Representation Hiding in JML

Curt Clifton
Rose-Hulman Institute of Technology

Steps for Specifying a Class

- 1. Specify the data…
  - Using **spec_public** fields for now
  - Later we'll see how to be more abstract
- 2. Specify **invariant**s
- 3. Specify each public method/constructor using:
  - **requires**, **assignable**, and **ensures**

Hiding Implementations: Specifying Interfaces

- We've seen examples of JML specs where the fields are **spec_public**
- But that violates information hiding!
- Solution:
  - **model** fields
  - *data groups*
  - **represents** clauses

Model Fields

- JML lets us add *model fields* to interfaces
- Model fields are just an abstract representation of object state
- Interfaces cannot be instantiated
- Can choose a model representation that is logically clear, even if it is inefficient

Warning!

- I’m using Lifts for the example and quiz
- For P3 you probably want to use the more direct specifications like we did last time and on HW10

Lift Interface Model

INDICATES THAT THIS IS A "SPECIFICATION ONLY" FIELD

FIELDS IN JAVA INTERFACES ARE STATIC BY DEFAULT

```
public interface LiftInterface {
    //@ public model instance int currentFloor;
    //@ public model instance int topFloor;
    //@ public model instance JMLValueSet requests;
    //@ public model instance int arrows;
    //@ public model instance int direction;
    //@ public model instance boolean doorsClosed;
```

UNLIKE Z, JML USES A REGULAR JAVA LIBRARY TO REPRESENT MATHEMATICAL IDEAS

Q: MODEL FIELDS

Lift Interface Invariant

```
/*@ public invariant
  @       1 <= currentFloor && currentFloor <= topFloor;
  @  public invariant (\forall int i; requests.contains(i);
  @                                   1 <= i && i <= topFloor);
  @  public invariant (direction == UP
  @                             || direction == DOWN
  @                             || direction == STOP);
  @  public invariant
  @              !doorsClosed ==> (arrows == direction)
  @  public invariant
  @              doorsClosed ==> (arrows == STOP)
  @*/
```

CARTOON OF THE DAY

DON'T FORGET TO BUY BEER FOR YOUR MONKEYS

Relating Concrete Fields to the Model

- Data groups
  - Let us say that whenever a model field changes, this concrete field can change
  - Denoted by **in** and **maps**
- Represents clauses
  - Relate private implementation to public model
  - **represents modelField <- *<expr>***
  - **represents modelField \such_that *<pred>***

Data Groups and Represents Clauses—1

DATA GROUP

```
class Lift implements LiftInterface {
    private int location; //@ in currentFloor;
        //@ private represents currentFloor <- location + 1;
```

PRONOUNCED "AS"

Q: BEGIN "REPRESENTATION"

Data Groups and Represents Clauses—2

```
private boolean[] floorRequestLight;
    //@ maps floorRequestLight[*] \into requests;
    /*@ private represents requests \such_that
      @     (\forall int i; 0<=i && i<floorRequestLight.length;
      @         floorRequestLight[i] <==> requests.contains(i+1));
      @ private represents topFloor <- floorRequestLight.length;
      @*/

private boolean upArrowLight; //@ in arrows;
    /*@ private represents arrows \such_that
      @         upArrowLight <==> (arrows == UP);
      @*/

private boolean downArrowLight; //@ in arrows;
    /*@ private represents arrows \such_that
      @         downArrowLight <==> (arrows == DOWN);
      @*/
```

Q: FINISH "REPRESENTATION"

In the beginning…

INITIALLY CLAUSES CONSTRAIN ALL CONSTRUCTORS

```
/*@ public initially requests.isEmpty()
  @  public initially currentFloor == 1
  @  public initially arrows == STOP
  @  public initially direction == STOP
  @  public initially doorsClosed;
  @*/
```

SPECIAL DATA GROUP CONTAINING ALL FIELDS

```
/*@ requires numberOfFloors > 0;
  @ assignable objectState;
  @ ensures topFloor == numberOfFloors;
  @*/
public Lift(int numberOfFloors) { … }
```

Q: INITIALLY BLOCK & CONSTRUCTOR

Does Implementation Meet Specification?

LIFT'S CODE `this.location = 0;`

BY SEMANTICS OF ASSIGNMENT

CONCRETE POSTCONDITION `location == 0;`

BY REPRESENTS CLAUSE

MODEL'S POSTCONDITION `currentFloor == 1;`

CONSTRUCTOR'S POSTCONDITION SATISFIES INITIALLY CLAUSE

JUST A SMALL SAMPLE FROM LIFT'S CONSTRUCTOR

Method Specifications Using the Model

```
/*@ requires 1 <= floor && floor <= topFloor;
  @ assignable requests;
  @ ensures requests.contains(floor);
  @*/
public void buttonPressed(int floor);
```

COMPARE TO ORIGINAL SPEC USING CONCRETE REPRESENTATION

```
/*@ requires 0 < floor && floor <= floorRequestLight.length;
  @ assignable floorRequestLight[floor-1];
  @ ensures floorRequestLight[floor-1];
  @*/
```

WITH MODEL-BASED SPECIFICATION

NO ADDITIONAL SPEC IS NEEDED FOR CONCRETE IMPLEMENTATION

Q: FINISH

Warning!

- I used Lifts for the example and quiz
- For P3 you probably want to use the more direct specifications like we did last time and on HW10