Bad smells of code

Lecture#10

Haroon Zafar

NUCES, Peshawar Campus

Bad smells in code

- Martin Fowler, Refactoring: Improving the design of existing code. Addison Wesley.

Introduction

- Bad Smells=“Bad smelling code”
  - Indicate that your code is ripe for refactoring

- Refactoring is about
  - How to change code by apply refactoring.

- Bad smells are about
  - When to modify it

Bad Smells

- Allow us to identify
  - What needs to be change in order to improve the code
- A recipe book to help us to choose the right refactoring path
  - No precise criteria
  - More to give an intuition and indications
- Goal: a more “Habitable” code.

Habitable code

- Habitable code is in which developer feels at home while developing a software system.

  (even when the code was not written by them)

- Symptoms of inhabitable code include
  - Overuse of abstraction or inappropriate compression.
- Habitable code should be easy to read, easy to change
- Software needs to be habitable because it always has to change.

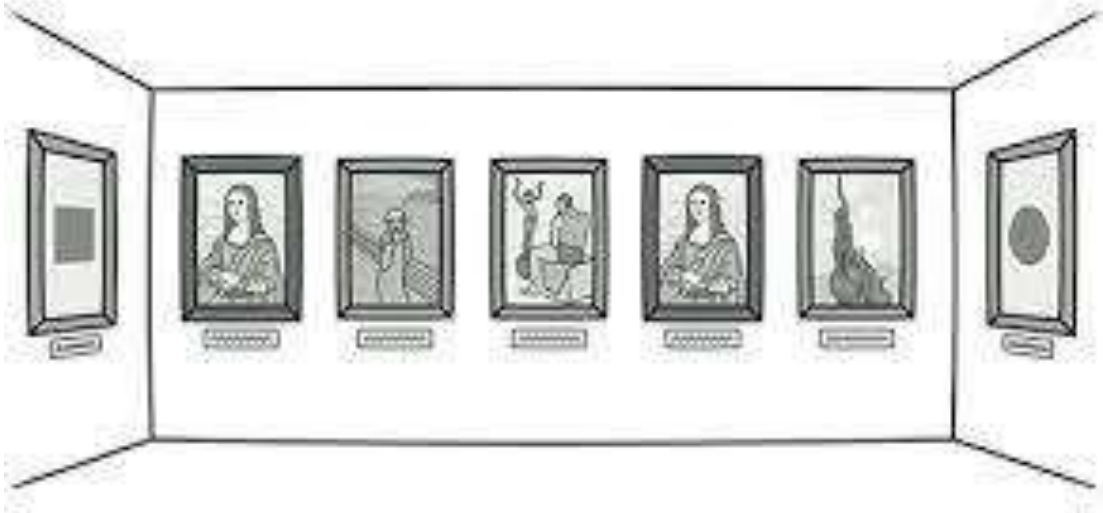
Bad smells Classification

- Bloaters (Top crime)
- Class/method organization
- Lack of loose coupling or cohesion
- Too much or too little delegations
- Non Object Oriented control or data structure
- Mixed methods

Alternative Classification

- Bloaters
- Object Oriented abusers
- Change preventers
- Dispensables
- Couplers
- Other smells

Bloaters/Top crime



Code duplication

Code Duplication

- Duplicated code is the number 1 in the stink parade:

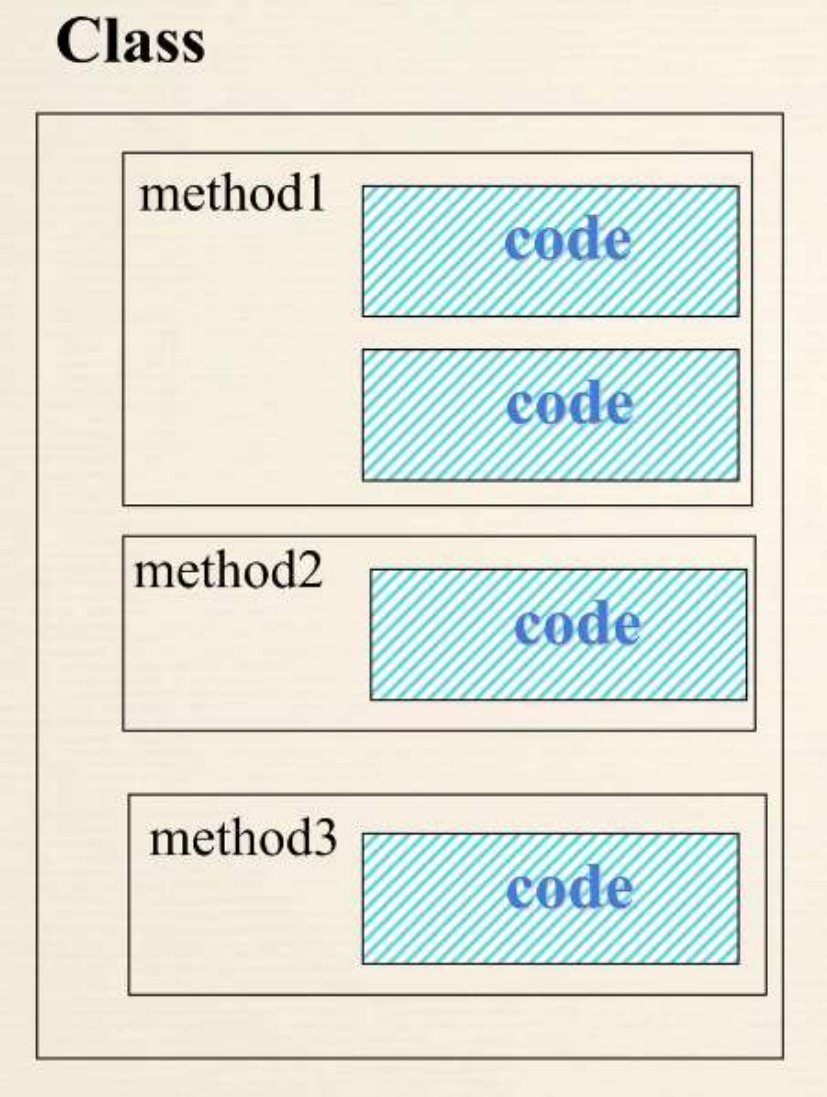
  We have duplicated code when we have the same code structure in more then one place

- Why is duplicated code bad?

Code duplication: Example

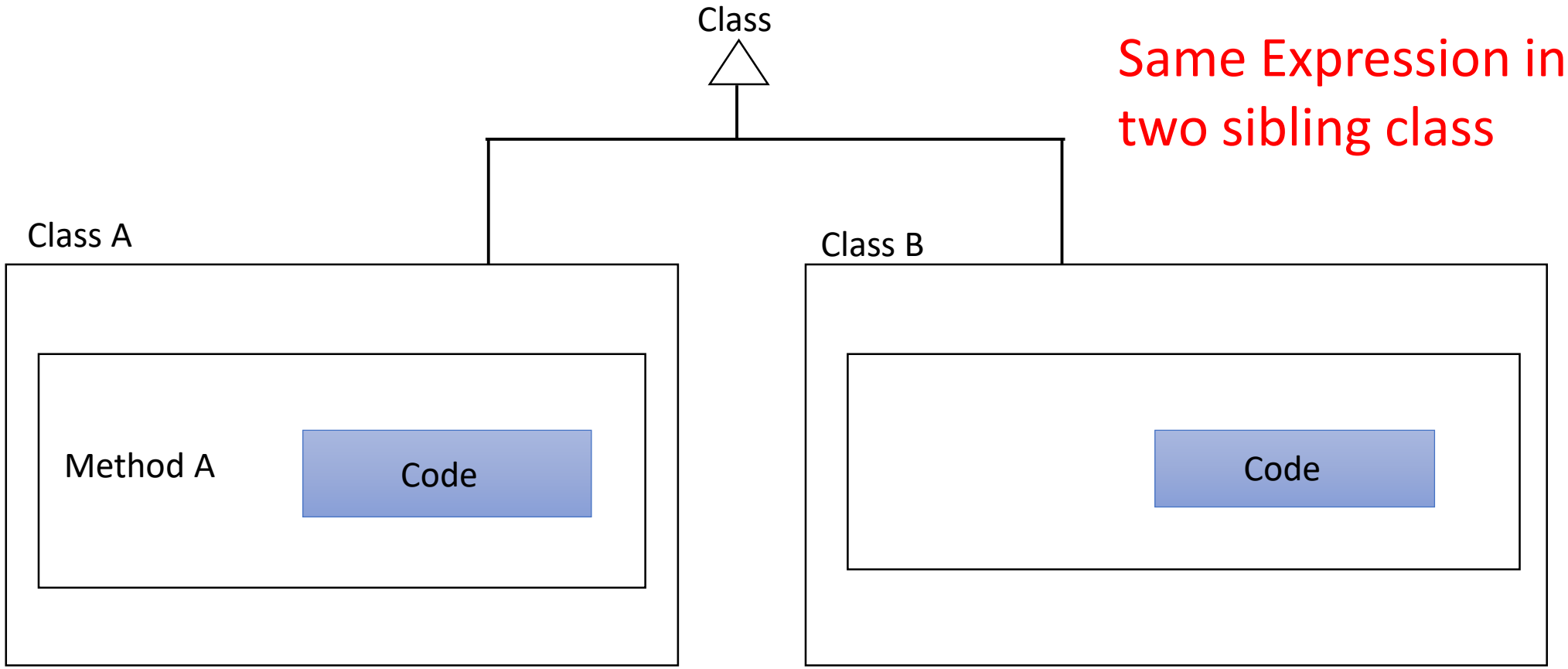
```
public double ringSurface(r1,r2) {
        // calculate the surface of the first circle
        double surf1 = bigCircleSurface(r1);
        // calculate the surface of the second circle
        double surf2 = smallCircleSurface(r2);
        return surf1 - surf2;
}
private double bigCircleSurface(r1) {
        pi = 4* ( arctan 1/2 + arctan 1/3 );
        return pi*sqr(r1);
}
private double smallCircleSurface(r2) {
        pi = 4* ( arctan 1/2 + arctan 1/3 );
        return pi*sqr(r2);
}
```

Code duplication: Example 2



Same expression in two or more methods of the same class

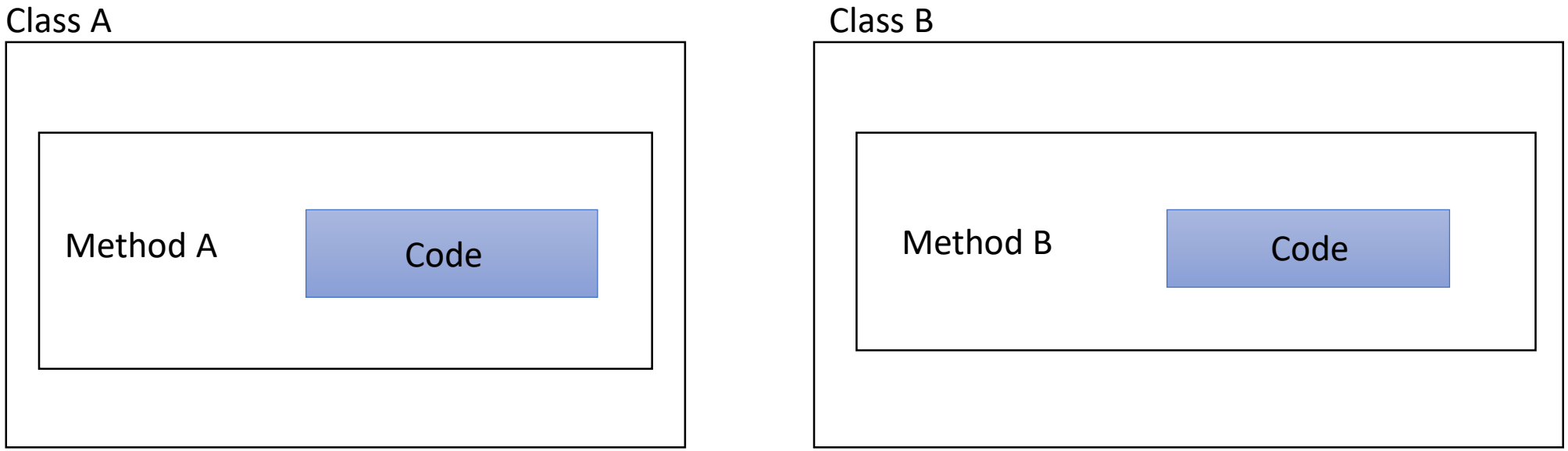
Code Duplication: Example 3



Same Expression in two sibling class

Code Duplication: Example 4

Same expression in
two unrelated class



Class/Method Organization

- Large class
- long Method
- Long Parameter List
- Lazy class
- Data class

Large Class

- A large class is a class that is trying to do too much tasks
- Often show up as too many instance variables
- Use Extract class or Extract subclass to bundle variable
  - Choose variable that belong together in the extracted class
  - Common prefixes and suffixes may suggest which ones may go together e.g depositAmont and depositCurrency

Large Class

- A class may also be too large in the sense that it has too much code
  - Likely some code inside the class is duplicated
  - Solve it by extracting the duplicated code in separate methods using ***Extract Method***
  - Or move part of the code to a new class, using ***Extract Class*** or ***Extract Subclass***
  - If need be, move existing or extracted methods to another class using Move Method

Long Parameter List

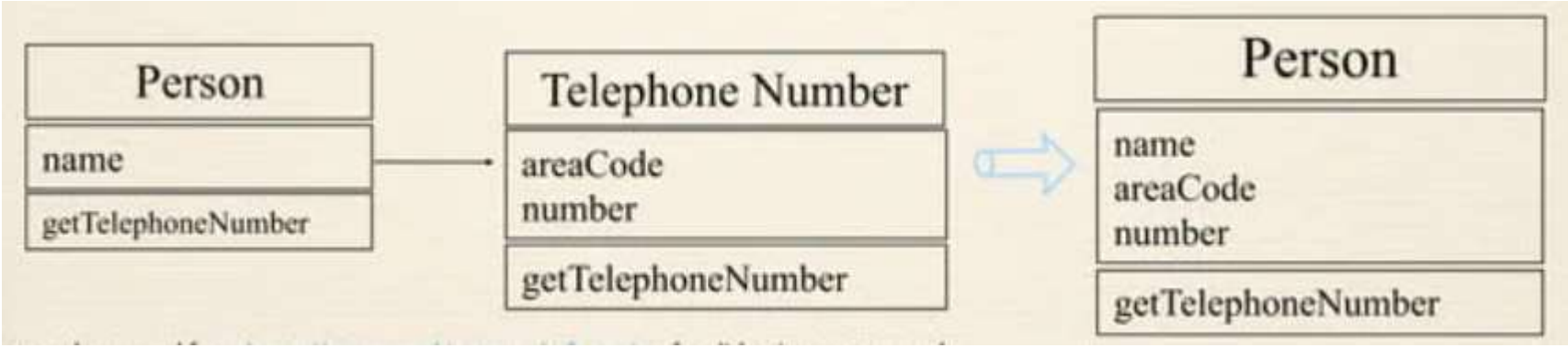
- In procedural programming languages, we pass as parameters everything needed by a subroutine
- Because the only alternative is ***global variable***
- With objects you don’t pass everything the method needs

Long Parameter List

- Long Parameter list are hard to understand
- Pass only the needed number of variables
- Use Replace parameter with methods when you can get the data in one parameter by making a request of an object you already known about

Lazy Class

- Each class cost money (and brain cells) to maintain and understand
- A class that isn't doing enough to pay for itself should be eliminated
- It might be a class that was added because of changes that were planned but not made
- Use Collapse hierarchy or Inline Class to eliminated the class



Data Class

- Classes with just fields, getter, setter and nothing else
- If there are public fields, use Encapsulation Field
- For fields that should not be changed use remove setting method

Long Method

- Object programs live best and longest with short methods
- New OO programs feel that OO programs are endless sequence of delegation
- Older languages carried an overhead in subroutine calls which deterred people from that small methods
  - There is still an overhead to the reader of the code because you have to switch context to see what subprocedure does.

- Important to have a good name for small method
  - Rename method

Long Method: Example

```
void printOwing() {
        Enumeration e = _orders.elements();
        double outstanding = 0.0;
        // Print banner
        System.out.println("*******************");
        System.out.println("****** Customer ******");
        System.out.println("*******************");
        // Calcultate outstanding
        While (e.hasMoreElements()) {
                Order each = (Order) e.nextElement();
                outstanding += each.getAmount();
        }
        // Print details
        System.out.println("name: " + _name);
        System.out.println("amount" + outstanding);
}
```

Long Method: Refactoring Patterns

- 99% of the time , all we have to do to shorten a method is Extract Method
  - Find parts of the methods that seems to go together nicely and extract them into a new method
- It can lead to problems…
  - Many temps: use Replace Temp with query
  - Long lists of parameters can be slimmed down with Introduce Parameter Object

Long Method: Refactoring Patterns

- But how to identify the clumps of the code to extract?
- Look for comments…
  - A block od statements with a comments that tells you what it is doing can be replaced b a method whose based on the comments
- Loops also give for extraction…
  - Extract the loop and code within the loop into its own method.

Long Method: Example revisited

```
void printOwing() {
        Enumeration e = _orders.elements();
        double outstanding = 0.0;
        // Print banner
        System.out.println("*******************");
        System.out.println("****** Customer ******");
        System.out.println("*******************");
        // Calcultate outstanding
        While (e.hasMoreElements()) {
                Order each = (Order) e.nextElement();
                outstanding += each.getAmount();
        }
        // Print details
        System.out.println("name: " + _name);
        System.out.println("amount" + outstanding);
}
```

Long Method Example revisited

```
void printOwing() {
        Enumeration e = _orders.elements();
        double outstanding = 0.0;
        // Print banner
        System.out.println("**********");
        System.out.println("***** Customer *****");
        System.out.println("**********");
        // Calcultate outstanding
        While (e.hasMoreElements()) {
                Order each = (Order) e.nextElement();
                outstanding += each.getAmount();
        }
        // Print details
        System.out.println("name: " + _name);
        System.out.println("amount" + outstanding);
}
```

Long Method Example revisited

```
void printOwing() {
        Enumeration e = _orders.elements();
        double outstanding = 0.0;
        printBanner();
        // Calcultate outstanding
        While (e.hasMoreElements()) {
                Order each = (Order) e.nextElement();
                outstanding += each.getAmount();
        }
        // Print details
        System.out.println("name: " + _name);
        System.out.println("amount" + outstanding);
}
void printBanner() {
        System.out.println("*******************");
        System.out.println("****** Customer *****");
        System.out.println("*******************");
}
```

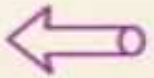
1.
Extract Method

Long Method Example revisited

```
void printOwing() {
        Enumeration e = _orders.elements();
        double outstanding = 0.0;
        printBanner();
        // Calcultate outstanding
        While (e.hasMoreElements()) {
                Order each = (Order) e.nextElement();
                outstanding += each.getAmount();
        }
        // Print details
        System.out.println("name: " + _name);
        System.out.println("amount" + outstanding);
}
void printBanner() {
        System.out.println("**********");
        System.out.println("****** Customer *****");
        System.out.println("**********");
}
```

Long Method Example revisited

```
void printOwing() {
        Enumeration e = _orders.elements();
        double outstanding = 0.0;
        printBanner();
        // Calcultate outstanding
        While (e.hasMoreElements()) {
                Order each = (Order) e.nextElement();
                outstanding += each.getAmount();
        }
        printDetails(outstanding);
}
void printDetails(double outstanding) {
        System.out.println("name: " + _name);
        System.out.println("amount" + outstanding);
}
void printBanner() { ... }
```



2.
Extract Method
Using local variable

Long Method Example revisited

```
void printOwing() {
        Enumeration e = _orders.elements();
        double outstanding = 0.0;
        printBanner();
        // Calcultate outstanding
        While (e.hasMoreElements()) {
                Order each = (Order) e.nextElement();
                outstanding += each.getAmount();
        }
        printDetails(outstanding);
}
void printDetails(double outstanding) {
        System.out.println("name: " + _name);
        System.out.println("amount" + outstanding);
}
void printBanner() { ... }
```

Long Method Example revisited

```
void printOwing() {
        printBanner();
        double outstanding = getOutstanding();
        printDetails(outstanding);
}
double getOutstanding() {
        Enumeration e = _orders.elements();
        double result = 0.0;
        While (e.hasMoreElements()) {
                Order each = (Order) e.nextElement();
                result += each.getAmount();
        }
        return result;
}
void printDetails(double outstanding) {…}
void printBanner() { … }
```

3.
Extract Method

Reassigning a Local Variable