Patterns from nanoGoogle I

- Decorator
  - Attach responsibilities dynamically
  - Concrete class 'is-a' and 'has-a' decorator
  - Avoid rewriting existing code, write new code

- BufferedReader and `java.io` classes
  - From string, from web, from …
- Filter classes
  - Boolean: accept/reject word
  - Altering: remove punctuation, lowercase, …

Decorator Details

- **Name: also-knowns-as *Wrapper***
  - **Wrap existing object with more responsibilities**
  - **HFDP: tall, decaf, skim, latte**

- **Forces:**
  - **Add responsibilities to objects without affecting other objects (and remove the responsibilities)**
  - **Extension by subclass impractical: class explosion or no access to parent class for subclassing**

Patterns from nanoGoogle II

- Strategy
  - Change algorithm, policy without altering existing code, but by writing new code
  - Program to interface, not implementation
  - Algorithm varies independently from client
- What to do when processing words
  - Count them, store them, dump them to disk
- How to print results after processing words
  - XML, to file, standard output, …

Strategy Details

- **Name: also known as *Policy***
  - **Make algorithms/policies interchangeable**
  - **HFDP: how to quack, how to fly**

- **Forces:**
  - **Re-use policies between contexts or change them at runtime**
  - **Context has-a policy, uses it, can change policy**
  - **Don't hardwire policy behavior into client**