

Conclusion



Richard Warburton

@richardwarburto www.monotonic.co.uk



Contents

Common Uses

Examples of interface
use in the wild

The Java Language

Recap key Java
interface features

Principles and Tradeoffs

The good, the bad and
the ugly



Common Uses



Dependency

A relationship between two components where the functionality of one component relies on another component



Example of Dependency Injection

```
public class ClientEngagmentController {  
  
    final ClientEngagementRepository repo;  
  
    ClientEngagmentController(  
        final ClientEngagementRepository repo)  
        this.repo = repo;  
  
    ...  
}
```



Testing

Stubs

Provide specific answers
to method calls

Used by code under
test to isolate it

Stubs often implement
abstract methods

Mocks

Verify that certain
methods are called

Used to test the behavior
of code under test

Often mock
interfaces



```
ClientEngagementRepository stubRepo = Mockito.mock(  
    ClientEngagementRepository.class);
```

```
List<ClientEngagement> engagements = ...
```

```
Mockito.when(stubRepo.find(any())) .thenReturn(engagements)
```

Stubbing Using Mockito

Provide a pre-canned value to return from the find method

This is used by code under test



Mocking Using Mockito

```
ClientEngagementRepository mockRepo = Mockito.mock(
    ClientEngagementRepository.class);

ClientEngagementController controller = new
    ClientEngagementController(mockRepo);

controller.saveEngagement(httpRequest, httpResponse);

Mockito.verify(mockRepo).add(eq(engagement));
```



Design Patterns

A solution to a common problem in software design

The solution should be general in the sense of being a template for other implementations of that solution



```
public interface ActionListener extends EventListener {  
    public void actionPerformed(ActionEvent e);  
}
```

```
JButton button = new JButton();  
button.addActionListener(clickListener);
```

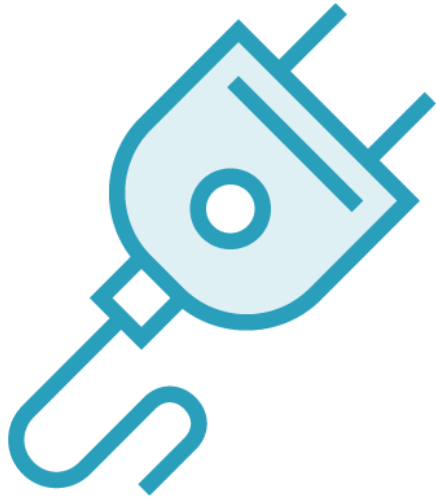
Observer Pattern

Example of interfaces being used for the Observer Pattern

ActionListener defines a callback to listen to click events

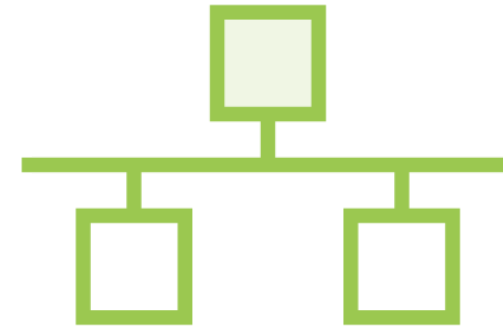


Other Examples



Plugins

Interfaces often used to declare an API for plugin writers



Ports and Adapters

Hexagonal architecture
Interfaces used to define ports
Adapters implement them



The Java Language



Polymorphism

Objects of a child class can be referred to by their parent's class, methods called on the parent bind to the child's implementation



```
interface RevenueCalculator  
{
```

```
class HourlyRateCalculator implements RevenueCalculator  
{
```

Interfaces



Methods

Abstract Classes

Methods without keyword have bodies

`abstract` keyword let's you remove the body

Methods can be `public`, `private`, `protected` or `package-private` – the default

Interfaces

Methods without keywords don't have bodies

`default` keyword lets you add a body

All methods are `public`



Fields

Abstract Classes

Can have fields

Non-private fields visible
in subclasses

Interfaces

Cannot have instance fields

No sharing of state



Inheritance



Abstract Classes
Single inheritance



Interfaces
Multiple inheritance



Principles and Tradeoffs



Why We Use Abstractions

Extensibility

Add behavior without
modifying the class

Polymorphism

Method invocation decided
at runtime



Potential Cons

False Abstractions

Poor Naming

**Single/Incomplete
Implementations**

YAGNI



Summary



Summary



Interfaces are a great Java language feature

Help write cleaner and more maintainable code

Using them effectively is key to good object oriented programming

