

SECTION 2: HW3 SETUP AND TOOLS

cse331-staff@cs.washington.edu

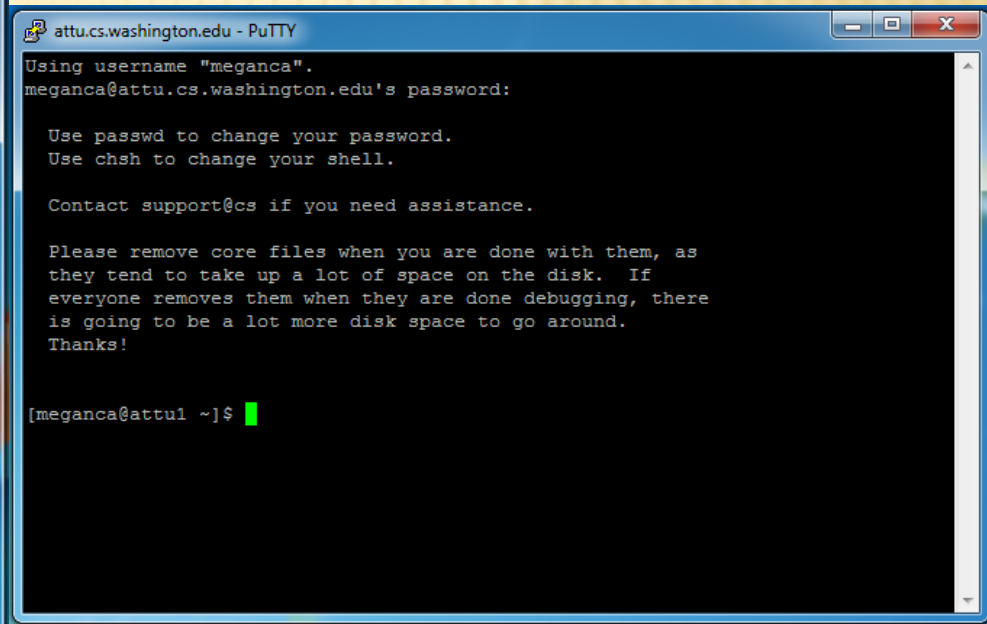
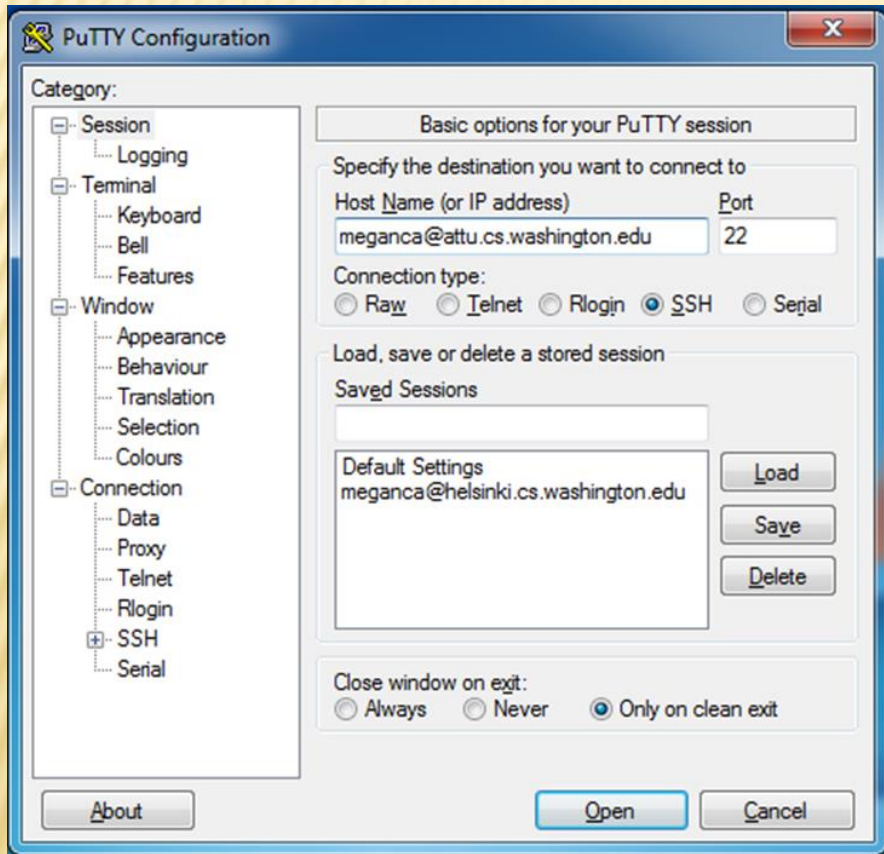
DEVELOPER TOOLS

- Eclipse and Java versions
- Remote access
- Version control
- Eclipse debugging

WHAT IS AN SSH CLIENT?

- Uses the secure shell protocol (SSH) to connect to a remote computer
 - + Enables you to work on a lab machine from home
 - + Similar to remote desktop
- Windows users: PuTTY and WinSCP
 - + PuTTY: ssh connection
 - + WinSCP: transfer or edit files
- Mac/Linux users: Terminal application
 - + Go to Applications/Utilities/Terminal
 - + Type in “ssh cseNetID@attu.cs.washington.edu”
 - + “ssh -XY cseNetID@attu.cs.washington.edu” lets you use GUIs

PuTTY

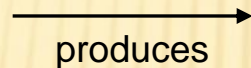


TERMINAL (LINUX, MAC)

```
meganca@charmader: ~  
meganca@charmader:~$ ssh meganca@attu.cs.washington.edu  
meganca@attu.cs.washington.edu's password:  
Last login: Wed Sep 24 17:13:13 2014 from c-24-19-57-209.hsd1.wa.comcast.net  
  
Use passwd to change your password.  
Use chsh to change your shell.  
  
Contact support@cs if you need assistance.  
  
Please remove core files when you are done with them, as  
they tend to take up a lot of space on the disk. If  
everyone removes them when they are done debugging, there  
is going to be a lot more disk space to go around.  
Thanks!  
  
[meganca@attu3 ~]$ █
```

ECLIPSE

- Get Java 7
- Important: Java separates compile and execution, eg:
 - + javac Example.java
Example.class
 - + Both compile and execute have to be the same Java!



DEMO #1

[http://courses.cs.washington.edu/courses/cse331/15wi/
tools/WorkingAtHome.html](http://courses.cs.washington.edu/courses/cse331/15wi/tools/WorkingAtHome.html)

WHAT IS UNIX?

- ✘ Multiuser modular operating system
 - + Traditionally command-line based
 - + Mac OS X is Unix-based!

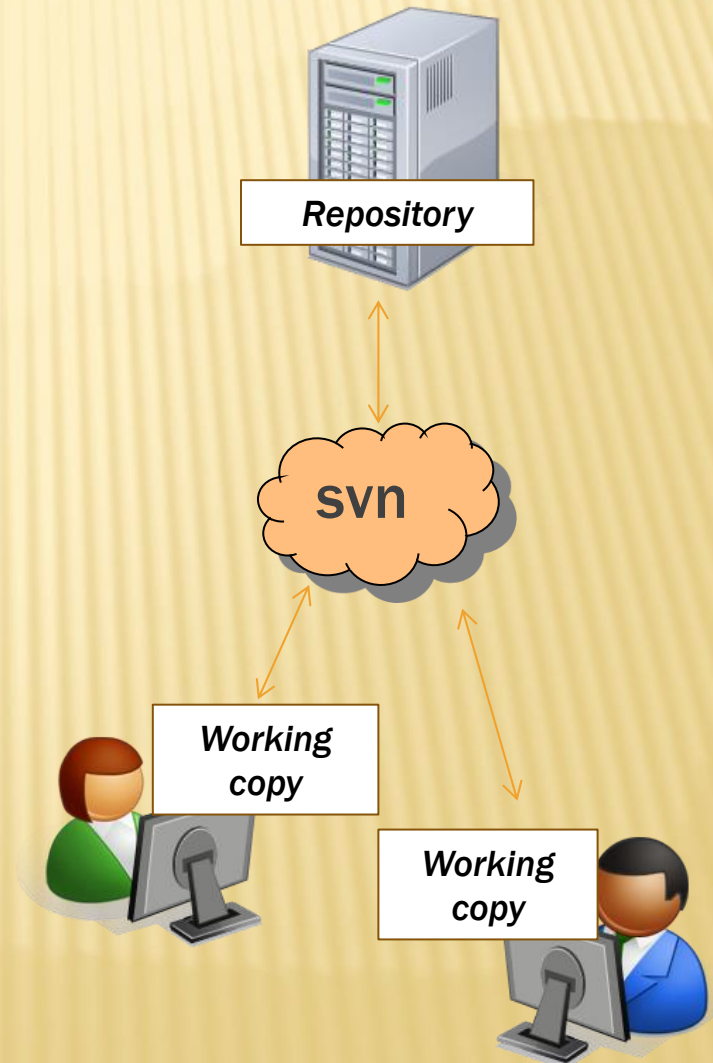
Command	What it does
pwd	p rints the name of the w orking d irectory
ls	lists the files in a directory (i.e., l ists s tuff)
cd	c hanges a d irectory
cp	c opies a file or directory
mv	m ove/rename a file or directory
rm	r emoves a file
mkdir	m ake a new d irectory
rmdir	r emove an empty d irectory
man	pulls up the m anual pages

WHAT IS VERSION CONTROL?

- Also known as source control/revision control
- System for tracking changes to code
 - + Software for developing software
- Essential for managing projects
 - + See a history of changes
 - + Revert back to an older version
 - + Merge changes from multiple sources
- We'll be talking about Subversion, but there are alternatives
 - ✓ Git, Mercurial, CVS
 - × Email, Dropbox, USB sticks

VERSION CONTROL ORGANIZATION

- ✘ A *repository* stores the master copy of the project
 - + Someone creates the repo for a new project
 - + Then nobody touches this copy directly
 - + Lives on a server everyone can access
- ✘ Each person *checks out* her own *working copy*
 - + Makes a local copy of the repo
 - + You'll always work off of this copy
 - + The version control system syncs the repo and working copy (with your help)



REPOSITORY

- Can create the repository anywhere
 - + Can be on the same computer that you're going to work on, which might be ok for a personal project where you just want rollback protection
- But, usually you want the repository to be robust:
 - + On a computer that's up and running 24/7
 - × Everyone always has access to the project
 - + On a computer that has a redundant file system
 - × No more worries about that hard disk crash wiping away your project!
- We'll use attu! (attu.cs.washington.edu)

VERSION CONTROL COMMON ACTIONS

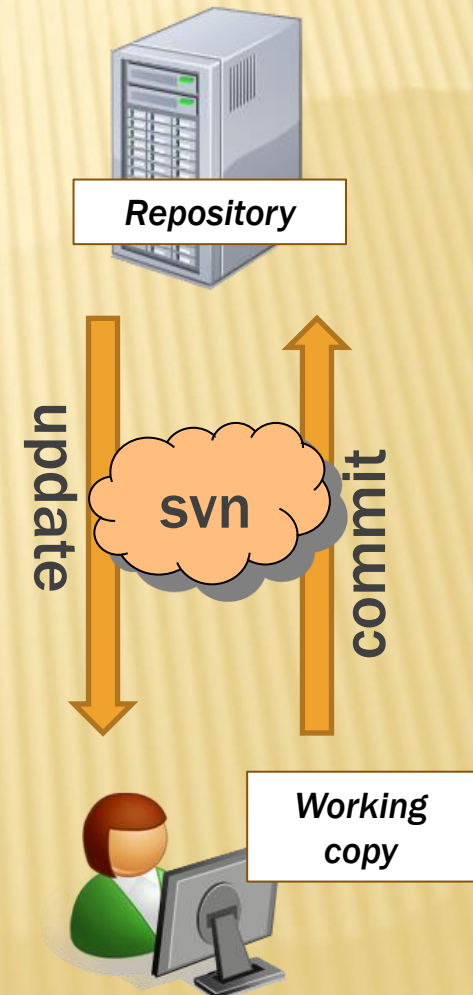
Most common commands:

- ✗ **Commit / checkin**

- + integrate changes *from* your working copy *into* the repository

- ✗ **Update**

- + integrate changes *into* your working copy *from* the repository



VERSION CONTROL COMMON ACTIONS (CONT.)

More common commands:

✘ Add, delete

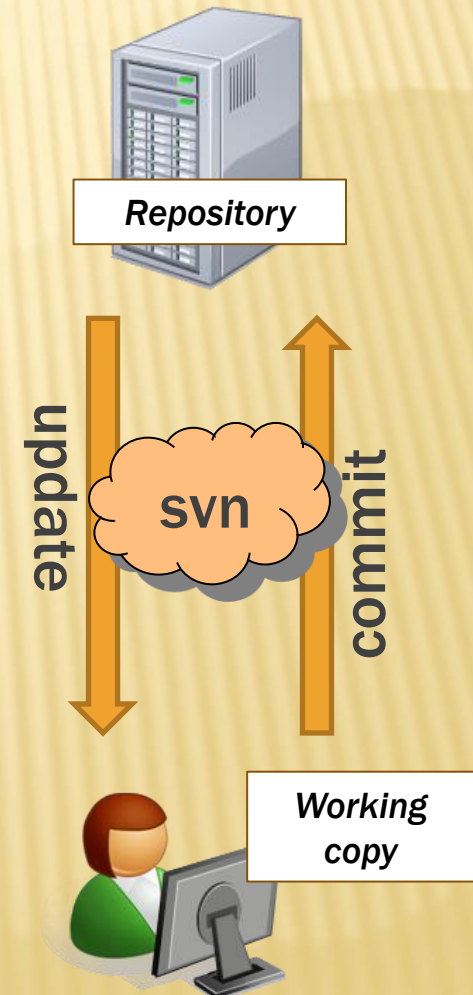
- + add or delete a file in the repository
- + just putting a new file in your working copy does not add it to the repo!

✘ Revert

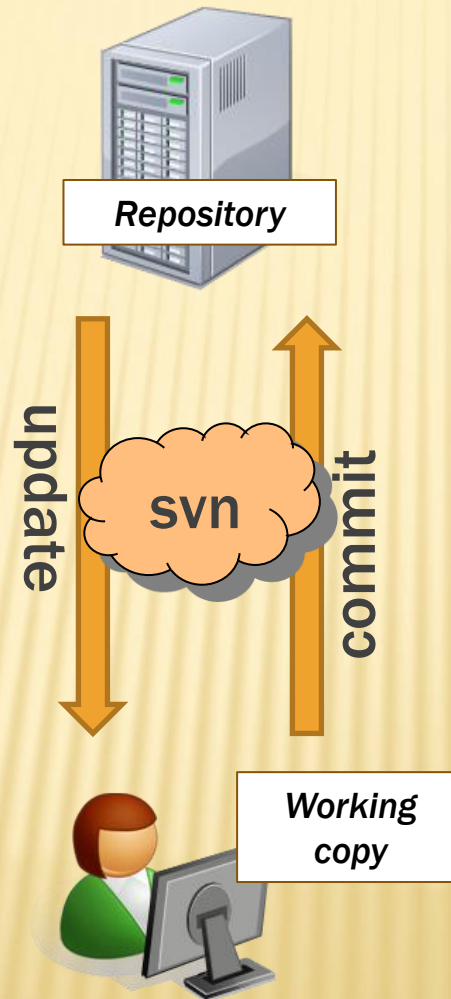
- + wipe out your local changes to a file

✘ Resolve, diff, merge

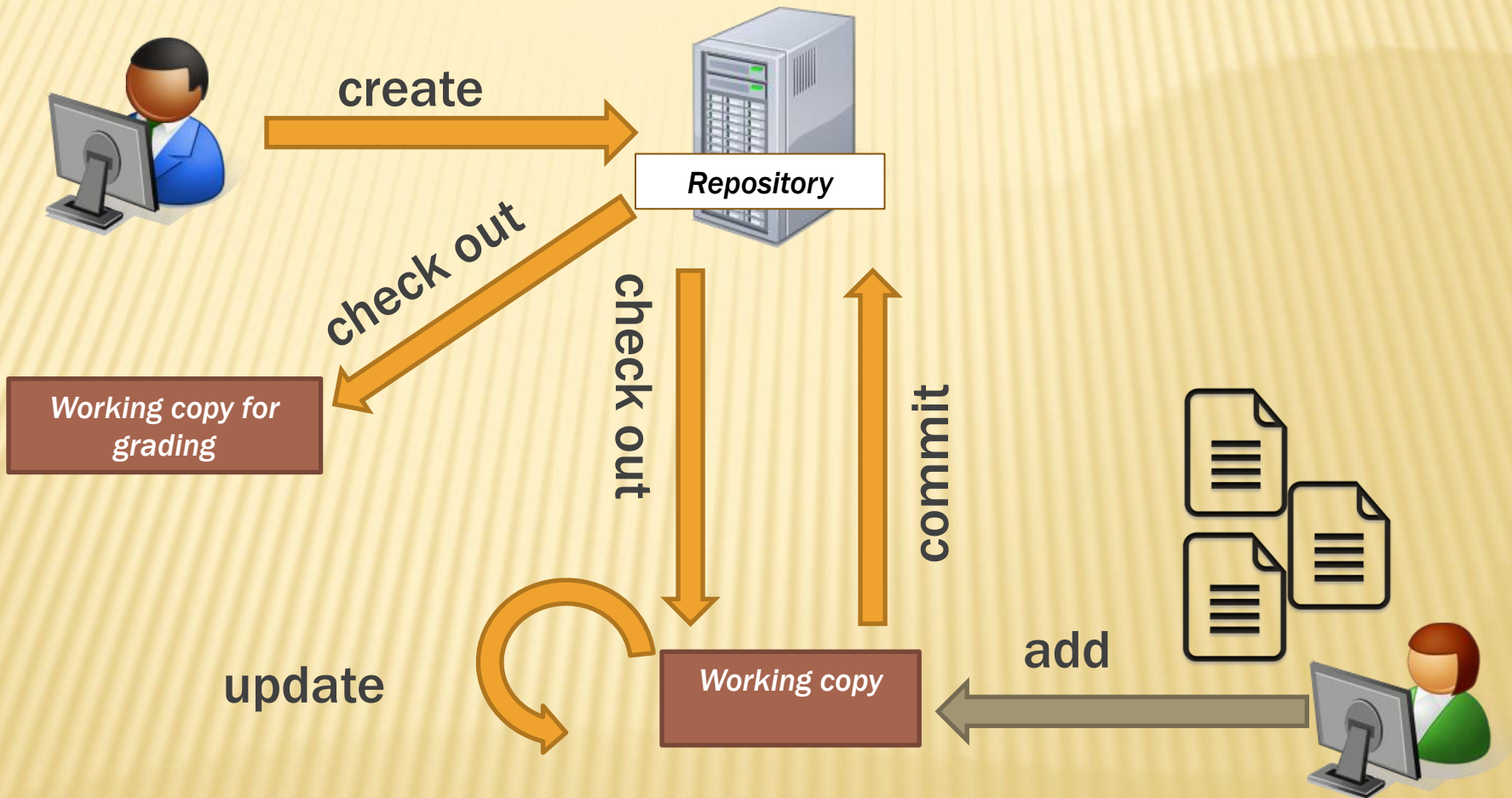
- + handle a conflict – two users editing the same code



VERSION CONTROL



331 VERSION CONTROL



331 VERSION CONTROL

- Your repo is at
`/projects/instr/15wi/cse331/YourCSENetID/R
EPOS/cse331`
- Only check out once (unless you're working
in a lot of places)
- Don't forget to add files!!
- Check in your work!

VERSION CONTROL: COMMAND-LINE

command	description
svn co <i>repo</i>	check out
svn ci [<i>files</i>]	commit / check in changed files
svn add <i>files</i>	schedule files to be added at next commit
svn help [<i>command</i>]	get help info about a particular command
svn merge <i>source1 source2</i>	merge changes
svn revert <i>files</i>	restore local copy to repo's version
svn resolve <i>files</i>	resolve merging conflicts
svn update [<i>files</i>]	update local copy to latest version
others: blame, changelist, cleanup, diff, export, ls/mv/rm/mkdir, lock/unlock, log, propset	

THIS QUARTER

- We distribute starter code by adding it to your **repo**
- You will **code** in Eclipse
- You turn in your files by **adding** them to the repo and **committing** your changes
- You will **validate** your homework by **SSHing** onto attu and running an Ant build file

HOW TO USE SUBVERSION

1. Eclipse plugin: Subclipse
2. GUI interface: TortoiseSVN, NautilusSVN
3. Command line: PuTTY

DEMO #2

<https://courses.cs.washington.edu/courses/cse331/15wi/tools/versioncontrol.html>

WHAT IS ECLIPSE?

- Integrated development environment (IDE)
- Allows for software development from start to finish
 - + Type code with syntax highlighting, warnings, etc.
 - + Run code straight through or with breakpoints (debug)
 - + Break code
- Mainly used for Java
 - + Supports C, C++, JavaScript, PHP, Python, Ruby, etc.
- Alternatives
 - + NetBeans, Visual Studio, IntelliJIDEA

ECLIPSE SHORTCUTS

Shortcut	Purpose
Ctrl + D	Delete an entire line
Alt + Shift + R	Refactor (rename)
Ctrl + Shift + O	Clean up imports
Ctrl + /	Toggle comment
Ctrl + Shift + F	Make my code look nice 😊

ECLIPSE DEBUGGING

- `System.out.println()` works for debugging...
 - + It's quick
 - + It's dirty
 - + Everyone knows how to do it
- ...but there are drawbacks
 - + What if I'm printing something that's null?
 - + What if I want to look at something that can't easily be printed (e.g., what does my binary search tree look like now)?
- Eclipse's debugger is powerful...if you know how to use it

ECLIPSE DEBUGGING

The screenshot displays the Eclipse IDE interface during a debugging session. The top toolbar includes standard development icons. Below it, the 'Quick Access' search bar is visible. The main workspace is divided into several panels:

- Debug Console:** Shows the execution stack with the following entries:
 - DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: not available
 - Method.invoke(Object, Object...) line: not available
 - FrameworkMethod\$1.runReflectiveCall() line: 45
 - FrameworkMethod\$1(ReflectiveCallable).run() line: 15
 - FrameworkMethod.invokeExplosively(Object, Object...) line: not available
 - InvokeMethod.evaluate() line: 20
 - BlockJUnit4ClassRunner(ParentRunner<T>).runLeaf(Statement) line: not available
 - BlockJUnit4ClassRunner.runChild(FrameworkMethod, Runnable) line: not available
 - BlockJUnit4ClassRunner.runChild(Object, RunNotifier) line: not available
 - ParentRunner\$3.run() line: 231
 - ParentRunner\$1.schedule(Runnable) line: 60
 - BlockJUnit4ClassRunner(ParentRunner<T>).runChildren(RunNotifier) line: not available
 - ParentRunner<T>.access\$000(ParentRunner, RunNotifier) line: not available
- Variables View:** A table showing the current state of variables:

Name	Value
this	RatPolyStackTest (id=33)
- Source Editor:** Displays the code for `RatPolyStackTest.java`. The current line is 157, which is highlighted in green:

```
151 ////////////////////////////////////////////////////  
152 /// Duplicate  
153 ////////////////////////////////////////////////////  
154  
155 @Test  
156 public void testDupWithOneVal() {  
157 RatPolyStack stk1 = stack("3");  
158 stk1.dup();  
159 assertStackIs(stk1, "33");  
160 stk1 = stack("123");  
161 stk1.dup();  
162 assertStackIs(stk1, "1123");
```
- Outline View:** Lists the methods in the class, with `testDupWithOneVal()` selected:
 - testClear(): void
 - testCtor(): void
 - testDifferentiate(): void
 - testDivMultiElems(): void
 - testDivTwoElems(): void
 - testDupWithMultVal(): void
 - testDupWithOneVal(): void**
 - testDupWithTwoVal(): void
 - testIntegrate(): void

ECLIPSE DEBUGGING

The screenshot displays the Eclipse IDE interface during a debug session. The top toolbar includes standard development icons. Below it, the 'Quick Access' search bar is visible. The main workspace is divided into several panels:

- Debug Console:** Shows a stack trace of the current execution, with the following methods listed:
 - DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: ...
 - Method.invoke(Object, Object...) line: not available
 - FrameworkMethod\$1.runReflectiveCall() line: 45
 - FrameworkMethod\$1(ReflectiveCallable).run() line: 15
 - FrameworkMethod.invokeExplosively(Object, Object...) line: ...
 - InvokeMethod.evaluate() line: 20
 - BlockJUnit4ClassRunner(ParentRunner<T>).runLeaf(Statem...
 - BlockJUnit4ClassRunner.runChild(FrameworkMethod, RunN...
 - BlockJUnit4ClassRunner.runChild(Object, RunNotifier) line: ...
 - ParentRunner\$3.run() line: 231
 - ParentRunner\$1.schedule(Runnable) line: 60
 - BlockJUnit4ClassRunner(ParentRunner<T>).runChildren(Ru...
 - ParentRunner<T>.access\$000(ParentRunner, RunNotifier) li...
- Variables View:** A table showing the current state of variables:

Name	Value
this	RatPolyStackTest (id=33)
- Code Editor:** Shows the source code for `RatPolyStackTest.java`. A green vertical bar on the left side of the editor indicates a breakpoint is set on line 57. The code includes comments and assertions.
- Outline View:** Located on the right side, showing the project's structure.

A text box overlaid on the code editor provides instructions: "Double click in the grey area to the left of your code to set a breakpoint. A breakpoint is a line that the Java VM will stop at during normal execution of your program, and wait for action from you."

ECLIPSE DEBUGGING

The screenshot shows the Eclipse IDE interface. The top toolbar has the Bug icon highlighted with a green box. A text box points to it with the instruction: "Click the Bug icon to run in Debug mode. Otherwise your program won't stop at your breakpoints." The Debug console on the left shows a stack trace of method calls. The Variable Explorer on the right shows a table with one entry: "Value" and "RatPolyStackTest (id=33)". The Java editor at the bottom shows the source code of "RatPolyStackTest.java" with a breakpoint set at line 157.

Click the Bug icon to run in Debug mode. Otherwise your program won't stop at your breakpoints.

Value
RatPolyStackTest (id=33)

```
151 ////////////////////////////////////////////////////
152 /// Duplicate
153 ////////////////////////////////////////////////////
154
155 @Test
156 public void testDupWithOneVal() {
157 RatPolyStack stk1 = stack("3");
158 stk1.dup();
159 assertStackIs(stk1, "33");
160 stk1 = stack("123");
161 stk1.dup();
162 assertStackIs(stk1, "1123");
}
```

- testClear() : void
- testCtor() : void
- testDifferentiate() : void
- testDivMultiElems() :
- testDivTwoElems() :
- testDupWithMultVal
- testDupWithOneVal(
- testDupWithTwoVal(
- testIntegrate() : void

ECLIPSE DEBUGGING

The screenshot displays the Eclipse IDE interface during a debugging session. At the top, the toolbar contains several icons for program control: a green play button (Run), a red stop button (Stop), a blue refresh button (Refresh), and a yellow arrow button (Step Over). These four icons are enclosed in a green rectangular box. Below the toolbar is the 'Quick Access' search bar. The main workspace is divided into several panes:

- Debug Console:** Shows a list of stack frames for the current thread. The frames include:
 - DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: not available
 - Method.invoke(Object, Object...) line: not available
 - FrameworkMethod\$1.runReflectiveCall() line: 45
 - FrameworkMethod\$1(ReflectiveCallable).run() line: 15
 - FrameworkMethod.invokeExplosively(Object, Object...) line: not available
 - InvokeMethod.evaluate() line: 20
 - BlockJUnit4ClassRunner(ParentRunner<T>).runLeaf(Statement) line: not available
 - BlockJUnit4ClassRunner.runChild(FrameworkMethod, Runnable) line: not available
 - BlockJUnit4ClassRunner.runChild(Object, RunNotifier) line: not available
 - ParentRunner\$3.run() line: 231
 - ParentRunner\$1.schedule(Runnable) line: 60
 - BlockJUnit4ClassRunner(ParentRunner<T>).runChildren(RunNotifier) line: not available
 - ParentRunner<T>.access\$000(ParentRunner, RunNotifier) line: not available
- Source Editor:** Displays the source code for 'RatPolyStackTest.java'. Line 157 is highlighted in green and has a mouse cursor over it. The code is:

```
151 ///////////////////////////////////////////////////////////////////
152 /// Duplicate
153 ///////////////////////////////////////////////////////////////////
154
155 @Test
156 public void testDupWithOneVal() {
157     RatPolyStack stk1 = stack("3");
158     stk1.dup();
159     assertStackIs(stk1, "33");
160     stk1 = stack("123");
161     stk1.dup();
162     assertStackIs(stk1, "1123");
```
- Outline View:** Shows a list of methods in the current class, including 'testClear(): void', 'testCtor(): void', 'testDifferentiate(): void', 'testDivMultiElems(): void', 'testDivTwoElems(): void', 'testDupWithMultVal(): void', 'testDupWithOneVal(): void', 'testDupWithTwoVal(): void', and 'testIntegrate(): void'. The 'testDupWithOneVal()' method is selected and highlighted in green.

A text box on the right side of the image contains the following text: "Controlling your program while debugging is done with these buttons".

ECLIPSE DEBUGGING

The screenshot displays the Eclipse IDE interface during a debug session. The top toolbar shows the Run, Pause, and Stop buttons, with the Run button highlighted in a green box. A text box overlaid on the Variables view contains the text: "Play, pause, stop work just like you'd expect".

The Debug Console (top left) shows a stack trace of the current execution:

```
DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: ...  
Method.invoke(Object, Object...) line: not available  
FrameworkMethod$1.runReflectiveCall() line: 45  
FrameworkMethod$1(ReflectiveCallable).run() line: 15  
FrameworkMethod.invokeExplosively(Object, Object...) line: ...  
InvokeMethod.evaluate() line: 20  
BlockJUnit4ClassRunner(ParentRunner<T>).runLeaf(Statem...  
BlockJUnit4ClassRunner.runChild(FrameworkMethod, RunN...  
BlockJUnit4ClassRunner.runChild(Object, RunNotifier) line: ...  
ParentRunner$3.run() line: 231  
ParentRunner$1.schedule(Runnable) line: 60  
BlockJUnit4ClassRunner(ParentRunner<T>).runChildren(Ru...  
ParentRunner<T>.access$000(ParentRunner, RunNotifier) li
```

The Variables view (top right) shows the current object: `RatPolyStackTest (id=33)`. The variable `this` is listed.

The Source Editor (bottom left) shows the Java code for `RatPolyStackTest.java`. The current line of execution is highlighted in green:

```
151 ///////////////////////////////////////////////////  
152 /// Duplicate  
153 ///////////////////////////////////////////////////  
154  
155 @Test  
156 public void testDupWithOneVal() {  
157     RatPolyStack stk1 = stack("3");  
158     stk1.dup();  
159     assertStackIs(stk1, "33");  
160     stk1 = stack("123");  
161     stk1.dup();  
162     assertStackIs(stk1, "1123");
```

The Outline view (bottom right) shows the class structure with the following methods listed:

- testClear(): void
- testCtor(): void
- testDifferentiate(): v
- testDivMultiElems():
- testDivTwoElems(): :
- testDupWithMultVal
- testDupWithOneVal()
- testDupWithTwoVal()
- testIntegrate(): void

ECLIPSE DEBUGGING

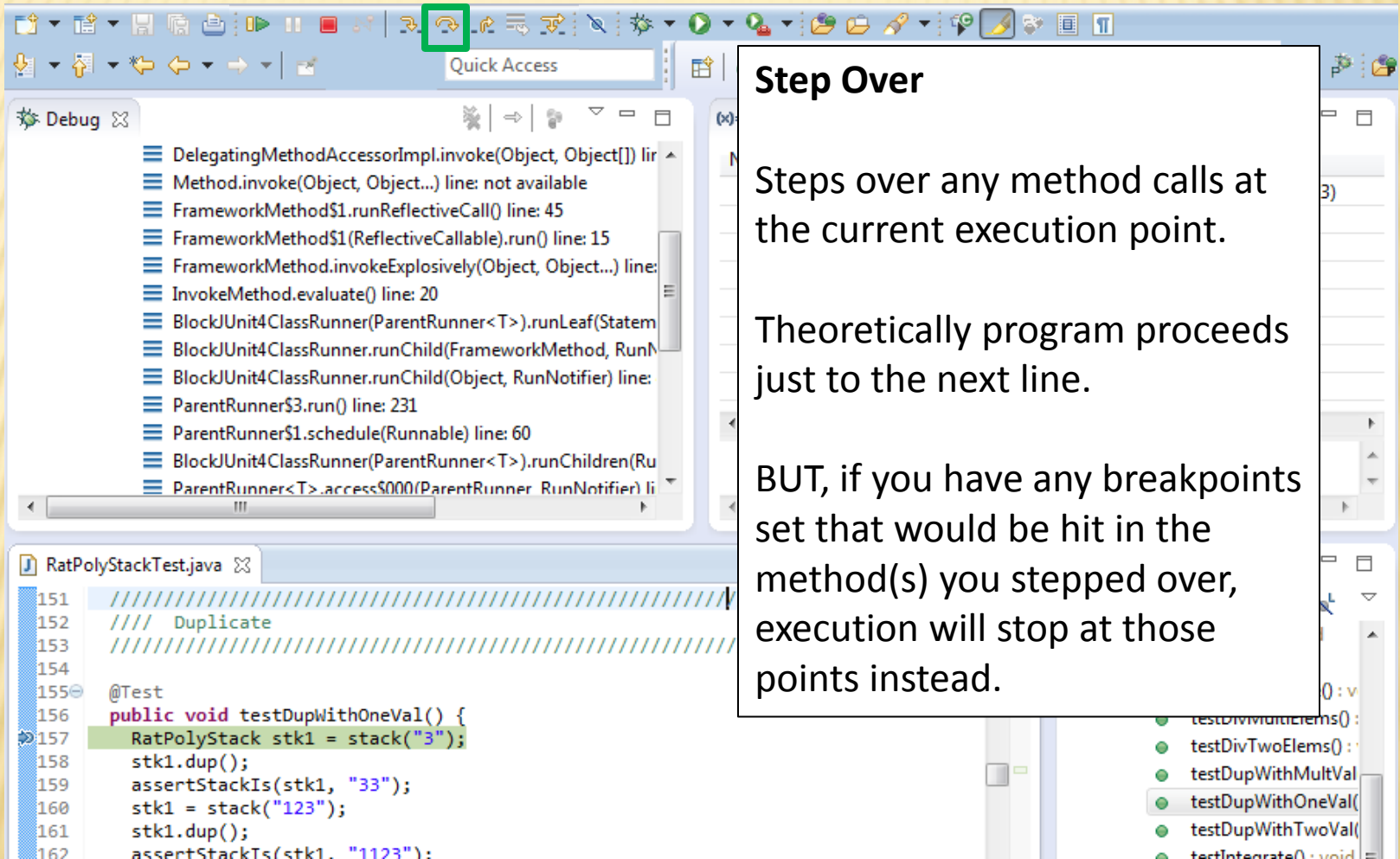
The screenshot shows the Eclipse IDE interface. The top toolbar has a green box around the 'Step Into' icon. The Debug console on the left lists several methods in the call stack. The code editor at the bottom shows a Java file named 'RatPolyStackTest.java' with a breakpoint at line 157. The code at line 157 is highlighted in green. On the right, a list of test methods is visible, with 'testDupWithOneVal()' selected.

Step Into

Steps into the method at the current execution point – if possible. If not possible then just proceeds to the next execution point.

If there's multiple methods at the current execution point step into the first one to be executed.

ECLIPSE DEBUGGING



Step Over

Steps over any method calls at the current execution point.

Theoretically program proceeds just to the next line.

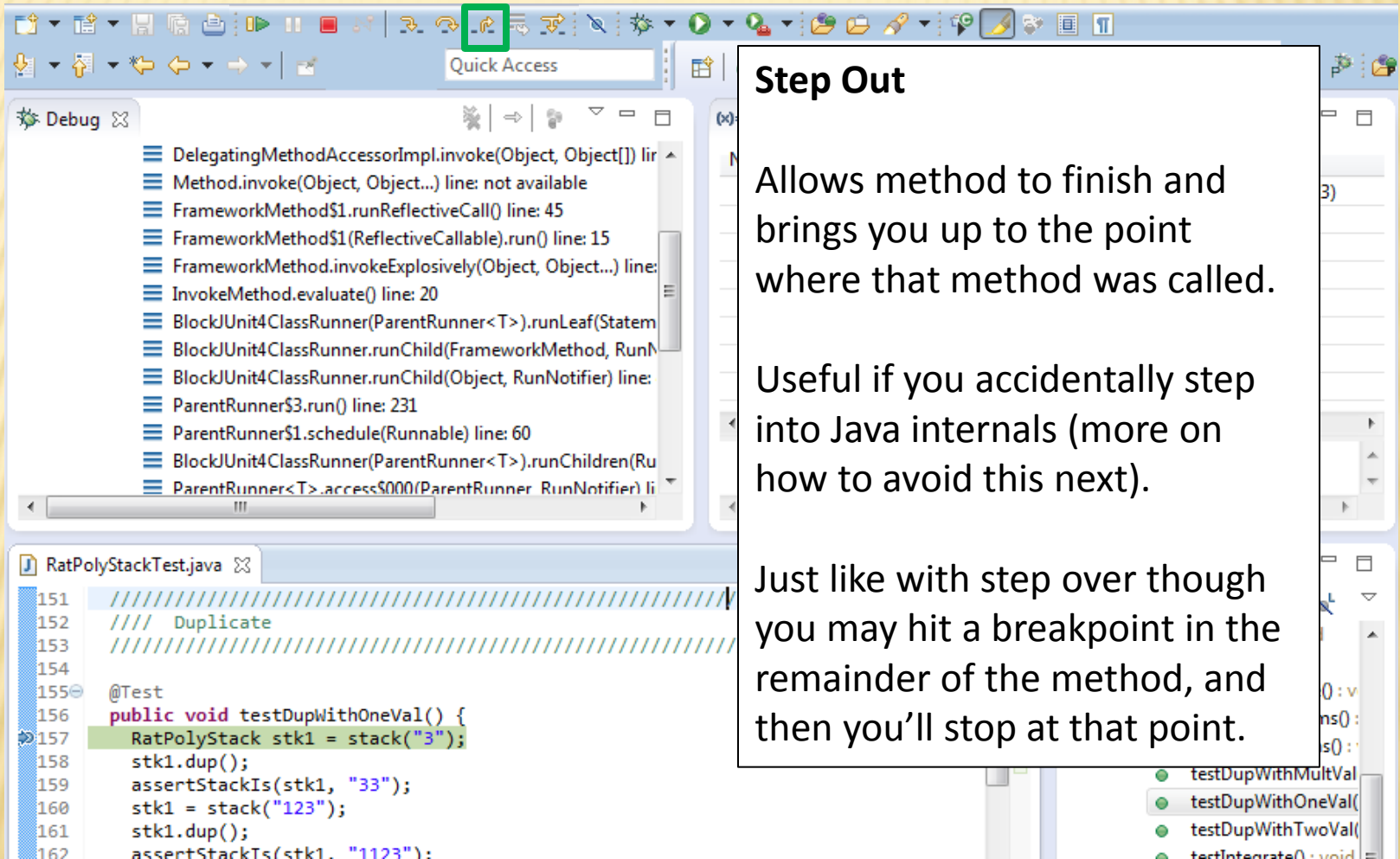
BUT, if you have any breakpoints set that would be hit in the method(s) you stepped over, execution will stop at those points instead.

```
Debug
- DelegatingMethodAccessorImpl.invoke(Object, Object[]) lir
- Method.invoke(Object, Object...) line: not available
- FrameworkMethod$1.runReflectiveCall() line: 45
- FrameworkMethod$1(ReflectiveCallable).run() line: 15
- FrameworkMethod.invokeExplosively(Object, Object...) line:
- InvokeMethod.evaluate() line: 20
- BlockJUnit4ClassRunner(ParentRunner<T>).runLeaf(Statem
- BlockJUnit4ClassRunner.runChild(FrameworkMethod, RunN
- BlockJUnit4ClassRunner.runChild(Object, RunNotifier) line:
- ParentRunner$3.run() line: 231
- ParentRunner$1.schedule(Runnable) line: 60
- BlockJUnit4ClassRunner(ParentRunner<T>).runChildren(Ru
- ParentRunner<T>.access$000(ParentRunner, RunNotifier) li

RatPolyStackTest.java
151 ////////////////////////////////////////////////////
152 /// Duplicate
153 ////////////////////////////////////////////////////
154
155 @Test
156 public void testDupWithOneVal() {
157 RatPolyStack stk1 = stack("3");
158 stk1.dup();
159 assertStackIs(stk1, "33");
160 stk1 = stack("123");
161 stk1.dup();
162 assertStackIs(stk1, "1123");

testDivWithMultElems():
testDivTwoElems():
testDupWithMultVal
testDupWithOneVal(
testDupWithTwoVal(
testIntegrate(): void
```

ECLIPSE DEBUGGING



Step Out

Allows method to finish and brings you up to the point where that method was called.

Useful if you accidentally step into Java internals (more on how to avoid this next).

Just like with step over though you may hit a breakpoint in the remainder of the method, and then you'll stop at that point.

```
151 ////////////////////////////////////////////////////
152 /// Duplicate
153 ////////////////////////////////////////////////////
154
155 @Test
156 public void testDupWithOneVal() {
157     RatPolyStack stk1 = stack("3");
158     stk1.dup();
159     assertStackIs(stk1, "33");
160     stk1 = stack("123");
161     stk1.dup();
162     assertStackIs(stk1, "1123");
```

ECLIPSE DEBUGGING

Enable/disable step filters

There's a lot of code you don't want to enter when debugging, internals of Java, internals of JUnit, etc.

You can skip these by configuring step filters.

Checked items are skipped.

The screenshot shows the Eclipse IDE interface. The top toolbar has a green box around the 'Step Filtering' icon. The 'Preferences' dialog is open, with the 'Step Filtering' section selected in the left-hand tree. The 'Step Filtering' section is expanded, and the 'Use Step Filters' checkbox is checked. Below this, a list of 'Defined step filters' is shown, with 'java.lang.ClassLoader' checked. Other filters include 'com.ibm.*', 'com.sun.*', 'java.*', 'javax.*', 'jrockit.*', 'junit.*', 'org.omg.*', 'sun.*', and 'sunw.*'. Buttons for 'Add Filter...', 'Add Class...', 'Add Packages...', 'Remove', 'Select All', and 'Deselect All' are visible. At the bottom of the dialog are 'Restore Defaults', 'Apply', 'OK', and 'Cancel' buttons. In the background, a code editor shows a list of test methods, with 'testClear() : void' and 'testCtor() : void' highlighted.

ECLIPSE DEBUGGING

The screenshot shows the Eclipse IDE interface. The top toolbar contains various icons for file operations, running, and debugging. Below the toolbar is the 'Quick Access' search bar. The main workspace is divided into several panes:

- Debug Console:** A window titled 'Debug' showing a stack trace. The stack trace lists several methods, including `DelegatingMethodAccessorImpl.invoke`, `Method.invoke`, `FrameworkMethod$1.runReflectiveCall`, `FrameworkMethod$1(ReflectiveCallable).run`, `FrameworkMethod.invokeExplosively`, `InvokeMethod.evaluate`, `BlockJUnit4ClassRunner(ParentRunner<T>).runLeaf`, `BlockJUnit4ClassRunner.runChild(FrameworkMethod, Runnable)`, `BlockJUnit4ClassRunner.runChild(Object, RunNotifier)`, `ParentRunner$3.run`, `ParentRunner$1.schedule(Runnable)`, `BlockJUnit4ClassRunner(ParentRunner<T>).runChildren(RunNotifier)`, and `ParentRunner<T>.access$000(ParentRunner, RunNotifier)`. The stack trace is highlighted with a green border.
- Code Editor:** A window titled 'RatPolyStackTest.java' showing the source code. A breakpoint is set at line 157, which is highlighted in green. The code includes a `@Test` annotation and a `public void testDupWithOneVal()` method. The code is as follows:

```
151 ////////////////////////////////////////////////////
152 /// Duplicate
153 ////////////////////////////////////////////////////
154
155 @Test
156 public void testDupWithOneVal() {
157     RatPolyStack stk1 = stack("3");
158     stk1.dup();
159     assertStackIs(stk1, "33");
160     stk1 = stack("123");
161     stk1.dup();
162     assertStackIs(stk1, "1123");
```
- Stack Trace Panel:** A panel titled 'Stack Trace' with a 'Name' column. It lists several methods, including `testDifferentiate()`, `testDivMultiElems()`, `testDivTwoElems()`, `testDupWithMultVal`, `testDupWithOneVal()`, `testDupWithTwoVal()`, and `testIntegrate(): void`. The `testDupWithOneVal()` method is selected.

Stack Trace

Shows what methods have been called to get you to current point where program is stopped.

You can click on different method names to navigate to that spot in the code without losing your current spot.

ECLIPSE DEBUGGING

Variables Window

Shows all variables, including method parameters, local variables, and class variables, that are in scope at the current execution spot. Updates when you change positions in the stackframe. You can expand objects to see child member values. There's a simple value printed, but clicking on an item will fill the box below the list with a pretty format.

```
159   assertStackIs(stk1, "33");
160   stk1 = stack("123");
161   stk1.dup();
162   assertStackIs(stk1, "1123");
```

Name	Value
this	RatPolyStackTest (id=33)

Some values are in the form of ObjectName (id=x), this can be used to tell if two variables are referring to the same object.

ECLIPSE DEBUGGING

Variables that have changed since the last break point are highlighted in yellow.

You can change variables right from this window by double clicking the row entry in the Value tab.

The screenshot shows the Eclipse IDE interface during a debug session. The top toolbar includes icons for Run, Break, and other debugging actions. The main window is divided into several panes:

- Variables View:** A table showing the current state of variables. The 'Value' tab is active, and the variable 'expt' is highlighted in yellow, indicating it has changed since the last breakpoint. The table is as follows:

Name	Value
this	RatTermTest (
t	RatTerm (id=4
coeff	RatNum (id=4
expt	5
- Code Editor:** Shows the source code for `RatPolyStackTest.java`. Line 157 is highlighted, corresponding to the breakpoint:

```
157 RatPolyStack stk1 = stack("3");
```
- Outline View:** Lists the methods in the class, with `testDupWithOneVal()` selected.

ECLIPSE DEBUGGING

Variables that have changed since the last break point are highlighted in yellow.

You can change variables right from this window by double clicking the row entry in the Value tab.

The screenshot displays the Eclipse IDE interface during a debug session. The top toolbar includes icons for file operations, running, and debugging. The main window is divided into several panes:

- Variables Window (top right):** A table showing the current state of variables. The 'Value' tab is active, and the variable 'expt' is highlighted in yellow, indicating it has changed since the last breakpoint. The table is as follows:

Name	Value
this	RatTermTest (
t	RatTerm (id=4
coeff	RatNum (id=4
expt	5
- Code Editor (bottom left):** Shows the source code for 'RatPolyStackTest.java'. Line 157 is highlighted, corresponding to the breakpoint:

```
157 RatPolyStack stk1 = stack("3");
```
- Outline (bottom right):** Lists the methods in the class, with 'testDupWithOneVal()' selected.

ECLIPSE DEBUGGING

There's a powerful right-click menu.

- See all references to a given variable
- See all instances of the variable's class
- Add watch statements for that variable's value (more later)

The screenshot shows the Eclipse IDE interface during a debug session. The top toolbar includes icons for file operations, execution, and search. The main editor displays Java code for a class named `Runner.class`. The code includes a `@Test` annotation and a `testDupWithOneVal()` method. The current line of execution is highlighted in green, showing the assignment `RatPolyStack stk1 = stack("3");`. The Variables view on the right shows the current stack frame with variables `this` (value: `RatTermTest (id=33)`) and `t`. Under `t`, the variables `coeff` and `expt` are listed. A right-click context menu is open over the `expt` variable, listing various actions such as `Select All`, `Copy Variables`, `Find...`, `Change Value...`, `All References...`, `All Instances...` (highlighted), `Instance Count...`, `New Detail Formatter...`, `Open Declared Type`, `Open Declared Type Hierarchy`, `Instance Breakpoints...`, `Watch`, and `Inspect`. The `Inspect` option is highlighted with a green border.

```
151 ///////////////////////////////////////////////////////////////////
152 /// Duplicate
153 ///////////////////////////////////////////////////////////////////Runner.class
154
155 @Test
156 public void testDupWithOneVal() {
157     RatPolyStack stk1 = stack("3");
158     stk1.dup();
159     assertStackIs(stk1, "33");
160     stk1 = stack("123");
161     stk1.dup();
162     assertStackIs(stk1, "1123");
```

Name	Value
this	RatTermTest (id=33)
t	
coeff	
expt	

- Select All (Ctrl+A)
- Copy Variables (Ctrl+C)
- Find... (Ctrl+F)
- Change Value...
- All References...
- All Instances... (Ctrl+Shift+N)
- Instance Count...
- New Detail Formatter...
- Open Declared Type
- Open Declared Type Hierarchy
- Instance Breakpoints...
- Watch
- Inspect (Ctrl+Shift+I)

ECLIPSE DEBUGGING

Show Logical Structure

Expands out list items so it's as if each list item were a field (and continues down for any children list items)

The screenshot shows the Eclipse IDE interface during a debug session. The top toolbar includes icons for Run, Stop, Step Over, Step Into, Step Return, and Break. The main window is divided into several panes:

- Debug Console:** Shows the execution stack with the following entries:
 - BlockJUnit4ClassRunner.runChild(Object, RunNotifier) line:
 - ParentRunner\$3.run() line: 231
 - ParentRunner\$1.schedule(Runnable) line: 60
 - BlockJUnit4ClassRunner(ParentRunner<T>).runChildren(Ru
 - ParentRunner<T>.access\$000(ParentRunner RunNotifier) li
- Variables View:** Displays the current state of variables. The variable `stk1` is expanded to show its logical structure:

Name	Value
this	RatPolyStackTest (id=33)
stk1	RatPolyStack (id=44)
polys	Stack<E> (id=49)
[0]	RatPoly (id=719)
terms	ArrayList<E> (id=728)
[0]	RatTerm (id=731)
coeff	RatNum (id=733)
expt	0
- Source Editor:** Shows the source code for `RatPolyStackTest.java`. The current line is 157, which is highlighted in green:

```
151 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
152 /// Duplicate
153 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
154
155 @Test
156 public void testDupWithOneVal() {
157   RatPolyStack stk1 = stack("3");
158   stk1.dup();
159   assertStackIs(stk1, "33");
160   stk1 = stack("123");
161   stk1.dup();
162   assertStackIs(stk1, "1123");
```
- Outline View:** Shows the class structure with the following methods listed:
 - testClear(): void
 - testCtor(): void
 - testDifferentiate(): v
 - testDivMultiElems():
 - testDivTwoElems(): :
 - testDupWithMultVal
 - testDupWithOneVal(
 - testDupWithTwoVal(
 - testIntegrate(): void

ECLIPSE DEBUGGING

Breakpoints Window

Shows all existing breakpoints in the code, along with their conditions and a variety of options.

Double clicking a breakpoint will take you to its spot in the code.

The screenshot displays the Eclipse IDE interface. The Breakpoints window is open, showing a list of breakpoints for the file 'RatPolyStackTest.java'. The breakpoints are:

- Ones [line: 33] - main(String[])
- ProjectEuler26 [line: 25] - main(String[])
- RatPolyStackTest [line: 157] - testDupWithOneVal()
- RatPolyStackTest [line: 159] [conditional] - testDupWithOneVal()
- RatPolyStackTest [line: 162] - testDupWithOneVal()

The Breakpoints window also shows options for Hit count, Suspend thread, Suspend VM, and Conditional (Suspend when 'true' or Suspend when value changes). A dropdown menu is open showing '<Choose a previously entered condition>' and the expression 'x == 6' is entered in the text field below.

The code editor shows the following code:

```
151 ////////////////////////////////////////////////////
152 /// Duplicate
153 ////////////////////////////////////////////////////
154
155 @Test
156 public void testDupWithOneVal() {
157     RatPolyStack stk1 = stack("3");
158     stk1.dup();
159     assertStackIs(stk1, "33");
160     stk1 = stack("123");
161     stk1.dup();
162     assertStackIs(stk1, "1123");
```

ECLIPSE DEBUGGING

Enabled/Disabled Breakpoints

Breakpoints can be temporarily disabled by clicking the checkbox next to the breakpoint. This means it won't stop program execution until re-enabled.

This is useful if you want to hold off testing one thing, but don't want to completely forget about that breakpoint.

```
156 public void testDupWithOneVal() {  
157     RatPolyStack stk1 = stack("3");  
158     stk1.dup();  
159     assertStackIs(stk1, "33");  
160     stk1 = stack("123");  
161     stk1.dup();  
162     assertStackIs(stk1, "1123");
```

The screenshot shows the Eclipse IDE's Breakpoints view. The Breakpoints view is open, displaying a list of breakpoints. The breakpoint for 'RatPolyStackTest [line: 162] - testDupWithOneVal()' is highlighted with a green box, and its checkbox is unchecked, indicating it is disabled. Other breakpoints are checked. The 'Conditional' checkbox is also checked, and the condition 'x == 6' is entered in the text field below. The background shows the Eclipse IDE interface with the Java editor and the Breakpoints view.

ECLIPSE DEBUGGING

Hit count

Breakpoints can be set to occur less-frequently by supplying a hit count of n .

When this is specified, only each n -th time that breakpoint is hit will code execution stop.

The screenshot shows the Eclipse IDE interface during a debug session. The main editor displays the source code for `DelegatingMethodAccessorImpl.invoke(Object, Object[])`. A breakpoint is set at line 157, which is highlighted in green. The breakpoint configuration dialog is open, showing a list of breakpoints for `RatPolyStackTest`. The selected breakpoint is at line 159 with the condition `x == 6`. The `Hit count` field is highlighted with a green box. The `Suspend thread` option is selected.

```
153 // ...
154 // ...
155 @Test
156 public void testDupWithOneVal() {
157     RatPolyStack stk1 = stack("3");
158     stk1.dup();
159     assertStackIs(stk1, "33");
160     stk1 = stack("123");
161     stk1.dup();
162     assertStackIs(stk1, "1123");
}
```

Breakpoint configuration dialog:

- Hit count:
- Suspend thread (selected)
- Suspend VM
- Conditional (checked)
- Suspend when 'true' (selected)
- Suspend when value changes
- Condition: `x == 6`

ECLIPSE DEBUGGING

Conditional Breakpoints

Breakpoints can have conditions. This means the breakpoint will only be triggered when a condition you supply is true. **This is very useful** for when your code only breaks on some inputs!

Watch out though, it can make your code debug very slowly, especially if there's an error in your breakpoint.

```
159   assertStackIs(stk1, "33");
160   stk1 = stack("123");
161   stk1.dup();
162   assertStackIs(stk1, "1123");
```

The screenshot shows the Eclipse IDE interface. The Breakpoints view is open, displaying a list of breakpoints. The breakpoint for `RatPolyStackTest [line: 159] - testDupWithOneVal()` is selected and highlighted. The configuration for this breakpoint is shown below the list, with a green box highlighting the conditional settings:

- Conditional Suspend when 'true' Suspend when value changes
- <Choose a previously entered condition>
- `x == 6`

The background shows the source code editor with the following code:

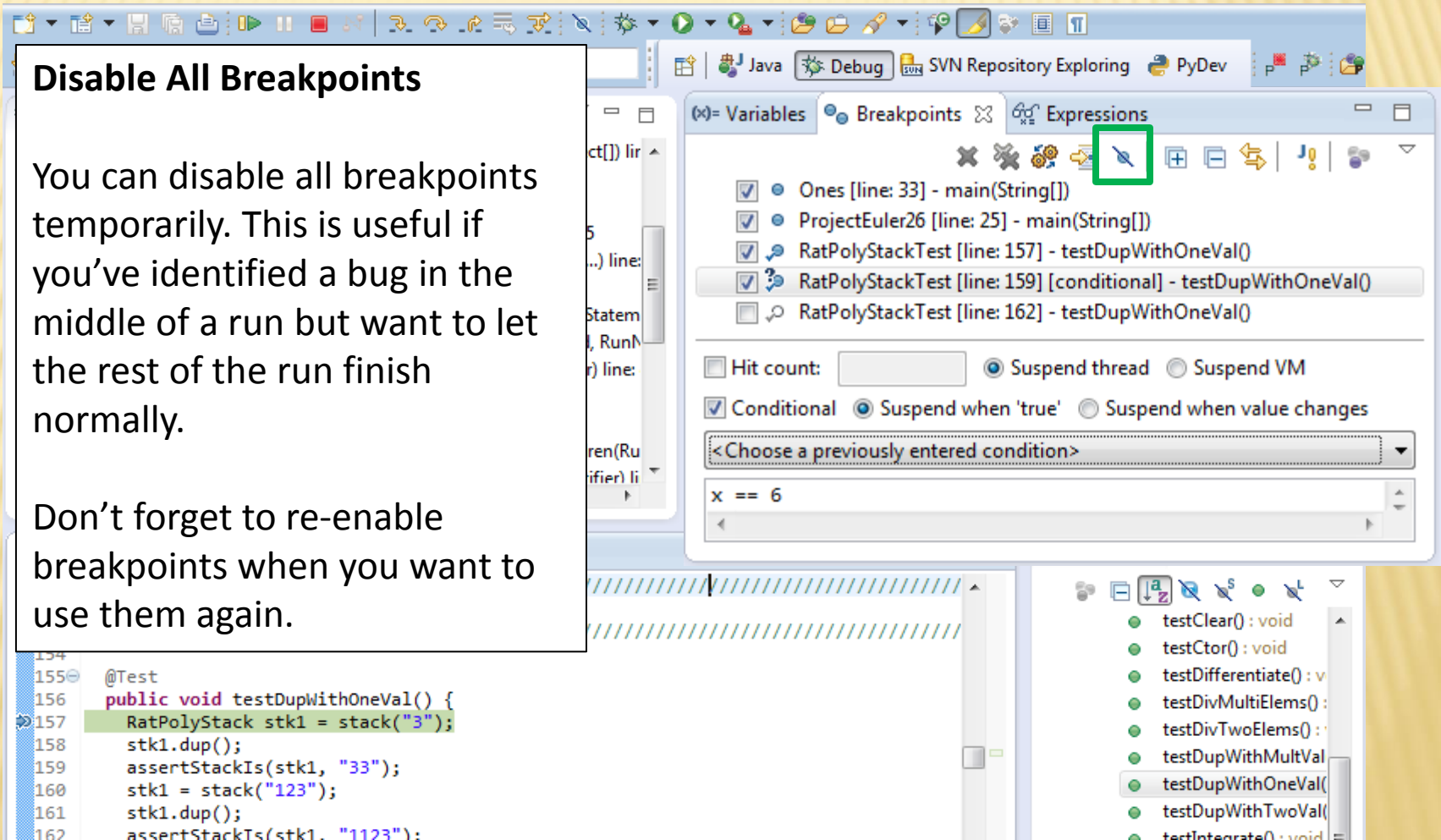
```
159   assertStackIs(stk1, "33");
160   stk1 = stack("123");
161   stk1.dup();
162   assertStackIs(stk1, "1123");
```

ECLIPSE DEBUGGING

Disable All Breakpoints

You can disable all breakpoints temporarily. This is useful if you've identified a bug in the middle of a run but want to let the rest of the run finish normally.

Don't forget to re-enable breakpoints when you want to use them again.



The screenshot shows the Eclipse IDE interface during a debug session. The Breakpoints view is open, displaying a list of breakpoints for the current project. A green box highlights the 'Disable All Breakpoints' icon (a crossed-out wrench) in the toolbar of the Breakpoints view. The list of breakpoints includes:

- Ones [line: 33] - main(String[])
- ProjectEuler26 [line: 25] - main(String[])
- RatPolyStackTest [line: 157] - testDupWithOneVal()
- RatPolyStackTest [line: 159] [conditional] - testDupWithOneVal()
- RatPolyStackTest [line: 162] - testDupWithOneVal()

Below the list, the 'Hit count' is set to 0, and the 'Suspend thread' option is selected. The 'Conditional' checkbox is checked, and the 'Suspend when 'true'' option is selected. The condition field contains the expression `x == 6`.

In the background, the Java editor shows the following code snippet:

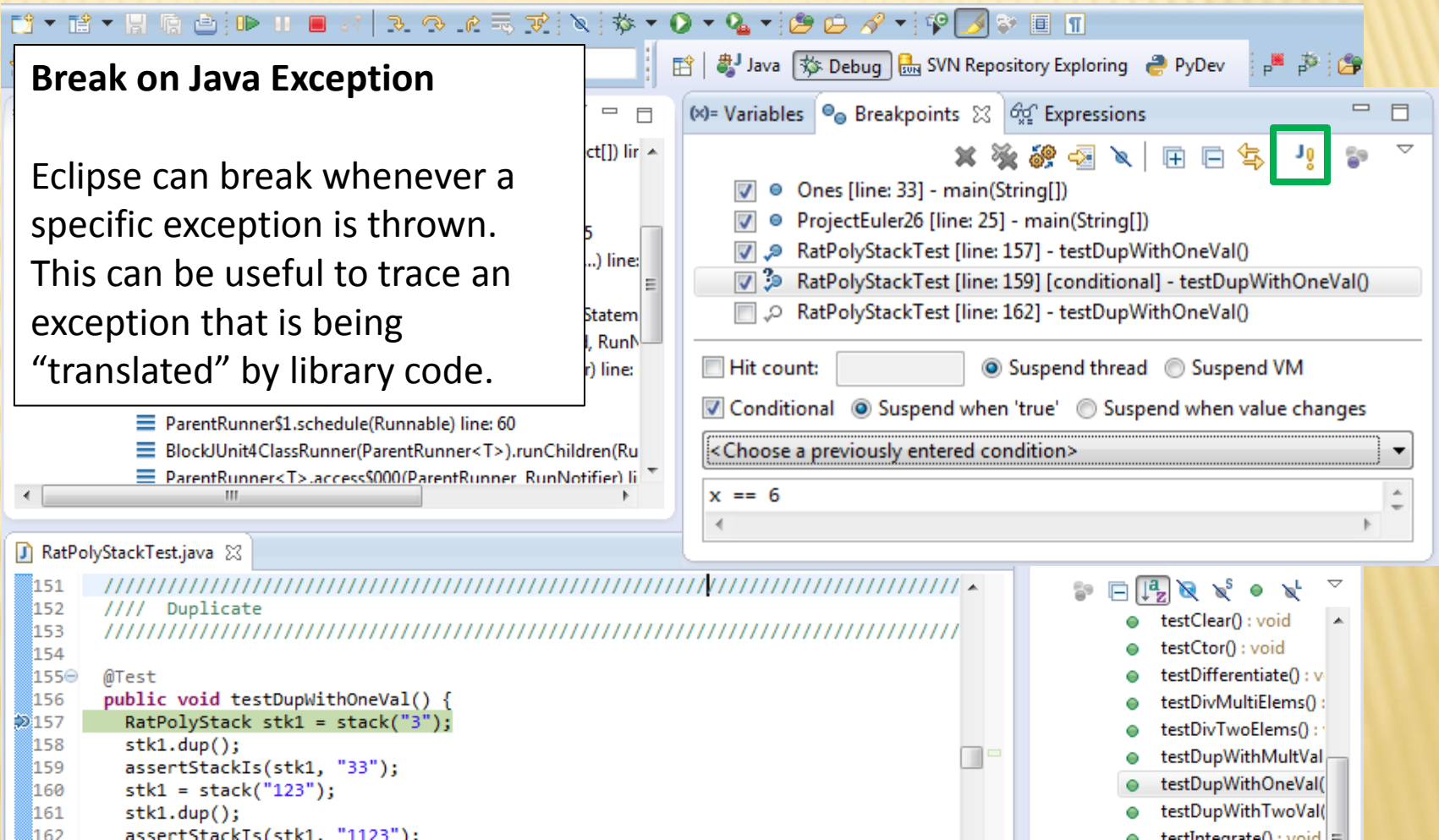
```
154 @Test
155 public void testDupWithOneVal() {
156     RatPolyStack stk1 = stack("3");
157     stk1.dup();
158     assertStackIs(stk1, "33");
159     stk1 = stack("123");
160     stk1.dup();
161     assertStackIs(stk1, "1123");
162 }
```

ECLIPSE DEBUGGING

Break on Java Exception

Eclipse can break whenever a specific exception is thrown. This can be useful to trace an exception that is being “translated” by library code.

```
ParentRunner$1.schedule(Runnable) line: 60
BlockJUnit4ClassRunner(ParentRunner<T>).runChildren(Ru
ParentRunner<T>.access$000(ParentRunner RunNotifier) li
```



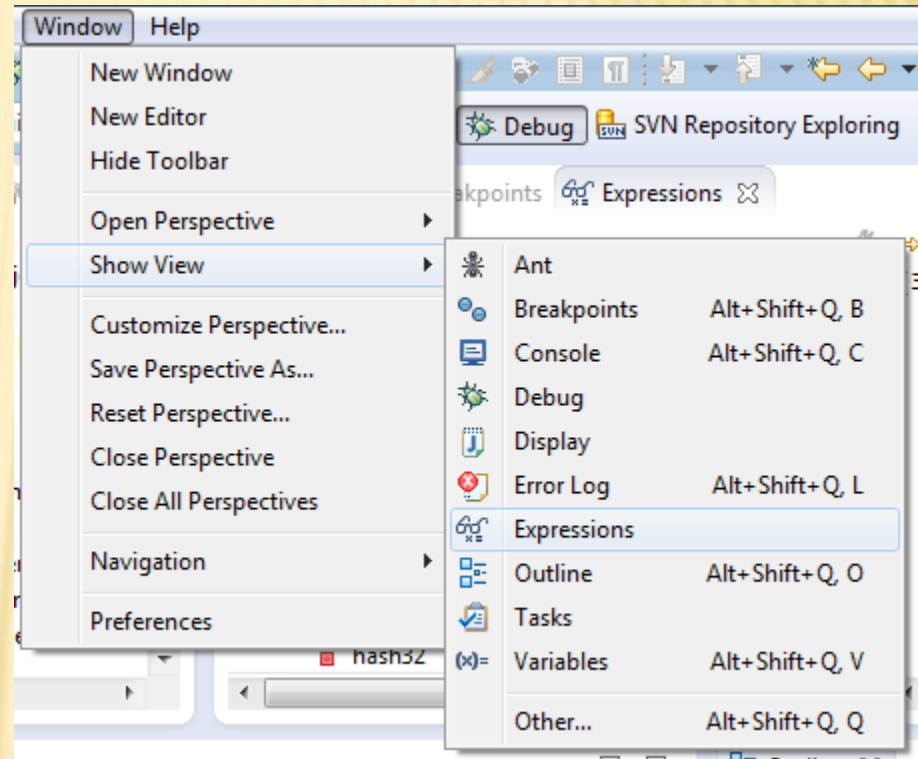
The screenshot shows the Eclipse IDE interface during a debugging session. A 'Breakpoints' dialog box is open in the foreground, allowing the user to configure a breakpoint. The dialog lists several breakpoints for the file 'RatPolyStackTest.java', including one at line 159 with a conditional expression. The 'Conditional' option is selected, and the expression 'x == 6' is entered. The 'Suspend thread' option is also selected. Below the dialog, the source code for 'RatPolyStackTest.java' is visible, with line 157 highlighted: `RatPolyStack stk1 = stack("3");`. The background shows the Eclipse IDE with various toolbars and panels.

ECLIPSE DEBUGGING

Expressions Window

Used to show the results of custom expressions you provide, and can change any time.

Not shown by default but highly recommended.



ECLIPSE DEBUGGING

Expressions Window

Used to show the results of custom expressions you provide, and can change any time.

Resolves variables, allows method calls, even arbitrary statements
"2+2"

Beware method calls that mutate program state – e.g. `stk1.clear()` or `in.nextLine()` – these take effect immediately

The screenshot shows the Eclipse IDE interface. The Expressions Window is open, displaying a table of variables and their values. The table has two columns: 'Name' and 'Value'. The variables listed are:

Name	Value
"this"	(id=33)
"stk1"	(id=57)
"stk1.polys"	(id=61)
capacityIncrement	0
elementCount	3
elementData	Object[10] (id=73)
modCount	3
"stk1.toString()"	hw4.RatPolyStack@...
hash	0
hash32	0

The background shows a code editor with the following Java code:

```
157 RatPolyStack stk1 = stack( 3 );
158 stk1.dup();
159 assertStackIs(stk1, "33");
160 stk1 = stack("123");
161 stk1.dup();
162 assertStackIs(stk1, "1123");
```

Below the code editor, a list of test methods is visible:

- testClear(): void
- testCtor(): void
- testDifferentiate(): void
- testDivMultiElems(): void
- testDivTwoElems(): void
- testDupWithMultVal(): void
- testDupWithOneVal(): void
- testDupWithTwoVal(): void
- testIntegrate(): void

ECLIPSE DEBUGGING

Expressions Window

These persist across projects, so clear out old ones as necessary.

The screenshot displays the Eclipse IDE interface during a debug session. The Expressions window is highlighted with a green border and contains the following data:

Name	Value
X+Y =? "this"	(id=33)
X+Y =? "stk1"	(id=57)
X+Y =? "stk1.polys"	(id=61)
◆ capacityIncrement	0
◆ elementCount	3
▶ ◆ elementData	Object[10] (id=73)
◆ modCount	3
X+Y =? "stk1.toString()"	hw4.RatPolyStack@...
■ hash	0
■ hash32	0

The background shows the source code for `RatPolyStackTest.java` with the following content:

```
151 ////////////////////////////////////////////////////
152 /// Duplicate
153 ////////////////////////////////////////////////////
154
155 @Test
156 public void testDupWithOneVal() {
157     RatPolyStack stk1 = stack("3");
158     stk1.dup();
159     assertStackIs(stk1, "33");
160     stk1 = stack("123");
161     stk1.dup();
162     assertStackIs(stk1, "1123");
```

ECLIPSE DEBUGGING

- The debugger is awesome, but not perfect
 - + Not well-suited for time-dependent code
 - + Recursion can get messy
- Technically, we talked about a “breakpoint debugger”
 - + Allows you to stop execution and examine variables
 - + Useful for stepping through and visualizing code
 - + There are other approaches to debugging that don't involve a debugger