Winter Semester 16/17

Software Engineering Design & Construction

Dr. Michael Eichberg
Fachgebiet Softwaretechnik
Technische Universität Darmstadt

Interface Segregation Principle

Interface Segregation Principle

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

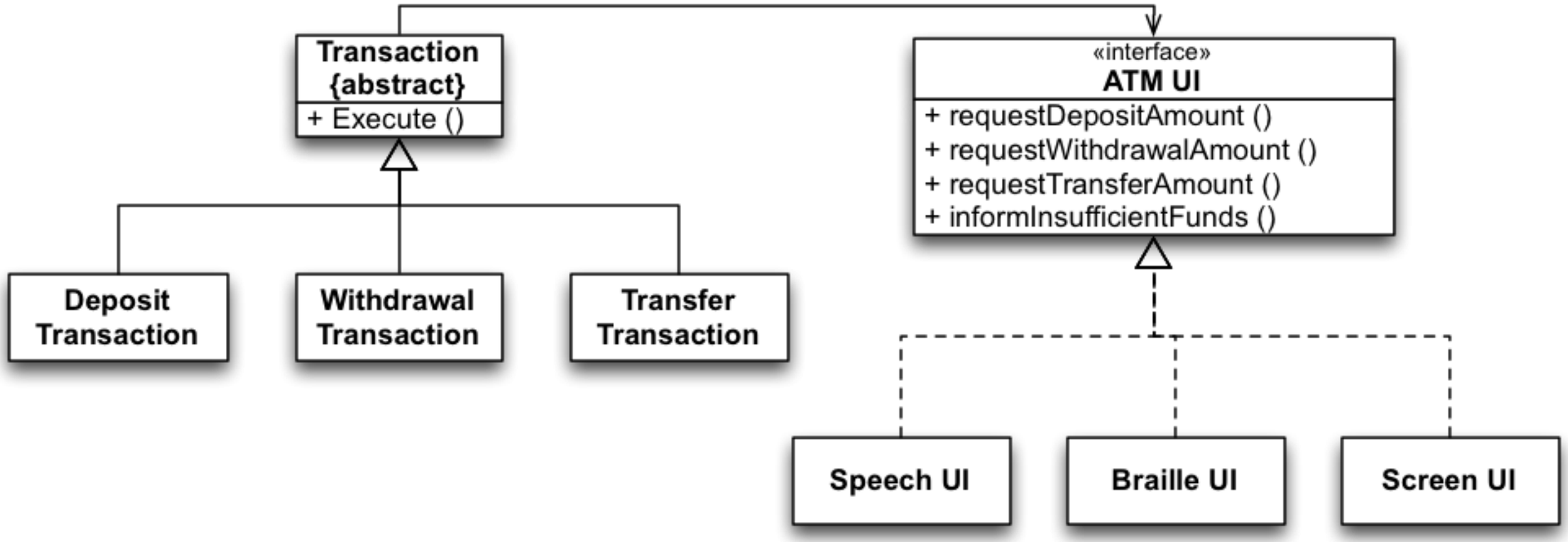
–Agile Software Development; Robert C. Martin; Prentice Hall, 2003

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?

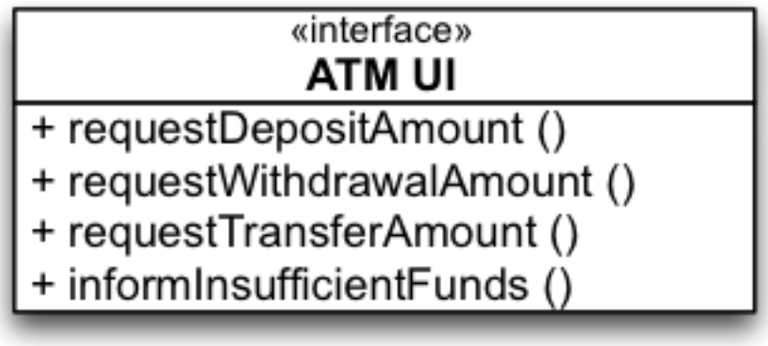
A Polluted Interface

ATM UI is a polluted interface!

- It declares methods that do not belong together.

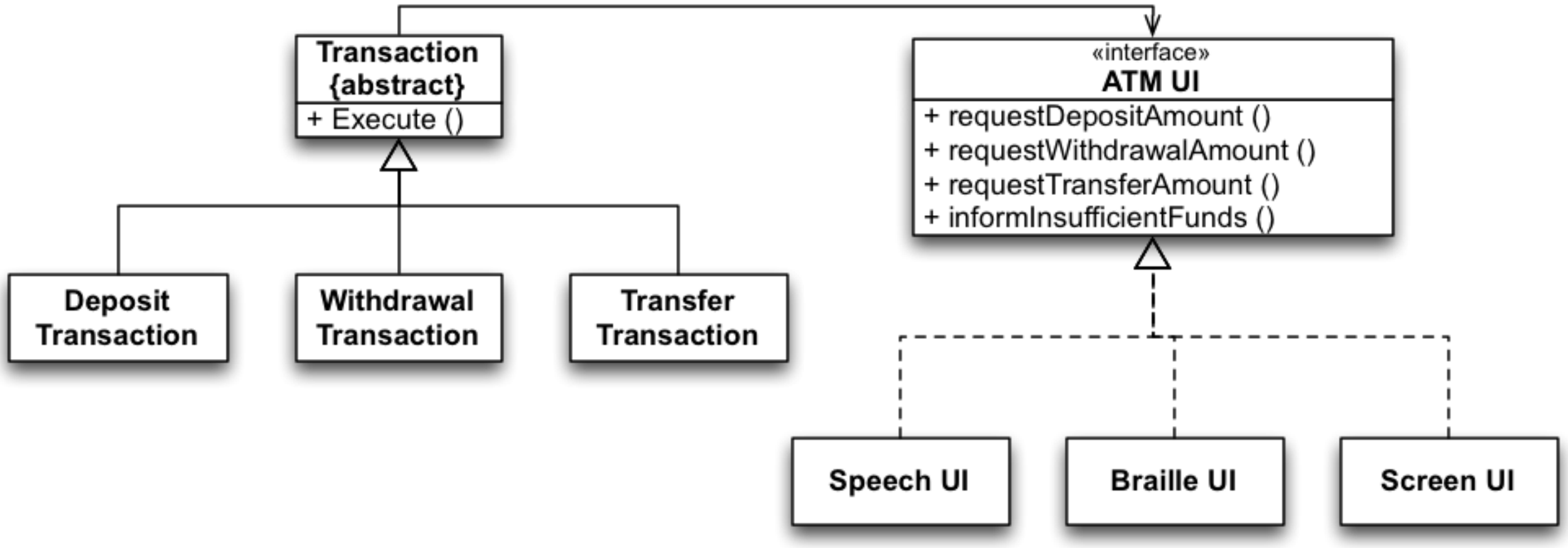
- It forces classes to depend on unused methods and therefore depend on changes that should not affect them.

- ISP states that such interfaces should be split.

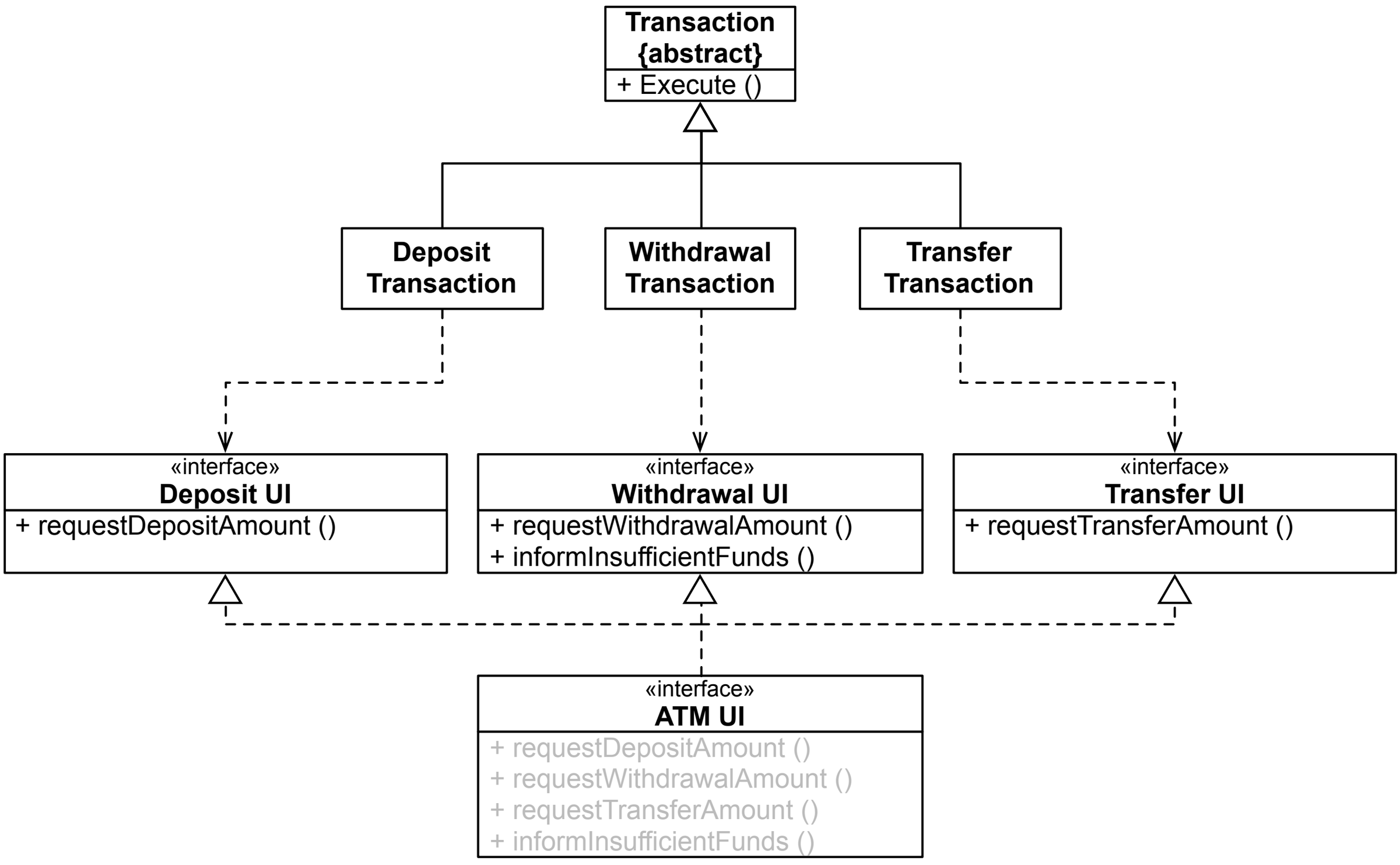


When clients depend on methods they do not use, they **become** subject to changes forced upon **these methods** by other clients.

How does an ISP compliant solution look like?



An ISP Compliant Solution



***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 or where different semantics are easily imaginable.

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

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

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

Proliferation of Interfaces/Traits

scala.collection.Traversable

Do we have an ISP violation?

```
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 or where different semantics are easily imaginable.

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