Weekly Report

Team Dec 15-12: @PaniniJ

Week 6: *Feb. 23rd - Mar. 1st*

Advisor	Dr. Rajan	
Client	Dr. Rajan	
Team Members	Dalton Mills	*Webmaster*
	David Johnston	*Team Lead*
	Kristin Clemens	*Communication Lead*
	Trey Erenberger	*Key Concept Holder*

Weekly Summary

The main work this past week was on the design of duck futures. We spent more time analyzing the existing implementation, categorizing different cases that we will want to support, and designing what the auto-generated duck future implementations should look like.

Technical Progress

All members of the group have been investigating and discussing issues related to the design and implementation of Duck futures. Some additional individual contributions include.

- Dalton began implementing support for automatic signature generation from `@Signature`-annotated client interfaces.
- David wrote-up some of his thoughts on the implementation of Duck futures on the GitHub wiki. David has also been discussing with Kristin some ideas for testing our capsule implementation.
- Trey is working on implementing duck generation, researching model testing apparatuses and test code transformation.
- Kristin investigated alternative strategies for implementing duck futures, in particular using composition to simplify implementation and using reflection to overcoming the problems posed by the future keyword. She has additionally worked with David to

formulate a testing mechanism which is able to use an arbitrary template class as an oracle for the capsules which it generates. This would ideally be built upon unit tests written by the user for the template class.

Meetings

Weekly Administrative Meeting

Members Present: All
Additional Participants: N/A
Date & Location: Tuesday 24 of February; Molecular Biology 1414
Minutes:

- Start: 3:00
- Explanation of proposed class hierarchy for the generated capsule classes.
    - Separation of runtime 'views' of the capsule and user defined signatures
    - Factoring functionality into the shared interfaces to separate the paninij functions from the user defined code.
- Discussion of @paninij vs @Capsule, @Signature
    - Prefer @capsule and @signature as it is more formal and declarative
    - Prefer @capsule and @signature in case additional types are added that cannot infer their type by class and interface
- Demonstration of wrapping primitives in ducks
    - Final class problem
    - Primitive types
        - cannot put args onto message queue as primitives will need to be wrapped in objects without extra cost of object creation.
- Work objectives
    - Dalton working on signature generation
    - Kristin working on testing generated classes and ‘what’ to test versus ‘how’
- Discussion and research of alternatives to ducks extending the classes they hold.
- End: 5:40

Bi-Weekly Advisor Meeting

Members Present: All
Additional Participants: Dr. Rajan
Date & Location: Friday 27 of February; Atanasoff 101
Minutes:

- Discussed David's Duck Future Design Write-Up
    - No unnecessary method calls.
    - Stick closely to the existing duck future implementation from PaniniJ for performance reasons.
- Plan for Java 1.9
    - For use with CheckerFramework.
- Discussion of testing
    - Flexible creation of generated tests for generated code.
    - Minimize hard-coded test cases.
    - Review tests that already exist for `pan-c` to get ideas for the kinds of tests to write.
    - Reuse pre-existing tests wherever possible.
    - To test performance, run PaniniJ generated code and compare to existing benchmarks for `pan-c` generated code.

Weekly Collaboration Meeting

Date & Location: Sunday 1 of March; Google Hangouts
Members Present: All
Additional Participants:
Minutes:

- Revisit of project configuration using maven tools
- discussion of tasks
    - Dalton: signature generation
    - Kristin: testing harness for panini programs
    - Trey: duck generation
    - David: capsule generation
- Panini and Testing
    - template model as oracle for panini generated system
    - converting user unit tests into panini equivalents
    - connecting user unit tests to generated
- Duck data structures

    - prevent duplicate duck classes
- Generation Constraints
    - inherited method's ducks problem
    - standardization of artifact creation

Individual Hourly Contributions

Trey Erenberger	7.5 Hours
David Johnston	7.3 Hours
Kristin Clemens	6 Hours
Dalton Mills	6.5 Hours

Culumulative Time Contribution

Trey Erenberger	50 Hours
David Johnston	65 Hours
Kristin Clemens	39.5 Hours
Dalton Mills	48.5 Hours

Tentative Plans for Week 7

The focus of the coming week will move from designing duck futures to implementing the duck future generation code. We should also finish drafting an implementation of signature-generation code.