Team Member Assessments

Preliminary Round

Second and Third can Influence Grade

Self and Team Member Assessments

- Why?
  - To reflect yourself on your contributions to-date.
  - To hear how others perceive your contribution and to be applauded, learn and improve based on their (averaged) feedback

Confidentiality

Individual values are kept in strict confidence

How it works:

- Split 100 points across your team – you decide how
- Add comments for each teammate
- Gail/Yuriy average the numerical value, and put this in the gradebook – private to each student
- Comments will be used by the staff and summarized for students *only* at the extremes
- Nothing is passed directly from the survey to students

Things to consider

- **Preparation** – Were they prepared when they came to team meetings/work sessions?
- **Contribution** – Did they contribute productively to team discussions and assignments?
- **Respect for others' ideas** – Did they encourage others to contribute their ideas?
- **Flexibility** – Were they flexible when disagreements occurred?
- **Responsibility** - Were they responsible members of the team in terms of communication and commitments?

Consider also any ground rules or responsibilities you may have discussed and agreed on as a team

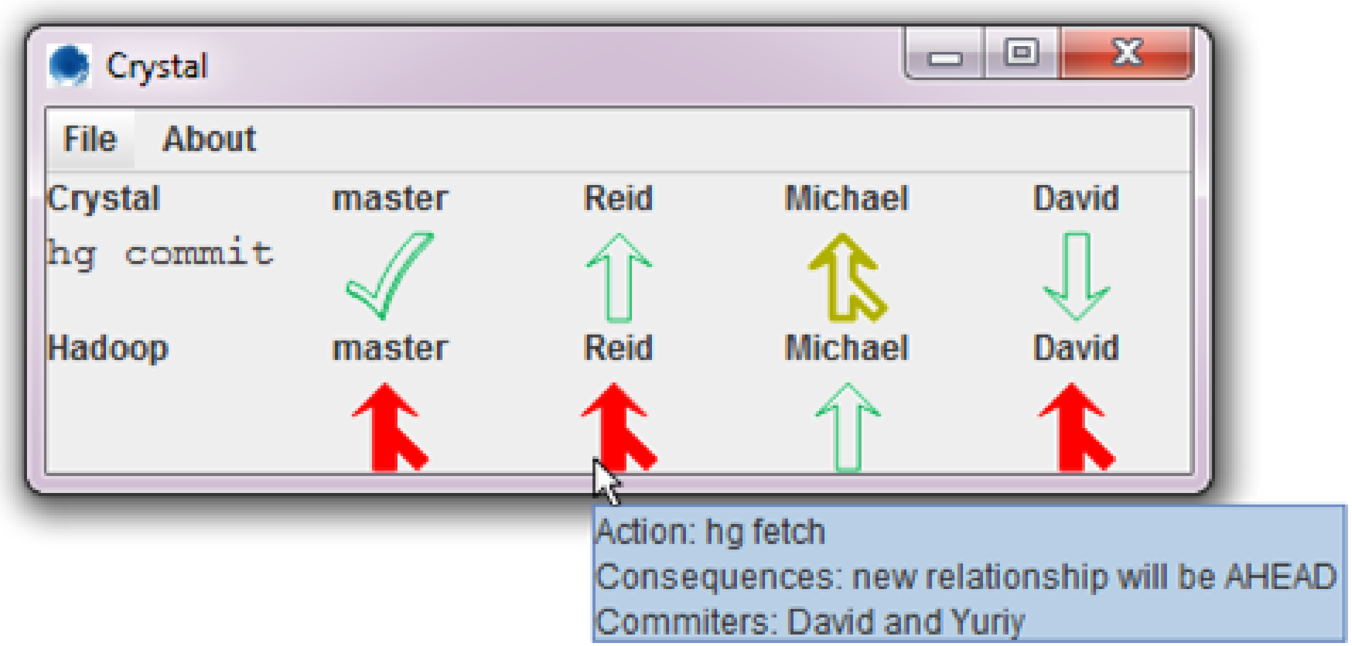
Survey is due Monday

- Survey links are on the class home page
- Complete by Monday at 11pm -- Required

Design Patterns

Creational, Structural, Behavioral

Crystal

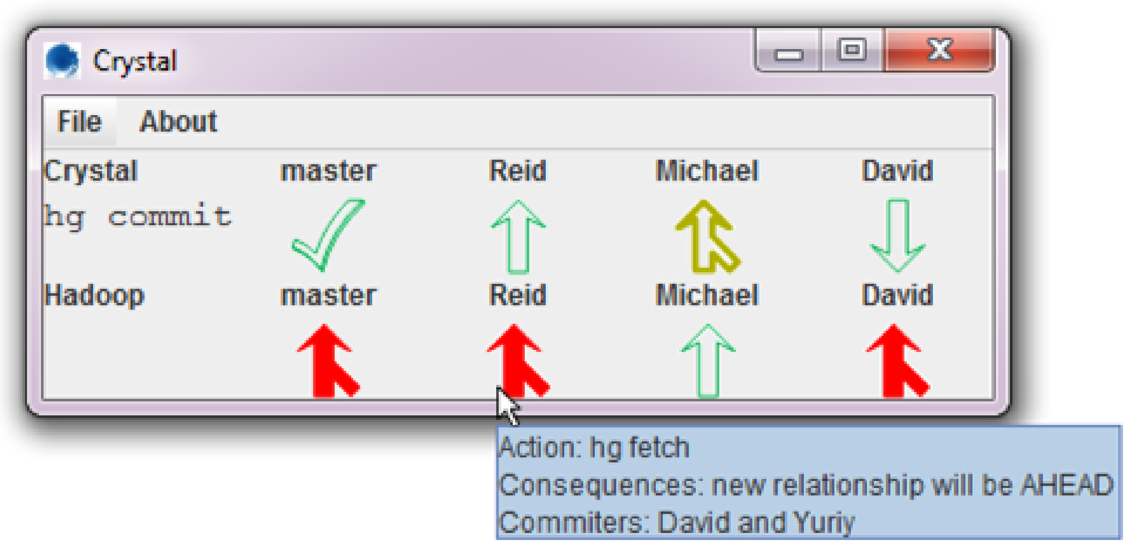


Crystal	master	Reid	Michael	David
hg commit	✓	⇧	⇧	⇩
Hadoop	master	Reid	Michael	David
	⬆	⬆	⇧	⬆

Action: hg fetch
Consequences: new relationship will be AHEAD
Commiters: David and Yuriy

Why use Crystal?

- Prevent conflicts
- Tells you when to communicate to avoid problems



Crystal	master	Reid	Michael	David
hg commit	✓	⇧	⇧	⇩
Hadoop	master	Reid	Michael	David
	⇧	⇧	⇧	⇧

Action: hg fetch
Consequences: new relationship will be AHEAD
Commiters: David and Yuriy

Tools we'll need

- Crystal

  http://www.cs.washington.edu/homes/brun/research/crystal

- Dropbox

  http://dropbox.com

- Mercurial

What’s hard about Crystal?

- The set up is awkward
- Beta release might contain bugs

- But, you have access to the developer

Steps to setting up Crystal

1. set up dropbox account
2. set up dropbox folder
3. set up hg repository clones
4. set up Crystal configuration

Step 1: set up dropbox account

1. Go to http://dropbox.com
2. Set up free account.

   if you want, you can invite each other to get more free space.

Step 2: set up dropbox folder

1. Create a project folder and share it with your group members.
2. In the project folder, create a folder for each group member and one for *master*.

Step 3: set up hg repository clones

1. Put the master in the *master* folder
2. Each group member:

   make a clone of the master in your folder

   (one person can do this for everyone)

You now can see everyone’s code versions

Step 4: set up Crystal configuration

Create a ~/.conflictClient.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<ccConfig tempDirectory="~/scratch/conflictClient/" hgPath="/usr/bin/hg" refresh="60">
    <project Kind="HG" ShortName="MyFirstProject" Clone="~/dropbox/myGroup/myName/" parent="master">
        <source ShortName="master" Clone="~/dropbox/myGroup/master/" commonParent="master" />
        <source ShortName="friend1" Clone="~/dropbox/myGroup/friend1/" commonParent="master" />
         <source ShortName="friend2" Clone="~/dropbox/myGroup/friend2/" commonParent="master" />
    </project>
</ccConfig>
```

Now just run Crystal

- Download the jar:

http://www.cs.washington.edu/homes/brun/research/crystal/crystal.jar

- Run

If you make changes to the ~/.conflictClient.xml file, restart Crystal

Design patterns outline

- Introduction to design patterns
- Creational patterns (constructing objects)
- Structural patterns (controlling heap layout)
- Behavioral patterns (affecting object semantics)

What is a design pattern?

- a standard solution to a common programming problem
  - a design or implementation structure that achieves a particular purpose
  - a high-level programming idiom
- a technique for making code more flexible
  - reduce coupling among program components
- shorthand for describing program design
  - a description of connections among program components
  - the shape of a heap snapshot or object model

Example 1: Encapsulation (data hiding)

- Problem: Exposed fields can be directly manipulated
  - Violations of the representation invariant
  - Dependences prevent changing the implementation
- Solution: Hide some components
  - Permit only stylized access to the object
- Disadvantages:
  - Interface may not (efficiently) provide all desired operations
  - Indirection may reduce performance

Example 2: Subclassing (inheritance)

- Problem: Repetition in implementations
  - Similar abstractions have similar members (fields, methods)
- Solution: Inherit default members from a superclass
  - Select an implementation via run-time dispatching
- Disadvantages:
  - Code for a class is spread out, and thus less understandable
  - Run-time dispatching introduces overhead

Example 3: Iteration

- Problem: To access all members of a collection, must perform a specialized traversal for each data structure
  - Introduces undesirable dependences
  - Does not generalize to other collections
- Solution:
  - The implementation performs traversals, does bookkeeping
  - Results are communicated to clients via a standard interface
- Disadvantages:
  - Iteration order is fixed by the implementation and not under the control of the client

Example 4: Exceptions

- Problem:
  - Errors in one part of the code should be handled elsewhere.
  - Code should not be cluttered with error-handling code.
  - Return values should not be preempted by error codes.
- Solution: Language structures for throwing and catching exceptions
- Disadvantages:
  - Code may still be cluttered.
  - It may be hard to know where an exception will be handled.
  - Use of exceptions for normal control flow may be confusing and inefficient.

Example 5: Generics

- Problem:
  - Well-designed data structures hold one type of object
- Solution:
  - Programming language checks for errors in contents
  - `List<Date>` instead of just `List`
- Disadvantages:
  - Slightly more verbose types

Creating generic classes

- Introduce a *type parameter* to a class

```
– public class Graph<N> implements Iterable<N> {
–   private final Map<N, Set<N>> node2neighbors;
–   public Graph(Set<N> nodes, Set<Tuple<N,N>> edges) {
–     ...
–   }
– }

– public interface Path<N, P extends Path<N,P>>
–   extends Iterable<N>, Comparable<Path<?, ?>> {
–   public Iterator<N> iterator();
– }
```

- Code can perform any operation permitted by the bound

Tips for designing generic classes

- First, write and test a concrete version
  - Consider creating a second concrete version
- Then, generalize it by adding type parameters
  - The compiler will help you to find errors

A puzzle about generics

- Integer is a subtype of Number

- List<Integer> is not a subtype of List<Number>
  - Compare specs: add(Integer) is not stronger than add(Number)
  - What goes wrong if List<Integer> is a subtype of List<Number>?
  - `List<Integer> li = new ArrayList<Integer>();`
  - `// legal if List<Integer> is subtype of List<Number>`
  - `List<Number> ln = li;`
  - `ln.add(new Float());`
  - `li.get(0); // we got a Float out of a List<Integer>!`

- Integer[] is a subtype of Number[]
  - Can we use similar code to break the Java type system?