CIS 771: Software Specifications

Introduction to JML

Java Modeling Language (JML)

- A behavioral interface specification language for Java
  - supporting design-by-contract (DBC)
- ... invented by Gary T. Leavens in the 90s
  - the de facto Java specification language in formal methods research community
  - over 100 research papers and 30 groups

Alloy, OCL, and JML

- JML features invariants, pre/postconditions, etc.
  - a code-level specification language
    - ... less abstract than Alloy and OCL
  - based on Java
    - ... more familiar to software developer
    - ... do not need to learn a *completely* different formalism
  - supported by various tools/techniques
    - e.g., runtime checkers, static analyzers

Overview of JML Features

- Type specifications: invariants, etc.
- Method specifications: pre/postconditions, frame conditions, etc.
- Specification expressions: Java expressions + JML-specific constructs
- ... we'll focus on lightweight specifications
- Note: throughout the lecture, we'll refer to sections of the JML Reference Manual using the following form: (§ section number)

JML Specifications

- ... are written inside Java comments immediately before Java program elements
  - newline comments of the form: `//@` ...
  - block comments: `/*@` ... `@*/`
  - multi-line block comments:
    ```
    /*@  ...
      @  ...
      @*/
    ```
  - note: no space between `/*` (or `//`) and `@`

Type Specifications (§8)

- User-defined types in Java are classes and interfaces
- JML supports specification of types

```
public class Container {
  //@ invariant this.data != null;
  Object data;

  ...
}
```

- the above is *instance* invariant constraining *object* field values

Invariants (§8.2)

- Invariants can be instance (default) or static
  - static inv. constrains *class* static fields
  - instance inv. constrains *object* fields
  - ... we'll focus only on instance invariants
- Instance invariants should be
  - established by non-helper constructors
  - preserved by non-helper instance methods
- `/*@ helper @*/` marks helper methods

Method Specifications (§9)

- ... we'll only consider lightweight specification cases (§9.4)
- Method (or constructor) clauses
  - preconditions: `requires` clauses (§9.9.2)
  - postconditions: `ensures` clauses (§9.9.3)
  - frame conditions: `assignable` clauses (§9.9.9)
  - etc.

requires Clause (§9.9.2)

- requires clause specifies method preconditions

```
public class Container {
  //@ invariant this.data != null;
  Object data;

  //@ requires data != null;
  public Container(Object data) {
    this.data = data;
  }
  ...
}
```

ensures Clause (§9.9.3)

- ensures clause specifies method postconditions

```
public class Container {
  ...

  //@ ensures \result == this.data;
  public Object getData() {
    return this.data;
  }
  ...
}
```

- \result is a JML expression for denoting a method's return value (§11.4.1)

ensures Clause (§9.9.3)

- postconditions may refer to “old” values (i.e., values at the method entry point)

```
/*@ requires other != null;
  @ ensures this.data == \old(other.data)
  @      && other.data == \old(this.data);
  @*/
public void swap(Container other) {
  Object temp = this.data;
  this.data = other.data;
  other.data = temp;
}
```

- \old() is a JML-specific expression for retrieving old values (§11.4.2)

assignable Clause (§9.9.9)

- assignable clause specifies the frame condition of a method
  - specifying what *may* be changed

```
/*@ requires other != null;
  @ assignable this.data, other.data;
  @ ensures this.data == \old(other.data)
  @      && other.data == \old(this.data);
  @*/
public void swap(Container other) {
  Object temp = this.data;
  this.data = other.data;
  other.data = temp;
}
```

- ... in lightweight specification, its unspecified

Variable Nullity (§6.2.12)

- Variable nullity is a source of problem in many Java programs
  - ... causes `NullPointerException`
- By default, JML assumes all variables have non-null values (as invariants)
  - `/*@ nullable @*/` can be used at variable declarations to indicate otherwise, e.g.,

    ```
    public void swap(/*@ nullable @*/ Container other)...
    ```
  - i.e., `/*@ non_null @*/` is the default
- good practice: always *explicitly* specify one way or the other for documentation purposes

For You To Do

- Revise the `Container` example to use JML `nullable` or `non_null` modifiers on appropriate variable declarations
  - ... can all non-null-ness variable preconditions safely be replaced to use the `non_null` modifier?
- Think about the possible input states of the `Container.swap()` method
  - ... is there a subtle input state that you do not expect but its contract still holds?

JML Tools

- Many research tools have been developed for JML
  - documentation, e.g., JMLDoc
  - runtime checking, e.g., JML RAC
  - static analyzer, e.g., ESC/Java, Kiasan
  - model checking, e.g., Bandera/Bogor
  - theorem proving, e.g., JACK, LOOP
- ... in this course, we will use Kiasan

Why use JML?

- Java only supports assertion statement
  - ... not until Java 1.4
- JML offers syntactic sugars for embedding assertions at various program points
  - requires: assertions at method entry points
  - ensures: assertions at (normal) method exit points
  - invariant: assertions at method entry and exit points

Why use JML?

- Software developers usually write design intentions/ contracts informally in Java documentation comments
  - parameter x is not null
  - object field y must not be negative
  - etc.
- ... it cannot be leveraged for checking the programs
  - outdated "contract" are undetected
  - clients may not read documentation
  - contracts in a natural language are often ambiguous
- JML provides a way to have checkable documentation

Why not just use `assert`?

- Often times, the same conditions should be checked at multiple program points
  - thus, it is tedious to just use Java's `assert` statement
  - ... we might miss placing assertions at some places (Murphy's law)
- Developers usually do not write assertions in code
  - ... assertions are mostly used for testing

Why not just use `assert`?

- Assertions “polute” codebase
- What about error handling?
  - Java offers feature to disable assertions
  - implemented as a conditional

```
if (!$assertionDisabled) {
  // check assertion
  ...
}
```

  - but they are still in the compiled code

Error Handling

- Tools can be developed to for JML to handle assertion errors
  - during testing or analysis, test reports can be generated
  - during deployment, error feedback can be accumulated in a remote database
    - ask users whether to send feedback
- ... Separation of Concerns (SoC)

Looking ahead...

- In the future, you will be asked to write contracts
- Companies such as Microsoft are already moving in such direction
  - Spec# programming system
    http://research.microsoft.com/projects/specsharp/
  - Code Contract: DBC for .NET
    http://research.microsoft.com/projects/contracts/