Code Refactoring smells (indicators)

* *Duplicated Code* - extract out the common bits into their own method (**extract method**) if code is in same class if two classes duplicate code, consider **extract class** to create a new class to hold the shared functionality.
* *Long Methods* - **extract method**!
* *Large Class* - Class trying to do too much often shows up as too many instance variables.
* *Long Parameter List*  - **replace parameter with method** (receiver explicitly asks sender for data via sender getter method) Example: day month, year, hour minute second ==> date
* *Divergent Change* - If you have a fixed class that does distinctly different things consider separating out the varying code into varying classes **(extract class)** that either subclass or are contained by the non-varying class.
* *Shotgun Surgery* - The smell: a change in one class repeatedly requires little changes in a bunch of other classes. try to **move method** and **move field** to get all the bits into one class since they are obviously highly dependent.
* *Feature Envy* - Method in one class uses lots of pieces from another class. **move method** to move it to the other class.
* *Data Clumps* - Data that's always hanging with each other (e.g. name street zip). Extract out a class (**extract class**) for the data. Will help trim argument lists too since name street zip now passed as one address object.
* *Switch (case) statements* - Use inheritance and polymorphism instead (example of this was in Fowler Chapter 1; this is one of the more difficult refactorings)
* *Lazy Class* - Class doesn't seem to be doing anything. Get rid of it!
  + **collapse heirarchy** if subclasses are nearly vacuous.
    - **inline class** (stick the class' methods and fields in the class that was using it and get rid of original class).
* *Speculative generality* - Class designed to do something in the future but never ends up doing it. Thinking too far ahead or you though you needed this generality but you didn't. like above, **collapse hierarchy**or **inline class**
* *Message chains* - Say you want to send a message to object D in class A but you have to go through B to get C and C to get D. use **hide delagate** to hide C and D in B, and add a method to B that does what A wanted to do with D.
* *Inappropriate Intimacy* - Directly getting in and munging with the internals of another class. To fix this, move methods, inline methods, to consolidate the intimate bits.
* *Incomplete Library Class* - If method missing from library, and we can't change the library, so either: o make this method in your object (**introduce foreign method**) If there is a lot of stuff you want to change: o make your own extension/subclass (**introduce local extension**)
* *Data Class* - We have already talked about this extensively: in data-centric design, there are some data classes which are pretty much structs: no interesting methods. first don't let other directly get and set fields (make them private) and don't have setter for things outsiders shouldn't change look who uses the data and how they use it and move some of that code to the data class via a combination of **extract method** and **move method** (see the Fowler chapter 1 example for several examples of this)
* *Comments* - Comments in the middle of methods are deodorant. You should really refactor so each