Contract-based Software Development

Rasmus Guldborg Pedersen

January 2015

Overview

Contracts in Design Patterns

Present a number of design patterns in which contracts play an essential role. You may use Code Contract to illustrate contracts.

Command Pattern

- Commands can execute a single operation

Command Pattern

- Commands can execute a single operation
- Interface with signature for the operatrion

Command Pattern

- Commands can execute a single operation
- Interface with signature for the operatrion
- The invoker does not know the concrete type

Command Pattern

- Commands can execute a single operation
- Interface with signature for the operatrion
- The invoker does not know the concrete type
- Used to queue operations

Command Pattern

- Commands can execute a single operation
- Interface with signature for the operatrion
- The invoker does not know the concrete type
- Used to queue operations
- Used to implement undo functionality

Command Pattern: Interface

```
[ContractClass(typeof(ICommandContract))]
public interface ICommand {
    IImmutableStack<int> Execute(
            IImmutableStack<int> stack);
    IImmutableStack<int> Undo(
            IImmutableStack<int> stack);
}
```

Command Pattern: Contract

```
[ContractClassFor(typeof(ICommand))]
abstract class ICommandContract {
    IImmutableStack<int> Execute(
            IImmutableStack<int> stack) {
        Contract.Requires(stack != null);
        Contract.Ensures(
            Undo(Contract.Result<IImmutableStack<int>>())
            == Contract.OldValue(stack));
    }
    // ...
}
```

Observer Pattern

Allow a number of observers to subscribe to changes on an observable object.

Observer Interface

```
public interface IObserver {
    void Update();
    bool IsUpdated();
}
```

Observable Interface

```
public interface IObservable {
    void Attach(IObserver observer);
    void Detach(IObserver observer);
    bool IsAttached(IObserver observer);
    IEnumerable GetObservers();
    void Notify();
}
```

Observable Contract

```
abstract class IObservableContract {
    void Attach(IObserver observer) {
        Contract.Requires(observer != null);
        Contract.Requires(!IsAttached(observer));
        Contract.Ensures(IsAttached(observer));
    }
    // ...
}
```

Observable Contract

```
abstract class IObservableContract {
    // ...
    bool IsAttached(IObserver observer) {
        Contract.Requires(observer != null);
        Contract.Ensures(Contract.Result<bool>()
            == GetObservers()
                .Contains(Contract.OldValue(observer)));
    }
}
```

Observable Contract

```
abstract class IObservableContract {
    // ...
    void Notify() {
        Contract.Ensures(Contract.ForAll(GetObservers(),
            x => x.IsUpdated()));
    }
}
```

The End

"Testing shows the presence, not the absence of bugs."
— Edsger W. Dijkstra