Summer Term 2018

Software Engineering Design & Construction

Dr. Michael Eichberg
Fachgebiet Softwaretechnik
Technische Universität Darmstadt

Interface Segregation Principle

I*nterface* ***S****egregation* ***P****rinciple*

Clients should not be forced to depend on methods that they do not use.

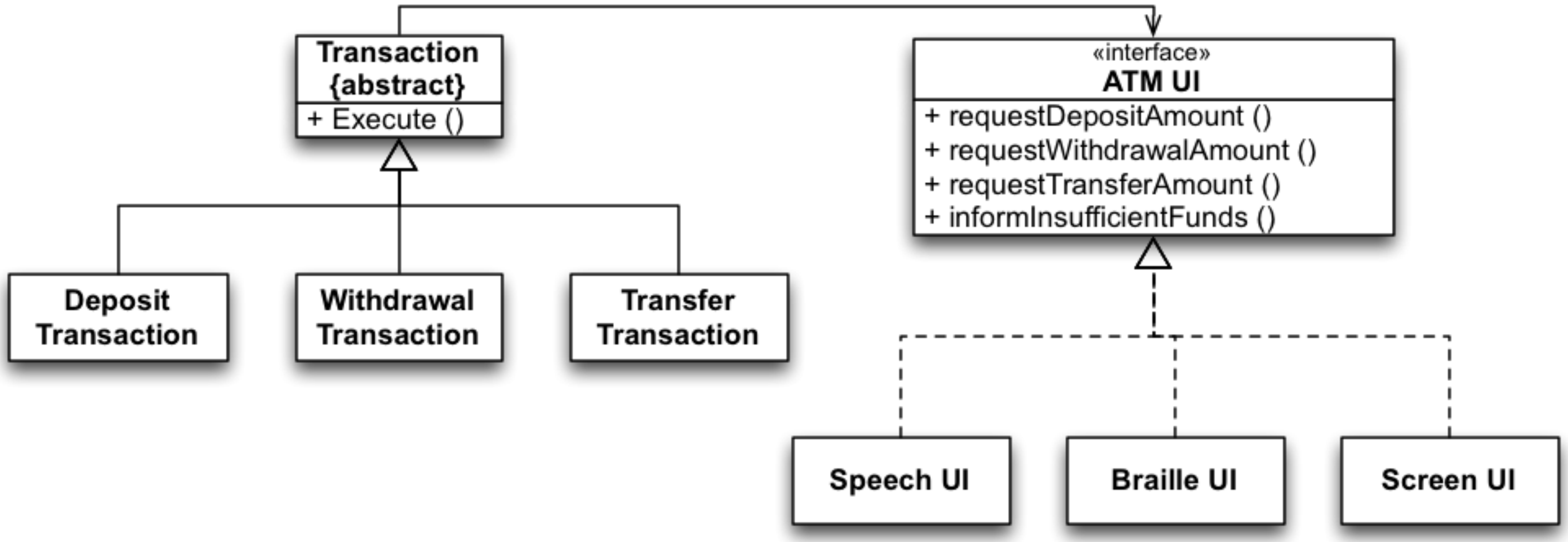
–Agile Software Development; Robert C. Martin; Prentice Hall, 2003

Introduction by Example

- Consider the development of software for an automated teller machine (ATM):
    - Support for the following types of transactions is required: **withdraw**, **deposit**, and **transfer**.
    - Support for different **languages** and support for different **kinds of UIs** is also required
    - Each transaction class needs to call methods on the GUI
      E.g., to ask for the amount to deposit, withdraw, transfer.

Introduction by Example

- Initial design of a software for an automatic teller machine (ATM):



What do you think?

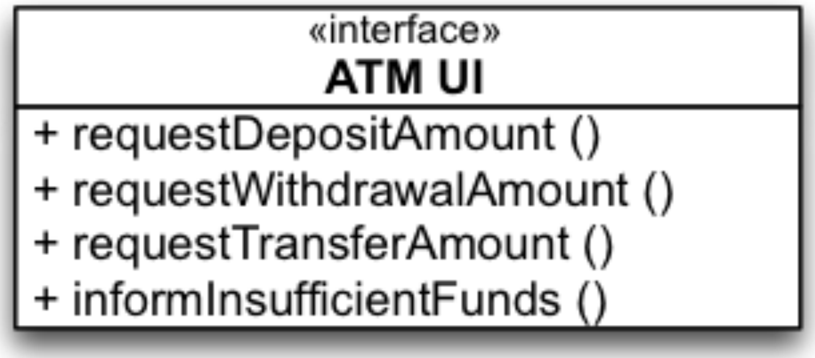
A Polluted Interface

ATM UI is a polluted interface!

- It declares methods that do not belong together.

- It forces classes to depend on unused methods and therefore depend on changes that should not affect them.

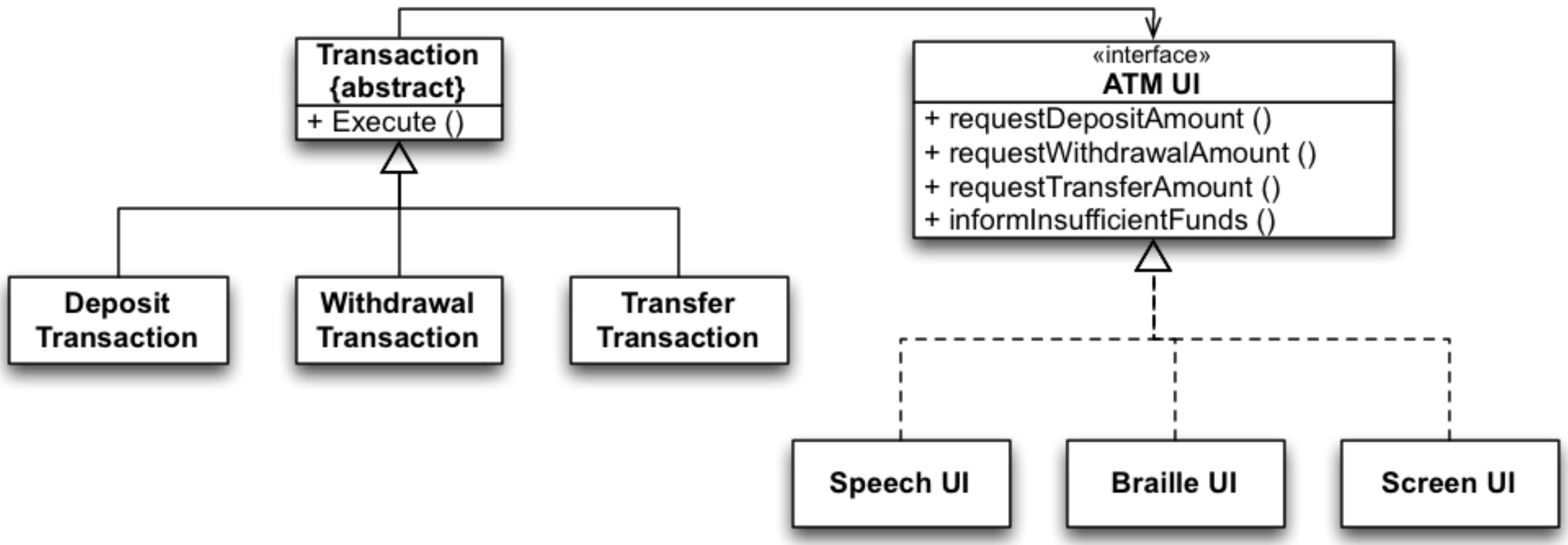
- ISP states that such interfaces should be split.


«interface»
ATM UI
+ requestDepositAmount ()
+ requestWithdrawalAmount ()
+ requestTransferAmount ()
+ informInsufficientFunds ()

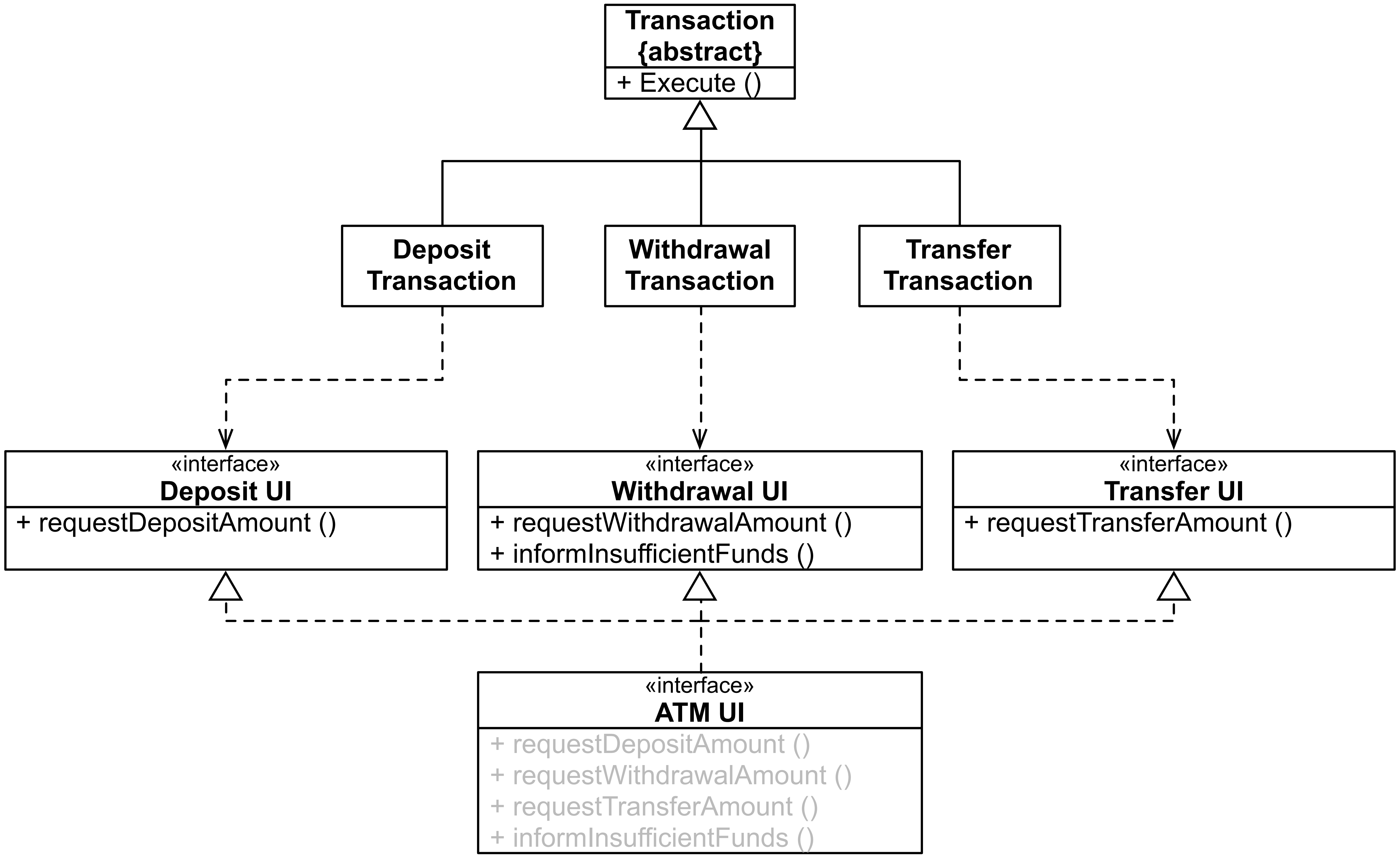
The Rationale Behind ISP

When clients depend on methods they do not use, they **become subject to changes forced upon these methods** by other clients.

How does an ISP compliant solution look like?



An ISP Compliant Solution


Transaction {abstract}
+ Execute ()
Deposit Transaction
Withdrawal Transaction
Transfer Transaction
«interface» Deposit UI
+ requestDepositAmount ()
«interface» Withdrawal UI
+ requestWithdrawalAmount ()
+ informInsufficientFunds ()
«interface» Transfer UI
+ requestTransferAmount ()
«interface» ATM UI
+ requestDepositAmount ()
+ requestWithdrawalAmount ()
+ requestTransferAmount ()
+ informInsufficientFunds ()

I*nterface /* ***T****rait* ***S****egregation* ***P****rinciple*

Clients should not be forced to depend on methods that they do not use.

–Agile Software Development; Robert C. Martin; Prentice Hall, 2003

General Strategy

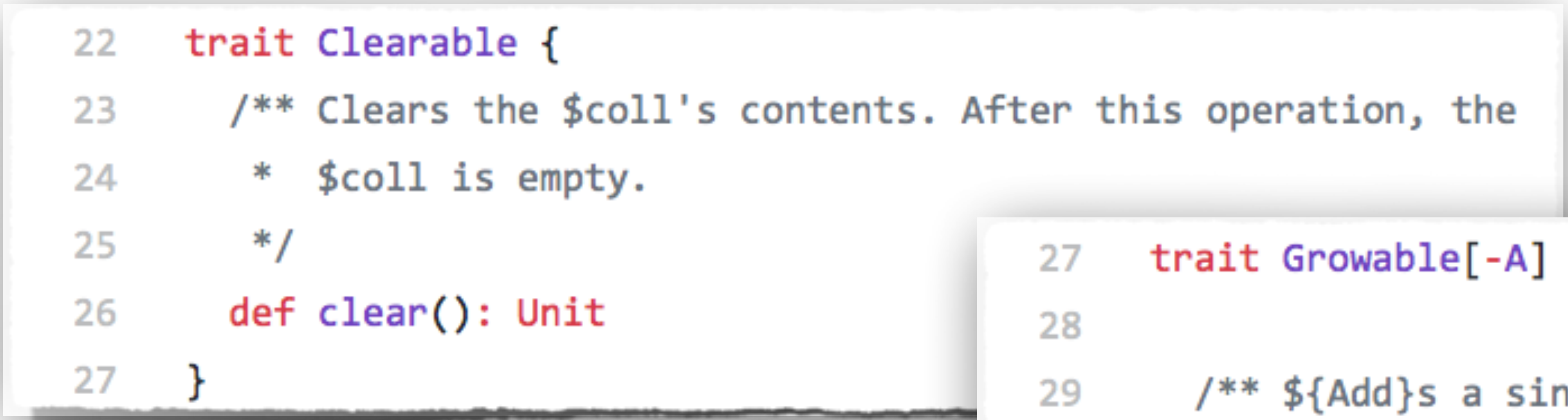
Try to group possible clients of a class and have an interface/trait for each group.

General Strategy

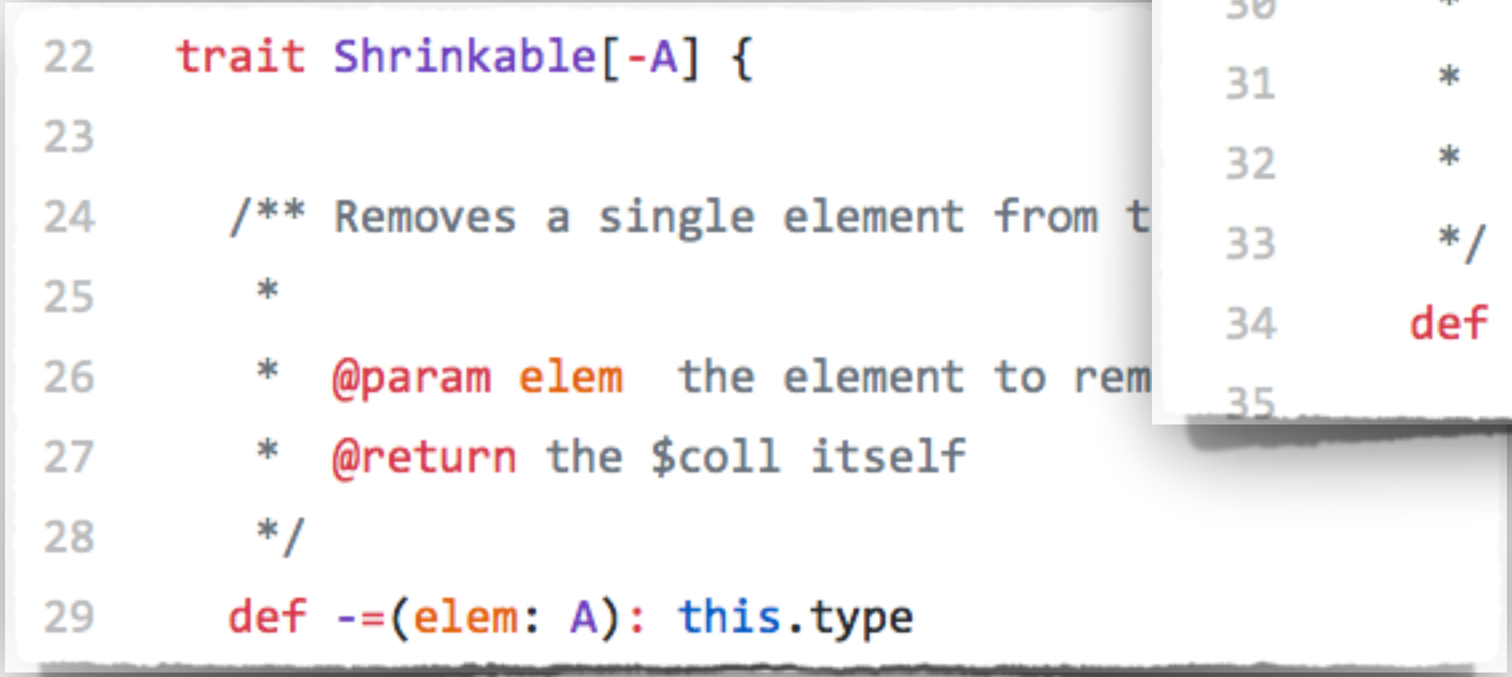
Try to group possible clients of a class and have an interface/trait for each group.

Proliferation of Interfaces/Traits

ISP in Scala (2.12.x) - Case Study

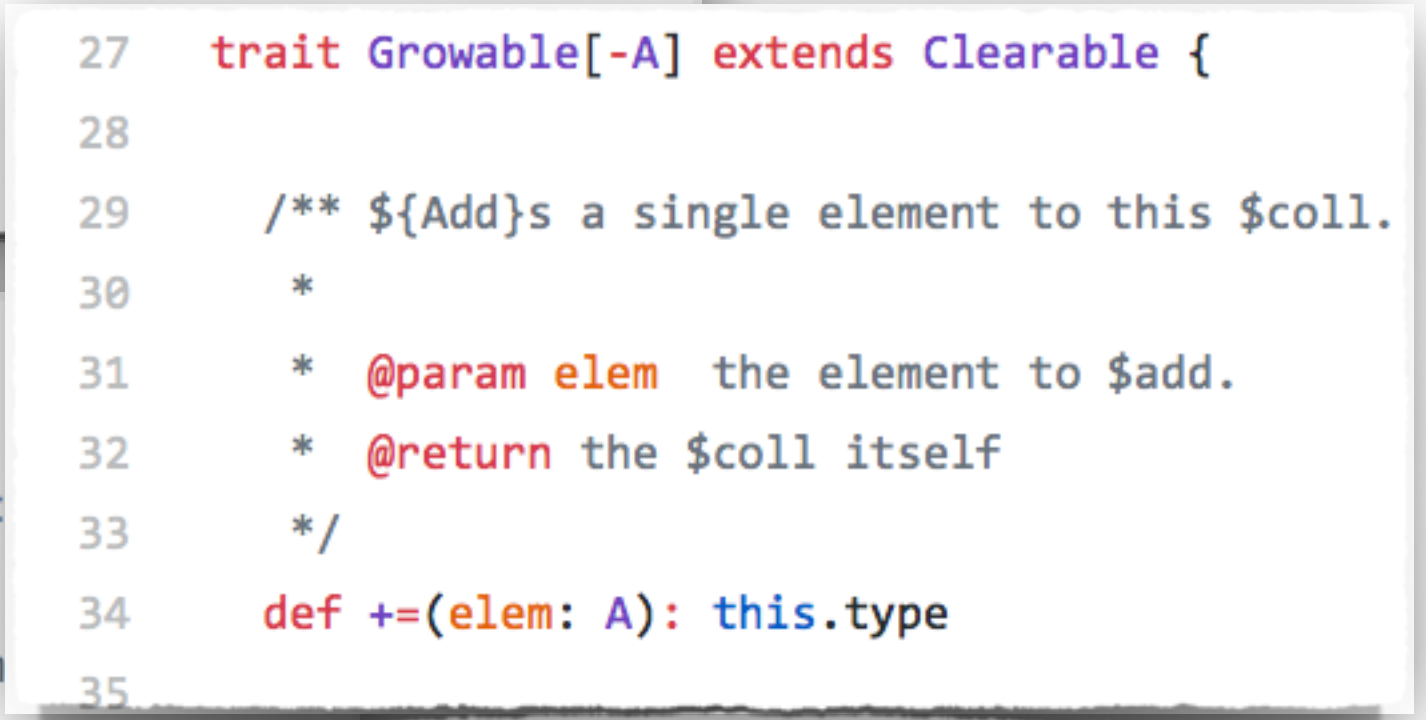


```scala
trait Clearable {
  /** Clears the $coll's contents. After this operation, the
   *  $coll is empty.
   */
  def clear(): Unit
}
```



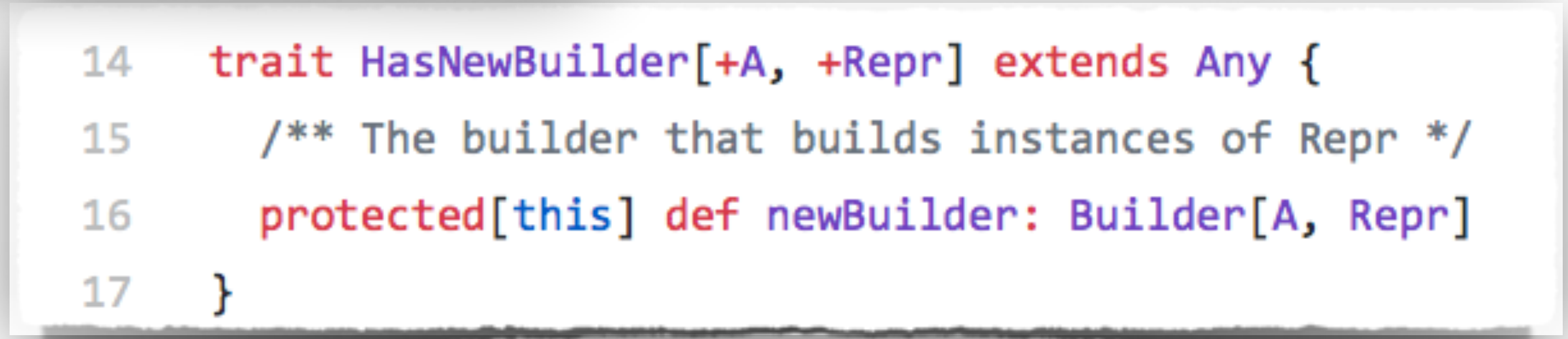
```scala
trait Shrinkable[-A] {

  /** Removes a single element from t
   *
   *  @param elem  the element to rem
   *  @return the $coll itself
   */
  def -=(elem: A): this.type
```



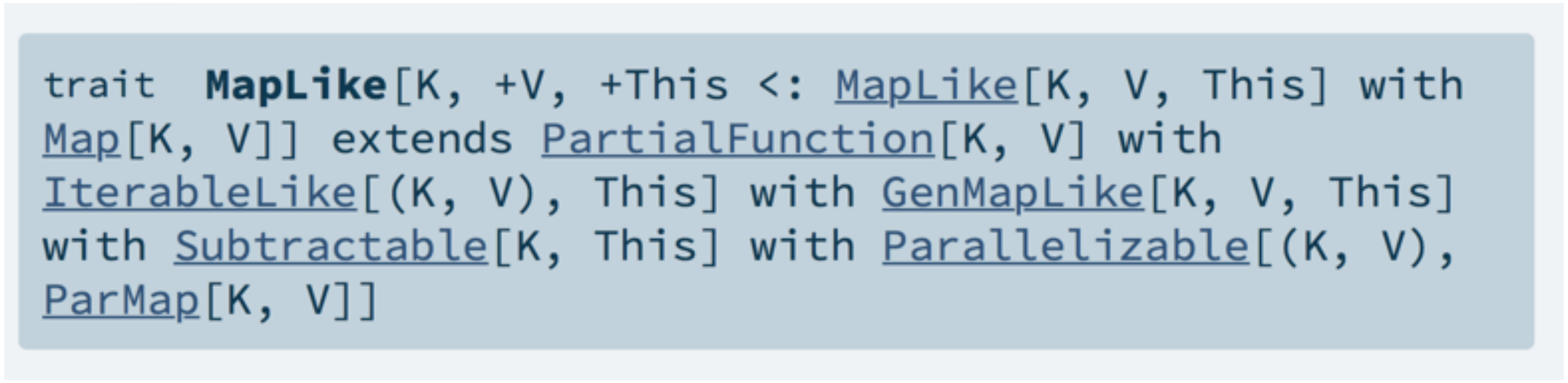
```scala
trait Growable[-A] extends Clearable {

  /** ${Add}s a single element to this $coll.
   *
   *  @param elem  the element to $add.
   *  @return the $coll itself
   */
  def +=(elem: A): this.type
```



```scala
trait HasNewBuilder[+A, +Repr] extends Any {
  /** The builder that builds instances of Repr */
  protected[this] def newBuilder: Builder[A, Repr]
}
```

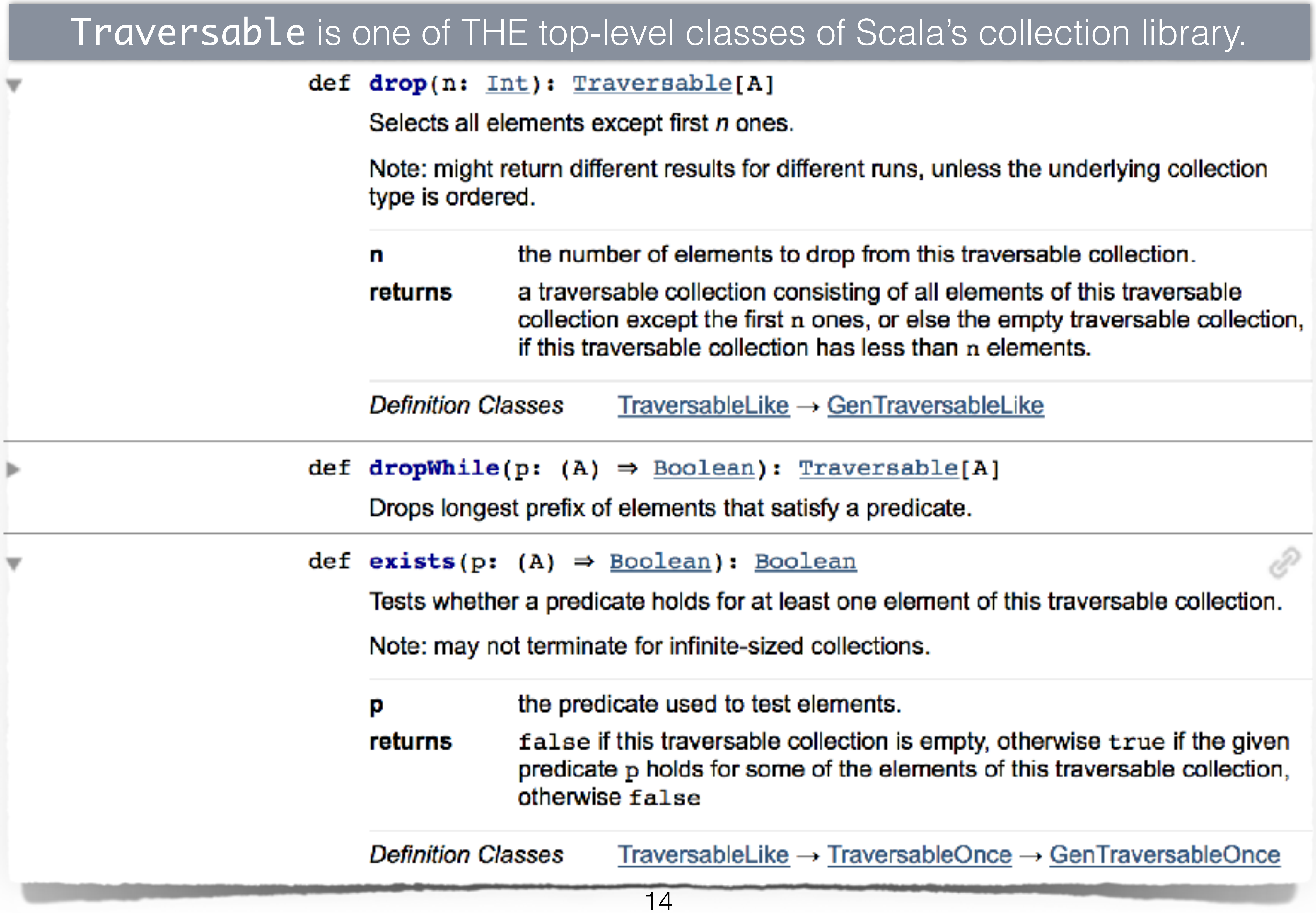
ISP in Scala (2.12.x) - Case Study



```
trait  MapLike[K, +V, +This <: MapLike[K, V, This] with
Map[K, V]] extends PartialFunction[K, V] with
IterableLike[(K, V), This] with GenMapLike[K, V, This]
with Subtractable[K, This] with Parallelizable[(K, V),
ParMap[K, V]]
```

Do we have an ISP violation?

`scala.collection.Traversable` (excerpt)



`Traversable` is one of THE top-level classes of Scala's collection library.

```
def drop(n: Int): Traversable[A]
```

Selects all elements except first *n* ones.

Note: might return different results for different runs, unless the underlying collection type is ordered.

n	the number of elements to drop from this traversable collection.
returns	a traversable collection consisting of all elements of this traversable collection except the first `n` ones, or else the empty traversable collection, if this traversable collection has less than `n` elements.

Definition Classes TraversableLike → GenTraversableLike

```
def dropWhile(p: (A) ⇒ Boolean): Traversable[A]
```

Drops longest prefix of elements that satisfy a predicate.

```
def exists(p: (A) ⇒ Boolean): Boolean
```

Tests whether a predicate holds for at least one element of this traversable collection.

Note: may not terminate for infinite-sized collections.

p	the predicate used to test elements.
returns	`false` if this traversable collection is empty, otherwise `true` if the given predicate `p` holds for some of the elements of this traversable collection, otherwise `false`

Definition Classes TraversableLike → TraversableOnce → GenTraversableOnce

***I**nterface (/ **T**rait) **S**egregation **P**rinciple*
(In case of Java 8 (/ Scala).)

Clients should not be forced to depend on methods that they do not use.

Subtypes should not be forced to inherit methods which have a specific semantics.

ISP violations in particular lead to …
(a) increased maintenance efforts and (b) reduced reusability.

–Agile Software Development; Robert C. Martin; Prentice Hall, 2003