Better Object Creation with Factories

Andrejs Doronins
TEST AUTOMATION ENGINEER

Dictionary.java

```java
public Dictionary(){

}
```

ClientApp.java

```java
Dictionary d =
            new Dictionary();
```

Dictionary.java

```java
public Dictionary(dep1, dep2){

this.dep1 = dep1;

this.dep2 = dep2;

}
```

ClientApp.java

```java
Dictionary d =
            new Dictionary(a,b);
```

Dictionary.java

```
public Dictionary(){

this.dep1 = new Dep1();

this.dep2 = new Dep2();

}
```

ClientApp.java

```
Dictionary d =

          new Dictionary();
```

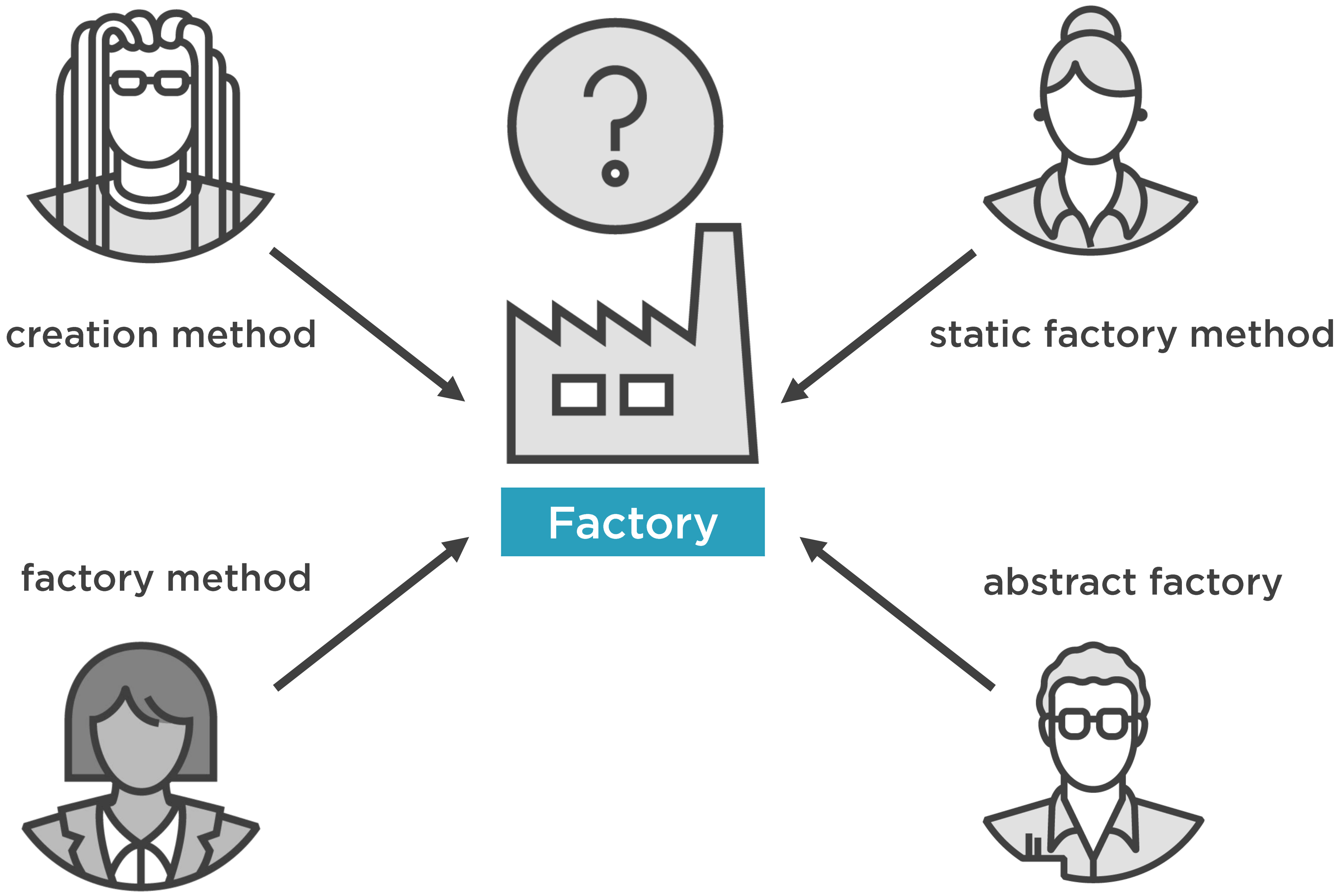
Factory Example

Dictionary.java

```java
public static Dictionary get(){

 return new Dictionary(dep1,dep2);

}
```

ClientApp.java

```java
Dictionary d = Dictionary.get();
```


creation method
static factory method
Factory
factory method
abstract factory

Creation/Factory Method

“Refactoring” by Martin F.
“Refactoring to Patterns” by Joshua K.

Static Factory Method

“Effective Java” by Joshua B.

Factory

Clarify & Demo

Simple (Parametrized) Factory

Book: “Head First Design Patterns”

Factory Method and Abstract Factory

“Design Patterns: Elements of Reusable Object-Oriented Software” Book

Factory

Creation/Factory Method

"Refactoring" by Martin F.
"Refactoring to Patterns" by Joshua K.

Static Factory Method

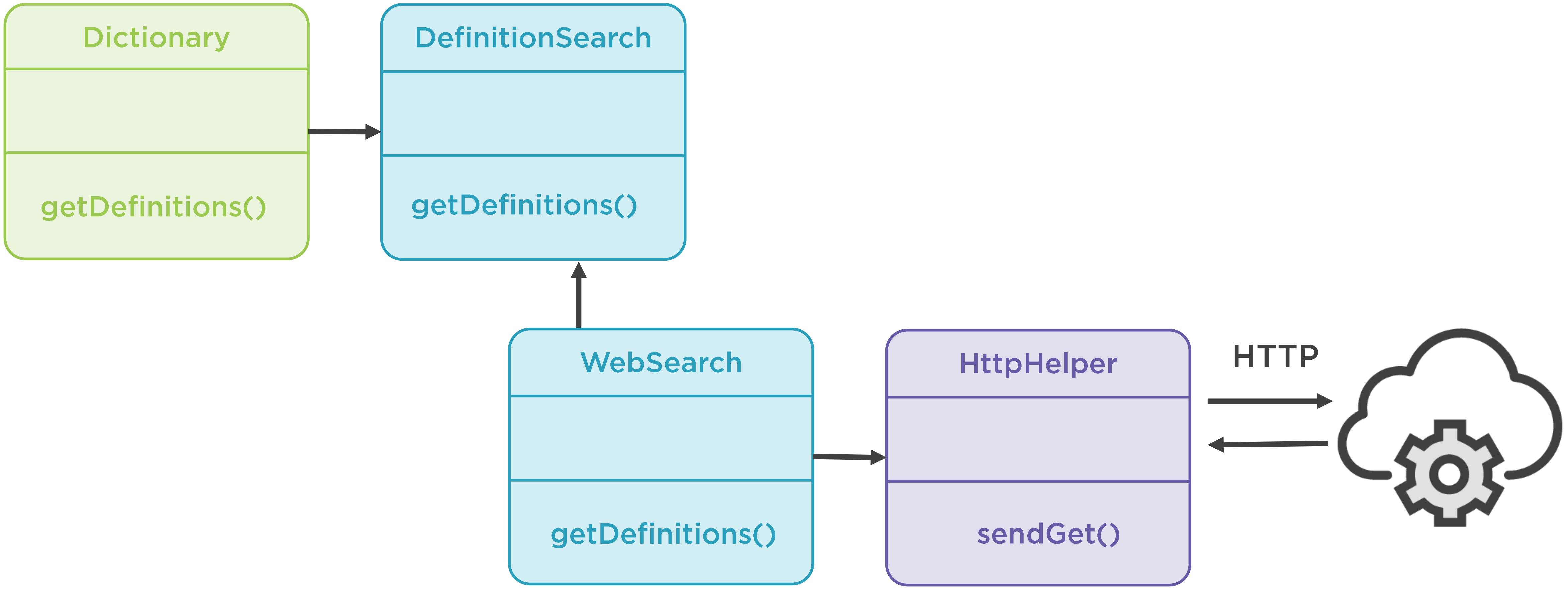
"Effective Java" by Joshua B.

Simple (Parametrized) Factory

Book: "Head First Design Patterns"

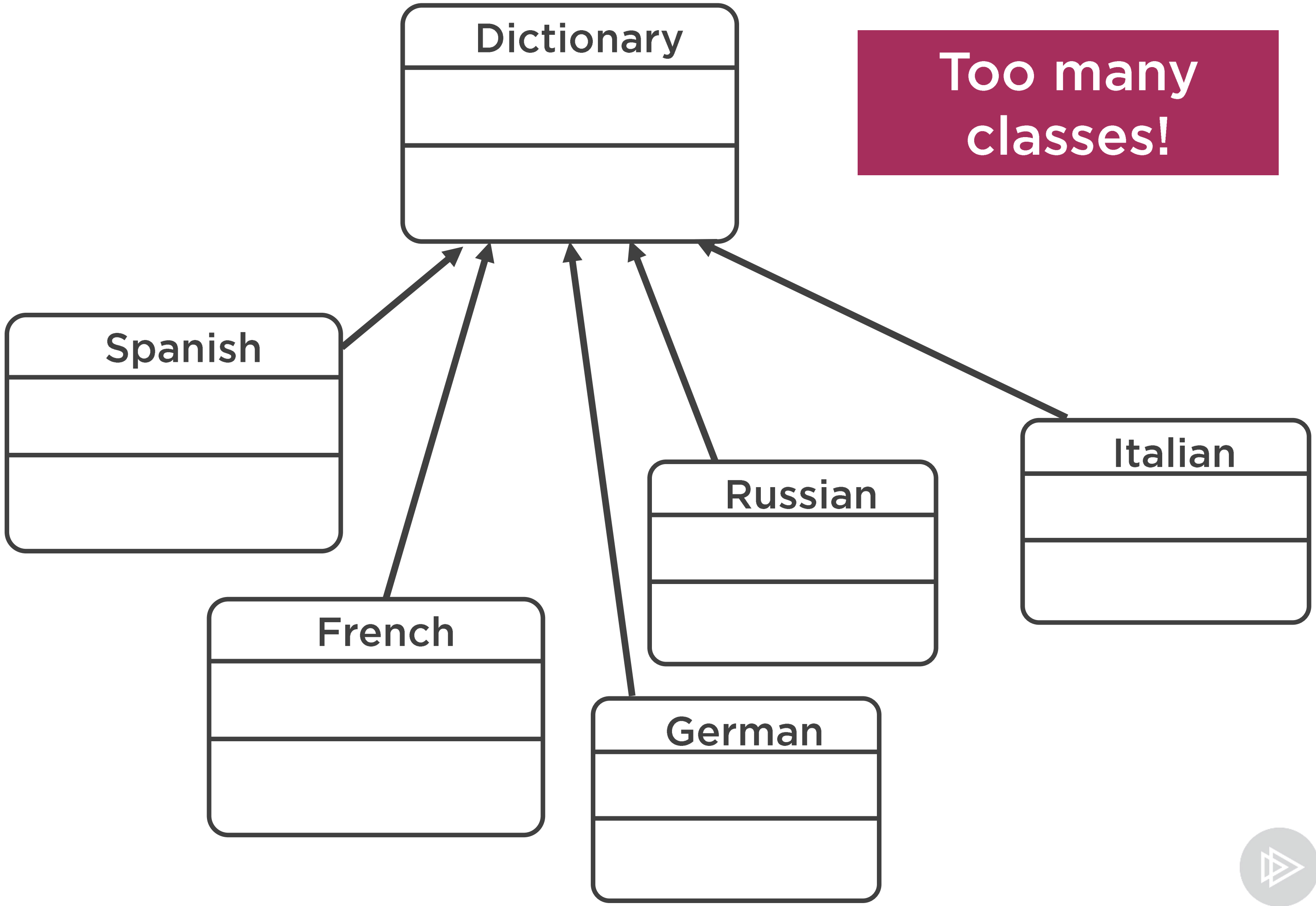
Factory Method and Abstract Factory

"Design Patterns: Elements of Reusable Object-Oriented Software" Book


Dictionary
getDefinitions()
DefinitionSearch
getDefinitions()
WebSearch
getDefinitions()
HttpHelper
sendGet()
HTTP

French!

Spanish!



Too many classes!

No-arg Constructor

```
Dictionary(){
// assign inside
}
```

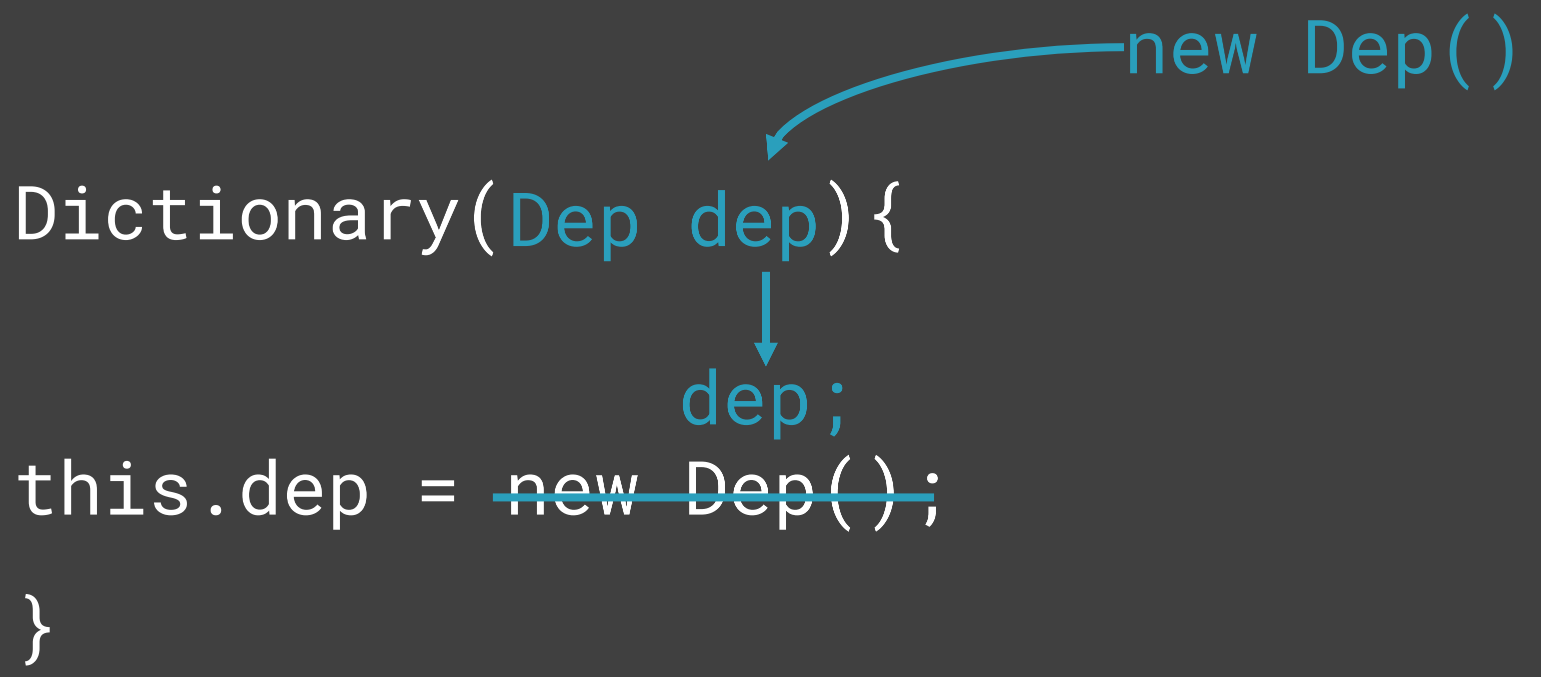
Pros:

- Simple client code

Cons:

- Inflexible
- Unstable tests

Dependency Injection



new Dep()

Dictionary(Dep dep){

dep;

this.dep = ~~new Dep();~~

}

Constructor with Arguments

```
Dictionary(T dep){
  // assign d
}
```

Pros:

- Stable Tests
- Flexible

Cons (new problem):

- More complex client code

Summary

```
Dictionary(){
    this.dep = new Dep();
}
```

↓

```
Dictionary(Dep dep){
    this.dep = dep;
}
```

✗ Inflexible

✗ Unstable tests

↓

+ Flexible

+ Testable

✗ Complex Client Code

Static Factory Method

Creation Method Example

DictionaryCopier.java

```
                static

public copy(Dictionary d){

Dictionary copy = /* clone it */;

return copy;

}
```

ClientApp.java

```
Dictionary d1 = new Dictionary();

Dictionary d2 = new
    DictionaryCopier~~()~~.copy(d);
```

Creation Methods
and
Static Factory Methods
are practically the same.

Static Factory Methods in Java

```
Calendar.getInstance();

String.valueOf(true);

LocalDate.of(2019, 01, 01);

Optional.empty();

Collections.unmodifiableCollection(...);
```

Summary

```
Dictionary(Dep dep){
    this.dep = dep;
}
```

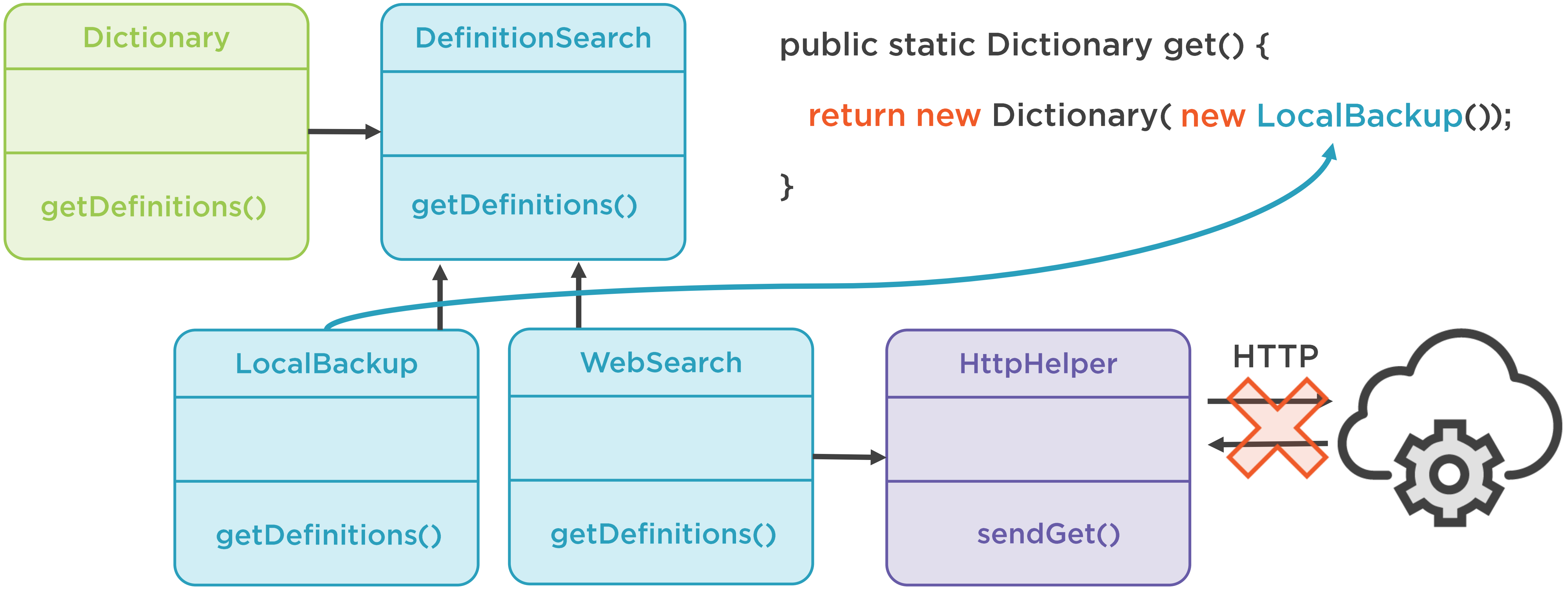
↓

```
public static Dictionary english(){
 return new Dictionary(new Dep());
   }
```

- ✚ Flexible
- ✚ Testable
- ✖ Complex Client Code

↓

- ✚ Simple Client Code
- ✚ Easy to change



```
public static Dictionary get() {
    return new Dictionary( new LocalBackup());
}
```

Factory Example

Dictionary.java

```
public static Dictionary get(){

 return new Dictionary( dep1  );

                        dep2
}
```

ClientApp.java

```
Dictionary d = Dictionary.get();
```

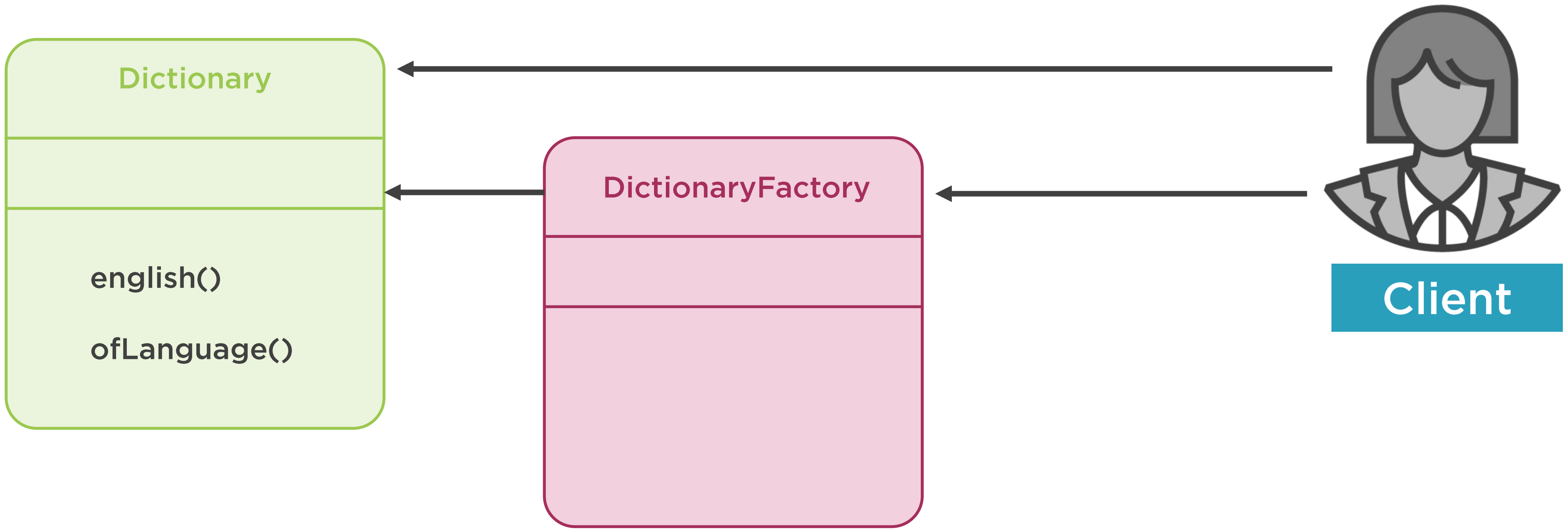
SOLID

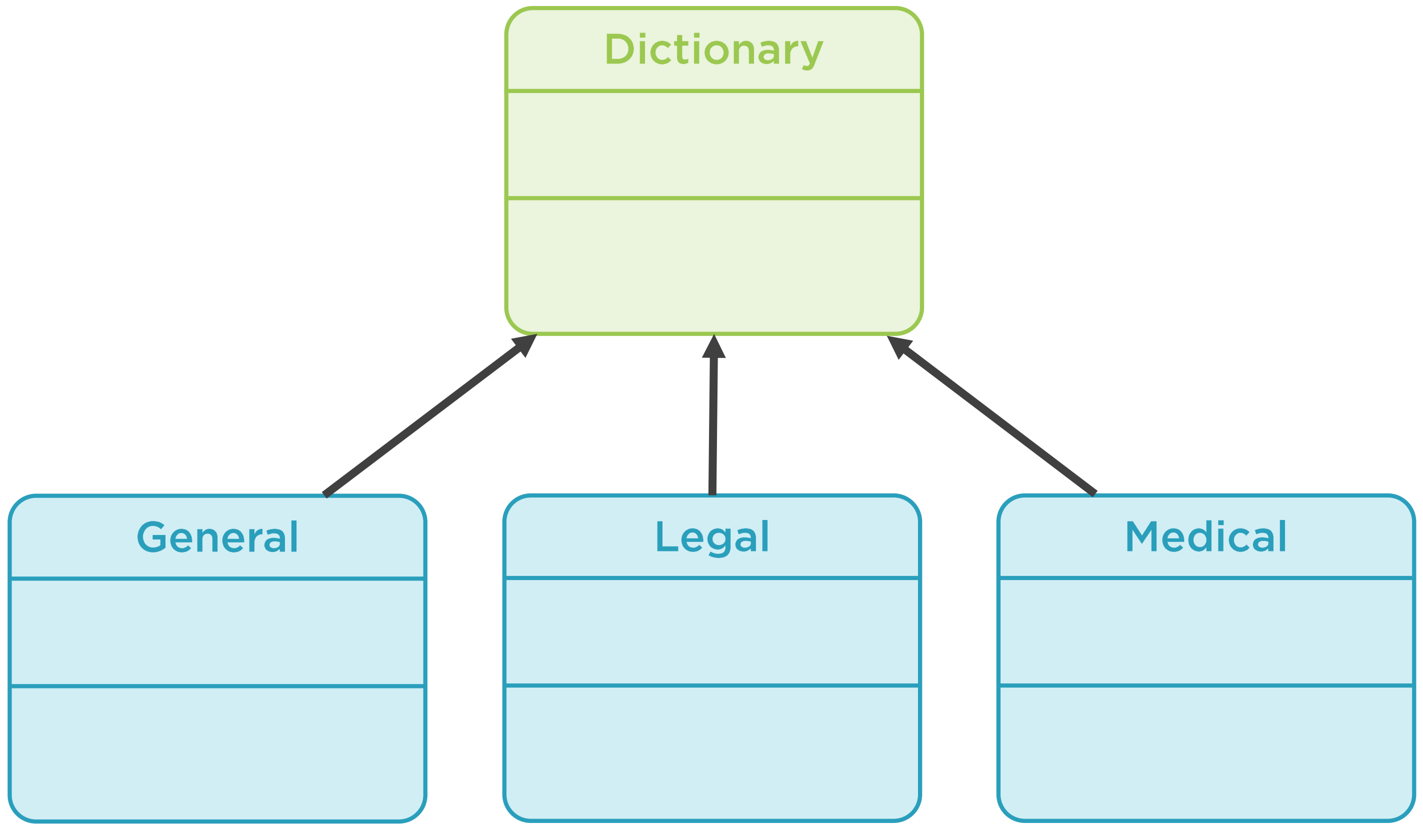
A class should have only one reason to change

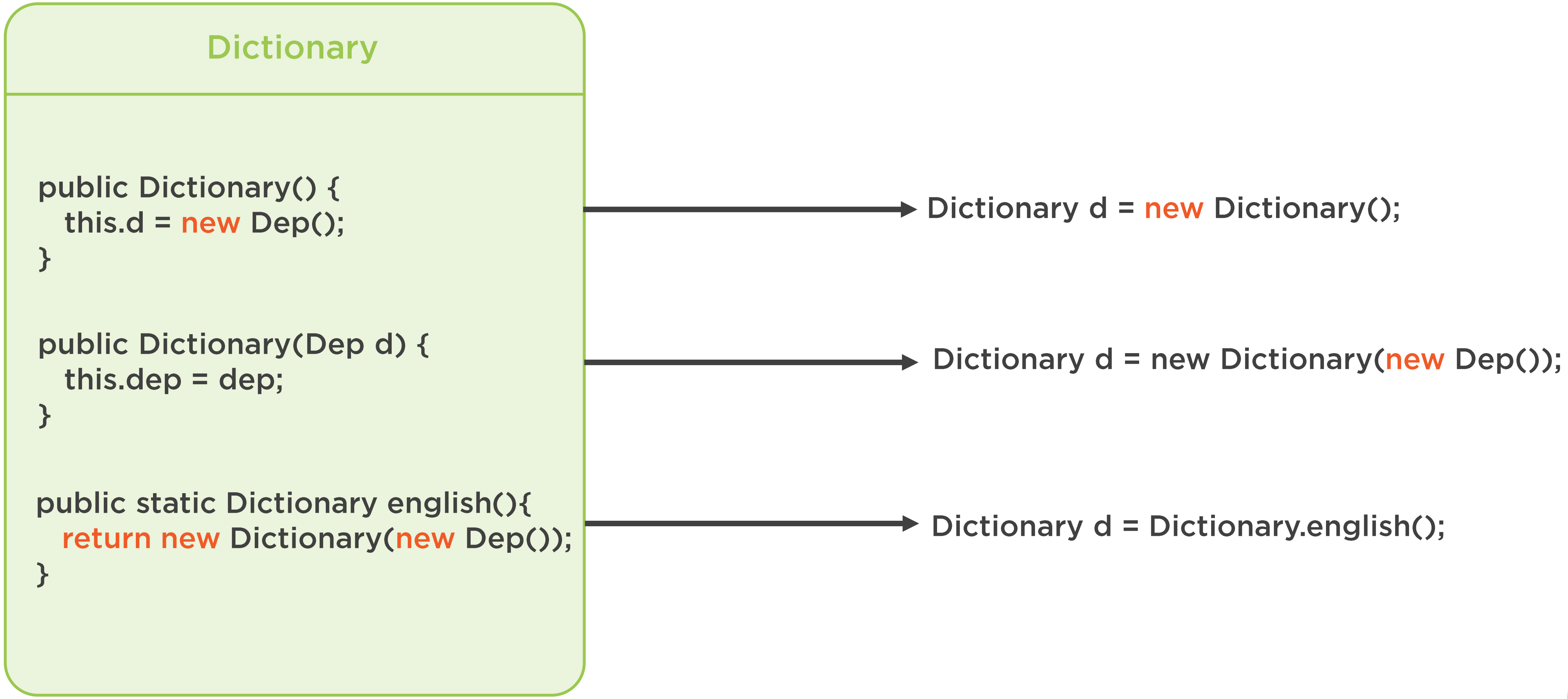
Extend code, not modify

Open-Closed Principle

Single Responsibility Principle


Dictionary
english()
ofLanguage()
DictionaryFactory
Client


Dictionary
General
Legal
Medical



Dictionary

```
public Dictionary() {
   this.d = new Dep();
}
```

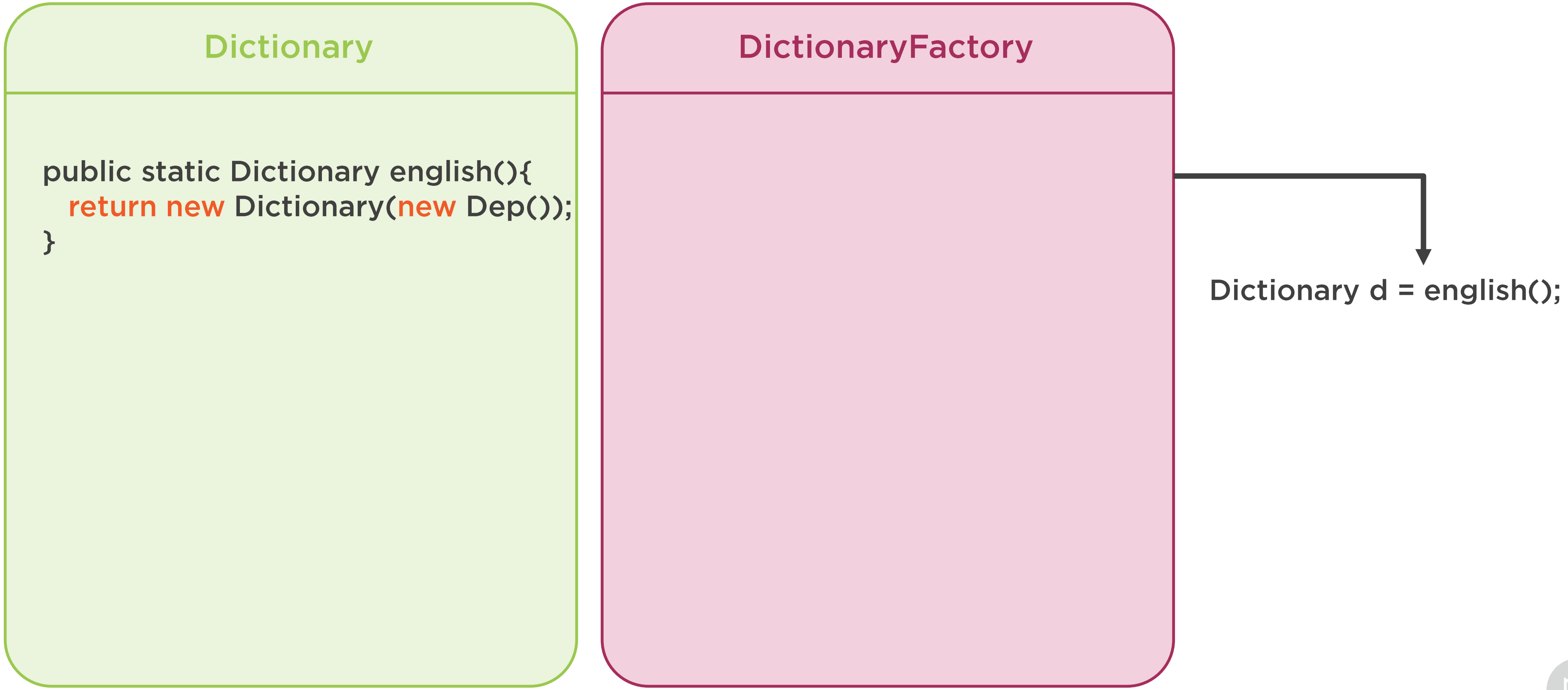
```
Dictionary d = new Dictionary();
```

```
public Dictionary(Dep d) {
   this.dep = dep;
}
```

```
Dictionary d = new Dictionary(new Dep());
```

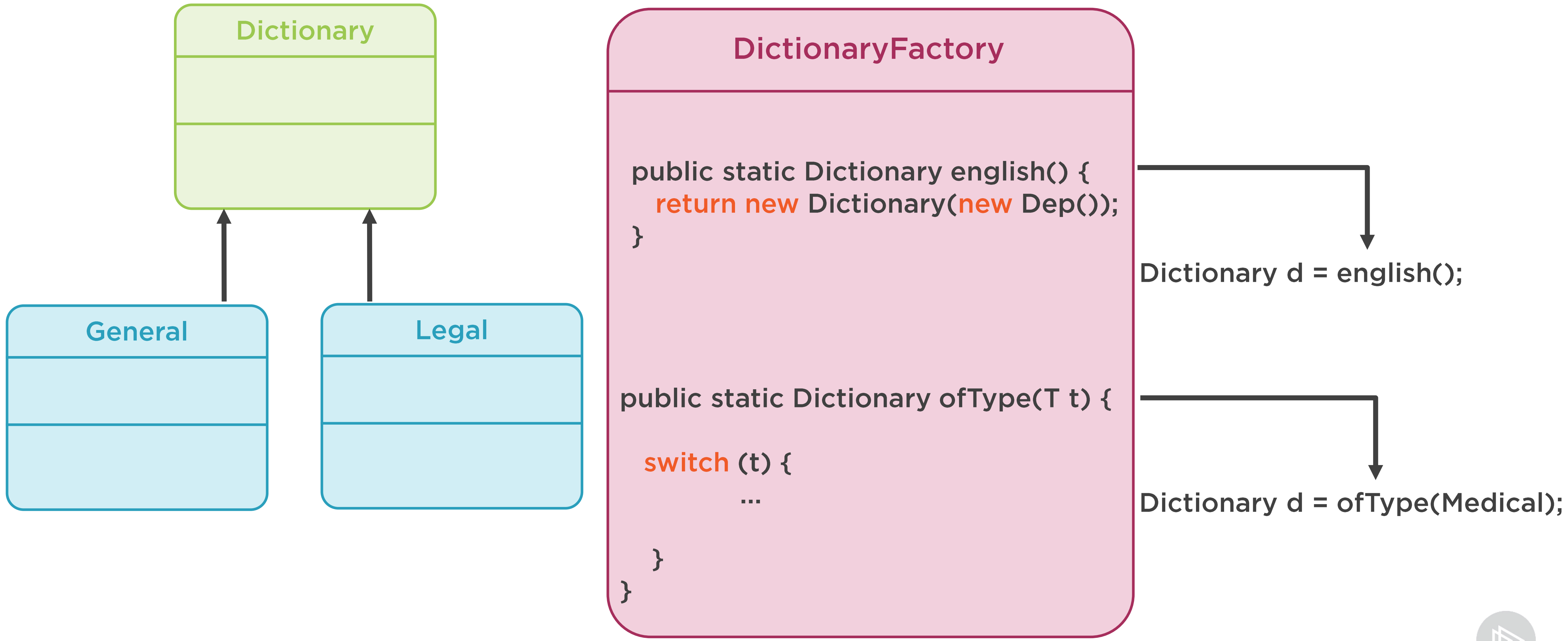
```
public static Dictionary english(){
   return new Dictionary(new Dep());
}
```

```
Dictionary d = Dictionary.english();
```



```
public static Dictionary english(){
   return new Dictionary(new Dep());
}
```

```
Dictionary d = english();
```



```
public static Dictionary english() {
  return new Dictionary(new Dep());
}
```

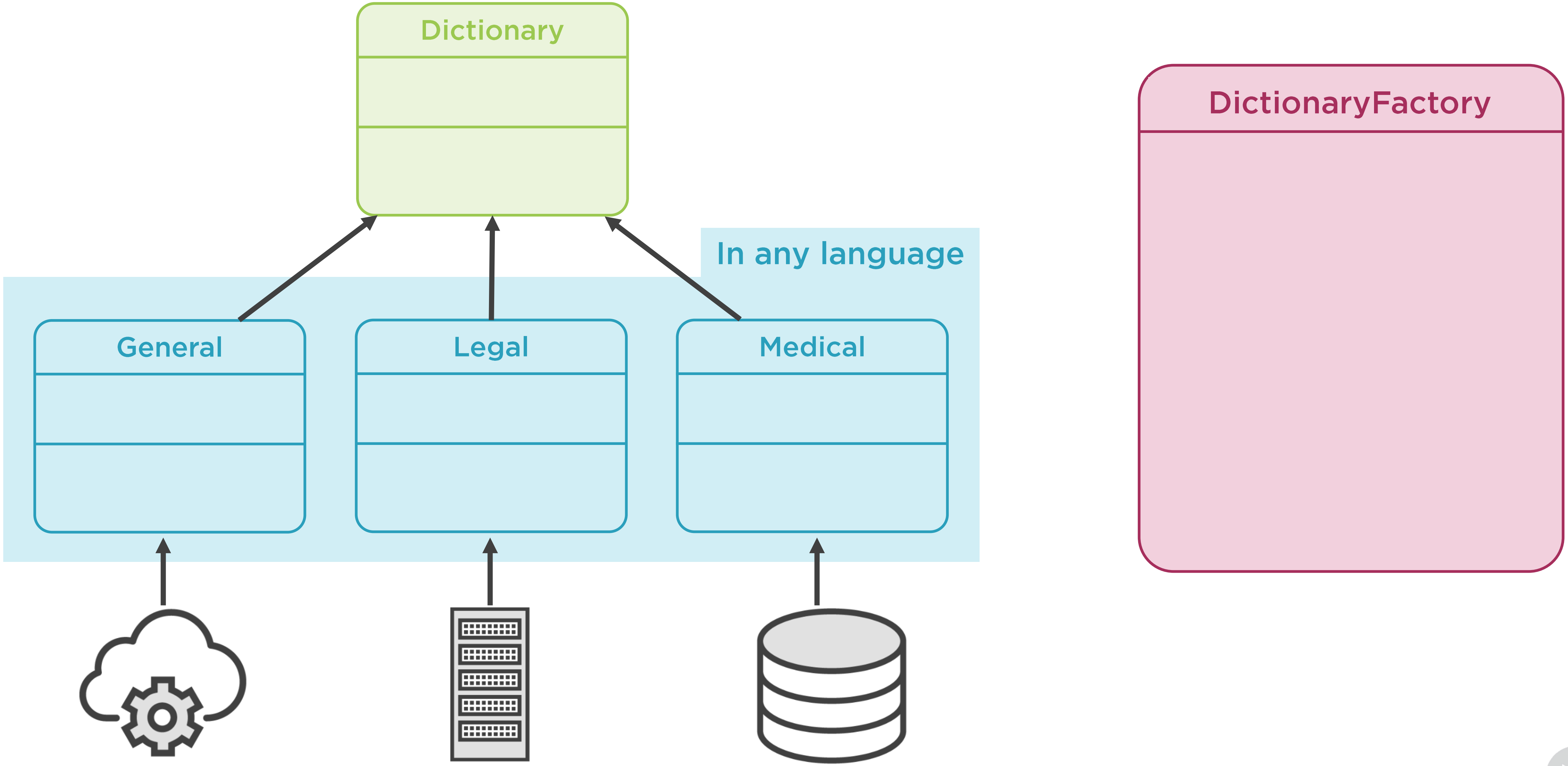
```
Dictionary d = english();
```

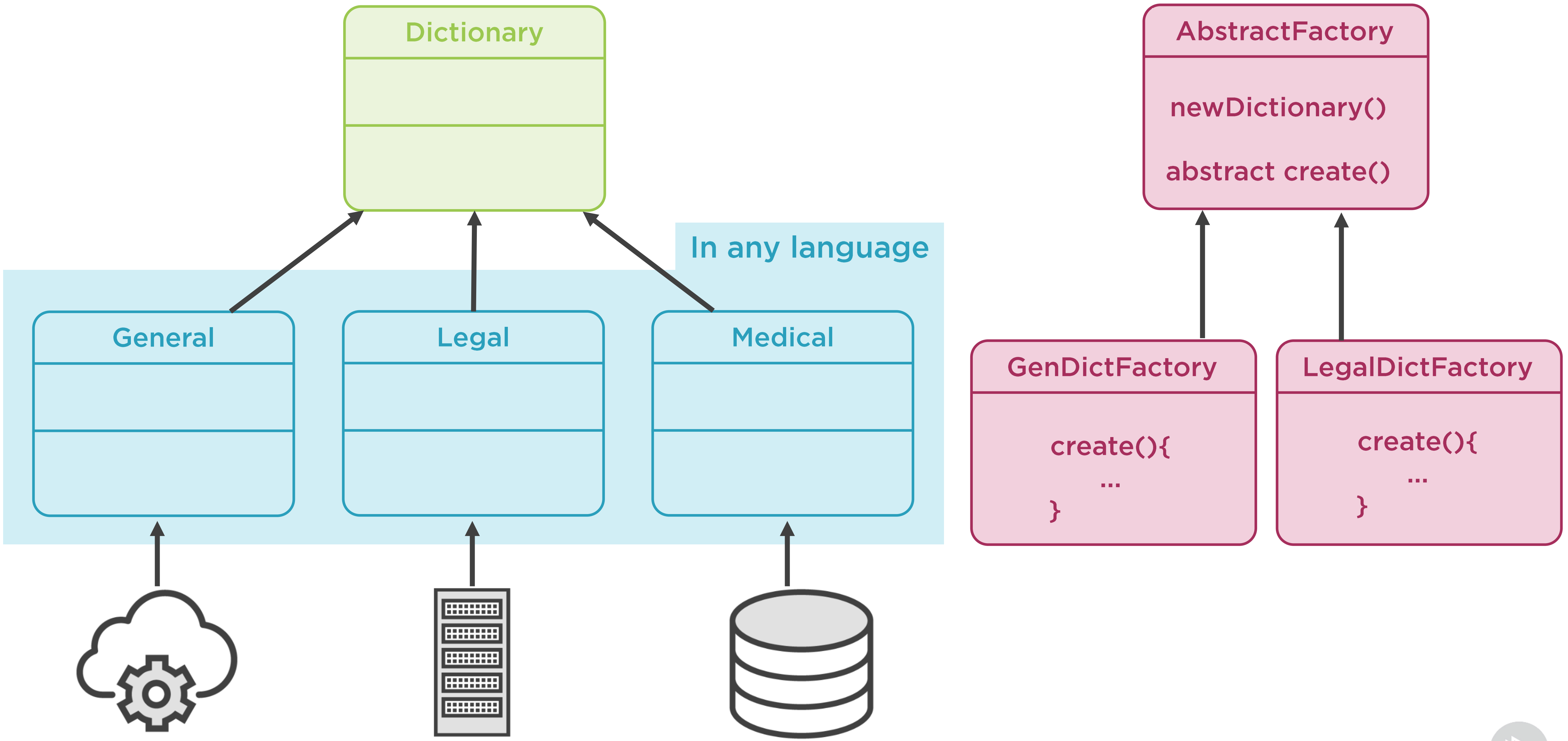
```
public static Dictionary ofType(T t) {

  switch (t) {
          ...

   }
}
```

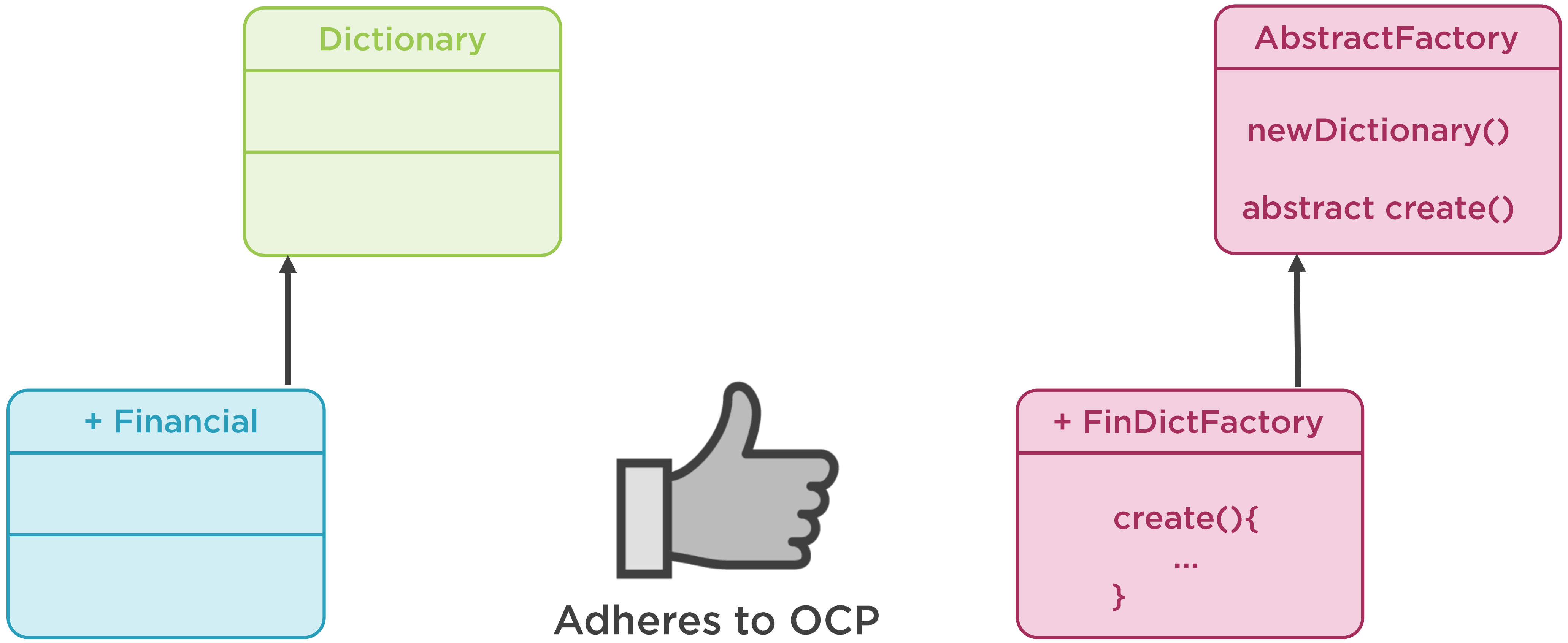
```
Dictionary d = ofType(Medical);
```

Future Requirements

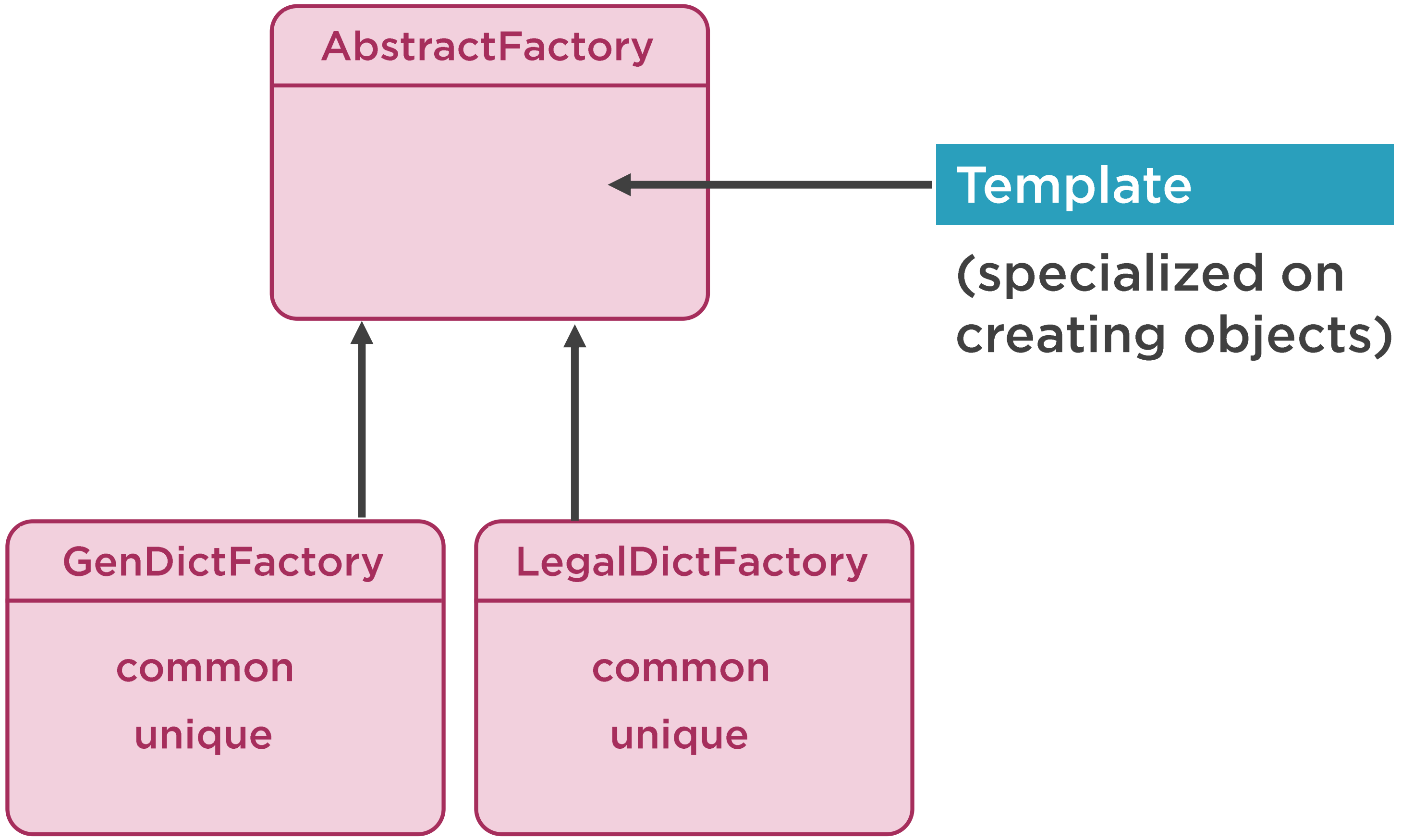
Any dictionary in
any language, please!


Dictionary
In any language
General
Legal
Medical
DictionaryFactory

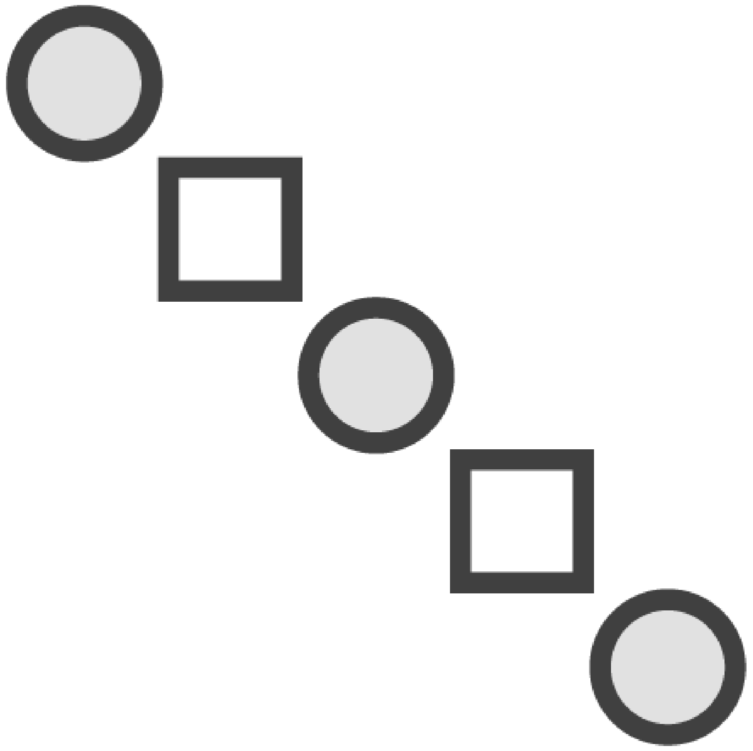

Dictionary
In any language
General
Legal
Medical
AbstractFactory
newDictionary()
abstract create()
GenDictFactory
create(){
...
}
LegalDictFactory
create(){
...
}



Adheres to OCP


AbstractFactory
Template
(specialized on creating objects)
GenDictFactory
common
unique
LegalDictFactory
common
unique

What We Didn’t Cover



Abstract Factory

- Quite complex
- Not very frequently used

Builder

- Not a “factory”
- Is frequently used

WithoutBuilder.java

```java
new Pizza(true, true, false);
```

WithBuilder.java

```java
Pizza.Builder()
    .cheese(true)
    .ham(true)
    .mushrooms(false)
    .build();
```

Summary

Assembling objects is complex

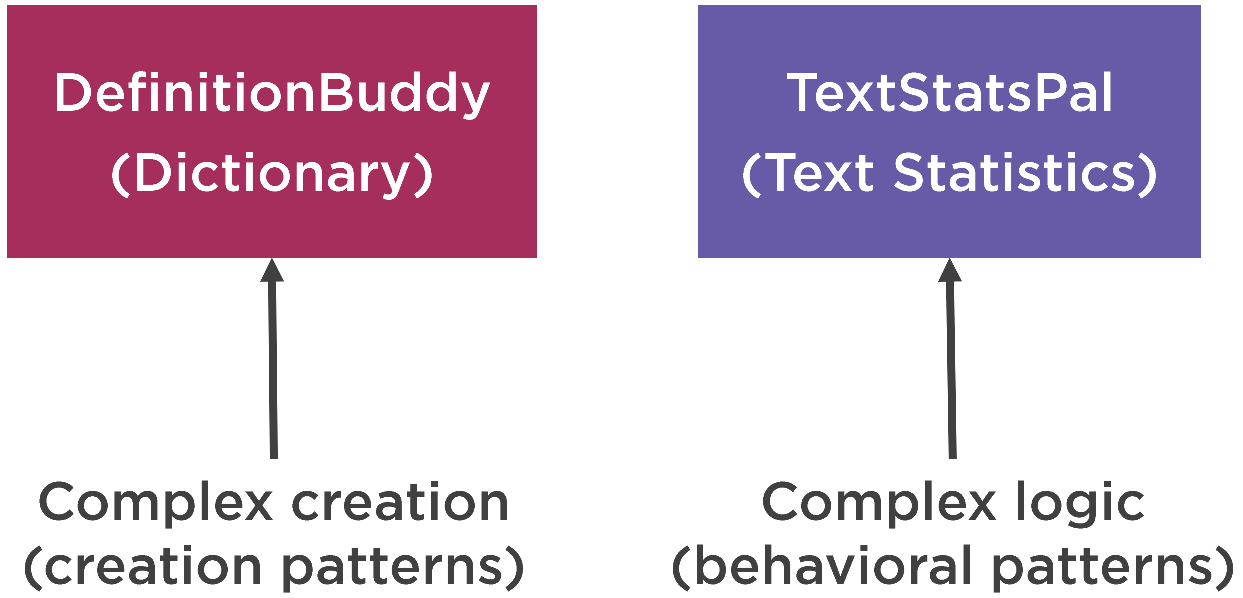
Static Factories have multiple benefits

- Reduced maintenance
- Complexity is hidden through better encapsulation

Dedicated simple factory class can be enough

Refactor to Factory Method if needed

Up Next


DefinitionBuddy
(Dictionary)
TextStatsPal
(Text Statistics)
Complex creation
(creation patterns)
Complex logic
(behavioral patterns)