a poll? When to use OOP:

- always
- sometimes
- never

“In Smalltalk, everything happens somewhere else.” - Adele Goldberg

---

How to find the classes (Meyer, Chapter 22)

Abstract data types -> Class -> object -> instance

The “Underline the nouns” fallacy. pgs 700, 720 (of 1370)

“...in function-oriented design you would concentrate on the verbs, which correspond to actions (“do this”); in object-oriented design you underline the nouns, which describe objects.”

“...do the objects of the system under discussion exhibit enough specific operations and properties of their own, *relevant to the system* and not covered by existing classes?”

*“A database record must be created every time the elevator **moves** from one floor to another.”*
versus
*“A database record must be created for every **move** of the elevator from one floor to another.”*

and

“*the system must support line insertion and deletion*”
versus
“*The editor must allow its users to insert or delete a line at the current cursor position.*“

725 Discovery and rejection
just as we need criteria for finding classes, we need criteria for rejecting candidate classes
eliminating bad ideas is just as important as finding good ones
“How to find the classes” means two things: not just how to come up with candidate abstractions but also how to unmask the inadequate among them

726 Danger signals
The grand mistake: calling “class” something which is in fact a routine
“My class performs…”

727

Class Name Rule:
A class name must always be either:
• A noun, possibly qualified.
• (Only for a deferred class describing a structural property) an adjective.

728
Single-routine classes
Premature classification
...inheritance is only relevant as a relation among well-understood abstractions
The only clear exception arises when you are dealing with an application domain for which a pre-existing taxonomy is widely accepted, as in some branches of science.

729
you should only design the inheritance hierarchy once you have at least a first grasp of the abstractions

If, early in a design process, you find the participants focusing on classification issues even though the classes are not yet well understood, they are probably putting the cart before the horse.

No-command classes: a class that has no routine at all, or only provides queries (ways to access objects) but no commands (procedures to modify objects).

[gives three examples where there is no mistake, one where the mistake was to omit commands, and one where the class was unjustified]:
...not a real data abstraction, simply some piece of passive information which might have been represented by a structure such as a list or array, or just by adding more attributes to another class.

730
...but if you do need a class and have not introduced it early enough, the adaptation may take some effort.

Mixed abstractions
Class Consistency principle: All the features of a class must pertain to a single, well-identified abstraction.

The ideal class:
clearly associated abstraction
class name is a noun or adjective
class represents a set of possible run-time objects, its instances
Several queries are available to find out properties of an instance
Several commands are available to change the state of an instance

Abstract properties can be stated, informally or (preferably) formally…

731 (761)
GENERAL HEURISTICS FOR FINDING CLASSES

737
Files

Find the Files

GUIs

Inheritance Hierarchies
The Diamond Problem

Videos:
super considered super?

Resources (links):

Bertrand Meyer (1997) Object Oriented Software Construction, 2nd ed. pdf:
http://web.uettaxila.edu.pk/CMS/AUT2011/seSCbs/tutorial%5CObject%20Oriented%20Software%20Construction.pdf

Excerpts (note: from eiffel.com):
https://archive.eiffel.com/doc/manuals/technology/oosc/page.html

Amazon ($118)
http://www.amazon.com/Object-Oriented-Software-Construction-CD-ROM-Edition/dp/0136291554

Gang of Four Design Patterns:
Gamma, Helm, Johnson, Vlissides (1994) Design Patterns: Elements of Reusable Object-Oriented Software. 430 pp
http://www.uml.org.cn/c++/pdf/DesignPatterns.pdf

Tutorialspoint website for Java
http://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm

---

Wiki page on paradigms:
https://en.wikipedia.org/wiki/Programming_paradigm#Multi-paradigm

---

Python specific design patterns:

Bruce Eckel (2001?). Thinking in Python: Design Patterns and Problem-Solving Techniques. 177 pp. MindView, Inc.
http://docs.linuxtone.org/ebooks/Python/Thinking_In_Python.pdf
http://www.mindview.net/Books/Python/ThinkingInPython.html
(*Python is executable pseudocode. Perl is executable line noise.*
Perl is like vice grips. You can do anything with it, and it's the wrong tool for every job.)

Rahul Verma, Chetan Giridhar (2011). Design Patterns in Python.
http://kennison.name/files/zopestore/uploads/python/DesignPatternsInPython_ver0.1.pdf
www.testingperspective.com

Alex Martelli (2007). Design Patterns in Python. 46 Slide Slideshow, with annotated references.
http://www.aleax.it/gdd_pydp.pdf

Basic Python intro with some useful advice:

Yang Li (?). Object-Oriented Design with Python: CSCI 5448: Object – Oriented A & D Presentation.
https://www.cs.colorado.edu/~kena/classes/5448/f12/presentation-materials/li.pdf

---

SourceMaking Code Smells (Organized list w/links):
https://sourcemaking.com/refactoring/smells

SM Home (Design Patterns, AntiPatterns, Refactoring, UML)
https://sourcemaking.com/

Martin Fowler Code Smell (2006)
http://martinfowler.com/bliki/CodeSmell.html

MF Catalog of Refactorings (December 2013)
http://www.refactoring.com/catalog/

MF Refactoring Databases:
http://martinfowler.com/books/refactoringDatabases.html

Industrial Logic Refactoring to Patterns
https://industriallogic.com/xp/refactoring/

IL Reference Guide pdf of Smells to Refactorings (Kerievsky, 2004)
Gives page numbers for two books: Fowler and Kerievsky
http://www.industriallogic.com/wp-content/uploads/2005/09/smellstorefactorings.pdf

Functional programming design patterns by Scott Wlaschin, 2014 NDC London.
https://vimeo.com/113588389

Loves smalltalk but...
Making fun of Enterprise OO is like shooting IMarineVertebrates in an AbstractBarrelProxyFactory

OO pattern/principle	FP equivalent
• Single Responsibility Principle	• Functions
• Open/Closed principle	• Functions
• Dependency Inversion Principle	• Functions, also
• Interface Segregation Principle	• Functions
• Factory pattern	• You will be assimilated!
• Strategy pattern	• Functions again
• Decorator pattern	• Functions
• Visitor pattern	• Resistance is futile!