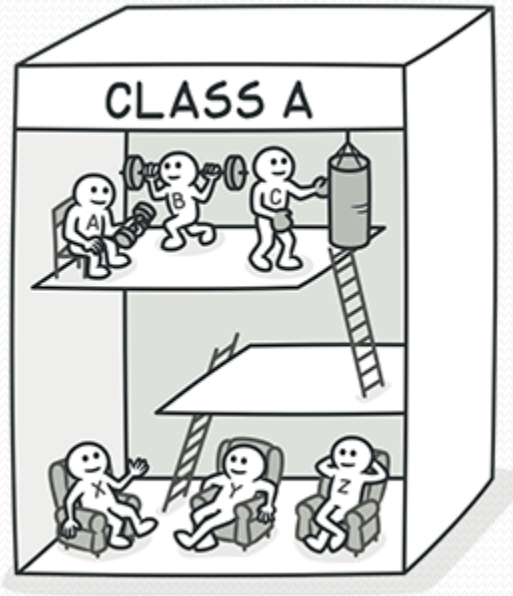
Bad Smells in Code

- **Duplicated Code**
- **Long Method**
- **Large Class**
- **Long Parameter List**
- **Divergent Change**
- **Shotgun Surgery**
- **Feature Envy**
- **Data Clumps**
- **Primitive Obsession**
- **Switch Statements**
- Parallel Interface Hierarchies
- Lazy Class
- Speculative Generality
- Temporary Field
- Message Chains
- Middle Man
- Inappropriate Intimacy
- Incomplete Library Class
- Data Class
- Refused Bequest

Temporary Field

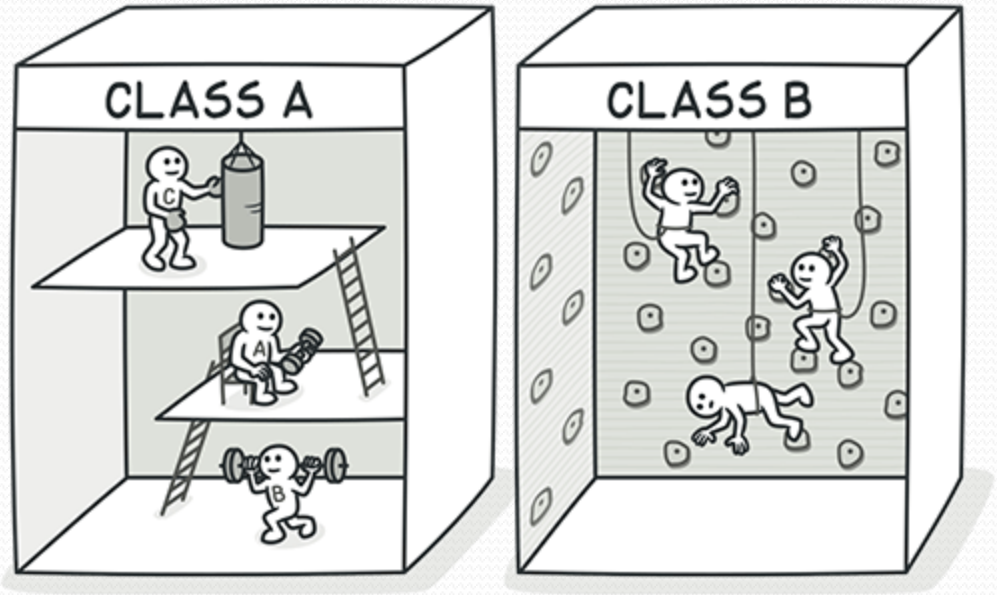
- **Signs and Symptoms:** Temporary fields get their values (and thus are needed by objects) only under certain circumstances. Outside of these circumstances, they're empty.
- **Reasons for the Problem :**Oftentimes, temporary fields are created for use in an algorithm that requires a large amount of inputs. So instead of creating a large number of parameters in the method, the programmer decides to create fields for this data in the class. These fields are used only in the algorithm and go unused the rest of the time.
- This kind of code is tough to understand. You expect to see data in object fields but for some reason they're almost always empty.

Temporary Field



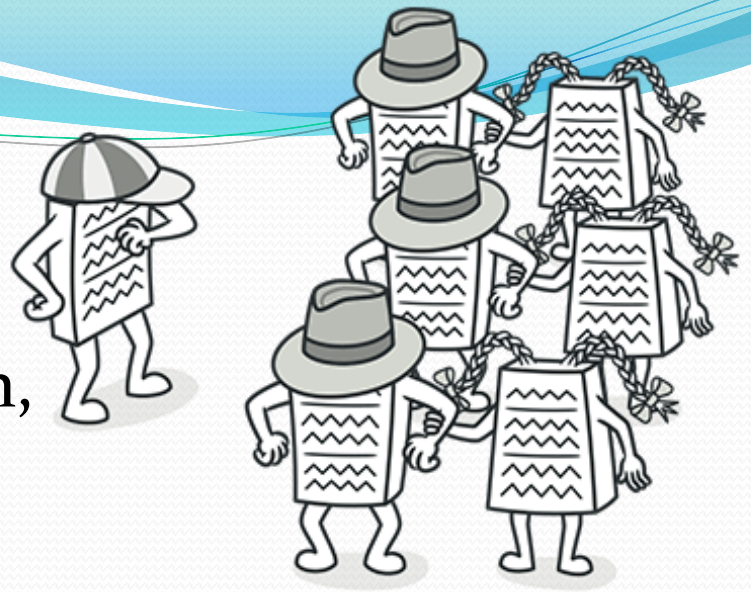
- **Treatment**
- Temporary fields and all code operating on them can be put in a separate class via Extract Class.
- Introduce Null Object and integrate it in place of the conditional code which was used to check the temporary field values for existence.

Temporary Field



- **Payoff**
- Better code clarity and organization.
- Code example:
- https://blog.ploeh.dk/2015/09/18/temporary-field-code-smell/

Middle Man



- **Signs and Symptoms**
- If a class performs only one action, delegating work to another class, why does it exist at all
- **Reasons for the Problem**
- This smell can be the result of overzealous elimination of Message Chains.
- It can be result of useful work of a class being gradually moved to other classes. The class remains as an empty shell that doesn't do anything other than delegate.
- **Treatment**
- If most of a method's classes delegate to another class, Remove Middle Man is in order.

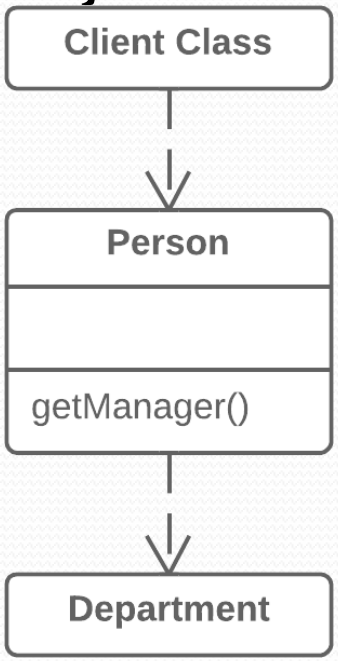
Middle Man

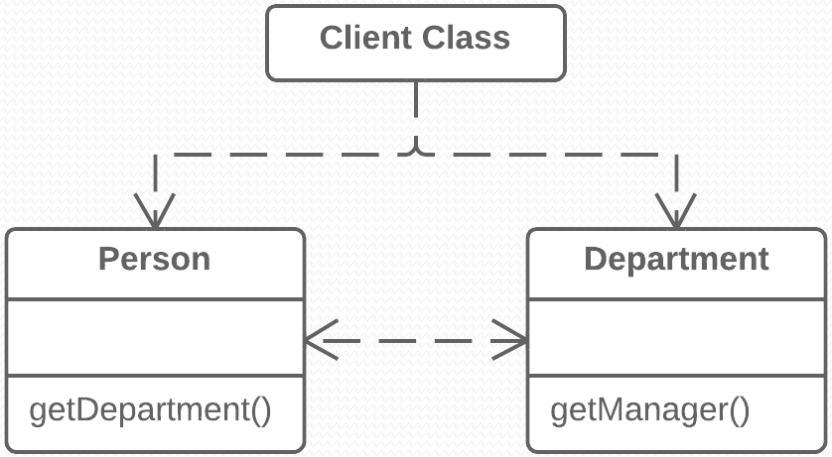
- **Payoff**
- Less bulky code.

- **When to Ignore**
- Don't delete middle man that have been created for a reason:
  - A middle man may have been added to avoid interclass dependencies.
  - Some design patterns create middle man on purpose (such as Proxy or Decorator).

Remove Middle Man

- **Problem**
- A class has too many methods that simply delegate to other objects.



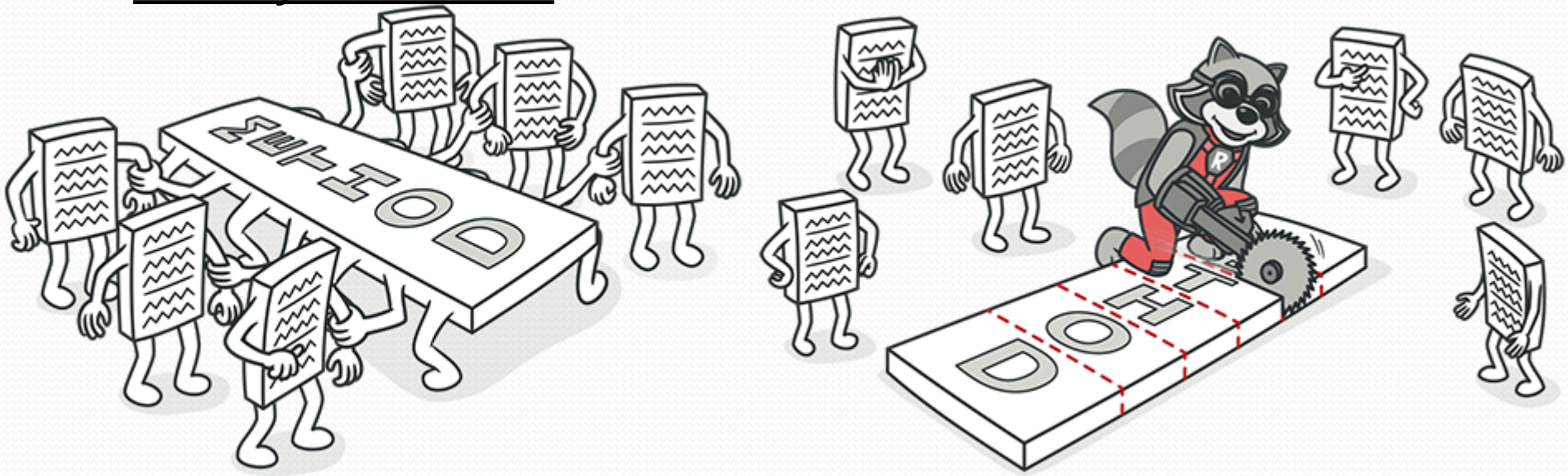


- **Solution**
- Delete these methods and force the client to call the end methods directly.

https://www.youtube.com/watch?v=xa6aS9ObJTg

Inappropriate Intimacy

- **Signs and Symptoms:** One class uses the internal fields and methods of another class. such as by directly accessing instance variables that aren't meant to be directly accessed.



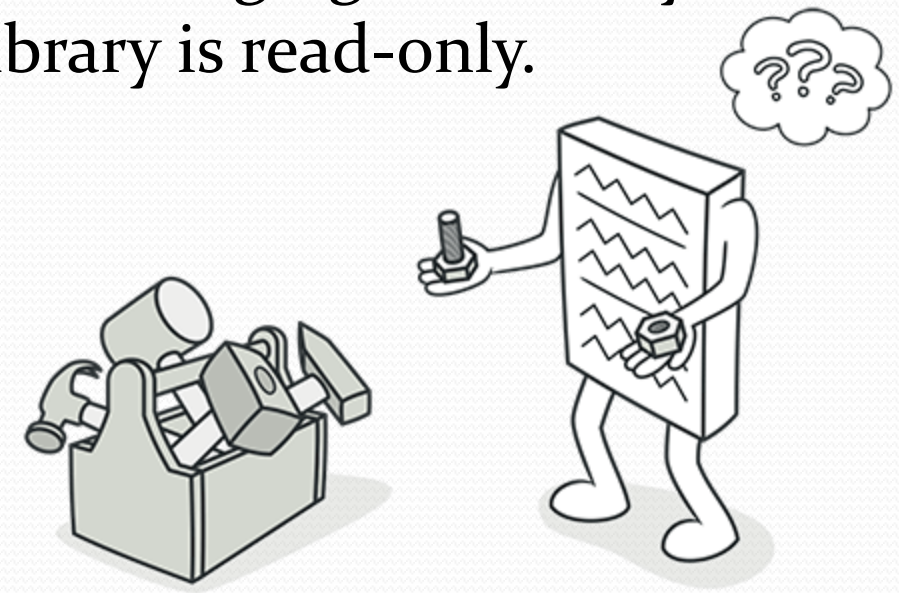
- **Reasons for the Problem**
- Keep a close eye on classes that spend too much time together. Good classes should know as little about each other as possible. Such classes are easier to maintain and reuse.
- **Feature Envy** is when a method uses more public features of another class than it does of its own.

Inappropriate Intimacy

- **Treatment**
- The simplest solution is to use Move Method and Move Field to move parts of one class to the class in which those parts are used. But this works only if the first class truly doesn't need these parts.
- Another solution is to use Extract Class and Hide Delegate on the class to make code relations "official".
- If the classes are mutually interdependent, you should use Change Bidirectional Association to Unidirectional.
- If this "intimacy" is between a subclass and the superclass, consider Replace Delegation with Inheritance.

Incomplete Library Class

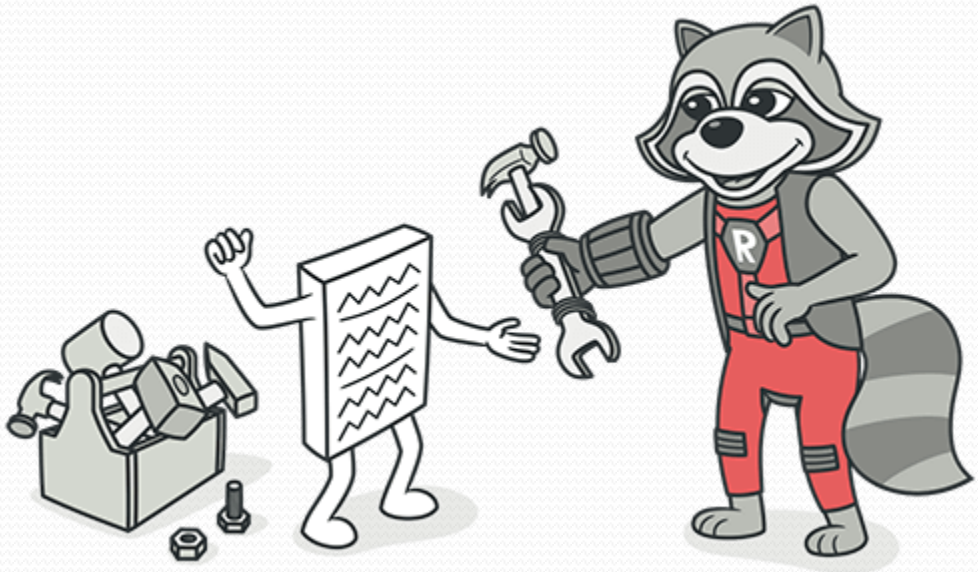
- **Signs and Symptoms**
- Sooner or later, libraries stop meeting user needs. The only solution to the problem – changing the library – is often impossible since the library is read-only.



- **Reasons for the Problem**
- The author of the library hasn't provided the features you need or has refused to implement them.

Incomplete Library Class

- **Treatment**
  - Introduce a few methods to a library class, use Introduce Foreign Method.
  - For big changes in a class library, use Introduce Local Extension.
- **Payoff**
- Reduces code duplication (instead of creating your own library from scratch, you can still piggy-back off an existing one).

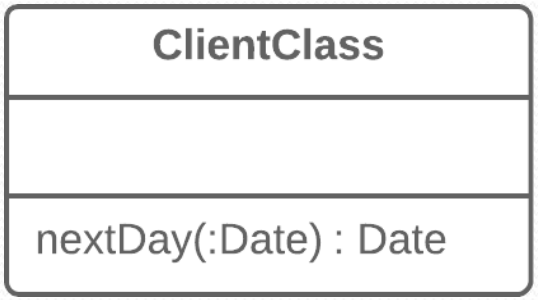


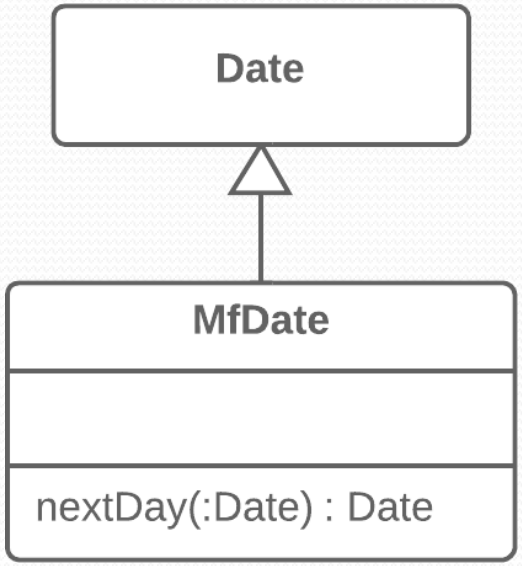
Introduce Foreign Method

- **Problem**
- A utility class doesn't contain the method that you need and you can't add the method to the class.
- **Solution**
- Add the method to a client class and pass an object of the utility class to it as an argument.

Introduce Local Extension

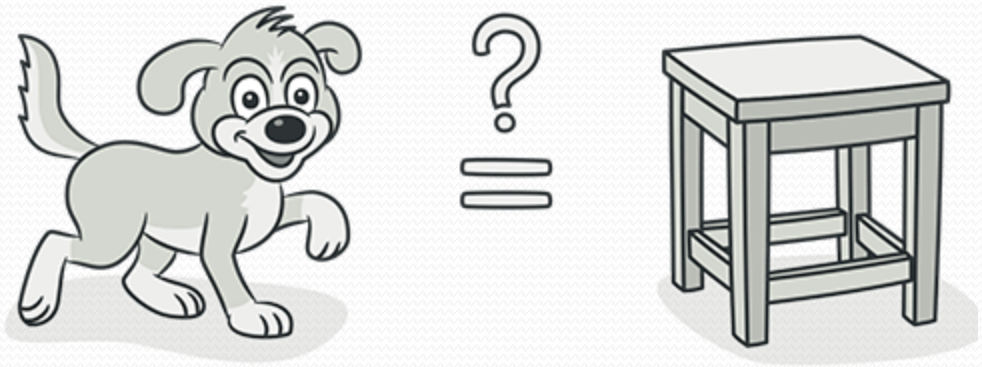
- **Problem**
- A utility class doesn't contain some methods that you need. But you can't add these methods to the class.
- **Solution**
- Create a new class containing the methods and make it either the child or wrapper of the utility class.

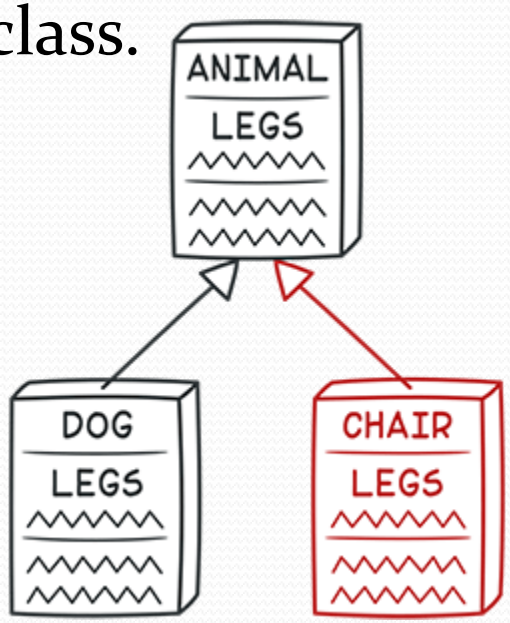




Refused Bequest

- This code smell is a little tricky to detect because this happens when a subclass doesn't use all the behaviors of its parent class. So it's as if the subclass "refuses" some behaviors ("bequest") of its parent class.





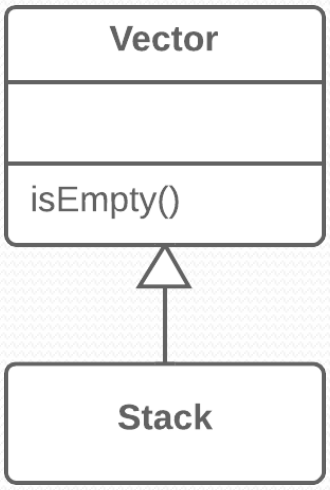
- **Reasons for the Problem**
- Someone was motivated to create inheritance between classes only by the desire to reuse the code in a superclass. But the superclass and subclass are completely different.

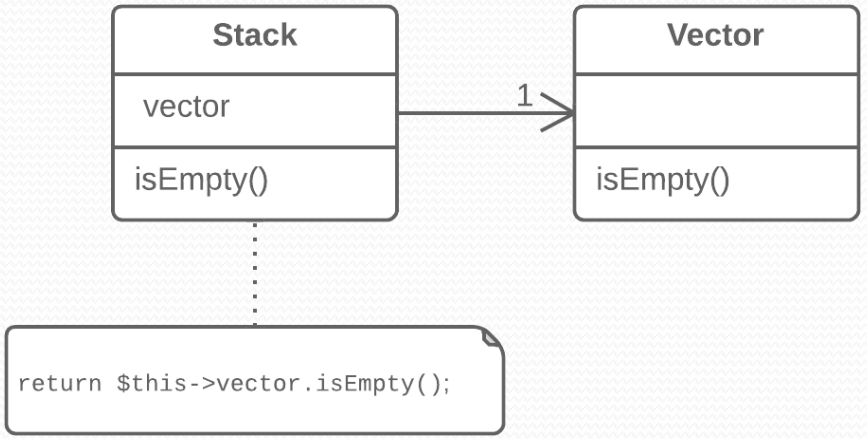
Refused Bequest

- **Treatment**
- If inheritance makes no sense and the subclass really does have nothing in common with the superclass, eliminate inheritance in favor of Replace Inheritance with Delegation.
- If inheritance is appropriate, get rid of unneeded fields and methods in the subclass. Extract all fields and methods needed by the subclass from the parent class, put them in a new subclass, and set both classes to inherit from it (Extract Superclass).

Replace Inheritance with Delegation

- **Problem:** You have a subclass that uses only a portion of the methods of its superclass (or it's not possible to inherit superclass data).
- **Solution:** Create a field and put a superclass object in it, delegate methods to the superclass object, and get rid of inheritance.



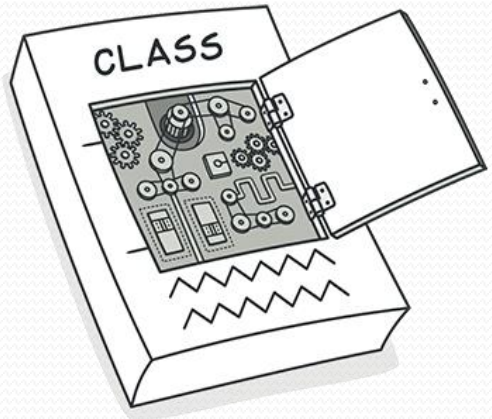


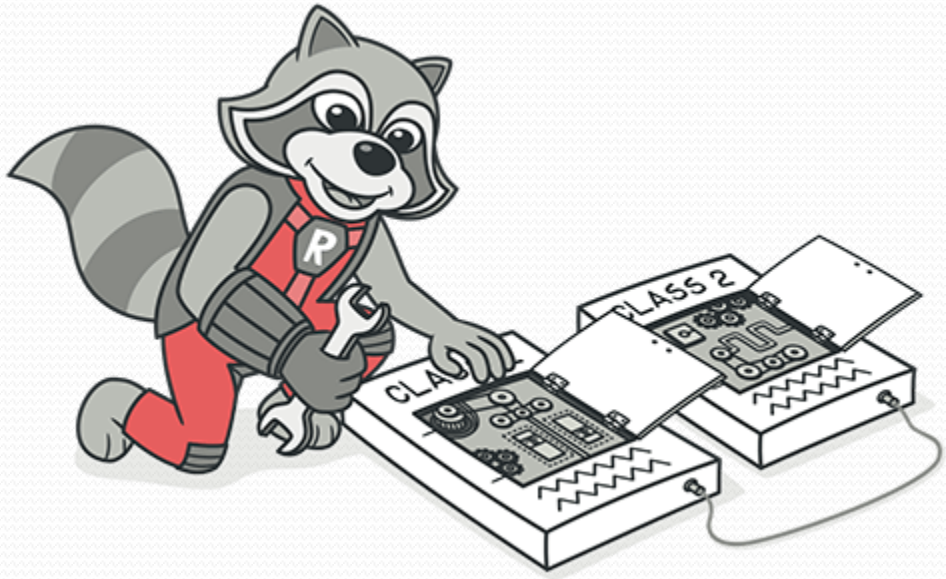
Divergent Change

- This is the case when you find yourself changing the same class for several different reasons. i.e., you are violating the Single Responsibility Principle.
- *Divergent Change* resembles Shotgun Surgery but is actually the opposite smell. *Divergent Change* is when many changes are made to a single class. *Shotgun Surgery* refers to when a single change is made to multiple classes simultaneously.
- **Signs and Symptoms**
- You find yourself having to change many unrelated methods when you make changes to a class. For example, when adding a new product type you have to change the methods for finding, displaying, and ordering products.

Divergent Change

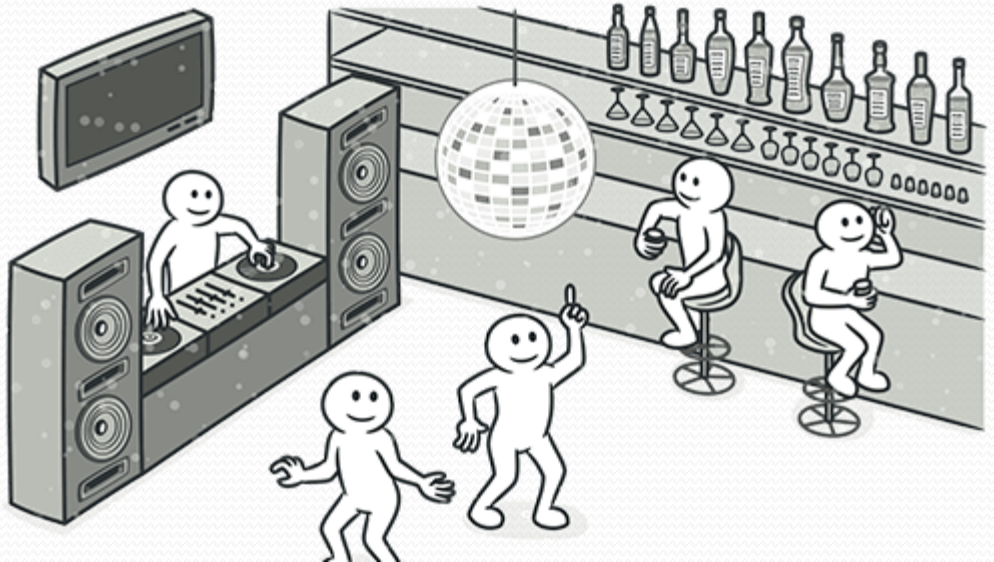
- **Treatment**
- Split up the behavior of the class via Extract Class.
- If different classes have the same behavior, you may want to combine the classes through inheritance (Extract Superclass and Extract Subclass).





Data Clump

- "data clump" is a name given to any group of variables which are passed around together (in a clump) throughout various parts of the program.
- **Signs and Symptoms**
- Sometimes different parts of the code contain identical groups of variables (such as parameters for connecting to a database). These clumps should be turned into their own classes.

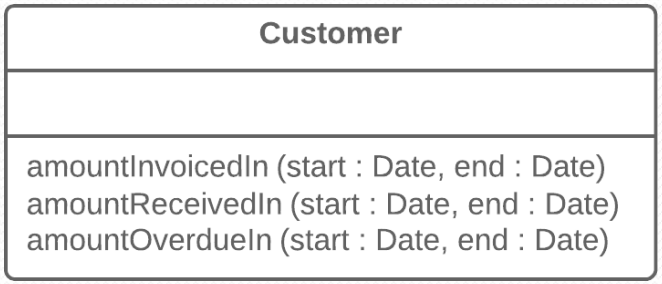


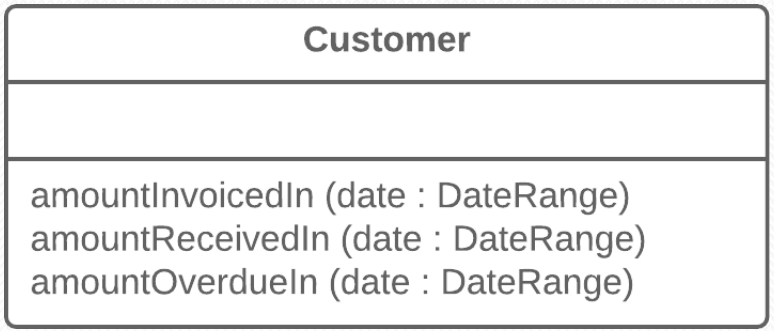
Data Clump

- **Reasons for the Problem**
- poor program structure or "copypasta programming".
- **Treatment**
  - If repeating data comprises the fields of a class, use Extract Class to move the fields to their own class.
  - If the same data clumps are passed in the parameters of methods, use Introduce Parameter Object to set them off as a class.
  - If some of the data is passed to other methods, think about passing the entire data object to the method instead of just individual fields. Preserve Whole Object will help with this.
  - Look at the code used by these fields. It may be a good idea to move this code to a data class.

Introduce Parameter Object

- **Problem**
- Your methods contain a repeating group of parameters.
- **Solution**
- Replace these parameters with an object.





Preserve Whole Object

- **Problem**
- You get several values from an object and then pass them as parameters to a method.

```
int low = daysTempRange.getLow();
int high = daysTempRange.getHigh();
boolean withinPlan =
plan.withinRange(low, high);
```

- **Solution:** Instead, try passing the whole object.

```
boolean withinPlan = plan.withinRange(daysTempRange);
```

Primitive Obsession

- This case is when we use primitives instead of value types for simple tasks. A simple example is a currency: we tend to put it in a float or double, instead of encapsulating it in a value type.
- **Signs and Symptoms**
  - Use of primitives instead of small objects for simple tasks (such as currency, ranges, special strings for phone numbers, etc.)
  - Use of constants for coding information (such as a constant USER_ADMIN_ROLE = 1 for referring to users with administrator rights.)
  - Use of string constants as field names for use in data arrays.

Primitive Obsession

- **Treatment**
  - If you have a large variety of primitive fields, it may be possible to logically group some of them into their own class. Even better, move the behavior associated with this data into the class too. For this task, try Replace Data Value with Object.
  - If the values of primitive fields are used in method parameters, go with Introduce Parameter Object or Preserve Whole Object.
  - When complicated data is coded in variables, use Replace Type Code with Class, Replace Type Code with Subclasses or Replace Type Code with State/Strategy.
  - If there are arrays among the variables, use Replace Array with Object.