- Refactoring Java Contracts
- By Iain Hull

Agenda

- What are Java Contracts
- Java Contracts View
- Refactoring Operations
  - Pull up
  - Push down
- Components

Java Contracts

- Representation of JML using Java code
  - By Patrice Chalin & Bobby
- Reuses JML 5 Annotations
  - By Kristina Taylor
- Inspired by Code Contracts in .net
  - By Microsoft

Java Contracts Example

```
import org.jmlspecs.annotation.NonNull;

public class Node<E> {
    private @NonNull E data;
    private @Nullable Node<E> next;

    public Node(final E data, final Node<E> next) {
        JC.requires(data != null);
        JC.ensures(this.data == null && this.next == next);

        this.data = data;
        this.next = next;
    }
```

Java Contract Types

- Method based
  - Invariants
  - Model Fields
- Statement based
  - Preconditions
  - Postconditions
- Block Based
  - SpecCase

Java Contracts View

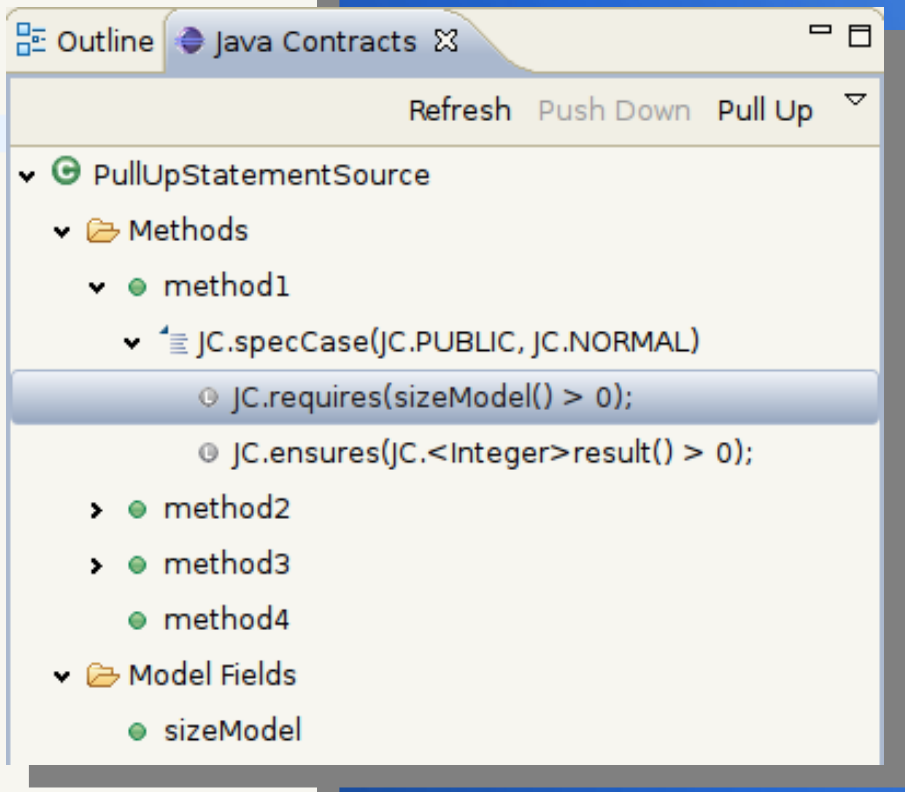
```java
import org.jmlspecs.annotation.ModelField;

 * Test Case class used as the source of PullUp Statement refactorings. The
public class PullUpStatementSource extends PullUpStatementDestination {
    @ModelField
    public int sizeModel() {
        return JC.someValue();
    }

     * Test case (fail) does use model
    @Override
    @Pure
    public int method1() {
        JC.requires(sizeModel() > 0);
        JC.ensures(JC.<Integer>result() > 0);
        return 1;
    }


     * Test case (success) does not use model
    @Override
    @Pure
    public int method2() {
        JC.requires(number > 0);
        JC.ensures(JC.result() == Integer.valueOf(1));

        return 1;
    }
```



Refactoring Operations

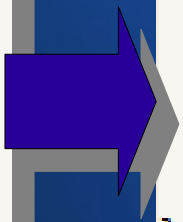
- Pull Up
  - Search for Super types
  - Search for code not available in destination
- Push Down
  - Search for Sub types
  - Search for code not available in destination

Pull-up Example

```
class A {
    public void foo(int a) {
        // Do something smart
    }
}

class B {
    void foo(int a) {
        JC.requires(a > 1);

        // Do something really smart
    }
}
```



```
class A {
    public void foo(int a) {
        JC.requires(a > 1);

        // Do something smart
    }
}

class B {
    void foo(int a) {
        // Do something really smart
    }
}
```

Plugin Structure