accenture
Latvia ATC

Test Automation Engineering Fundamentals: Java

Module 5: Object-Oriented Design and Principles

Module Objectives

- At the end of this module, participants will be able to:
  - Define the concept of software abstraction in Object-Oriented languages
  - Describe the benefits of an *outward-in* approach when designing classes
  - Explain the principles of *high cohesion* and *loose coupling* in object design

Software Abstraction

accenture
Latvia ATC

- Abstraction is the process of reducing or simplifying information or a concept in order to retain content relevant to the current context

- Data is abstracted by high level language constructs, such as:
  - Numbers
  - Letters
  - Higher level data concepts

- Functionality is abstracted by presenting interfaces that hide complex logic or algorithms, such as:
  - Functions
  - Methods

Abstraction and Object-Oriented Programming (OOP)

accenture Latvia ATC

An OOP language is built around the concept of a software entity called an ***object***

An object is defined as the binding of data and the *relevant* functions and behavior exposed through a public interface

Objects communicate through the use of public interfaces and do not need to know the inner workings of other objects

Objects are based on classes, which are used to define the attributes and behaviors common to all instances of that class

Classes have hierarchies that define abstractions in their classifications

Object-Oriented Programming (OOP)

accenture
Latvia ATC

- Object orientation is a powerful means of expressing a problem domain and its solution, but it is still subject to limitations and pitfalls.
- The mindset of an OO designer must be focused on creating small, simple re-usable components that can be combined to create more complex systems.

Design by Interface

accenture
Latvia ATC

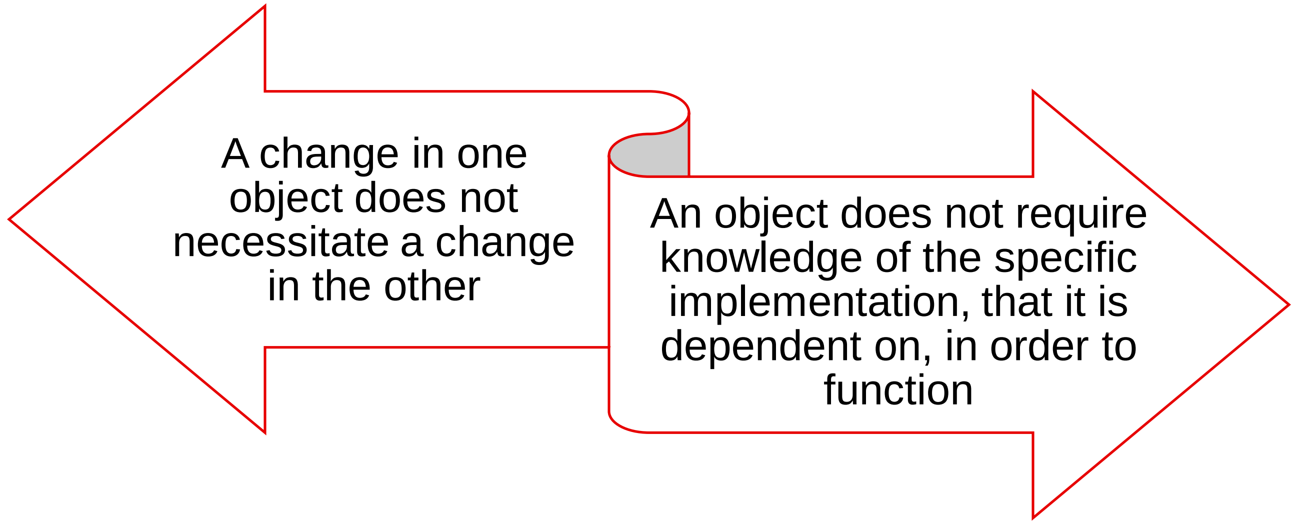
- An OO designer needs to separate the concerns of *behavior* from the concerns of *implementation.*
- It is a good principle to first define *behavior* by defining the interfaces that will be implemented by a class of objects.
- Defining interfaces first imposes a mind-set of determining how the different objects communicate with each other.
- Client objects should only be aware of the interfaces that are implemented by the objects they are dependent on.
- The exposed interfaces should present parameters and return types that are not dependent on the implementation details.

High Cohesion

- It is important to remember that an object binds ***relevant*** data and behavior together.
- A common pitfall in OO is designing an object to be able to do too many, possibly unrelated or loosely related things (low cohesion).
- An object exhibits high cohesion if all of its behavior is strongly related within the context of the object’s overall functionality.
- The functionality of the object itself must be designed with a very focused and specific goal.

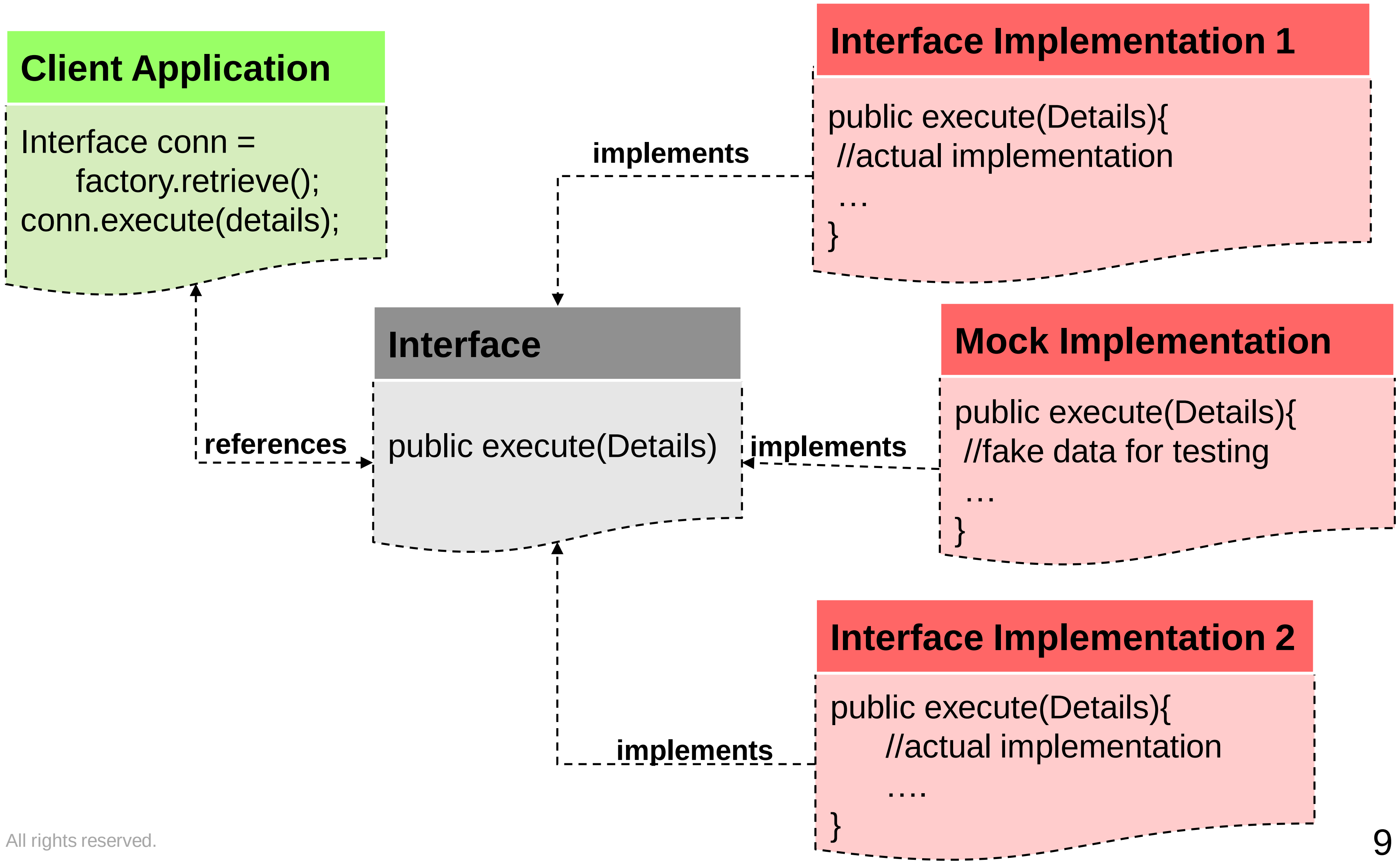
Loose Coupling

- An object is loosely coupled with another object when:


A change in one object does not necessitate a change in the other
An object does not require knowledge of the specific implementation, that it is dependent on, in order to function

- Is a natural result of high cohesion and designing by interfaces

Code by Interface



```
Interface conn =
      factory.retrieve();
conn.execute(details);
```

```
public execute(Details)
```

```
public execute(Details){
 //actual implementation
 …
}
```

```
public execute(Details){
 //fake data for testing
 …
}
```

```
public execute(Details){
      //actual implementation
      ….
}
```

Questions and Comments

accenture
Latvia ATC

- What questions or comments do you have?