Bad Smells in Code:

Duplicated code: Extract Method
- Same code in two sibling classes: *Pull Up Method*
- Similar code in sibling classes: *Form Template Method*
- Same code in unrelated classes: *Extract Class*

Long Method:
- Decompose into small methods (sometimes just 1 line long)
- *Extract Method* on blocks that can be separated out
- May need to *Replace Temp with Query* to enable extraction

Large Classes:
- Tries to do too many different things (too many instance variables/too much code)
- *Extract Class* or *Extract Subclass* to separate out bundles of data/responsibilities

Long Parameter List:
- Better to pass in an object so the method can get data
- *Preserve Whole Object* or *Introduce Parameter Object*

Divergent Change:
- Occurs when class changes in distinct ways for diff reasons
- Responsibilities of class are divergent
- Determine what changes for a single cause, *Extract Class*

Shotgun Surgery:
- Single change causes many little changes to several classes
- *Move Method* and *Move Field* to put changes into one class
- Sometimes best to *Inline Class*

Feature Envy:
- A class does calculations belonging elsewhere
- Put into proper class with *Move Method*

Data Clumps:
- Data clusters together in fields or parameter lists
- *Extract Class* to change clumps into an object
- Shrink parameter lists with *Introduce Parameter Object* or *Preserve Whole Object*

Primitive Obsession:
- Often better to use a class instead of primitive data type
    - Allows range checking, formatting, etc
    - *Replace Data Value with Object*
- If primitive is type code: *Replace Type Code with Class/Subclass/State/Strategy*

Switch Statements:
- Are rare in good OO code
- If switching on type code: *Replace Conditional with Polymorphism* (easier to add subclasses than switches)

Parallel Inheritance Hierarchies:
- When you make a subclass of one class, also make subclass of another. (Special shotgun surgery)
- Eliminate hierarchy by moving data, code to the other
- *Move Method* and *Move Field*

Lazy Class:
- Doesn't do enough to justify its existence
- Eliminate with *Collapse Hierarchy* or *Inline Class*

Speculative Generality:
- You added code for future expansion that never happened
- Remove useless abstract classes with *Collapse Hierarchy*
- Remove unneeded delegation with *Inline Class*
- Remove unused parameters with *Remove Parameter*

Temporary Field:
- An instance variable is set and used only part of the time
- *Extract Class*, moving over orphan variables and methods

Message Chains:
- Client follows chain of referring objects, sending message to last object. Change in intermediaries causes client changes
- *Hide Delegate* on first object in chain to return last object

Middle Man:
- Most methods of a class delegate to another class
- *Remove Middle Man* to talk to delegated object directly

Inappropriate Intimacy:
- Class knows too much about another's private parts
- *Move Method* and *Move Field* to the first class
- Or *Extract Class* to put commonality in a safe place
- *Replace Inheritance with Delegation* if subclass knows too much about its parents

Alternative Classes with Different Interfaces:
- 2+ classes do the same thing with inconsistent interfaces
- Use *Rename Method* and *Move Method* to give them identical interfaces
- If redundant, *Extract Superclass*

Incomplete Library Class:
- Can't use *Move Method* on code you can't change
- *Introduce Foreign Method* into client class (1-2 methods)
- *Introduce Local Extension* to create a subclass

Data Class:
- Class with no behavior, only get and set methods
- *Move Methods* into the data class
- May need to *Extract Method* first

Refused Bequest:
- Subclass doesn't use all methods and data it inherits
- Reorganize class hierarchy with *Push Down Method* and *Push Down Field* to create sibling with unused behavior
- If subclass doesn't support superclass interface: *Replace Inheritance with Delegation*

Comments that Explain Bad Code:
- *Extract Method* on commented blocks of code
- *Rename Method* to make purpose clearer

Some Refactoring Methods:

Form Template Method:

- Subclasses implement algorithms that contain similar steps
- Move structure and identical steps to superclass, leave implementation of the differing steps in the subclasses

Replace Temp with Query:

- You store expression in local variable for later use
- Move expression into a new method that returns the result

Preserve Whole Object:

- Use same object to get several results and pass into method
- Pass the object as a parameter, method can sort it out

Inline Class:

- A class does almost nothing
- Move all features from that class into another one

Replace Type Code with Class/Subclass/State/Strategy:

- You have a coded type variable that affects behavior
- Replace type with a new state object

Replace Conditional with Polymorphism:

- Conditional that performs actions based on properties
- Create subclasses for each branch of the conditional

Hide Delegate:

- Client requests result from object C via Client -> A -> B -> C
- Create new method in class A that delegates the call to C, now client doesn't need or know about the other classes

Remove Middle Man:

- Class has many methods that delegate to other objects
- Remove methods, make client call end methods directly

Replace Inheritance with Delegation:

- Subclass only uses a portion of its superclass (or not possible to have a superclass)
- Create a field and put a superclass object in it, delegate methods to the superclass object and remove inheritance.

Introduce Foreign Method:

- Utility class doesn't contain method you need, can't add method to the class directly
- Add the method to a client class, pass object of utility class as an argument

Introduce Local Extension:

- Utility class doesn't contain method you need, can't add method to the class directly
- Create a new class containing the methods, make it either the child or wrapper of the utility class