Software Change Contract

CS4271

Jooyong Yi
National University of Singapore

Hoare Triple

- {Pre} P {Post}
  - P: a given program
  - Pre: pre-condition that should hold before executing P
  - Post: post-condition that should hold after executing P
- Example:
  - {x>=0} y=abs(x); {y==x} {x<0} y=abs(x); {y==-x}

Today's Topics

- Software change contract
  - Hoare triple expressing program changes
- Java Modeling Language (JML)
  - Program-like Hoare triple
- We touch only specification (leaving verification a black box).

From Hoare Triple to JML

From Hoare Triple to JML

- Start with {x>=0} y=abs(x); {y==x}

From Hoare Triple to JML

- Start with {x>=0} y=abs(x); {y==x}
- In JML, specify Hoare triple above a method declaration

From Hoare Triple to JML

- Start with {x>=0} y=abs(x); {y==x}
- In JML, specify Hoare triple above a method declaration

```
//@ requires x>=0;
//@ ensures ???;
public int abs(int x) { /* body */ }
```


From Hoare Triple to JML

- Start with {x>=0} y=abs(x); {y==x}
- In JML, specify Hoare triple above a method declaration

```
//@ requires x>=0;
//@ ensures \result == x;
public int abs(int x) { /* body */ }
```

From Hoare Triple to JML

- Full specification

```
//@ requires x>=0;
//@ ensures \result==x;
//@ also
//@ requires x<0;
//@ ensures \result==-x;
public int abs(int x) { /* body */ }
```


Program Contract

```
//@ requires x>=0;
//@ ensures \result==x;
//@ also
//@ requires x<0;
//@ ensures \result==-x;
public int abs(int x) { ... }
```

Contract between method abs and its caller

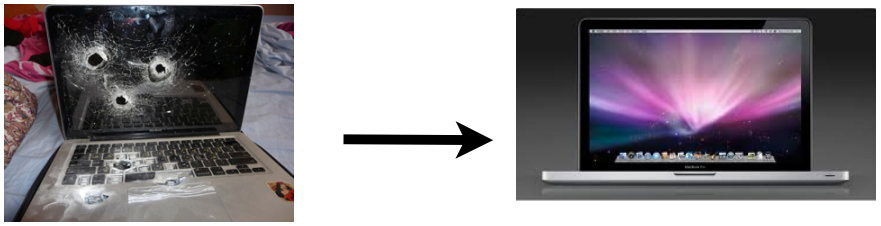
abs : service
caller: client

Change Contract

When we get things repaired ...

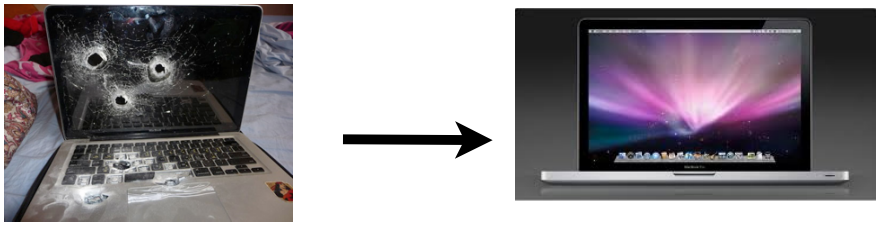
- We make a contract with a service person

When we get things repaired ...



- Contract specifies which part will be fixed or replaced.

When we get things repaired ...



- Contract specifies which parts will be fixed or replaced.
- Does not explicitly specify which parts will be remained.

Change Contract?

```
public int abs(int x) {

// previous impl

}
```

Change Contract →

```
public int abs(int x) {

// new impl

}
```

- contract between two versions of a method.
- describes how method behavior changes.

Change Contract Example

-10 10

```
public int abs(int x) {
  return (x>0)? x : -x;
}
```

result == 10

Change Contract Example

Integer.MIN_VALUE

```
public int abs(int x) {
  return (x>0)? x : -x;
}
```

```
\result == -Integer.MIN_VALUE
```


Change Contract Example

Integer.MIN_VALUE

```
public int abs(int x) {
  return (x>0)? x : -x;
}
```

~~\result == -Integer.MIN_VALUE~~

\result == Integer.MIN_VALUE

Change Contract Example

Integer.MIN_VALUE

-2^31 ≤ Integer ≤ 2^31-1

```
public int abs(int x) {
  return (x>0)? x : -x;
}
```

~~\result == -Integer.MIN_VALUE~~

\result == Integer.MIN_VALUE

Change Contract Example

Integer.MIN_VALUE

```
public int abs(int x) {
  // modified impl
}
```

~~\result == -Integer.MIN_VALUE~~

~~\result == Integer.MIN_VALUE~~

???

Change Contract Example

Integer.MIN_VALUE

```
public int abs(int x) {
  // modified impl
}
```

signals OutOfBoundsException

Change Contract Example

```
requires x == Integer.MIN_VALUE;
ensured \result == Integer.MIN_VALUE;
signals (OutOfBoundsException) true;
```

```
public int abs(int x) {

return (x>0)? x : -x;

}
```

Change Contract

```
public int abs(int x) {

 // new impl

}
```


Understanding Change Contract

```
requires x == Integer.MIN_VALUE;
ensured \result == Integer.MIN_VALUE;
signals (OutOfBoundsException) true;
```

Integer.MIN_VALUE

```
public int abs(int x) {

return (x>0)? x : -x;

}
```

```
public int abs(int x) {

 // new impl

}
```


Understanding Change Contract

```
requires x == Integer.MIN_VALUE;
ensured \result == Integer.MIN_VALUE;
signals (OutOfBoundsException) true;
```

Integer.MIN_VALUE

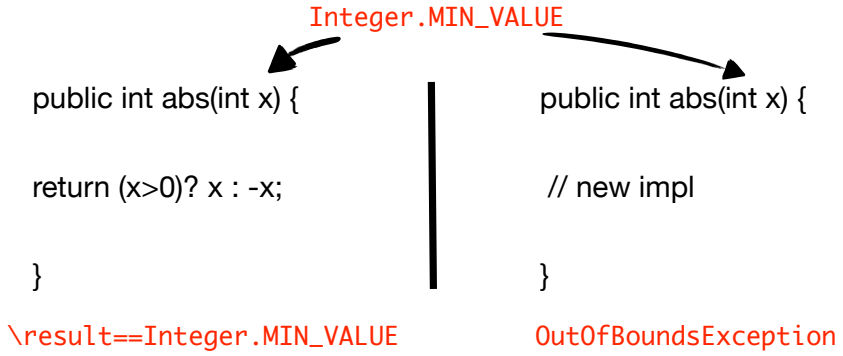
```
public int abs(int x) {

return (x>0)? x : -x;

}
```

```
public int abs(int x) {

 // new impl

}
```

\result==Integer.MIN_VALUE

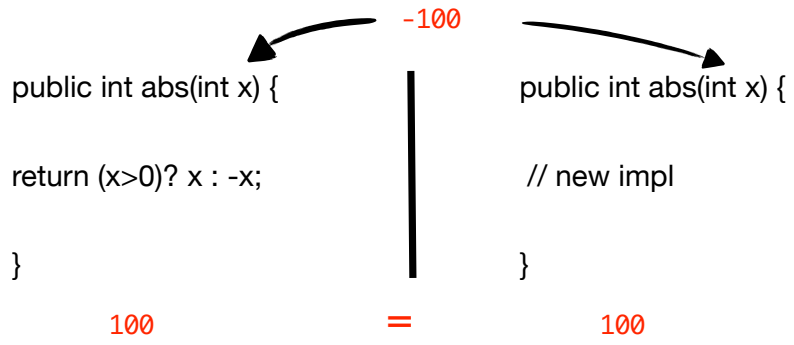
Understanding Change Contract

```
requires x == Integer.MIN_VALUE;
ensured \result == Integer.MIN_VALUE;
signals (OutOfBoundsException) true;
```



Equal Unless Specified

```
requires x == Integer.MIN_VALUE;
ensured \result == Integer.MIN_VALUE;
signals (OutOfBoundsException) true;
```



More Change Contract Examples

Unexpected Exception

```
signaled (NullPointerException) find(name)==null;
signals (NullPointerException) false;
```

```
void delete(String name) {
  File f = find(name);
  f.remove( );
}
```

→

```
void delete(String name) {
  File f = find(name);
  if (f != null) {
    f.remove( );
  }
}
```

Alternative Exception

```
signaled (NullPointerException) find(name)==null;
signals (NoSuchFileException) true;
```

```
void delete(String name) {
  File f = find(name);
  f.remove( );
}
```

```
void delete(String name) {
  File f = find(name);
  if (f == null) {
    throw new
    NoSuchFileException(name);
  } else { f.remove( ); }
}
```


Fixing Broken Assumption

```
ensured !\result.equals(\result.trim());
ensures \result.equals(\result.trim());
```

```
String getID( ) {

// previous impl

}
```

“ JohnSmith ”

```
String getID( ) {

// new impl

}
```

“JohnSmith”

Fixing Broken Assumption

```
requires x == Integer.MIN_VALUE;
ensured \result == Integer.MIN_VALUE;
ensures \result.longValue() ==
        -((long) \prev(\result));
```

```
int abs(int x) {
  return (x>0)? x : -x;
}
```

```
BigInteger abs(int x) {
  Integer i = new Integer(x);
  new BigInteger(i.toString( )).abs( );
}
```


Structural Changes (Refactoring)

```
boolean
withinRange(int low, int high) {
  return this.low <= low &&
  this.high >= high;
}
```

```
boolean
withinRange(Range r) {
  return this.low<=r.getLow( )
  && this.getHigh( )>=high;
}
```

- No behavioral change

Structural Changes (Refactoring)

- Remember “Equal Unless Specified”
- Yet need to relate structurally different two sets of input

```
boolean                          boolean
withinRange(int low, int high) { withinRange(Range r) {
  return this.low <= low &&        return this.low<=r.getLow( )
  this.high >= high;               && this.getHigh( )>=high;
}                                }
```


After Refactoring

```
old_param low:int, high:int;
new_param r:Range;
matches [                    ];
```

generated template

```
boolean                          boolean
withinRange(int low, int high) { withinRange(Range r) {
  return this.low <= low &&        return this.low<=r.getLow( )
  this.high >= high;               && this.getHigh( )>=high;
}                                }
```


Filling In the Blank

```
old_param low:int, high:int;
new_param r:Range;
matches r.getLow()==\prev(low) &&
        r.getHigh()==\prev(high) ;
```

```
boolean                          boolean
withinRange(int low, int high) { withinRange(Range r) {
  return this.low <= low &&        return this.low<=r.getLow( )
  this.high >= high;               && this.getHigh( )>=high;
}                                }
```


Why \prev Expression?

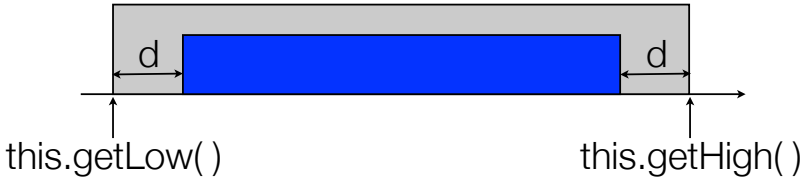
```
old_param low:int, high:int;
new_param r:Range;
matches r.getLow()==\prev(low) &&
        r.getHigh()==\prev(high);
```

```
boolean
withinRange(int low, int high) {
  // previous impl
}
```

```
class Range {
  int low, high;

  boolean
  withinRange(Range r) {
    // new impl
  }
```

New Feature Addition

```
boolean                              boolean
withinRange(Range r) {               withinRange(Range r, int d) {
  return this.low<=r.getLow( )         return this.low+d<=r.getLow( )
  && this.getHigh( )>=high;      →     && this.getHigh( )>=high-d;
}                                    }
```



New Feature Addition

```
old_param r:Range;
new_param r:Range, int d;
matches [          ];
```

```
boolean                              boolean
withinRange(Range r) {               withinRange(Range r, int d) {
  return this.low<=r.getLow( )         return this.low+d<=r.getLow( )
  && this.getHigh( )>=high;      →     && this.getHigh( )>=high-d;
}                                    }
```


New Feature Addition

```
old_param r:Range;
new_param r:Range, int d;
matches d == 0 ;
```

```
boolean                              boolean
withinRange(Range r) {               withinRange(Range r, int d) {
  return this.low<=r.getLow( )         return this.low+d<=r.getLow( )
  && this.getHigh( )>=high;      →     && this.getHigh( )>=high-d;
}                                    }
```


Similar Treatment of Fields

```
new_field i:int;
old_param i:int;
matches this.i==\prev(i) ;
```

```
class C {                      →     class C {
  void m(int i) { ... }                int i;
}                                      void m( ) { ... }
                                     }
```