

Code Smells

- a *code smell* is an indicator that the design is likely to cause trouble later
 - harder to change
 - easier to break
 - harder for someone else to read
- something *sniffable* – quick to spot
- not a bug – the code works and produces correct output
- not inherently problematic – warning signs worth investigating rather than the problem itself
- no single official list – many overlapping catalogs exist
 - Fowler and Beck, *Refactoring* (1999, 2nd ed 2018)
 - Jerzyk and Madeyski (2023), [codesmells.org](https://www.ibm.com/think/topics/code-smells)
 - <https://www.ibm.com/think/topics/code-smells>

Lexical Abusers

- mysterious name
 - a name that doesn't reveal what something holds or does

```
public boolean chk ( Card c, TableauColumn col ) { ... }  
public boolean canPlace ( Card card, TableauColumn column ) { ... }
```

- magic number
 - an unexplained literal number embedded in code

```
if ( card.getRank().getValue() == 13 ) { ... }  
if ( card.getRank() == Rank.KING ) { ... }
```

Bloaters

- long method
 - a method you have to scroll through – it does too many distinct things to describe in one sentence

```
processOrder() validates the order, calculates the total, applies a  
discount, charges payment, and sends a confirmation email
```

- large class
 - a class you have to scroll through – it has too many responsibilities to describe in one sentence

```
Order validates itself, saves itself to a database, sends  
notification emails, and generates a PDF invoice
```

- long parameter list
 - more than two parameters

Bloaters

- data clumps
 - the same small group of variables shows up repeatedly

```
public void printLabel ( String street, String city, String zip ) { ... }  
public void validateAddress ( String street, String city, String zip ) { ... }  
  
public void printLabel(Address address) { ... }  
public void validateAddress(Address address) { ... }
```

Dispensables

- duplicate code
 - the same logic written more than once – a fix in one place risks being missed in the other

both `Order.calculateDiscount()` and a method in `Invoice` contain the same discount computation

- dead code
 - code that is no longer called from anywhere
- comments as deodorant
 - comments added to cover up for code smells – explain a confusing piece of code rather than a better fix

```
// check if the customer qualifies for free shipping
if (o.getTotal() > 50 && o.getCustomer().getYears() > 1
    && !o.getItems().isEmpty()) { ... }

if (qualifiesForFreeShipping(order)) { ... }
```

25

Refactoring

- *refactoring* is changing a program's internal structure without changing its observable behavior
 - not fixing bugs
 - not adding new features
- only safe when there's a passing test suite already in place
 - passing the tests again after refactoring is what confirms that the refactoring didn't break anything
- small steps, verified often
 - not one big rewrite

Obfuscators

- complicated boolean expression
 - convoluted conditionals, especially involving double negatives or many `&&`, `||`, `!`

```
if (!(order.isCancelled() || !order.isPaid()) &&
    order.getTotal() > 0) { ... }

boolean readyToShip = order.isPaid() && !order.isCancelled()
    && order.getTotal() > 0;
if (readyToShip) { ... }
```

Lexical Abusers

- mysterious name
 - a name that doesn't reveal what something holds or does
- magic number
 - an unexplained literal number embedded in code

Bloaters

- long method
 - a method you have to scroll through – it does too many distinct things to describe in one sentence
- large class
 - a class you have to scroll through – it has too many responsibilities to describe in one sentence
- long parameter list
 - more than two parameters
- data clumps
 - the same small group of variables shows up repeatedly

Dispensables

- duplicate code
 - the same logic written more than once – a fix in one place risks being missed in the other
- dead code
 - code that is no longer called from anywhere
- comments as deodorant
 - comments added to cover up for code smells – explain a confusing piece of code rather than a better fix

Obfuscators

- complicated boolean expression
 - convoluted conditionals, especially involving double negatives or many &&, ||, !

Refactorings

smell	refactoring
mysterious name	give it a new descriptive name
magic number	replace with a named constant
long method	pull each distinct responsibility into its own method which the original method calls in sequence
large class	split the class along its distinct responsibilities
long parameter list	reduce the number of parameters – passing an existing object instead of several extracted values, bundle a recurring group of values into an object, let the method compute or look up a value itself, replace a boolean parameter with two separate well-named methods, group methods sharing a clump of parameters into a class with those parameters as instance variables
duplicate code	extract the shared logic into one place and call it multiple times
dead code	delete it

Refactorings

smell	refactoring
comments as deodorant	give the confusing code a clear name or pull it out into a small well-named method so the comments become unnecessary
data clumps	bundle the values into a single object
complicated boolean expression	pull the expression out into a well-named variable or method