OOP via Python: Session 04

Stephen Leach, Oct 2021

Organisation

Organisation - a big part of OOP's success

- Grouping of methods into classes, typically enforced
- One class per file, typically not enforced
- Grouping classes under a path-like "packages"
- Packages typically mapped to folders

Python - typically less organised

- Python provides modules and packages to group content
- Methods are grouped under classes (enforced)
- There are typically many classes in a module (file), though
- Python developers appear to mainly subscribe to the view that stuffing dozens of classes into a single file in some wacky order and then peppering the code with singletons and functions is a Good Thing
- Ooookay

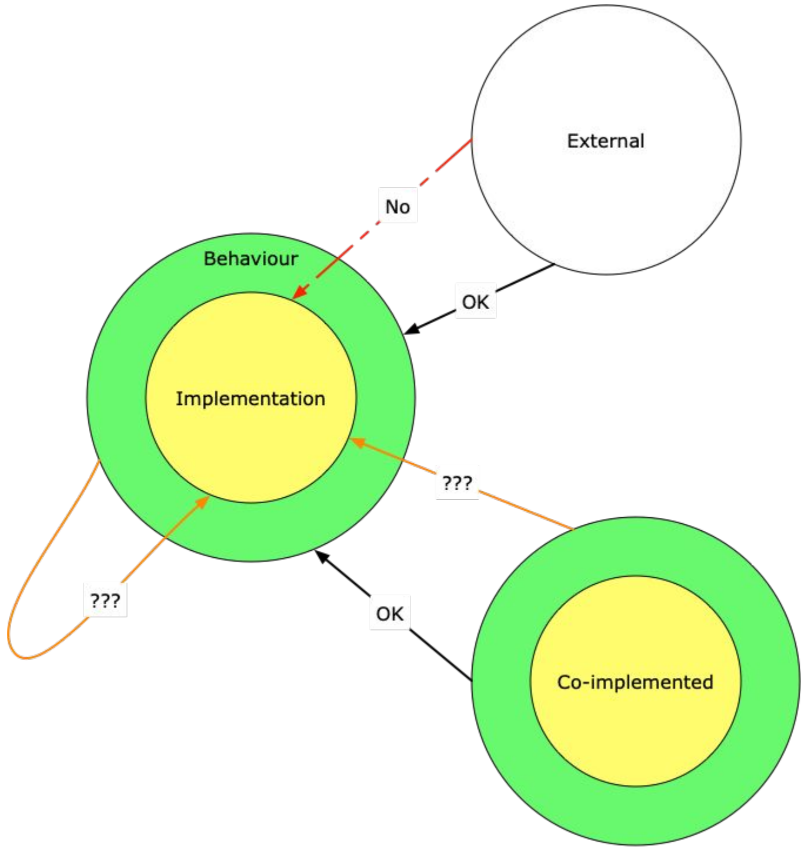
Reflection on Organisational Benefits

- Organisation tends to be imposed by the language and/or toolchain
- Removing individual "whimsical" variation is usually welcomed by mission-focussed teams
- We won't dwell further on this topic because although it is a very important insight, in practice we can't make much use of it as we are very constrained by the platform

Public vs Non-Public

Three notions

- Behaviour vs Implementation
- Access Control
- Scope



Public vs Non-Public

.Vision()

The set of public methods represent the conceptual behaviour of an object

Non-public methods are implementation artefacts that use the same dispatch mechanism as public methods

._reality()

Python has no set way to distinguish public from private methods

Most languages treat "public", "protected" and "private" as access controls, which is strictly speaking an orthogonal concern.

Luckily Python does not suffer from this problem

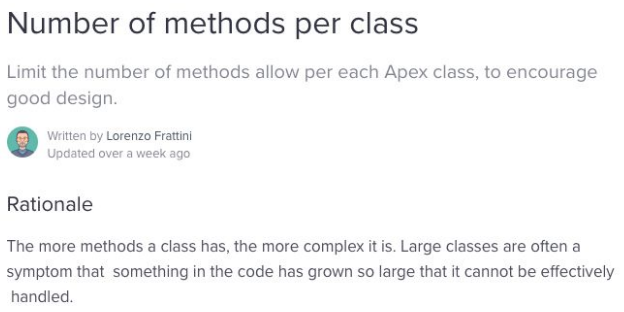
Underscore

- _foo is hidden from the listing available in help and hidden from `import *`
- __bar is name-mangled into _{classname}__bar

```
>>> class Piffle:
...   def __hello(self):
...     print('Found me')
...
>>> Piffle().__hello()
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
AttributeError: 'Piffle' object has no
attribute '__hello'
>>>
>>>
>>> Piffle()._Piffle__hello()
Found me
>>>
```

Core vs Non-Core

Questionable Advice?



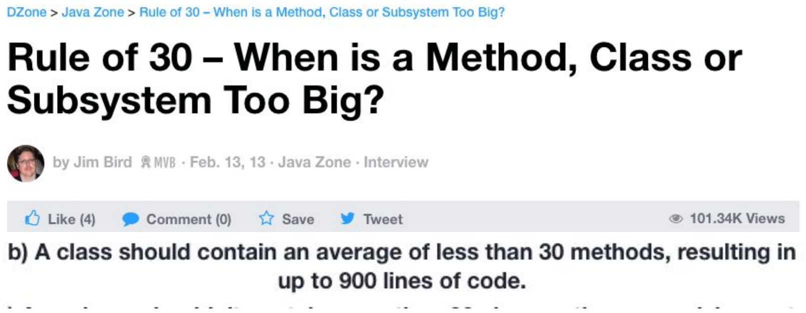
Number of methods per class

Limit the number of methods allow per each Apex class, to encourage good design.

Written by Lorenzo Frattini
Updated over a week ago

Rationale

The more methods a class has, the more complex it is. Large classes are often a symptom that something in the code has grown so large that it cannot be effectively handled.



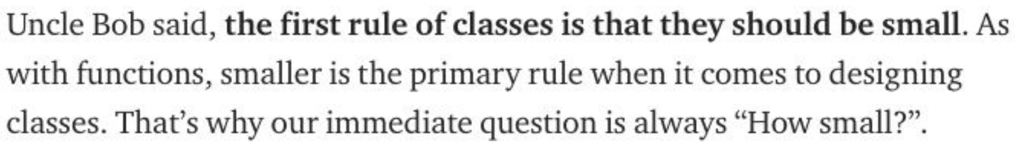
DZone > Java Zone > Rule of 30 – When is a Method, Class or Subsystem Too Big?

Rule of 30 – When is a Method, Class or Subsystem Too Big?

by Jim Bird MVB · Feb. 13, 13 · Java Zone · Interview

Like (4) Comment (0) Save Tweet 101.34K Views

b) A class should contain an average of less than 30 methods, resulting in up to 900 lines of code.



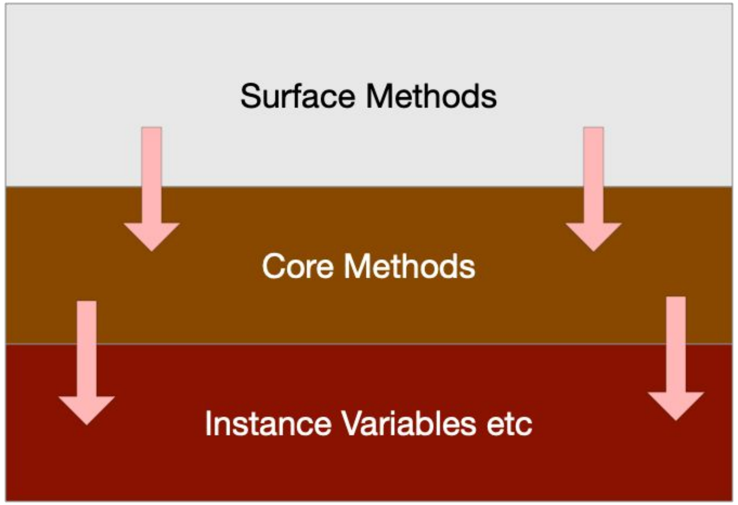
Uncle Bob said, **the first rule of classes is that they should be small**. As with functions, smaller is the primary rule when it comes to designing classes. That’s why our immediate question is always “How small?”.

Example java.String

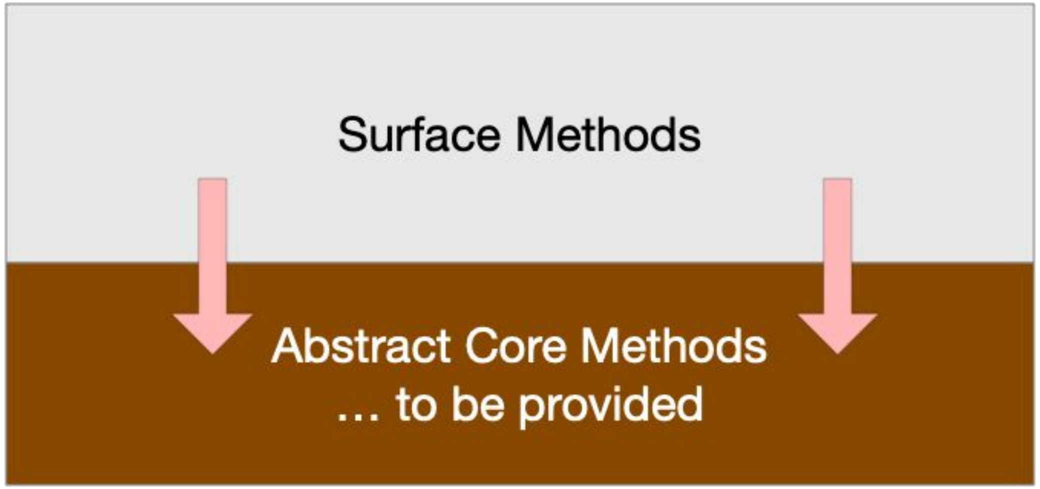
```
char charAt(int index)
int codePointAt(int index)
int codePointBefore(int index)
int codePointCount(int beginIndex, int endIndex)
int compareTo(String anotherString)
int compareToIgnoreCase(String str)
String concat(String str)
boolean contains(CharSequence s)
boolean contentEquals(CharSequence cs)
boolean contentEquals(StringBuffer sb)
static String copyValueOf(char[] data)
static String copyValueOf(char[] data, int offset, int count)
boolean endsWith(String suffix)
boolean equals(Object anObject)
boolean equalsIgnoreCase(String anotherString)
static String format(Locale l, String format, Object... args)
static String format(String format, Object... args)
byte[] getBytes()
byte[] getBytes(Charset charset)
byte[] getBytes(String charsetName)
void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
int hashCode()
int indexOf(int ch)
int indexOf(int ch, int fromIndex)
int indexOf(String str)
int indexOf(String str, int fromIndex)
String intern()
boolean isEmpty()
static String join(CharSequence delimiter, CharSequence... elements)
static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)
int lastIndexOf(int ch)
int lastIndexOf(int ch, int fromIndex)
int lastIndexOf(String str)
int lastIndexOf(String str, int fromIndex)
int length()
boolean matches(String regex)
int offsetByCodePoints(int index, int codePointOffset)
boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)
boolean regionMatches(int toffset, String other, int ooffset, int len)
String replace(char oldChar, char newChar)
String replace(CharSequence target, CharSequence replacement)
String replaceAll(String regex, String replacement)
String replaceFirst(String regex, String replacement)
String[] split(String regex)
String[] split(String regex, int limit)
boolean startsWith(String prefix)
boolean startsWith(String prefix, int toffset)
CharSequence subSequence(int beginIndex, int endIndex)
String substring(int beginIndex)
String substring(int beginIndex, int endIndex)
char[] toCharArray()
String toLowerCase()
String toLowerCase(Locale locale)
String toString()
String toUpperCase()
String toUpperCase(Locale locale)
String trim()
static String valueOf(boolean b)
static String valueOf(char c)
static String valueOf(char[] data)
static String valueOf(char[] data, int offset, int count)
static String valueOf(double d)
static String valueOf(float f)
static String valueOf(int i)
static String valueOf(long l)
static String valueOf(Object obj)
```

Wot?! No reverse??? 😡

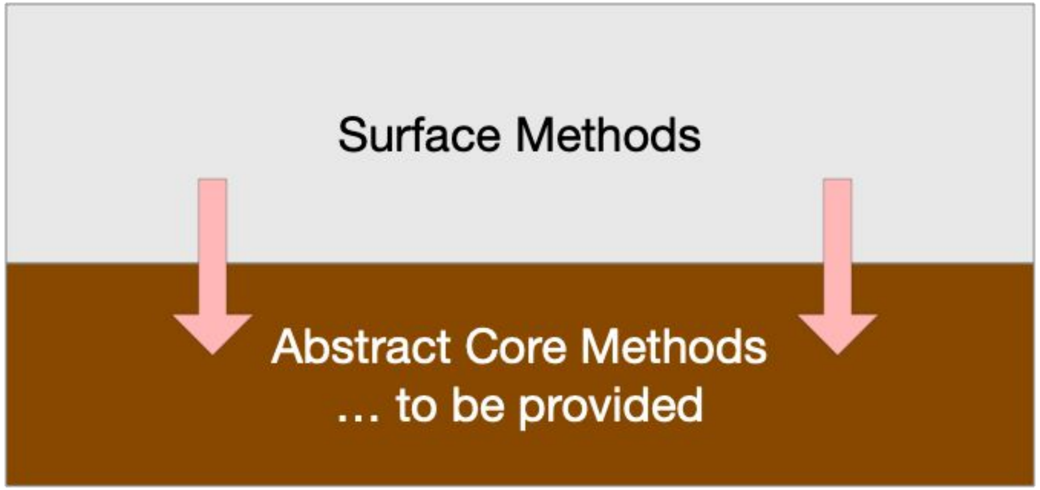
Implementation Layers

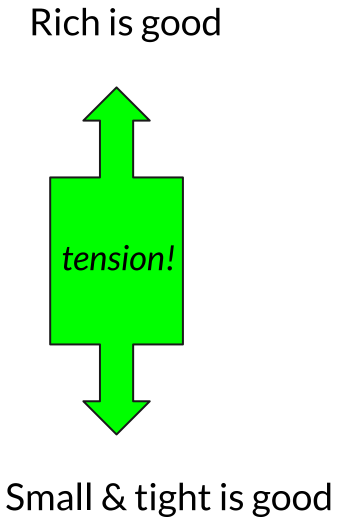


Abstract Class



Design Tension resolved by Abstract Methods



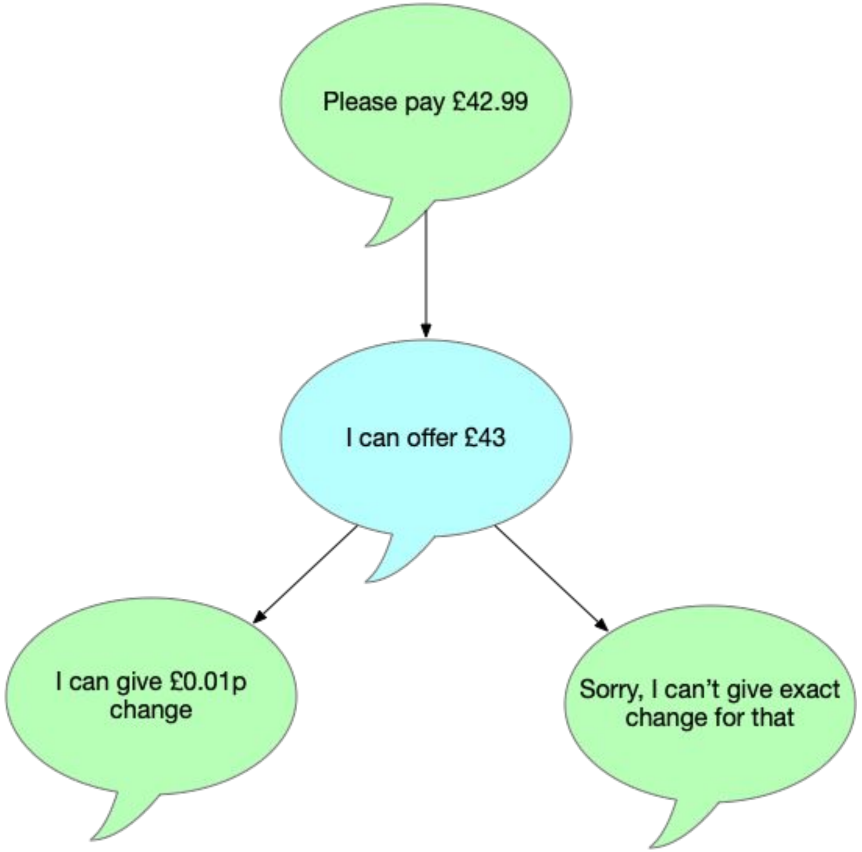


Purse - Our New Example

A Pouch for Carrying Money

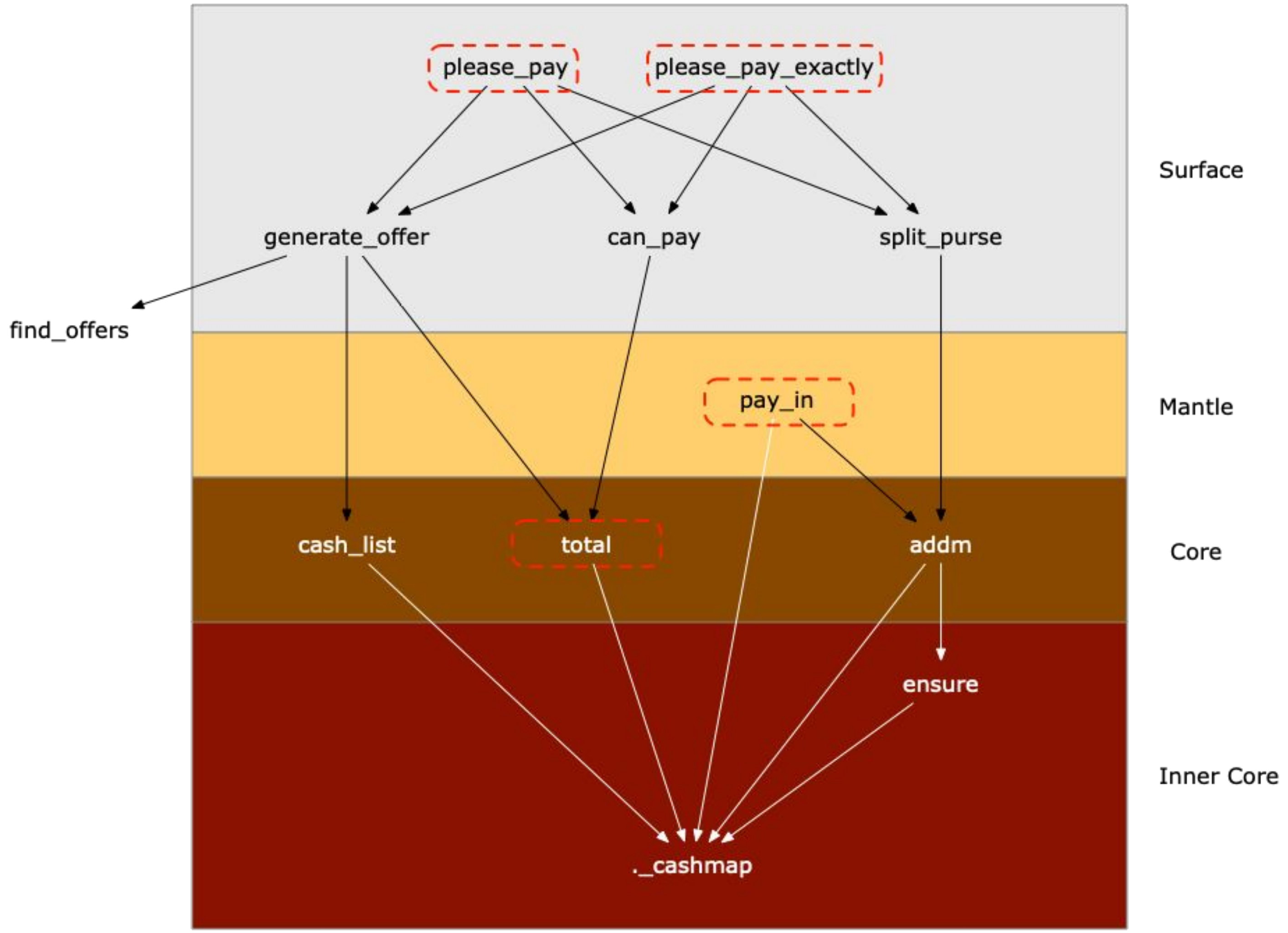
- Only interested in the money for this example
- Unordered container for coins and/or notes
- Each coin and note has a face value (the printed value)
- No limits on the number of coins and/or notes
- The money in a purse is private

Transaction



Exercise

- Identifying public and non-public methods
- Classification of core and surface methods
    - Also sub-core
    - And 'mantle'


please_pay
please_pay_exactly
Surface
generate_offer
can_pay
split_purse
find_offers
pay_in
Mantle
cash_list
total
addm
Core
ensure
Inner Core
._cashmap

Exercise

- Write an alternative implementation of Purse that satisfies
- Test transactions between the two implementation