Extract Class

This refactoring involves taking part of a class and moving it to a separate class

Smell Code

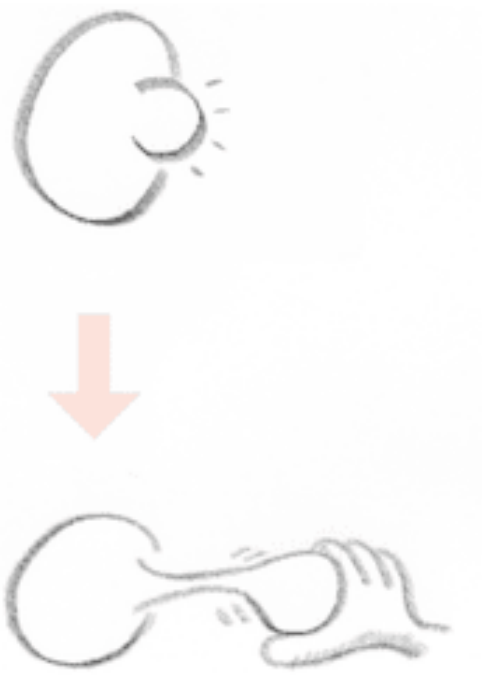
The Book has two responsibilities.

```
class Book:
  def __init__(...)
    _title =
    _isbn =
    _authorName =
    _authorEmail =

=>

class Book:
  def __init__(...)
    _title =
    _isbn =
    _author = Author

class Author:
   ...
```



Code Smell 2

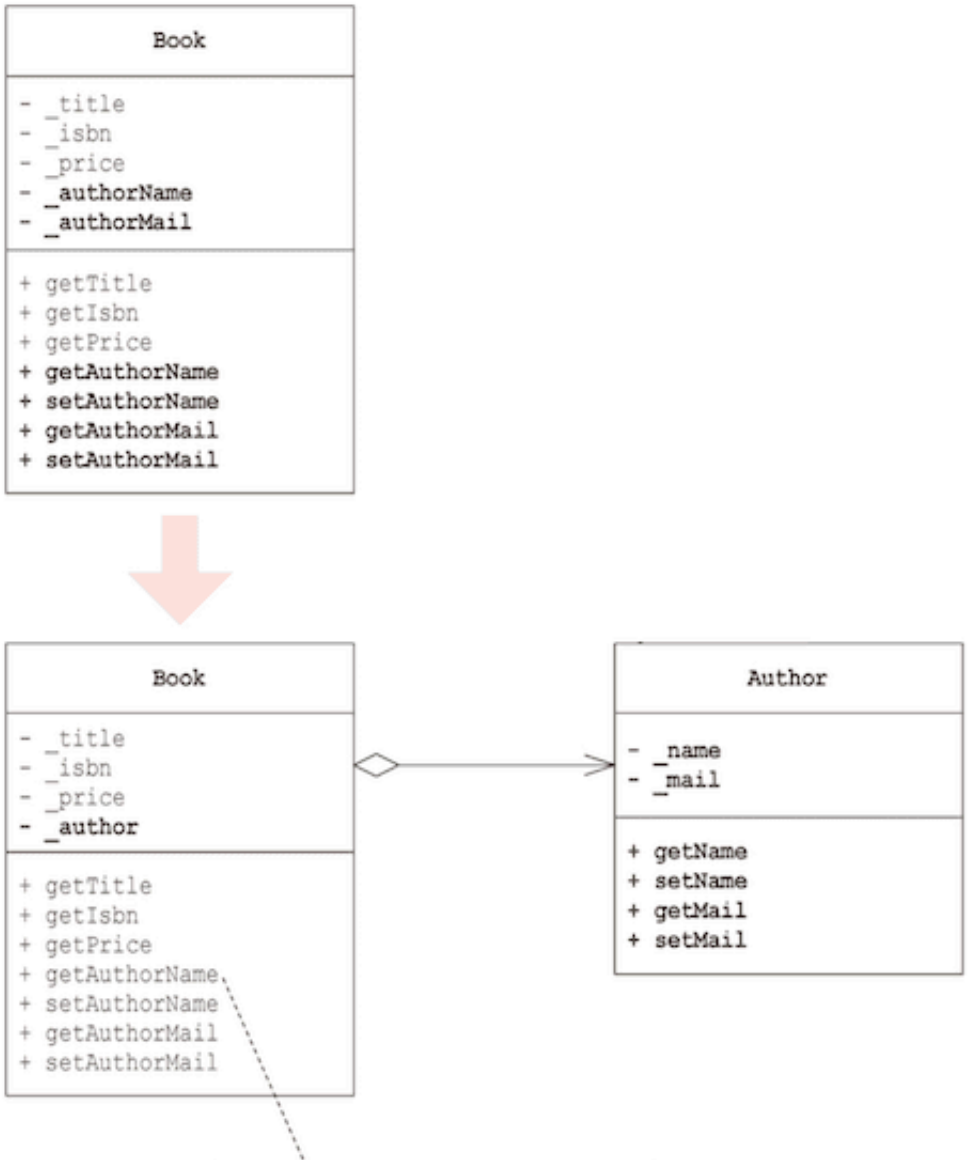
Refactor the user of the Book.

```
class Book:
  _authorName ...
  def getAuthorName(): return _authorName

=>

class Book:
  _author = Author("Great Author")
  def getAuthorName(): return _author.getName()
```

The Book has no responibility of the Author.



Example: Book

Before:

```
class Book:
    """Book class with embedded author information (before refactoring)"""

    def __init__(self, title, isbn, price, author_name, author_mail):
        self.title = title
        self.isbn = isbn
        self.price = price
        self.author_name = author_name
        self.author_mail = author_mail
...
```

???

```
def main():
    refactoring = Book(
        "Refactoring: improving the design of existing code",
        "ISBN0201485672",
        "$44.95",
        "Martin Fowler",
        "fowler@acm.org")
```

The to_xml method generates the book info in the XML format.

We ask a question: is it OK to deal with the author and book?

```
def to_xml(self):
    author = self.tag("author",
                      self.tag("name", self.or_name) +
                      self.tag("mail", self.or_mail))
    book = self.tag("book",
                    self.tag("title", self.title) +
                    self.tag("isbn", self.isbn) +
                    self.tag("price", self.price) +
                    author)
    return book
```

After:

```
class Author:
    """Author class (extracted from Book)"""

    def __init__(self, name, mail):
        self.name = name
        self.mail = mail

class Book:
    """Book class using composition with Author (after refactoring)"""

    def __init__(self, title, isbn, price, author_name, author_mail):
        self.title = title
        self.isbn = isbn
        self.price = price
        self.author = Author(author_name, author_mail)

    def get_author_name(self):
        return self.author.get_name()
```

The to_xml method should be refactored too.

```
def to_xml(self):

    author = self.tag("author",
        self.tag("name", self.or_name) +
        self.tag("mail", self.or_mail))
=>
    author = self.tag("author",
        self.tag("name", self.author.get_name()) +
        self.tag("mail", self.author.get_mail()))
    ...
    return book
```

Tip - The importance of API

We don't change anything from the caller.

```
def get_author_name(self):
    return self.author_name
=>
def get_author_name(self):
    return self.author.get_name()
---
name = self.get_author_name() # no change
```

Tip - Single direction link

- After the refactoring, we may want to make a reverse link from the Author to the Book.
- However, it makes code too complicated, and hard to manage.
- Keep the connection as one direction, not double direction.

Unit Test

Run the test to check the before and after the refactoring.

```
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Inline Class: Reverse of Extracting Class

- When a class is too short, we can embed the class using the inline class refactoring.
- With inline class, we can reduce the number of classes.

Discussion

Benefits of Extract Class refactoring

1. **Single Responsibility** - each class has one clear purpose
2. **Better organization** - related data and methods are grouped together
3. **Easier testing** - smaller classes are easier to test
4. **Improved reusability** - extracted class can be used elsewhere

Code Smell

- **Class is too large** - too many methods and fields
- **Multiple responsibilities** - class violates Single Responsibility Principle
- **Groups of related data** - data that always changes together
- **Subset of methods** - some methods only work with certain fields
- **Hard to understand** - class complexity makes it difficult to maintain