SECTION 2:
HW3 Setup

Justin Bare and Deric Pang

slides borrowed and adapted from Alex Mariakis and CSE 390a

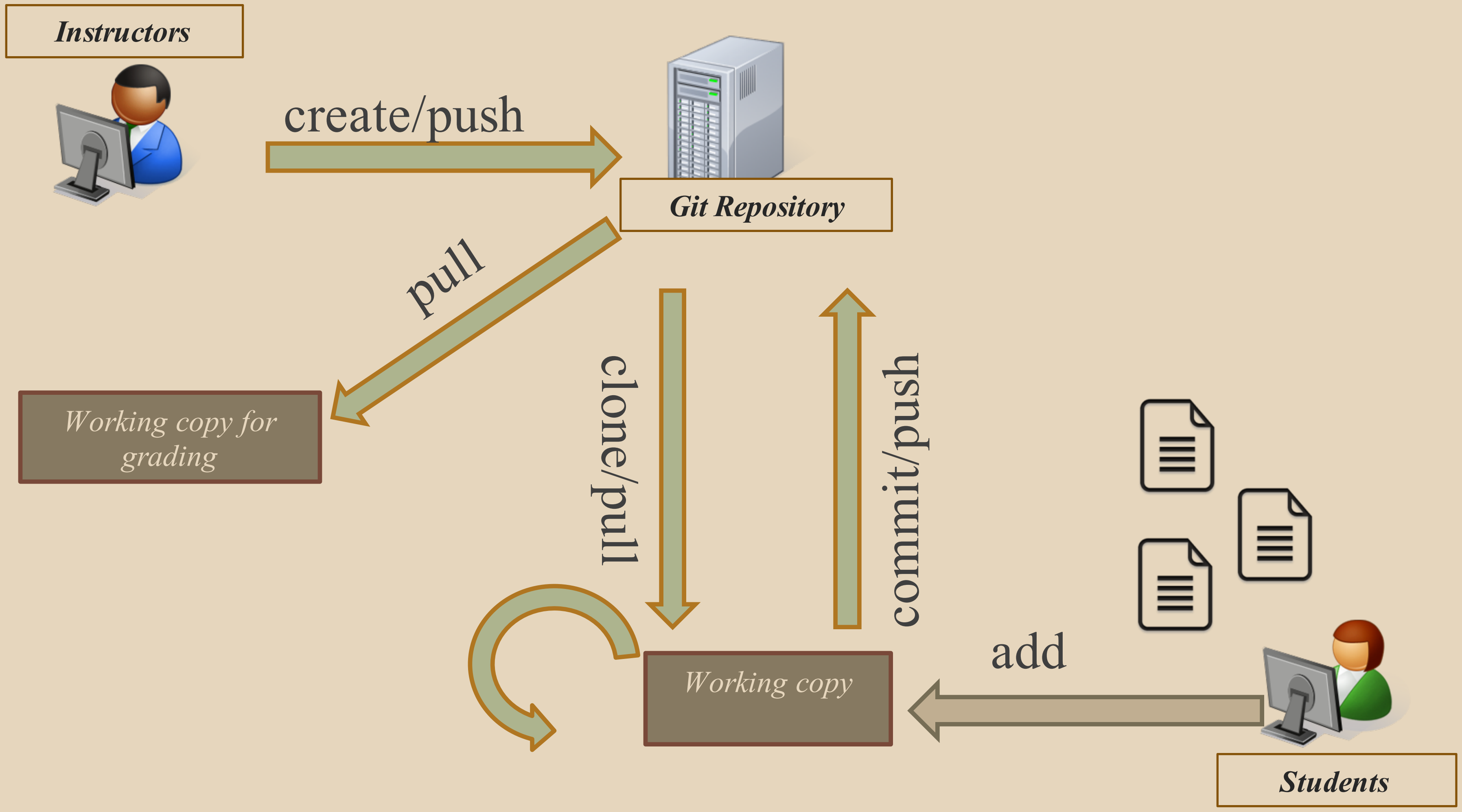
OUTLINE

- Version Control – Git
- Eclipse and Java versions
- SSH
- Ant
- Eclipse Debugging

LINKS TO DETAILED SETUP AND USAGE INSTRUCTIONS

- All References
  - http://courses.cs.washington.edu/courses/cse331/16wi/#ref
- Basic Setup
  - http://courses.cs.washington.edu/courses/cse331/16wi/tools/basicSetup.html
- Working from home: Java, Eclipse, SSH
  - http://courses.cs.washington.edu/courses/cse331/16wi/tools/WorkingAtHome.html
- Editing, Compiling, Running, and Testing Programs
  - http://courses.cs.washington.edu/courses/cse331/16wi/tools/editing-compiling.html
- Eclipse Reference
  - http://courses.cs.washington.edu/courses/cse331/16wi/tools/eclipse_reference.html
- Version Control - Git
  - http://courses.cs.washington.edu/courses/cse331/16wi/tools/versioncontrol.html
- Assignment Submission
  - http://courses.cs.washington.edu/courses/cse331/16wi/tools/turnin.html

331 VERSION CONTROL



GIT BEST PRACTICES

- Add/commit/push your code **EARLY** and **OFTEN!!!**
    - You really, really, really don’t want to deal with merge conflicts
    - Keep your repository up-to-date all the time
- Use the combined ‘Commit and Push’ tool in Eclipse
- Do not rename folders and files that we gave you – this will mess up our grading process and you could get a bad score
- Use the repo only for the homework
    - Adding other stuff (like notes from lecture) may mess up our grading process

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)

- 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 and Java

- Get Java **8**

- Please use **Eclipse 4.5 (Mars), “Eclipse for Java Developers”**

DEMO

- Eclipse
- Git cloning

DEVELOPMENT PROCESS

- We distribute starter code by adding it to your GitLab repo. You retrieve it with **git clone** the first time then **git pull** for later assignments
- You will write code using Eclipse
- You update your files on the repo by **adding** them to the repo, **committing** your changes, and eventually **pushing** accumulated changes to GitLab
- You “turn in” an assignment when you’re finished by **tagging** your repo and **pushing** the tag to GitLab
- You will validate your homework submission by **SSHing** onto attu, **cloning** your repo, and running **ant validate**

HW 3

- Many small exercises to get you used to version control and tools and a Java refresher
- More information on homework instructions

DEMO

- Editing HW3
- Git pull in Eclipse
- Add/commit/push in Eclipse

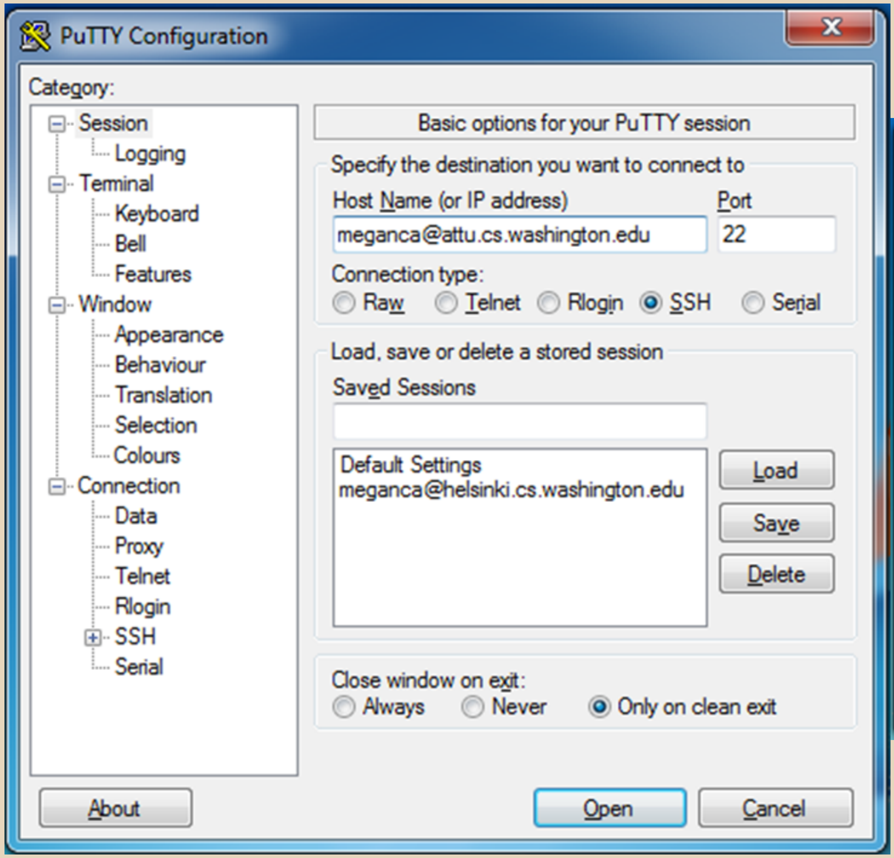
Turning in HW3

- Add/commit/push your final code
- Create a **hw3-final** tag on the last commit and push the tag to the repo (this can and should be done in Eclipse)
  - You can push a new hw3-final tag that overwrites the old one if you realize that you still need to make changes
    - But keep track of how many late days you have left!
- After the final commit and tag pushed, remember to log on to attu and run ant validate

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”

PUTTY



Specify the destination you want to connect to

Host Name (or IP address): meganca@attu.cs.washington.edu

Port: 22

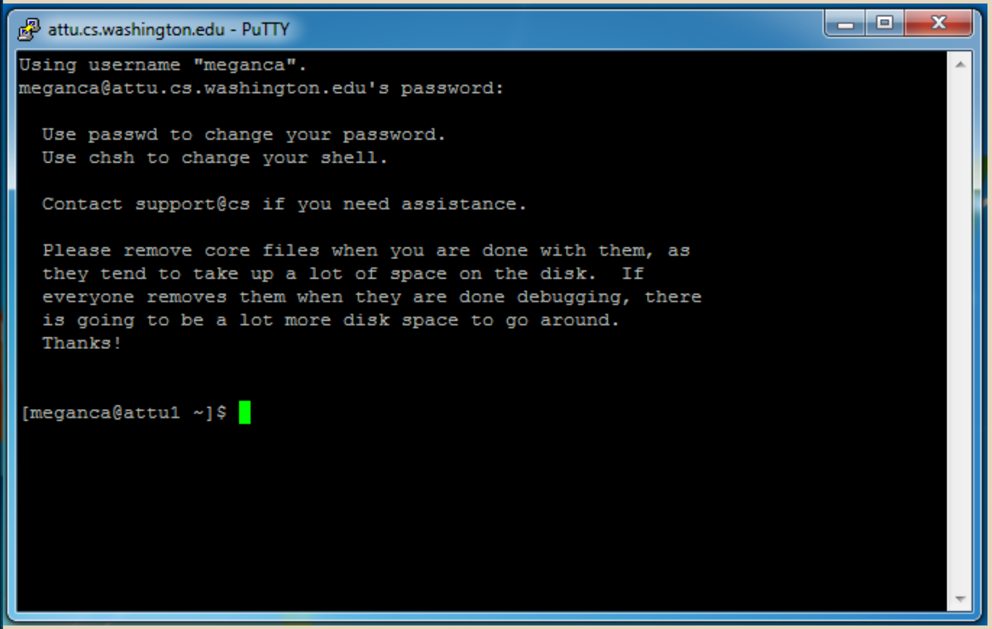
Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

Saved Sessions

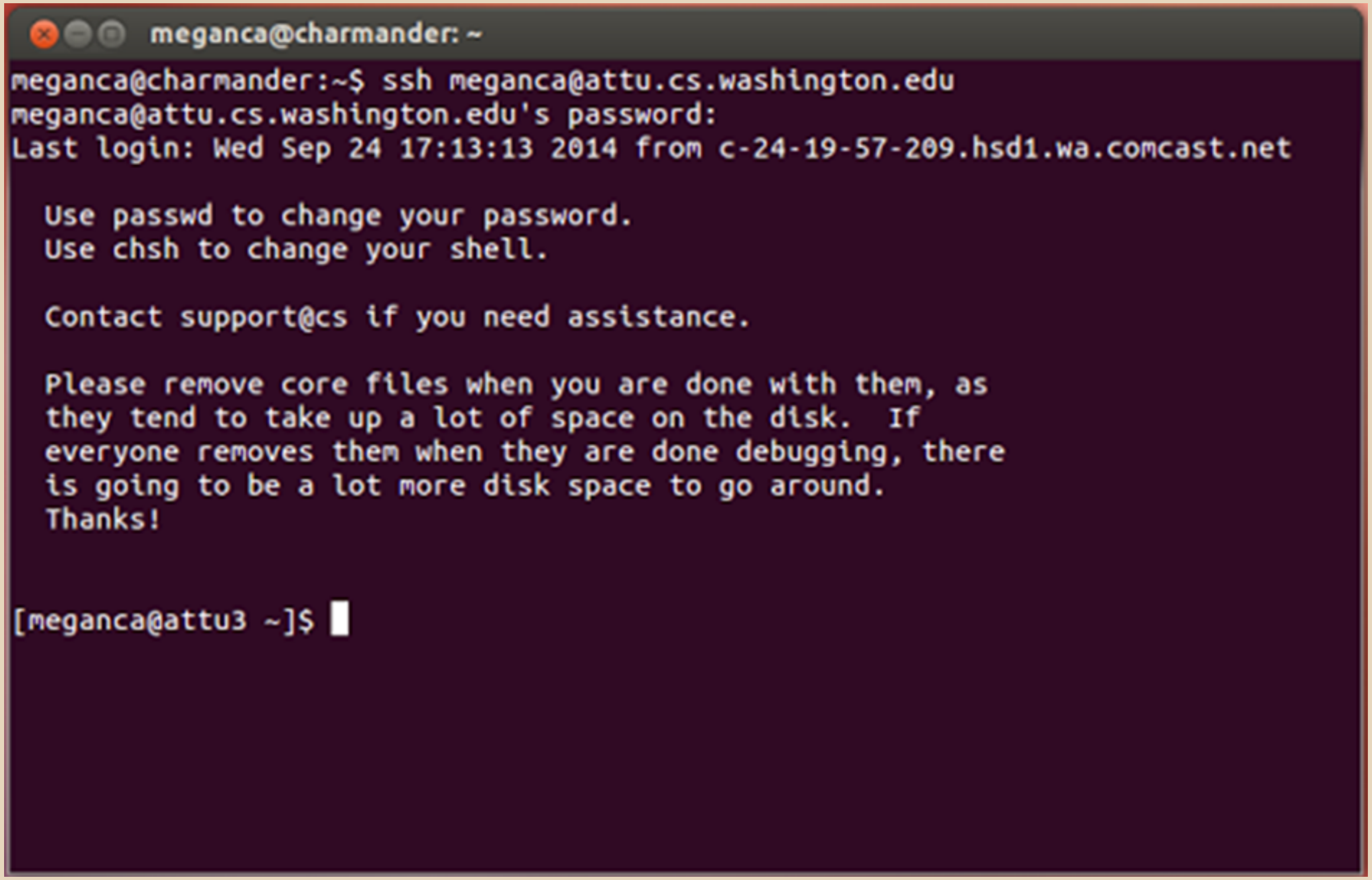
Default Settings
meganca@helsinki.cs.washington.edu

Close window on exit: Always Never [✓ Only on clean exit]



```
Using username "meganca".
meganca@attu.cs.washington.edu's password:

  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@attu1 ~]$
```

TERMINAL (LINUX, MAC)



```
meganca@charmander:~$ 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 ~]$ 
```

Ant Validate

- **What will this do?**
  - You start with a freshly cloned copy of your repo and do “git checkout hw3-final” to switch to the files you intend for us to grade, then run ant validate
  - Makes sure you have all the **required** files
  - Make sure your homework builds without errors
  - Passes specification and implementation tests in the repository
    - **Note**: this does not include the additional tests we will use when grading
    - This is just a sanity check that your current tests pass

Ant Validate

- **How do you run ant validate?**
  - Has to be done on attu from the command line since that is the environment your grading will be done on

  - Do not use the Eclipse ant validate build tool!

  - Be *sure* to use a fresh copy of your repo, and discard that copy when you’re done
    - If you need to fix things, do it in your primary working copy (eclipse)

Ant Validate

- **How do you run ant validate?**
  - Steps
    - Log into attu via SSH
    - In attu, checkout a brand new local copy (clone) of your repository through the command-line
      - **Note:** Now, you have two local copies of your repository, one on your computer through Eclipse and one in attu
    - Go to the hw folder which you want to validate through the 'cd' command, then switch to the hw3 tag
      - For example: cd ~/cse331/src/hw3
        git checkout hw3-final
    - Run ant validate

Ant Validate

- **How do you know it works?**
  - If successful, will output **Build Successful** at the bottom

  - If unsuccessful, will output **Build Failed** at the bottom with information on why
    - If ant validate failed, discard the validate copy of the repo on attu, fix and commit changes through eclipse, go back to attu, clone a fresh copy of the repo, and try ant validate again

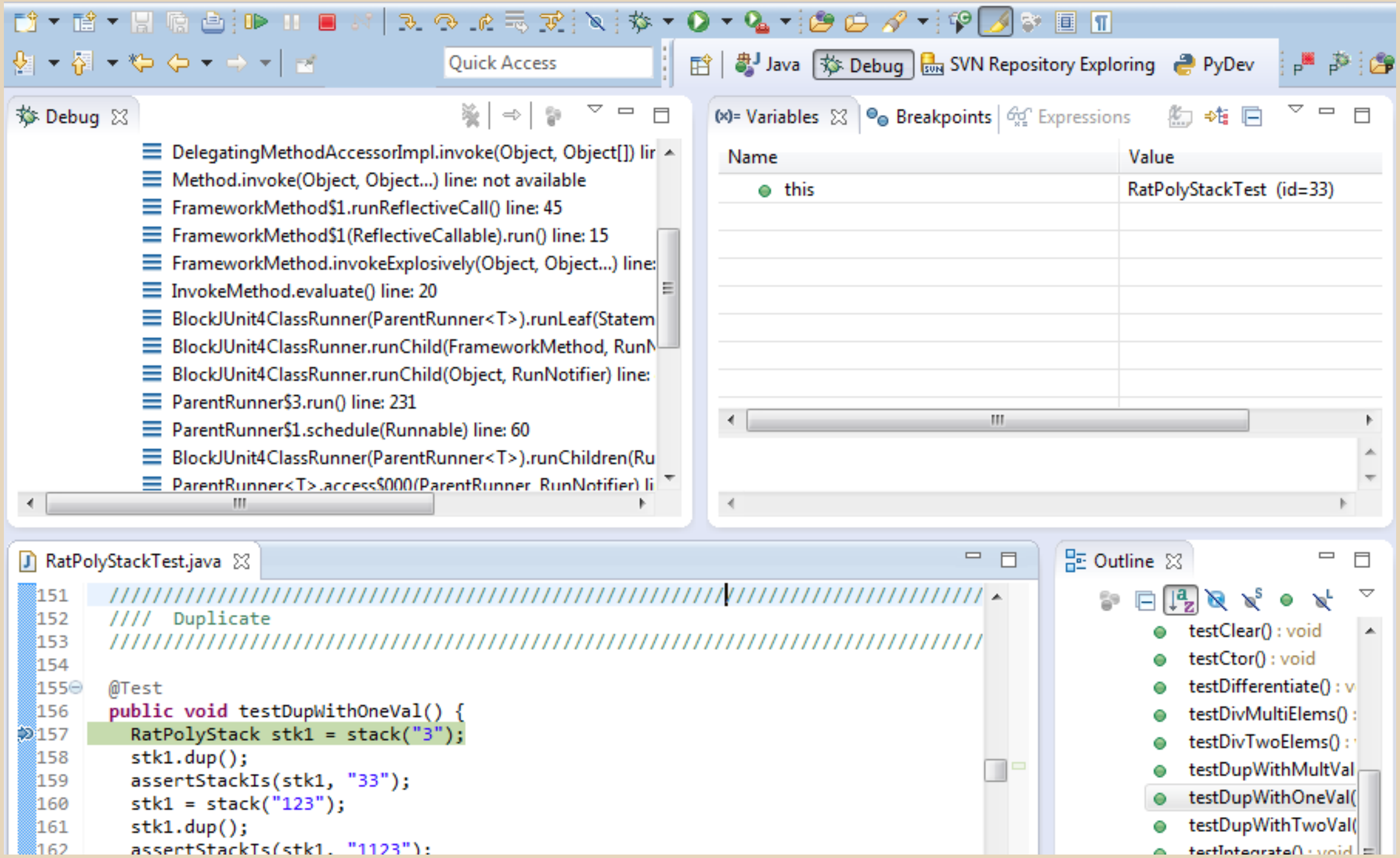
DEMO

- Tagging your final submission
- SSH into attu
- Running ant validate on attu

ECLIPSE DEBUGGING (if time)

- 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



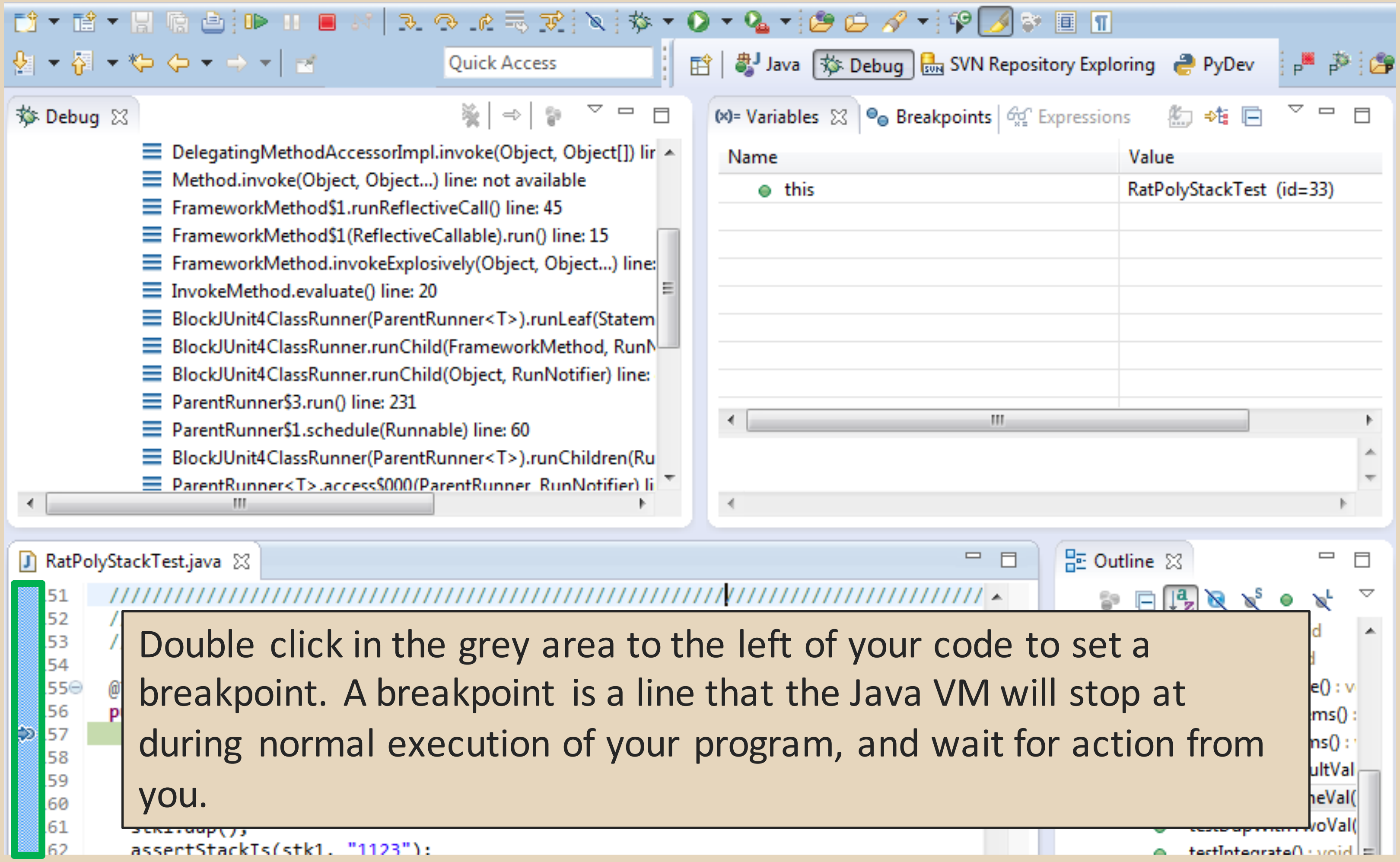
```
DelegatingMethodAccessorImpl.invoke(Object, Object[]) lir
Method.invoke(Object, Object...) line: not available
FrameworkMethod$1.runReflectiveCall() line: 45
FrameworkMethod$1(ReflectiveCallable).run() line: 15
FrameworkMethod.invokeExplosively(Object, Object...) line:
InvokeMethod.evaluate() line: 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
```

Name	Value
this	RatPolyStackTest (id=33)

```
////////////////////////////////////////////////////////////////////////////
////  Duplicate
////////////////////////////////////////////////////////////////////////////

@Test
public void testDupWithOneVal() {
  RatPolyStack stk1 = stack("3");
  stk1.dup();
  assertStackIs(stk1, "33");
  stk1 = stack("123");
  stk1.dup();
  assertStackIs(stk1, "1123");
```

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

ECLIPSE DEBUGGING



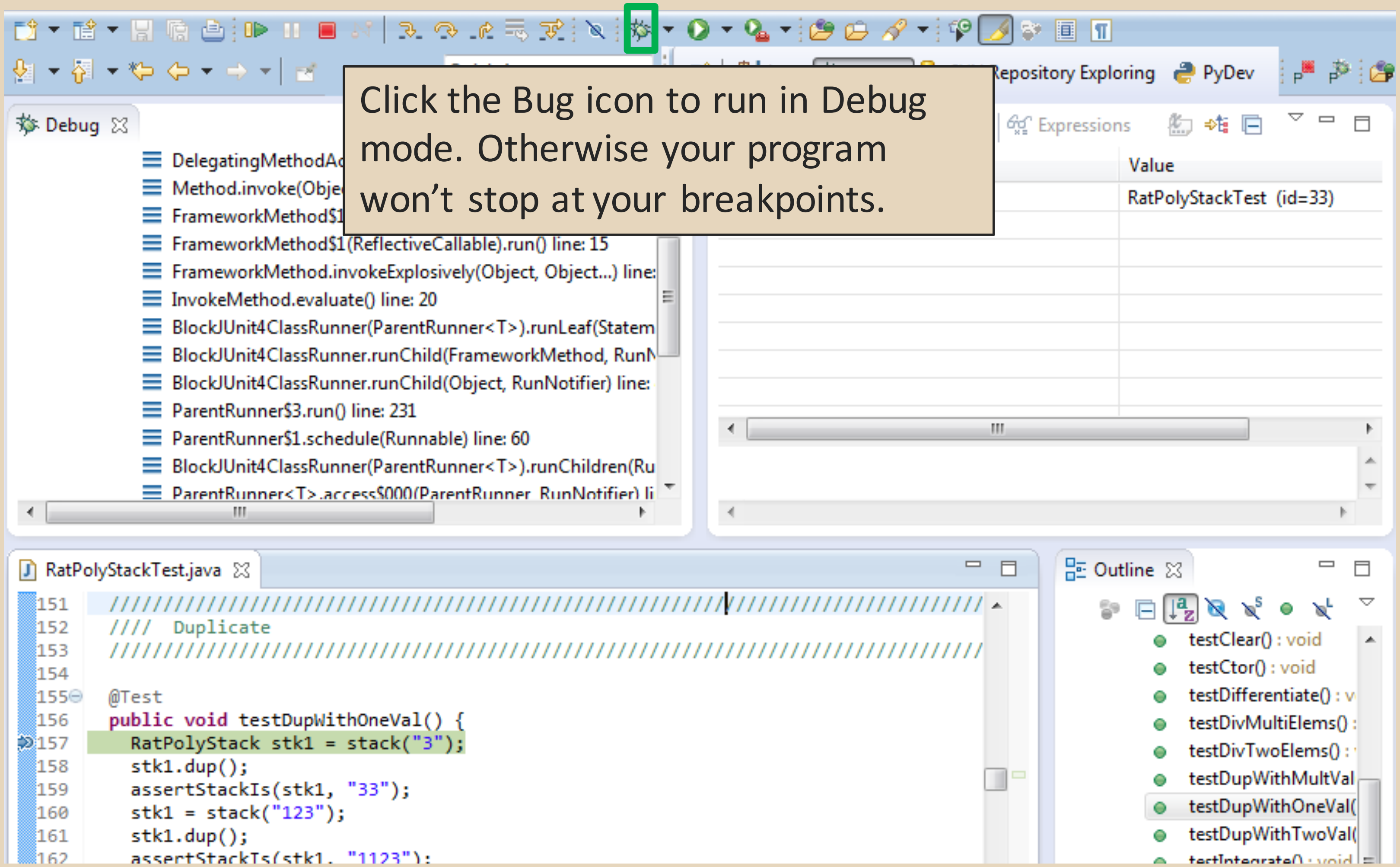
```
DelegatingMethodAccessorImpl.invoke(Object, Object[]) lir
Method.invoke(Object, Object...) line: not available
FrameworkMethod$1.runReflectiveCall() line: 45
FrameworkMethod$1(ReflectiveCallable).run() line: 15
FrameworkMethod.invokeExplosively(Object, Object...) line:
InvokeMethod.evaluate() line: 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
```

Name	Value
this	RatPolyStackTest (id=33)

```
////////////////////////////////////////////////////////////////////////////////
/
/

@
p



    stk1.dup();
    assertStackIs(stk1, "1123");
```

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



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

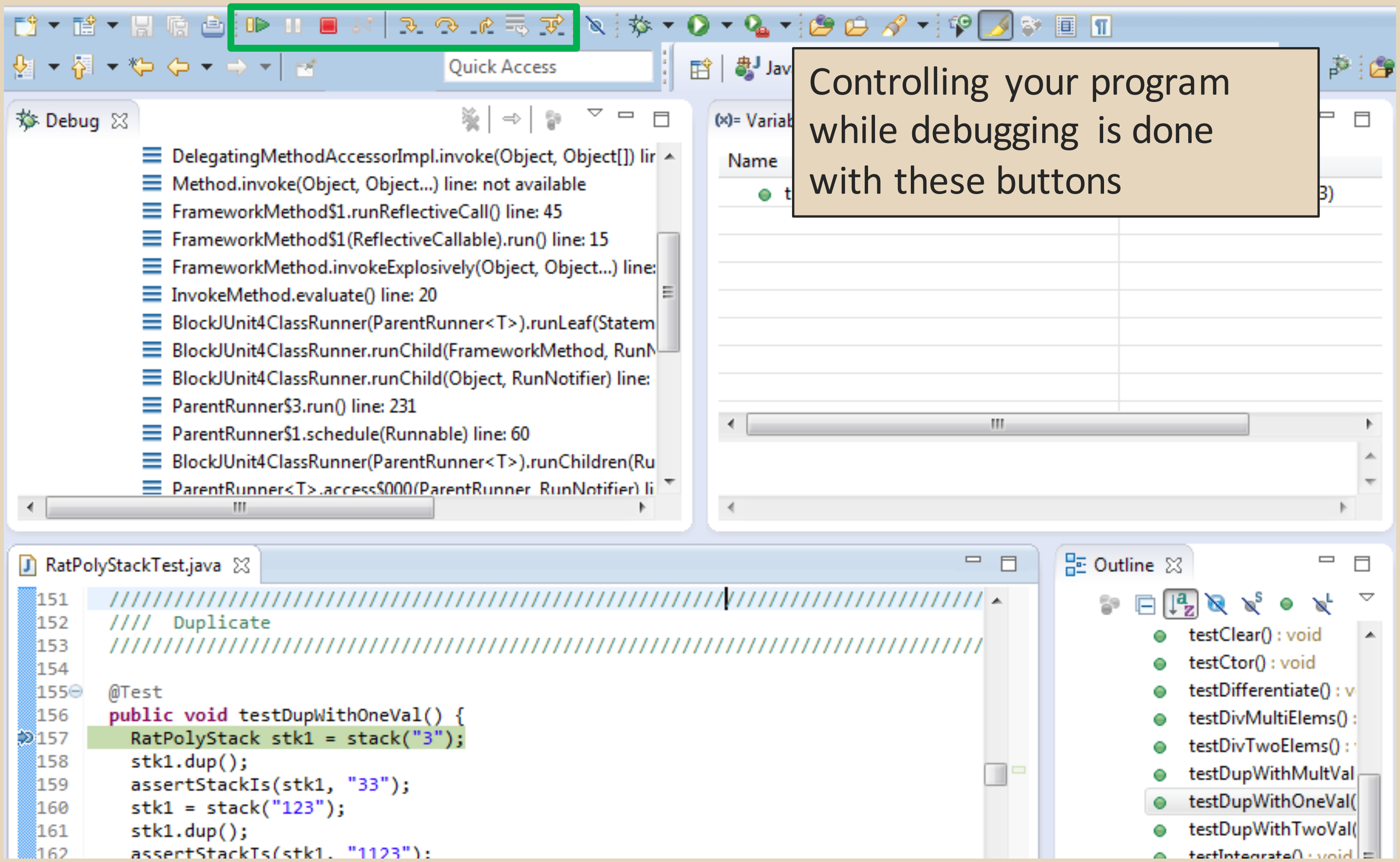
```
DelegatingMethodAc
Method.invoke(Obje
FrameworkMethod$1
FrameworkMethod$1(ReflectiveCallable).run() line: 15
FrameworkMethod.invokeExplosively(Object, Object...) line:
InvokeMethod.evaluate() line: 20
BlockJUnit4ClassRunner(ParentRunner<T>).runLeaf(Statem
BlockJUnit4ClassRunner.runChild(FrameworkMethod, 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
```

Value: RatPolyStackTest (id=33)

```
/////////////////////////////////////////////////////////////////////////////////
////  Duplicate
/////////////////////////////////////////////////////////////////////////////////

@Test
public void testDupWithOneVal() {
  RatPolyStack stk1 = stack("3");
  stk1.dup();
  assertStackIs(stk1, "33");
  stk1 = stack("123");
  stk1.dup();
  assertStackIs(stk1, "1123");
```

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

ECLIPSE DEBUGGING

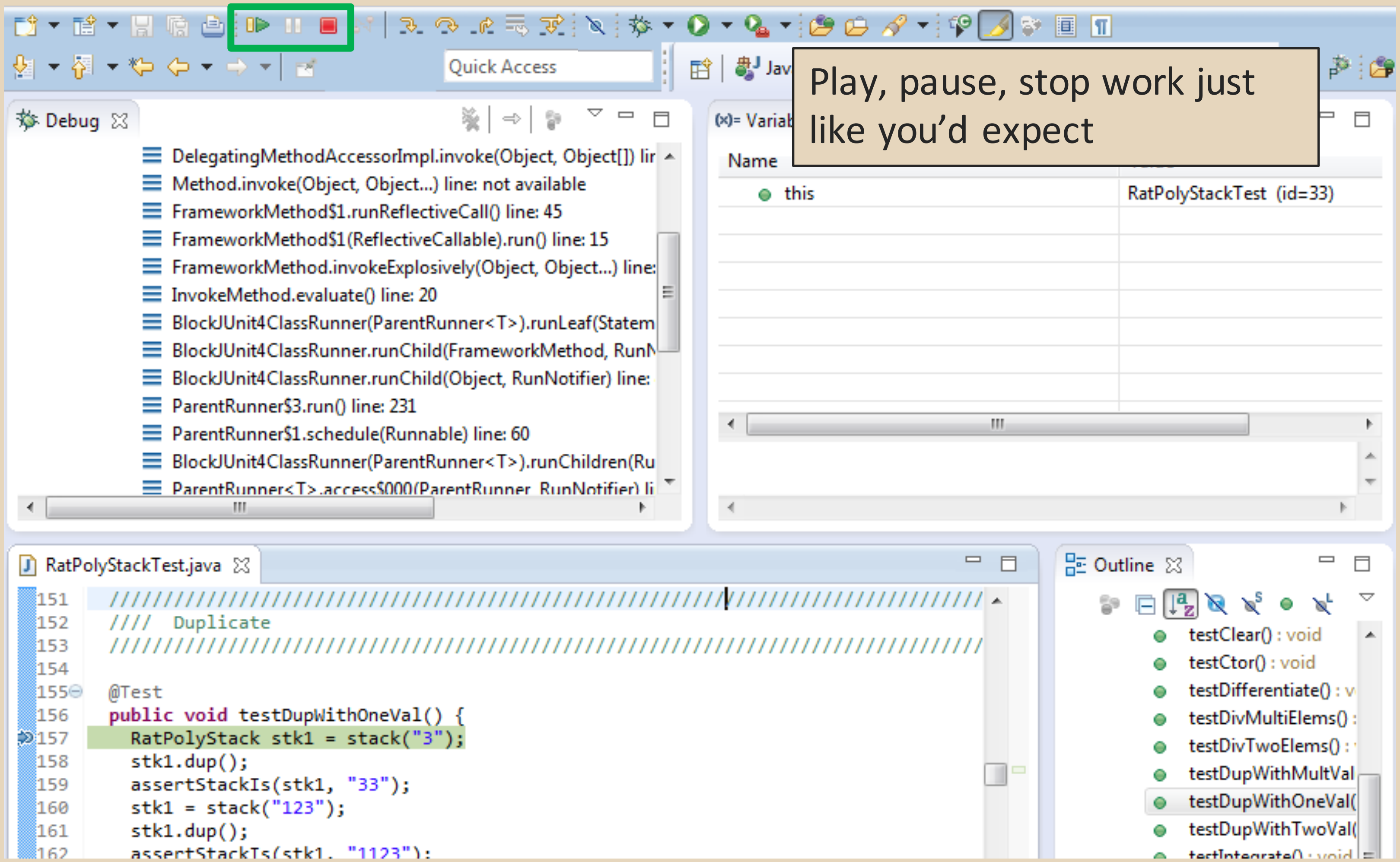

Controlling your program while debugging is done with these buttons

```
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
```

```
////////////////////////////////////////////////////////////////////////////////
////  Duplicate
////////////////////////////////////////////////////////////////////////////////

@Test
public void testDupWithOneVal() {
  RatPolyStack stk1 = stack("3");
  stk1.dup();
  assertStackIs(stk1, "33");
  stk1 = stack("123");
  stk1.dup();
  assertStackIs(stk1, "1123");
```

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

ECLIPSE DEBUGGING



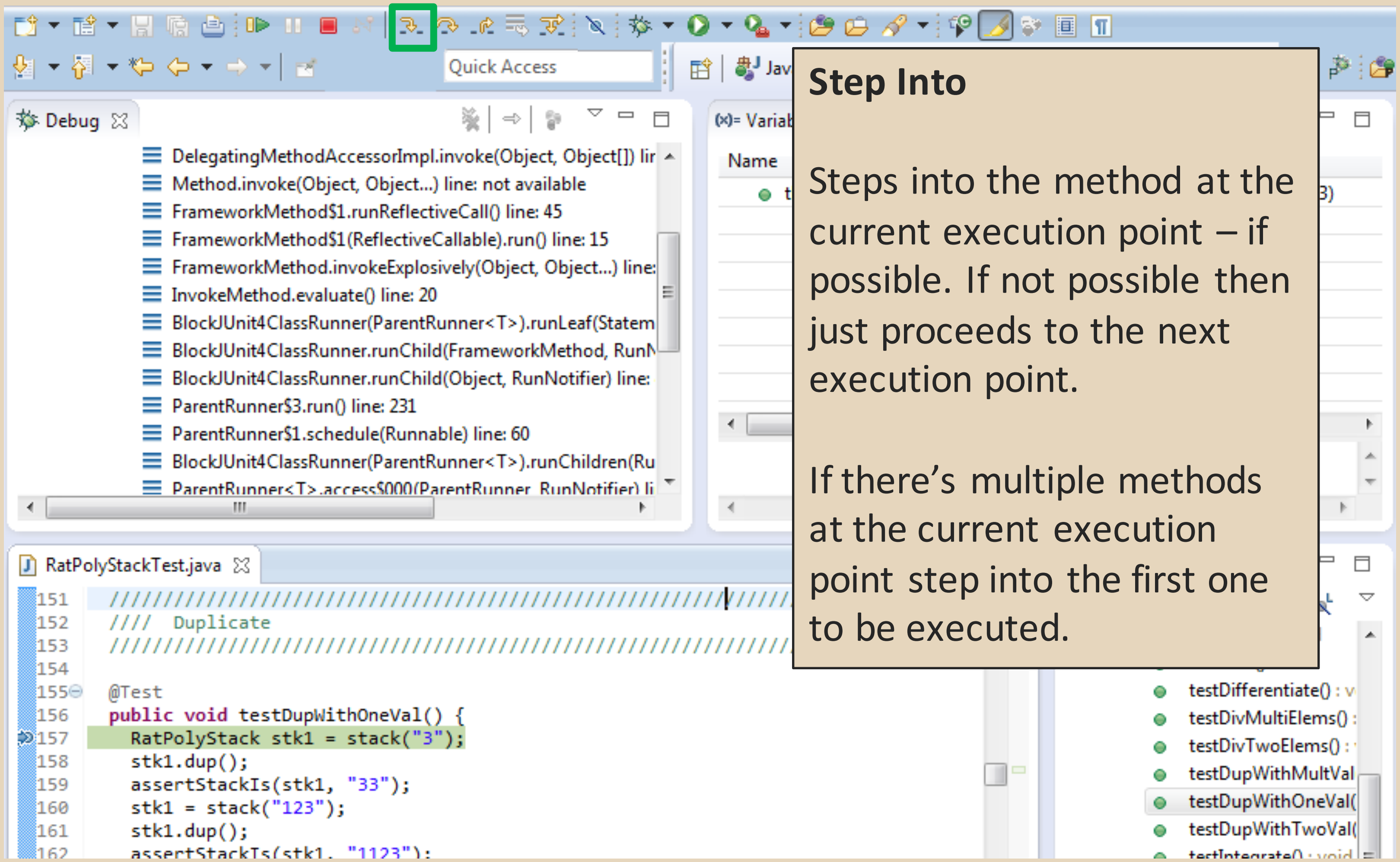
```
DelegatingMethodAccessorImpl.invoke(Object, Object[]) lir
Method.invoke(Object, Object...) line: not available
FrameworkMethod$1.runReflectiveCall() line: 45
FrameworkMethod$1(ReflectiveCallable).run() line: 15
FrameworkMethod.invokeExplosively(Object, Object...) line:
InvokeMethod.evaluate() line: 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
```

Name	Value
this	RatPolyStackTest (id=33)

```
//////////////////////////////////////////////////////////////////////////////
////  Duplicate
//////////////////////////////////////////////////////////////////////////////

@Test
public void testDupWithOneVal() {
  RatPolyStack stk1 = stack("3");
  stk1.dup();
  assertStackIs(stk1, "33");
  stk1 = stack("123");
  stk1.dup();
  assertStackIs(stk1, "1123");
```

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

ECLIPSE DEBUGGING



```
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
```

```
    ////////////////////////////////////////////////////////////
    ////  Duplicate
    ////////////////////////////////////////////////////////////

    @Test
    public void testDupWithOneVal() {
      RatPolyStack stk1 = stack("3");
      stk1.dup();
      assertStackIs(stk1, "33");
      stk1 = stack("123");
      stk1.dup();
      assertStackIs(stk1, "1123");
```

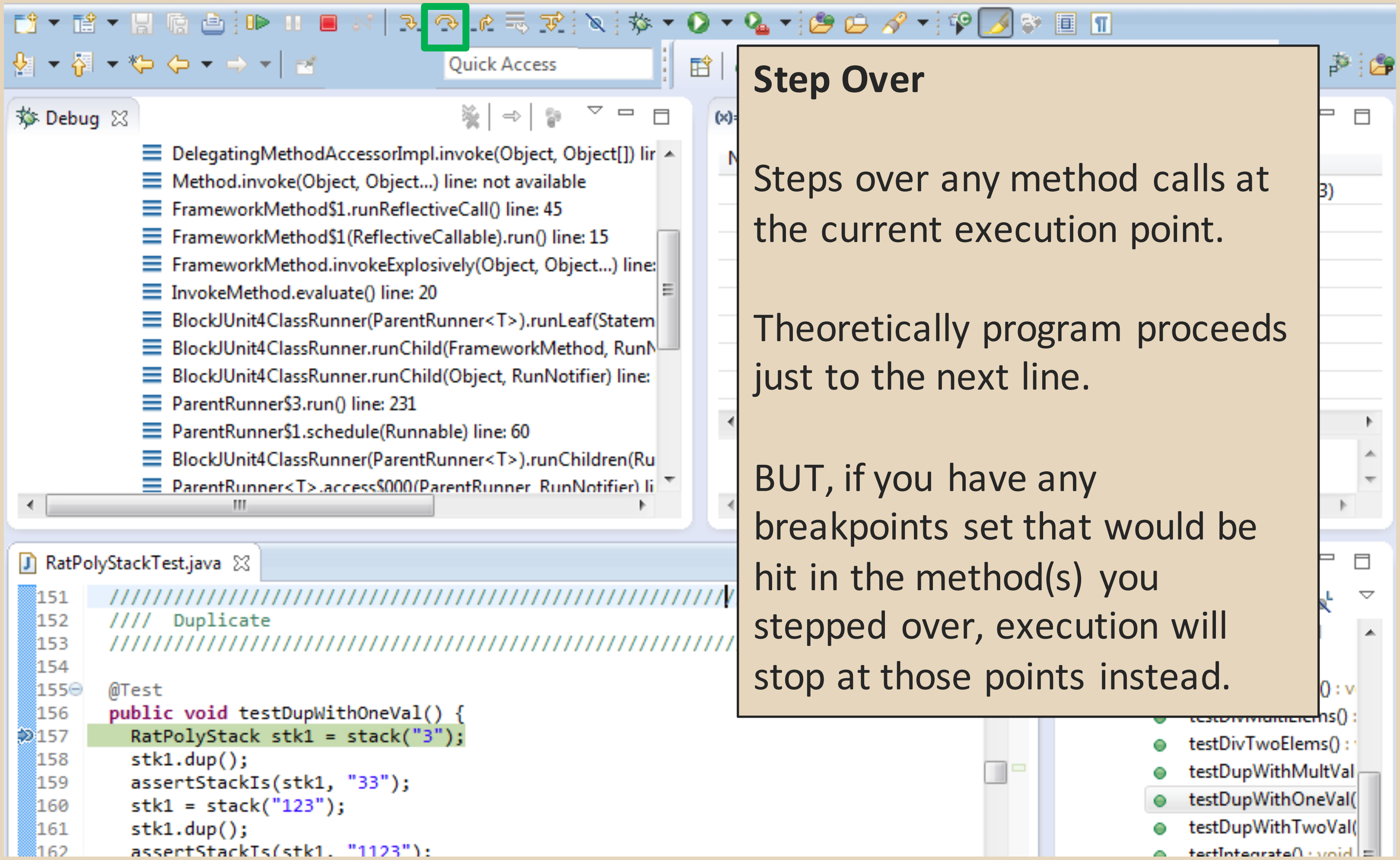
```
testDifferentiate() : v
testDivMultiElems() :
testDivTwoElems() :
testDupWithMultVal
testDupWithOneVal(
testDupWithTwoVal(
testIntegrate() : void
```

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



```
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
```

```
////////////////////////////////////////////////////////////
////  Duplicate
////////////////////////////////////////////////////////////

@Test
public void testDupWithOneVal() {
  RatPolyStack stk1 = stack("3");
  stk1.dup();
  assertStackIs(stk1, "33");
  stk1 = stack("123");
  stk1.dup();
  assertStackIs(stk1, "1123");
```

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

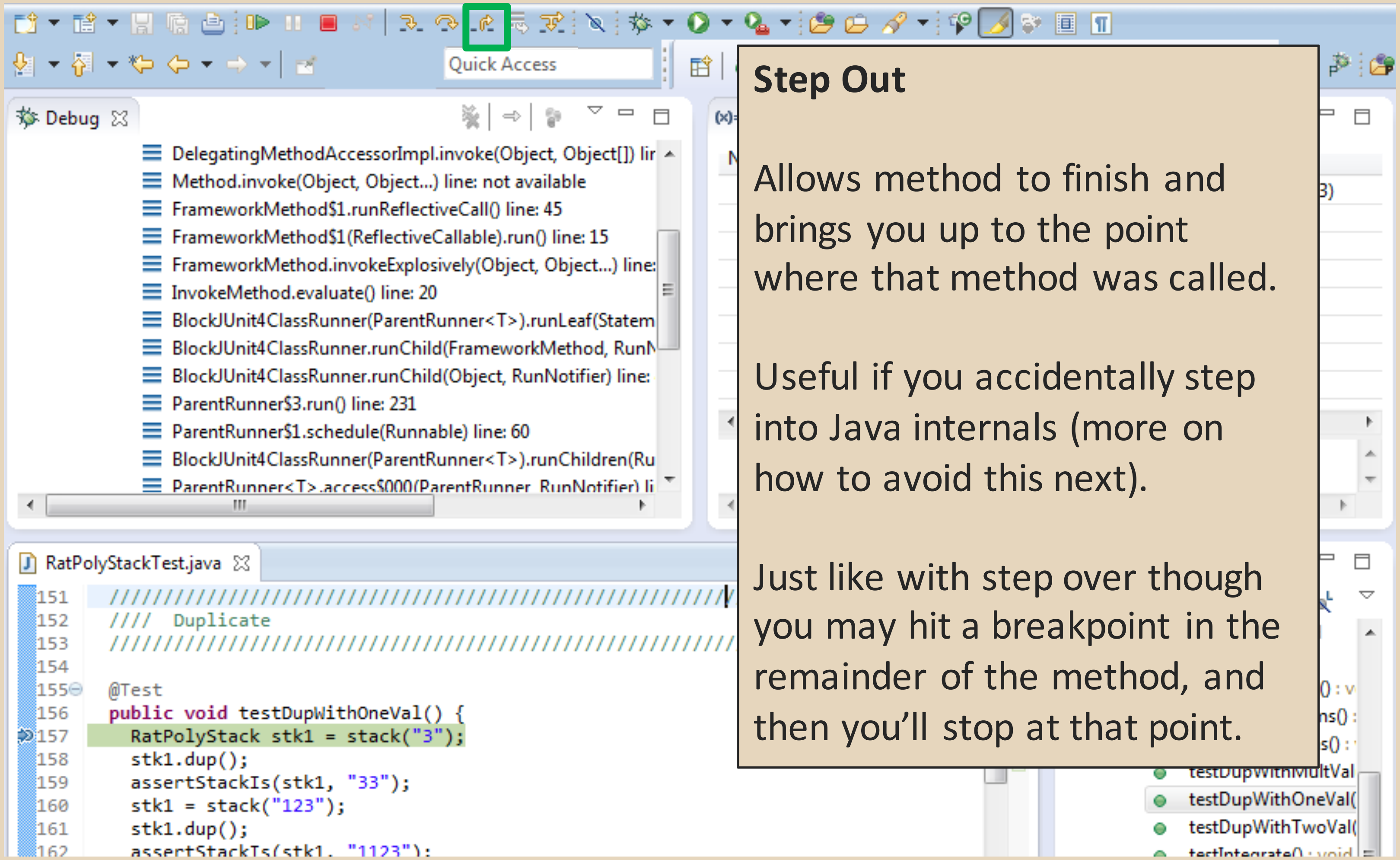
Step Over

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.

ECLIPSE DEBUGGING



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

```
    ///////////////////////////////////////////////////////////////
    ////  Duplicate
    ///////////////////////////////////////////////////////////////

    @Test
    public void testDupWithOneVal() {
      RatPolyStack stk1 = stack("3");
      stk1.dup();
      assertStackIs(stk1, "33");
      stk1 = stack("123");
      stk1.dup();
      assertStackIs(stk1, "1123");
```

testDupWithMultVal
testDupWithOneVal(
testDupWithTwoVal(
testIntegrate() : void

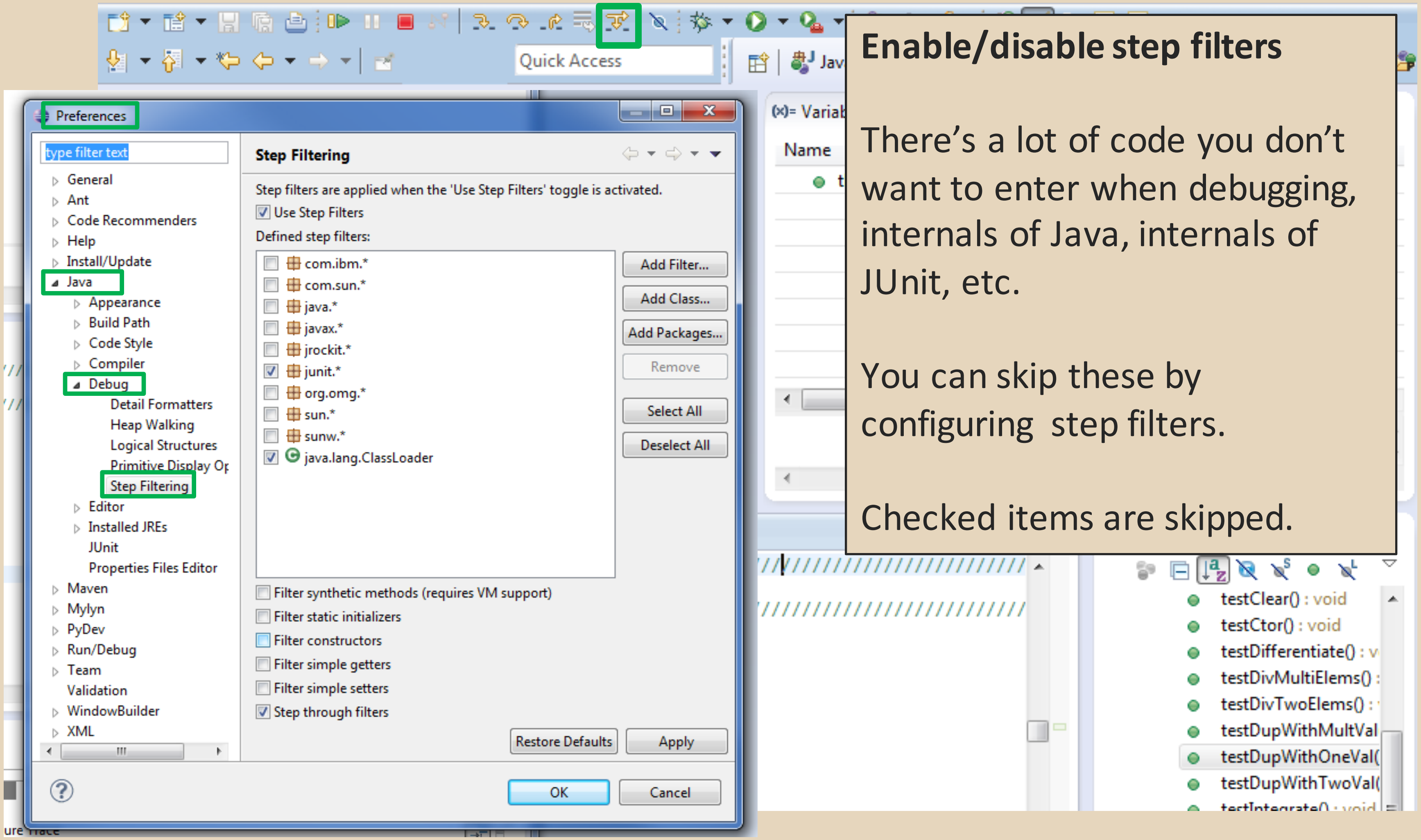
Step Out

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.

ECLIPSE DEBUGGING



Step Filtering

Step filters are applied when the 'Use Step Filters' toggle is activated.

- [✓] Use Step Filters

Defined step filters:

- [ ] com.ibm.*
- [ ] com.sun.*
- [ ] java.*
- [ ] javax.*
- [ ] jrockit.*
- [✓] junit.*
- [ ] org.omg.*
- [ ] sun.*
- [ ] sunw.*
- [✓] java.lang.ClassLoader

- [ ] Filter synthetic methods (requires VM support)
- [ ] Filter static initializers
- [ ] Filter constructors
- [ ] Filter simple getters
- [ ] Filter simple setters
- [✓] Step through filters

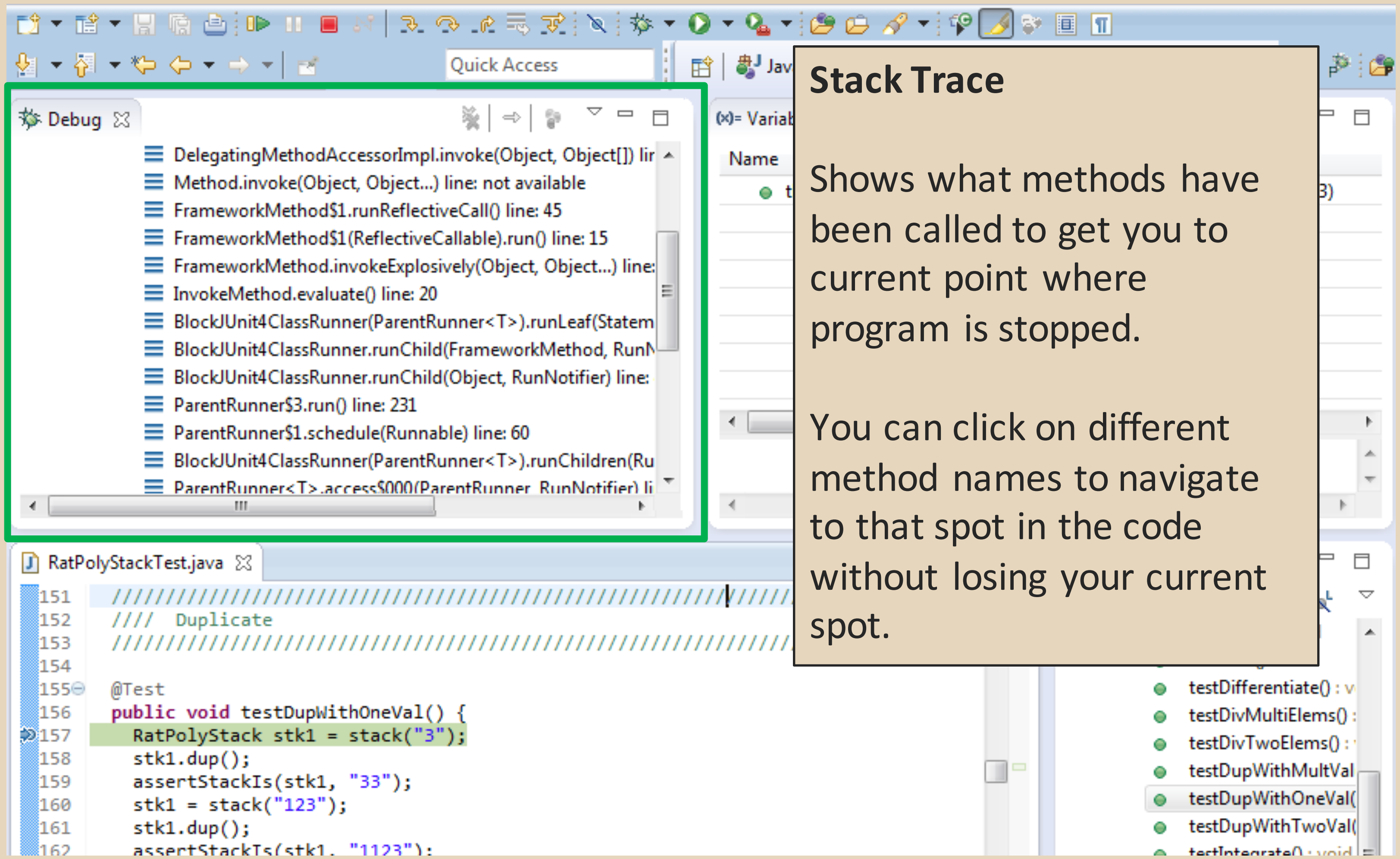
Enable/disable step filters

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.

ECLIPSE DEBUGGING



```
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
```

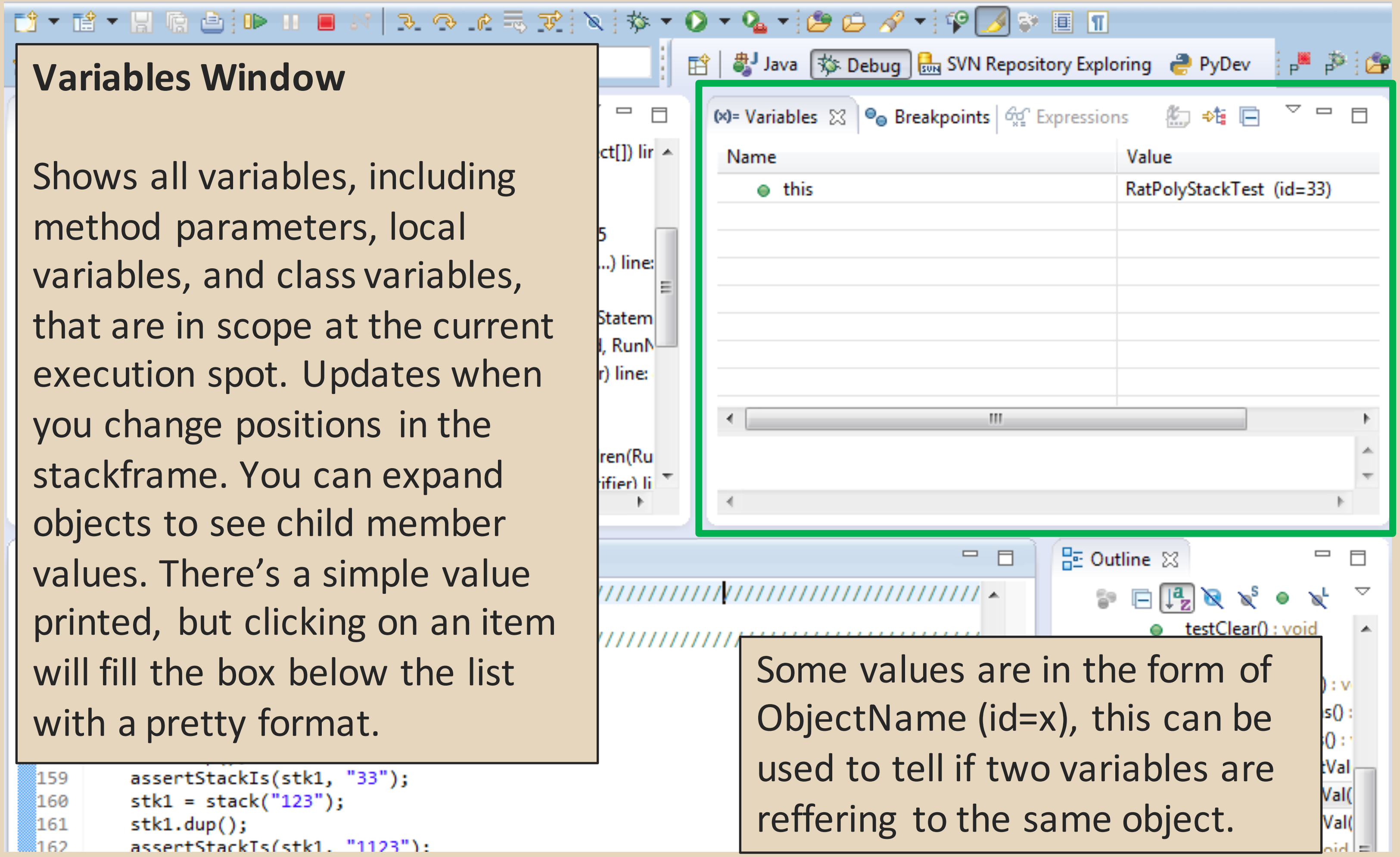
```java
//////////////////////////////////////////////////////////////
////  Duplicate
//////////////////////////////////////////////////////////////

@Test
public void testDupWithOneVal() {
  RatPolyStack stk1 = stack("3");
  stk1.dup();
  assertStackIs(stk1, "33");
  stk1 = stack("123");
  stk1.dup();
  assertStackIs(stk1, "1123");
```

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



Name	Value
this	RatPolyStackTest (id=33)

```
assertStackIs(stk1, "33");
stk1 = stack("123");
stk1.dup();
assertStackIs(stk1, "1123");
```

Variables Window

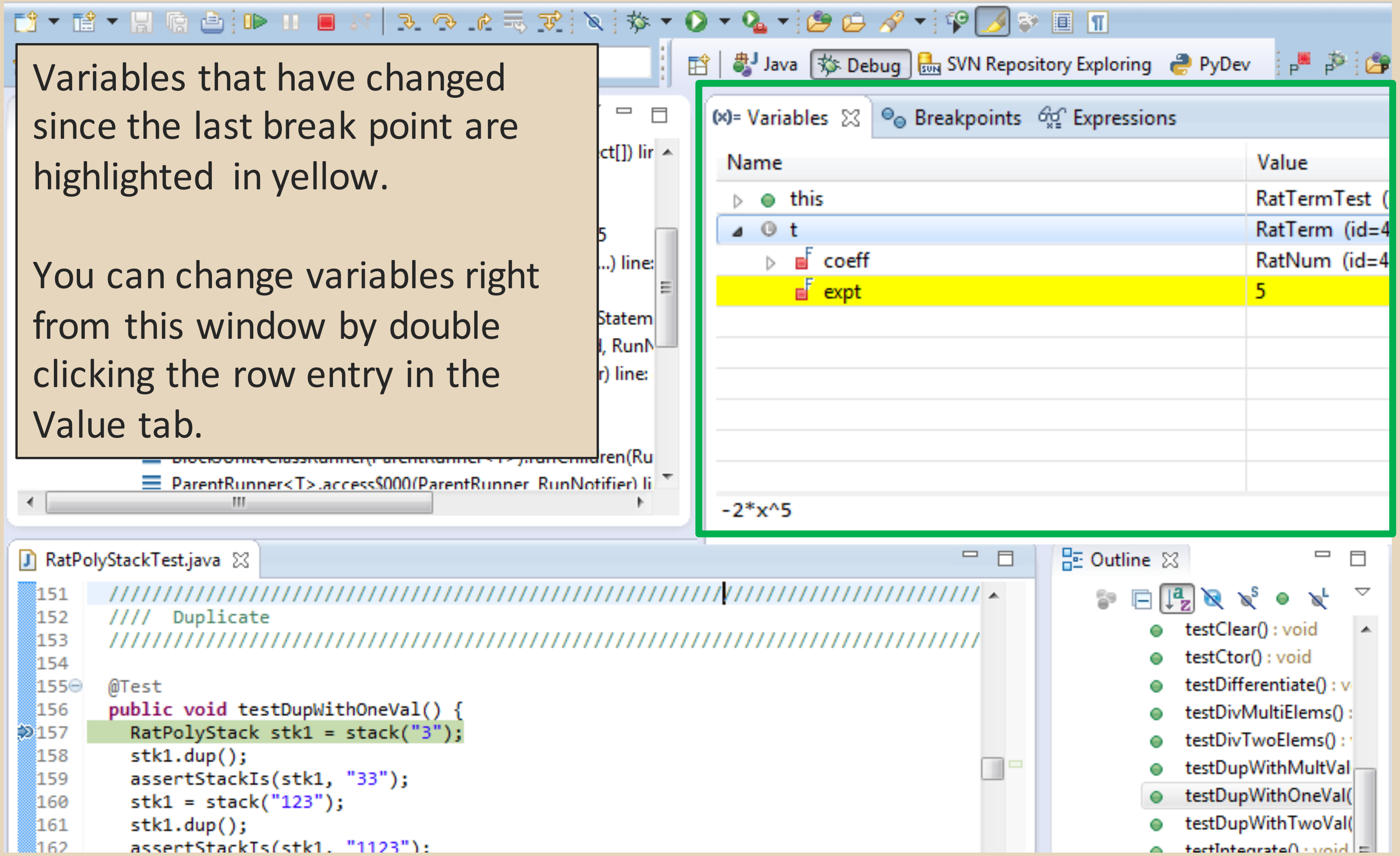
Shows all variables, including method parameters, local variables, and class variables, that are in scope at the current execution spot. Updates when you change positions in the stackframe. You can expand objects to see child member values. There's a simple value printed, but clicking on an item will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

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



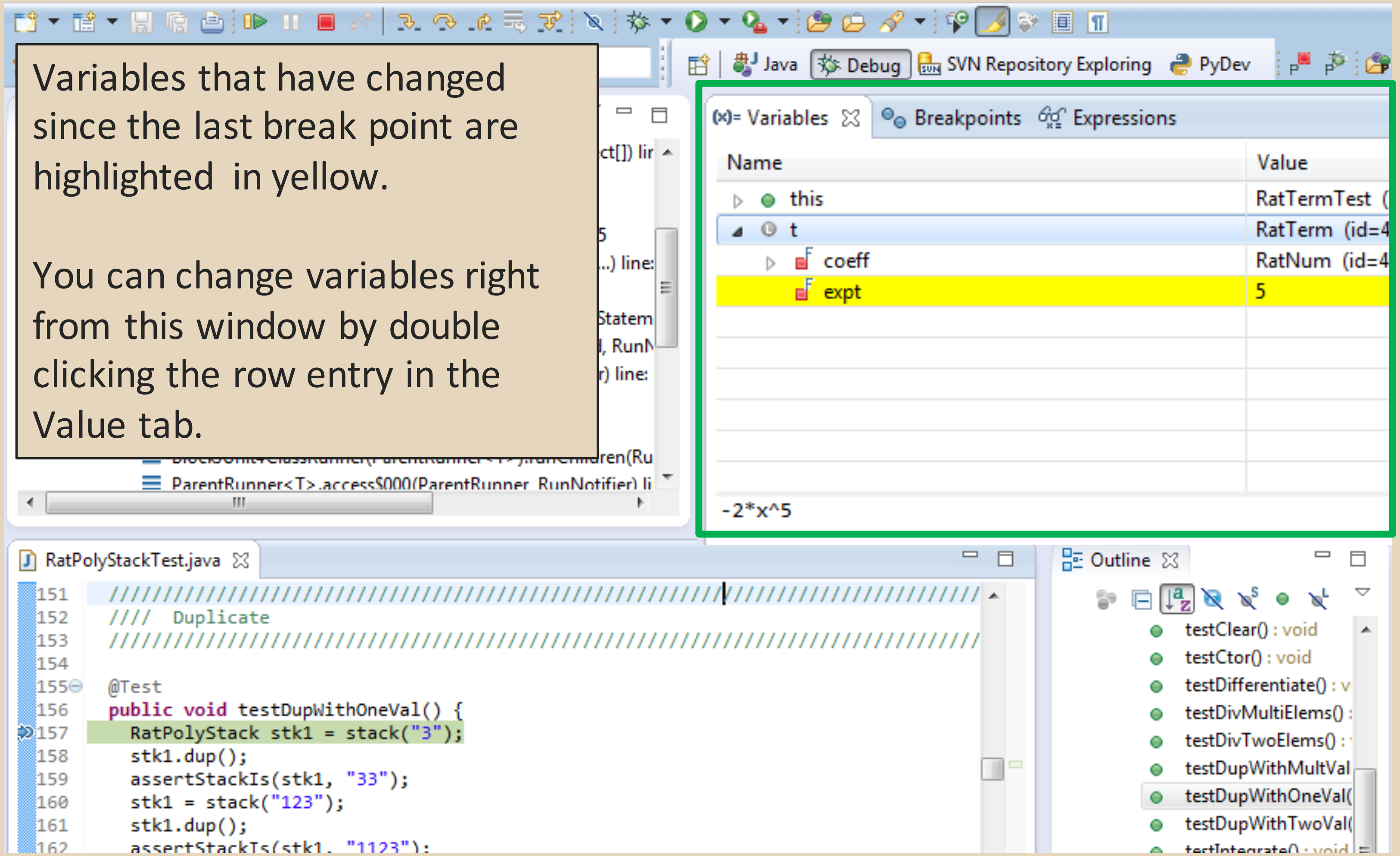
Name	Value
this	RatTermTest (
t	RatTerm (id=4
coeff	RatNum (id=4
expt	5

-2*x^5

```
/////////////////////////////////////////////////////////////////////////////
////  Duplicate
/////////////////////////////////////////////////////////////////////////////

@Test
public void testDupWithOneVal() {
  RatPolyStack stk1 = stack("3");
  stk1.dup();
  assertStackIs(stk1, "33");
  stk1 = stack("123");
  stk1.dup();
  assertStackIs(stk1, "1123");
```

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

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.

Name	Value
this	RatTermTest (
t	RatTerm (id=4
coeff	RatNum (id=4
expt	5

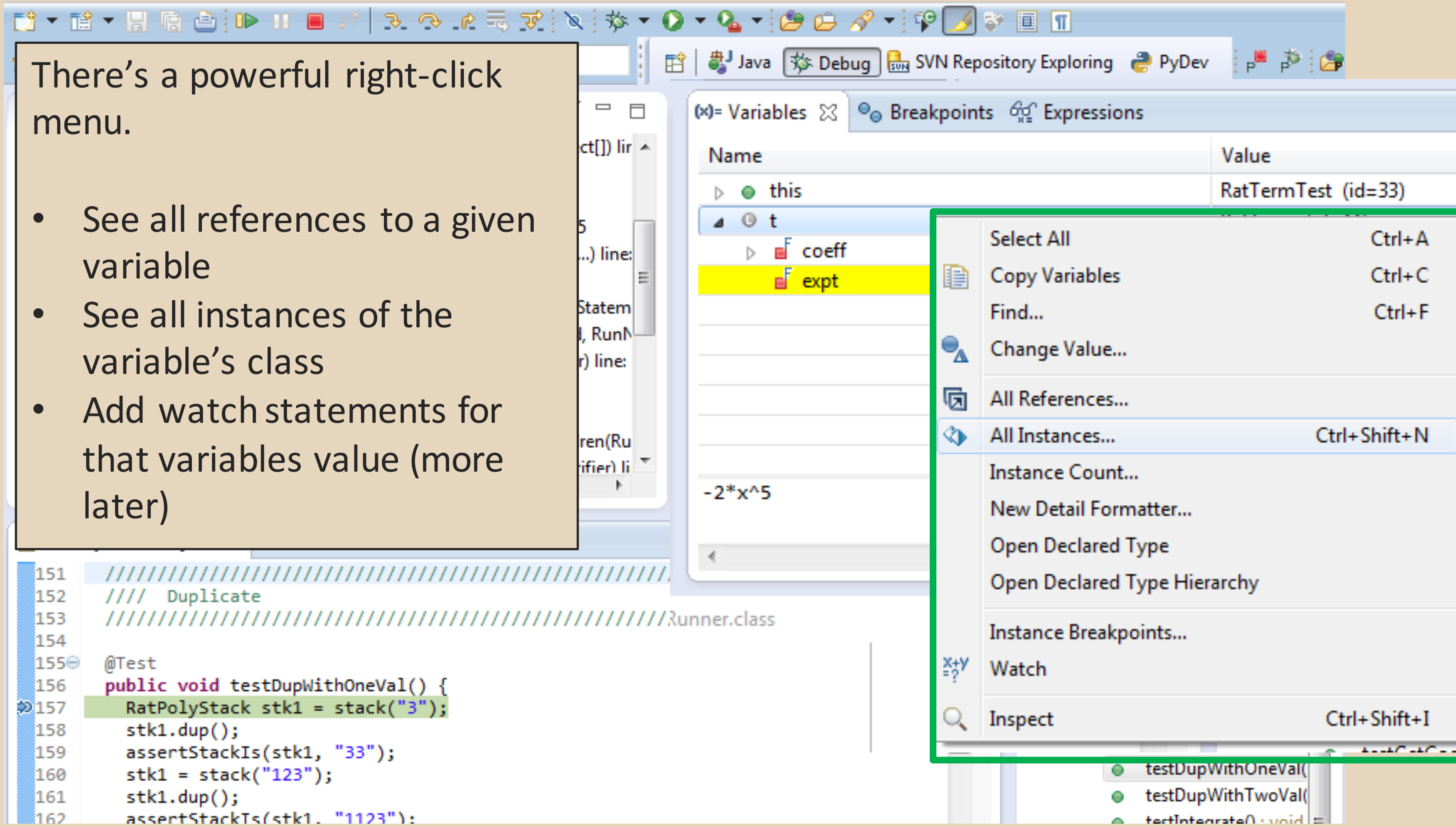
```
-2*x^5
```

```
//////////////////////////////////////////////////////////////////////////
////  Duplicate
//////////////////////////////////////////////////////////////////////////

@Test
public void testDupWithOneVal() {
  RatPolyStack stk1 = stack("3");
  stk1.dup();
  assertStackIs(stk1, "33");
  stk1 = stack("123");
  stk1.dup();
  assertStackIs(stk1, "1123");
```

ECLIPSE DEBUGGING

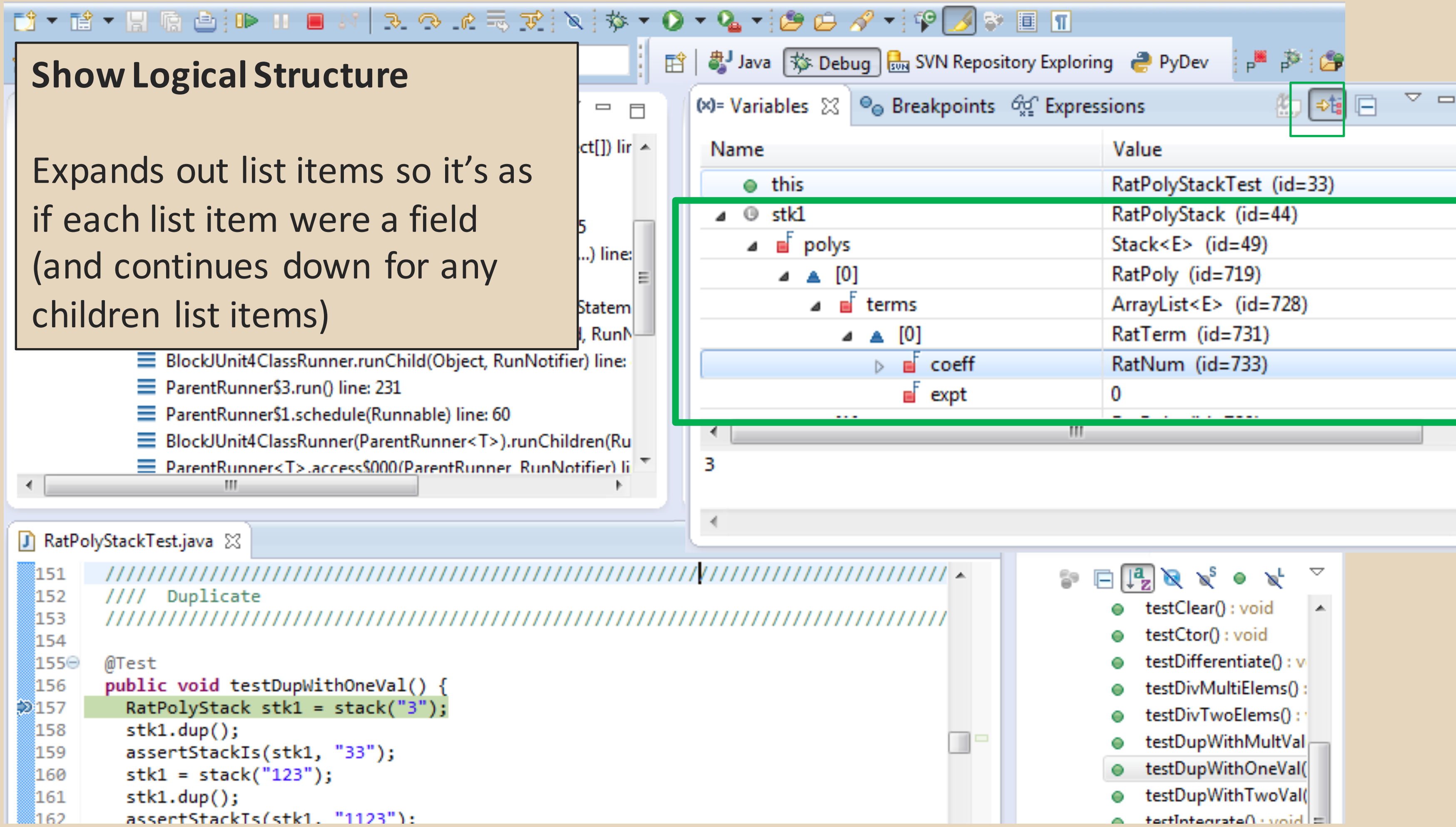
There's a powerful right-click menu.

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



```
    ////////////////////////////////////////////////////////////
    ////  Duplicate
    ////////////////////////////////////////////////////////////

    @Test
    public void testDupWithOneVal() {
      RatPolyStack stk1 = stack("3");
      stk1.dup();
      assertStackIs(stk1, "33");
      stk1 = stack("123");
      stk1.dup();
      assertStackIs(stk1, "1123");
```

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)

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

3

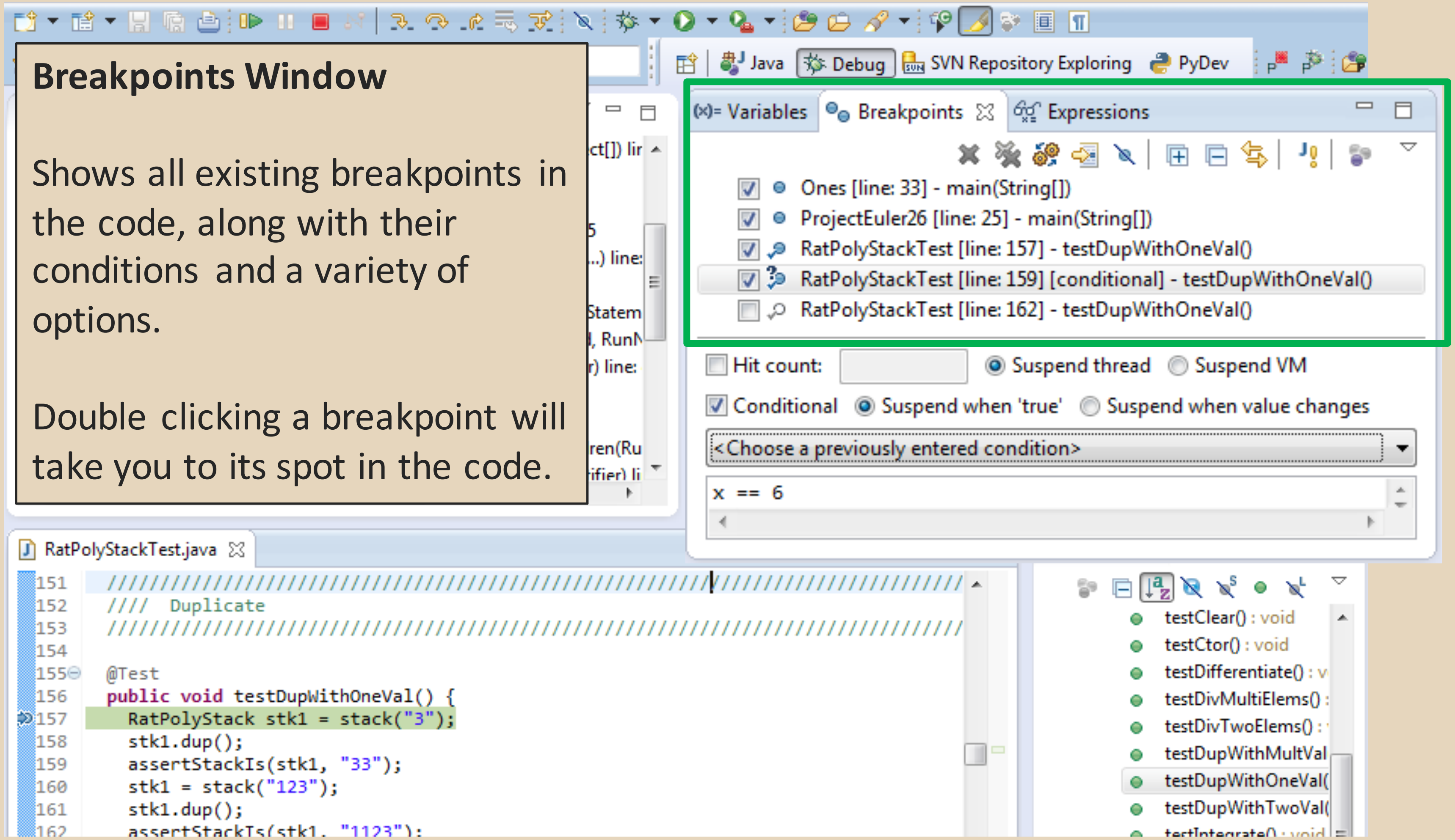
```
BlockJUnit4ClassRunner.runChild(Object, RunNotifier) line:
ParentRunner$3.run() line: 231
ParentRunner$1.schedule(Runnable) line: 60
BlockJUnit4ClassRunner(ParentRunner<T>).runChildren(Ru
ParentRunner<T>.access$000(ParentRunner, RunNotifier) li
```

RatPolyStackTest.java

```
//////////////////////////////////////////////////////////////////////
////  Duplicate
//////////////////////////////////////////////////////////////////////

@Test
public void testDupWithOneVal() {
  RatPolyStack stk1 = stack("3");
  stk1.dup();
  assertStackIs(stk1, "33");
  stk1 = stack("123");
  stk1.dup();
  assertStackIs(stk1, "1123");
```

```
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.

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

Hit count: [✓ Suspend thread] Suspend VM

[✓] Conditional [✓ Suspend when 'true'] Suspend when value changes

<Choose a previously entered condition>

```
x == 6
```

RatPolyStackTest.java

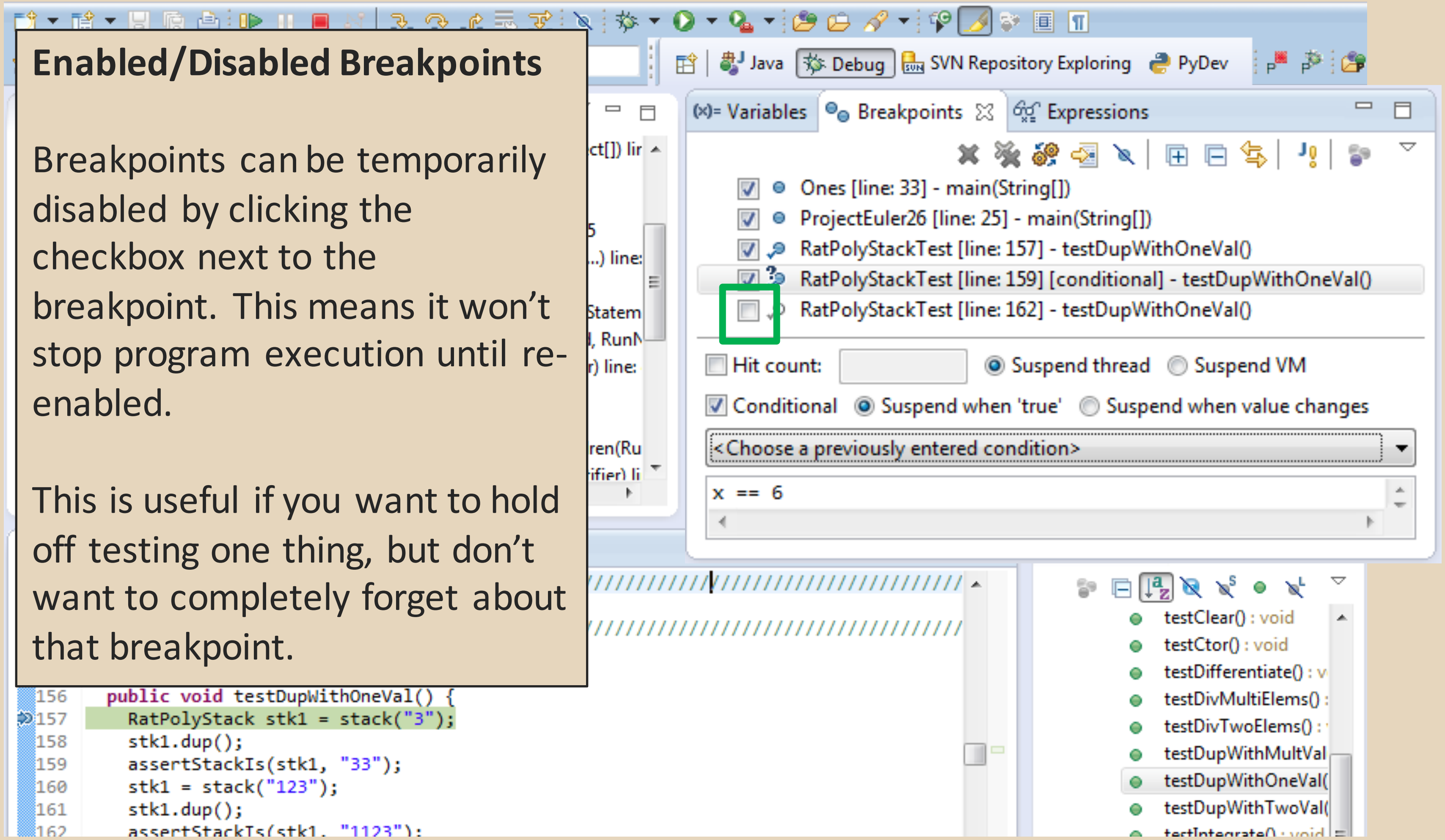
```
    ////////////////////////////////////////////////////////////////////////////////
    ////  Duplicate
    ////////////////////////////////////////////////////////////////////////////////

    @Test
    public void testDupWithOneVal() {
      RatPolyStack stk1 = stack("3");
      stk1.dup();
      assertStackIs(stk1, "33");
      stk1 = stack("123");
      stk1.dup();
      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.



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

[ ] Hit count: [✓ Suspend thread] Suspend VM

[✓] Conditional [✓ Suspend when 'true'] Suspend when value changes

<Choose a previously entered condition>

```
x == 6
```

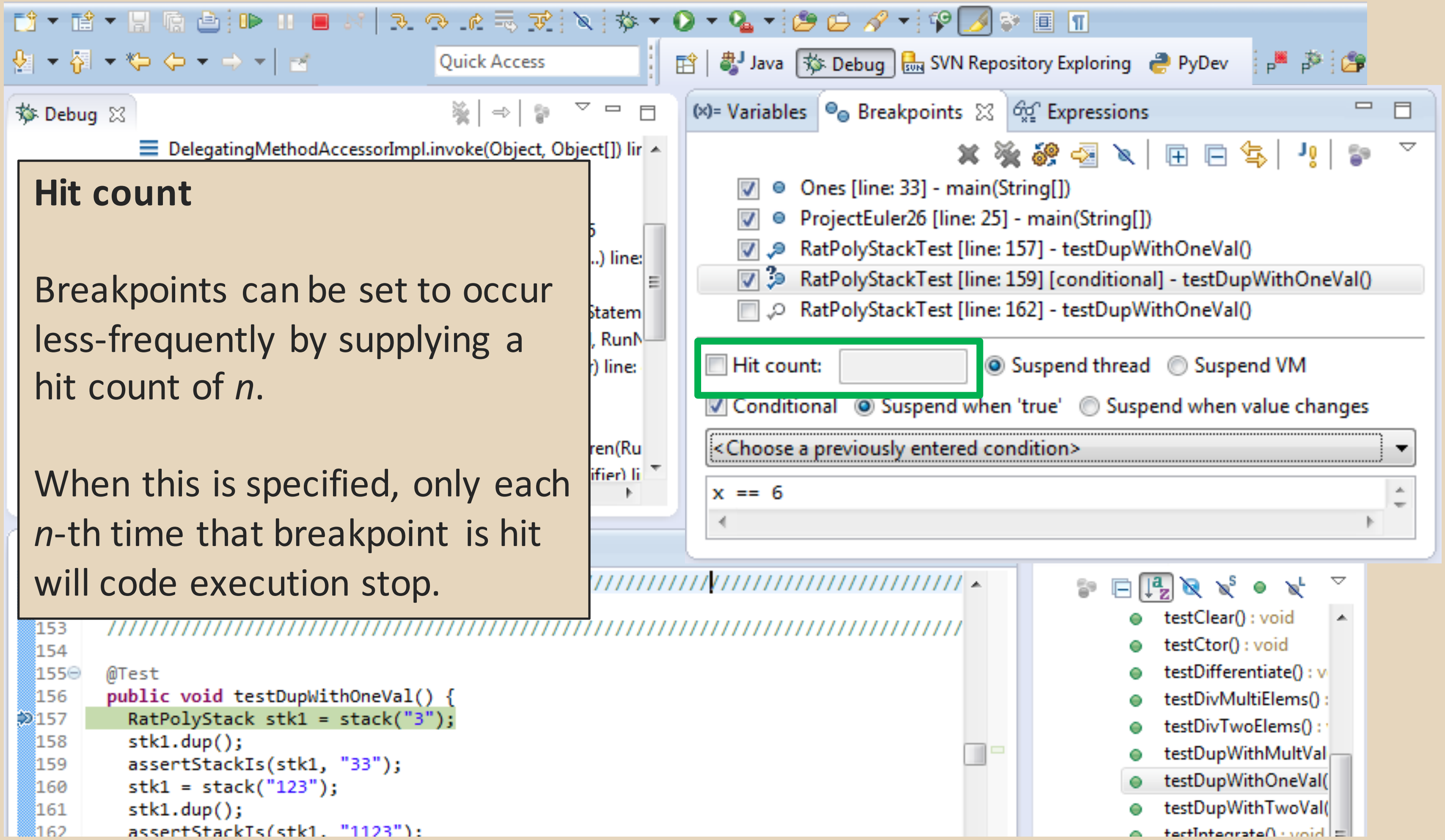
```
156   public void testDupWithOneVal() {
157     RatPolyStack stk1 = stack("3");
158     stk1.dup();
159     assertStackIs(stk1, "33");
160     stk1 = stack("123");
161     stk1.dup();
162     assertStackIs(stk1, "1123");
```

ECLIPSE DEBUGGING

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.



DelegatingMethodAccessorImpl.invoke(Object, Object[]) lir

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

[ ] Hit count: ____ [✓ Suspend thread] Suspend VM

[✓] Conditional [✓ Suspend when 'true'] Suspend when value changes

<Choose a previously entered condition>

```
x == 6
```

```
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");
```

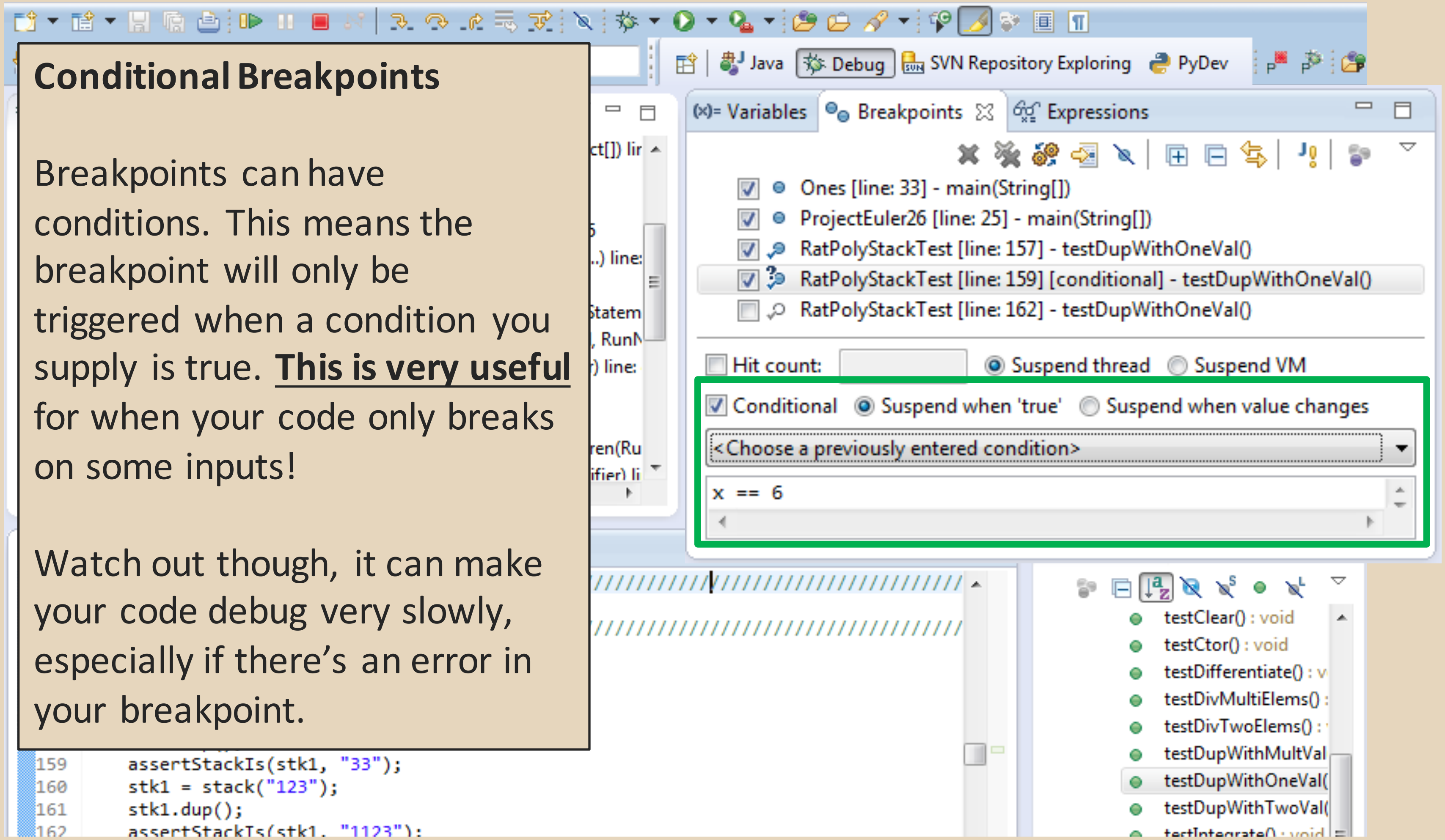
- testClear() : void
- testCtor() : void
- testDifferentiate() : v
- testDivMultiElems() :
- testDivTwoElems() :
- testDupWithMultVal
- testDupWithOneVal(
- testDupWithTwoVal(
- testIntegrate() : void

ECLIPSE DEBUGGING

Conditional Breakpoints

Breakpoints can have conditions. This means the breakpoint will only be triggered when a condition you supply is true. **<u>This is very useful</u>** 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.



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

[ ] Hit count: [✓ Suspend thread] Suspend VM

[✓] Conditional [✓ Suspend when 'true'] Suspend when value changes

<Choose a previously entered condition>

```
x == 6
```

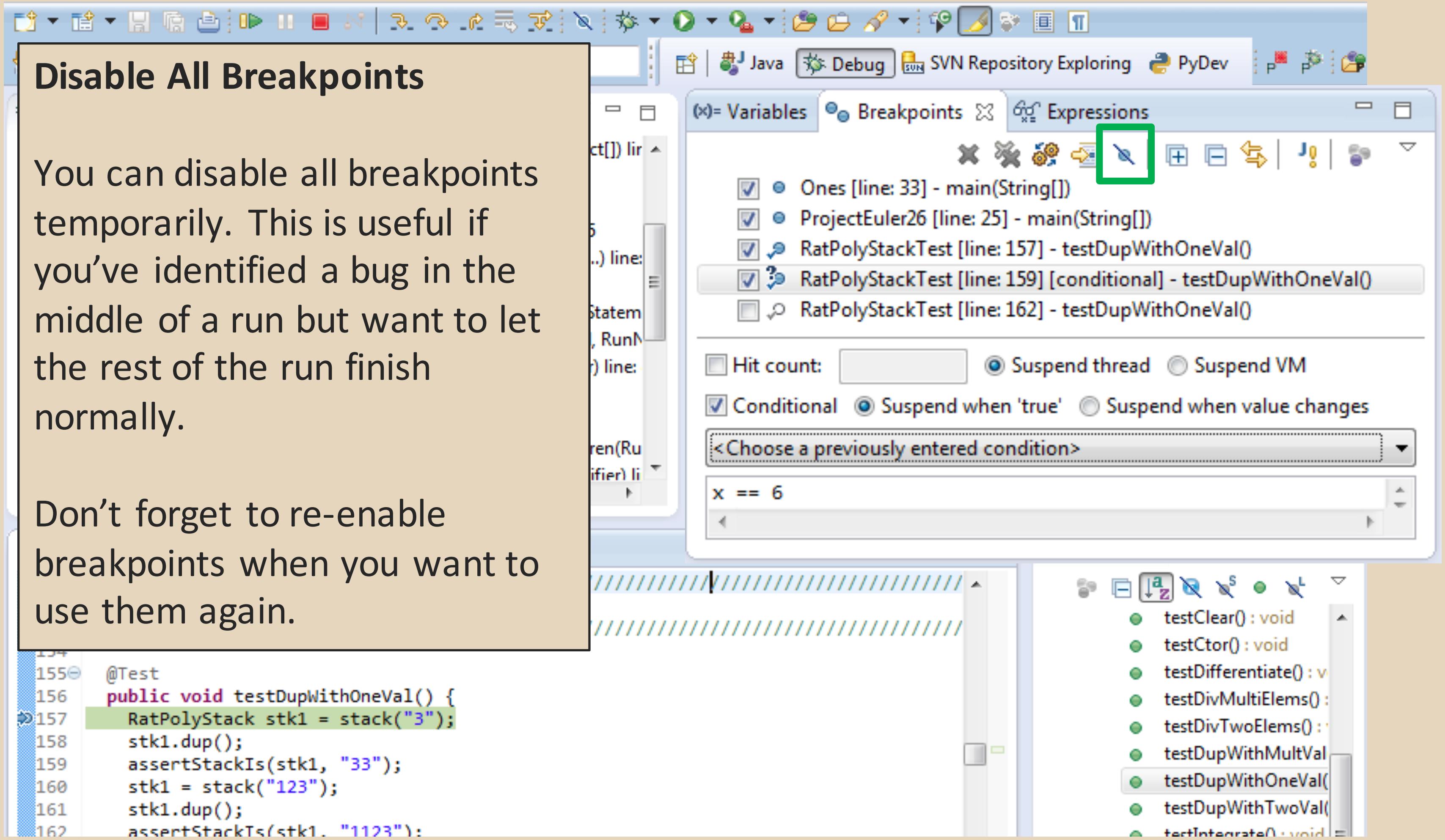
```
assertStackIs(stk1, "33");
stk1 = stack("123");
stk1.dup();
assertStackIs(stk1, "1123");
```

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.



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

[ ] Hit count: [✓ Suspend thread] Suspend VM

[✓] Conditional [✓ Suspend when 'true'] Suspend when value changes

<Choose a previously entered condition>

```
x == 6
```

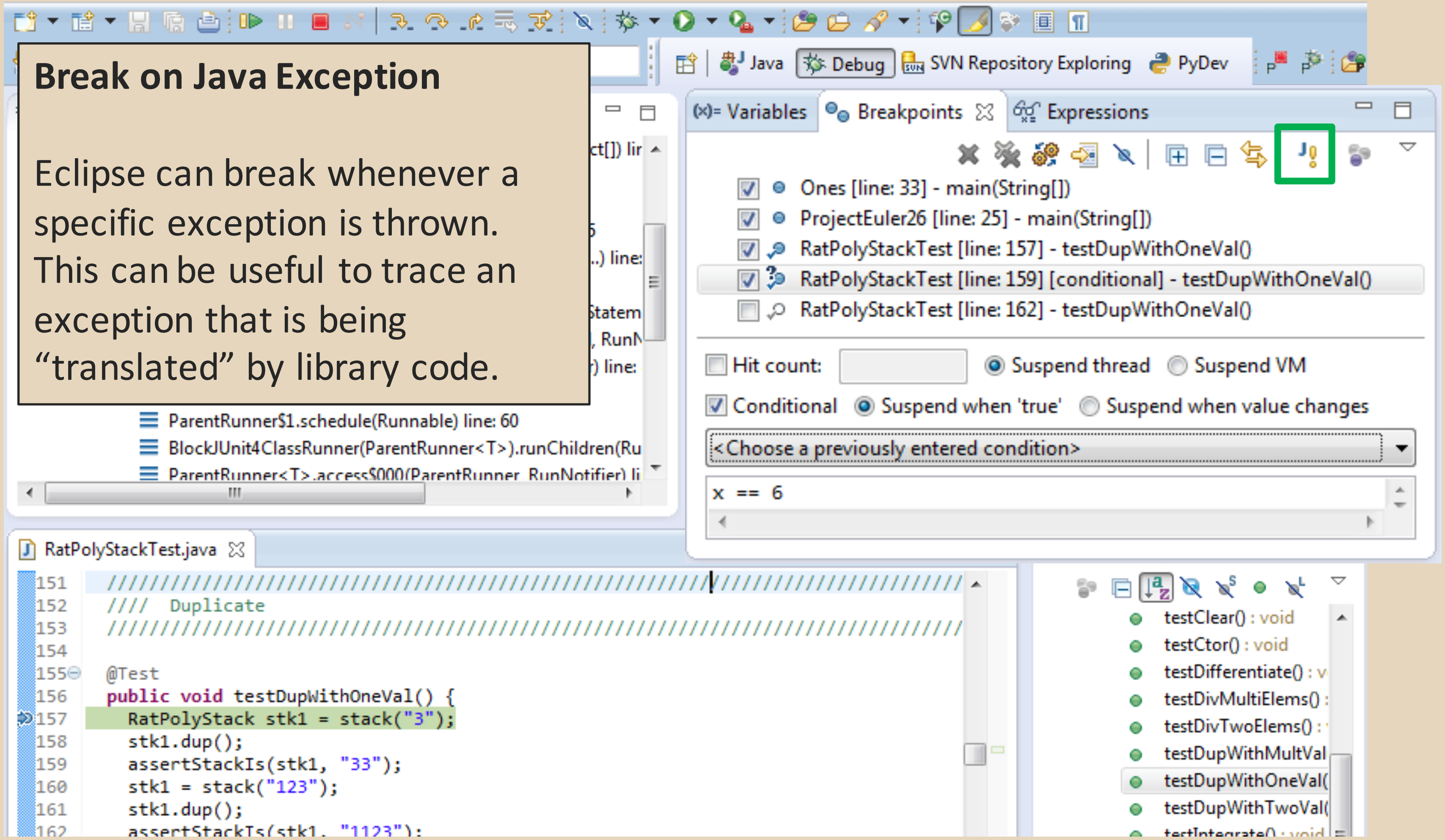
```
@Test
public void testDupWithOneVal() {
  RatPolyStack stk1 = stack("3");
  stk1.dup();
  assertStackIs(stk1, "33");
  stk1 = stack("123");
  stk1.dup();
  assertStackIs(stk1, "1123");
```

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

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.



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

[ ] Hit count: [✓ Suspend thread] Suspend VM

[✓] Conditional [✓ Suspend when 'true'] Suspend when value changes

<Choose a previously entered condition>

```
x == 6
```

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

```
////////////////////////////////////////////////////////////////////////////////
////  Duplicate
////////////////////////////////////////////////////////////////////////////////

@Test
public void testDupWithOneVal() {
  RatPolyStack stk1 = stack("3");
  stk1.dup();
  assertStackIs(stk1, "33");
  stk1 = stack("123");
  stk1.dup();
  assertStackIs(stk1, "1123");
```

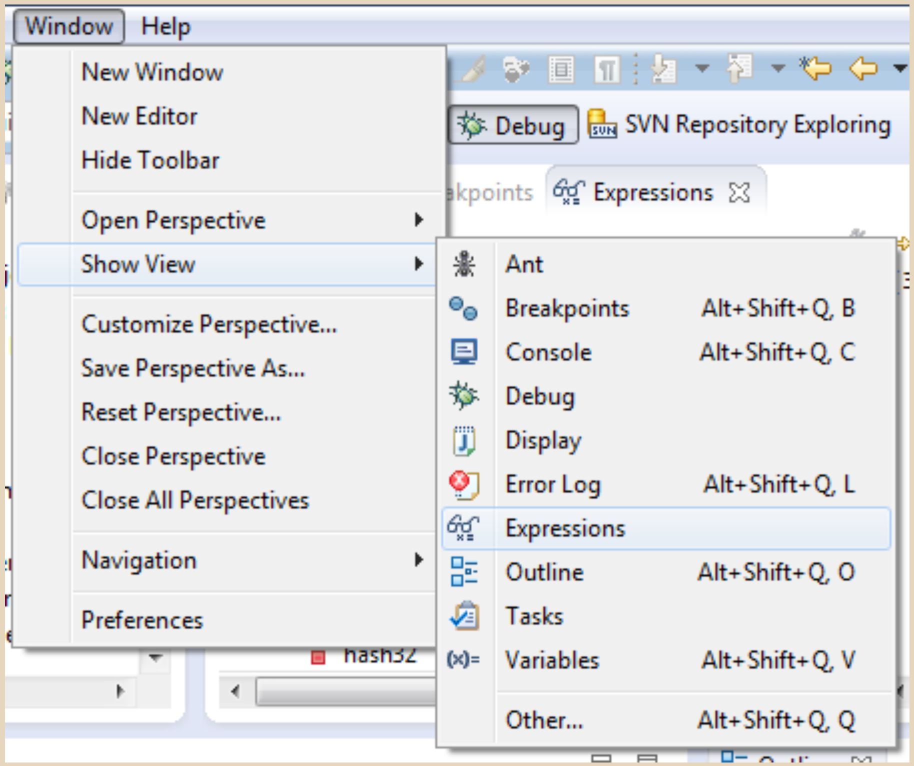
testClear() : void
testCtor() : void
testDifferentiate() : v
testDivMultiElems() :
testDivTwoElems() :
testDupWithMultVal
testDupWithOneVal(
testDupWithTwoVal(
testIntegrate() : void

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.



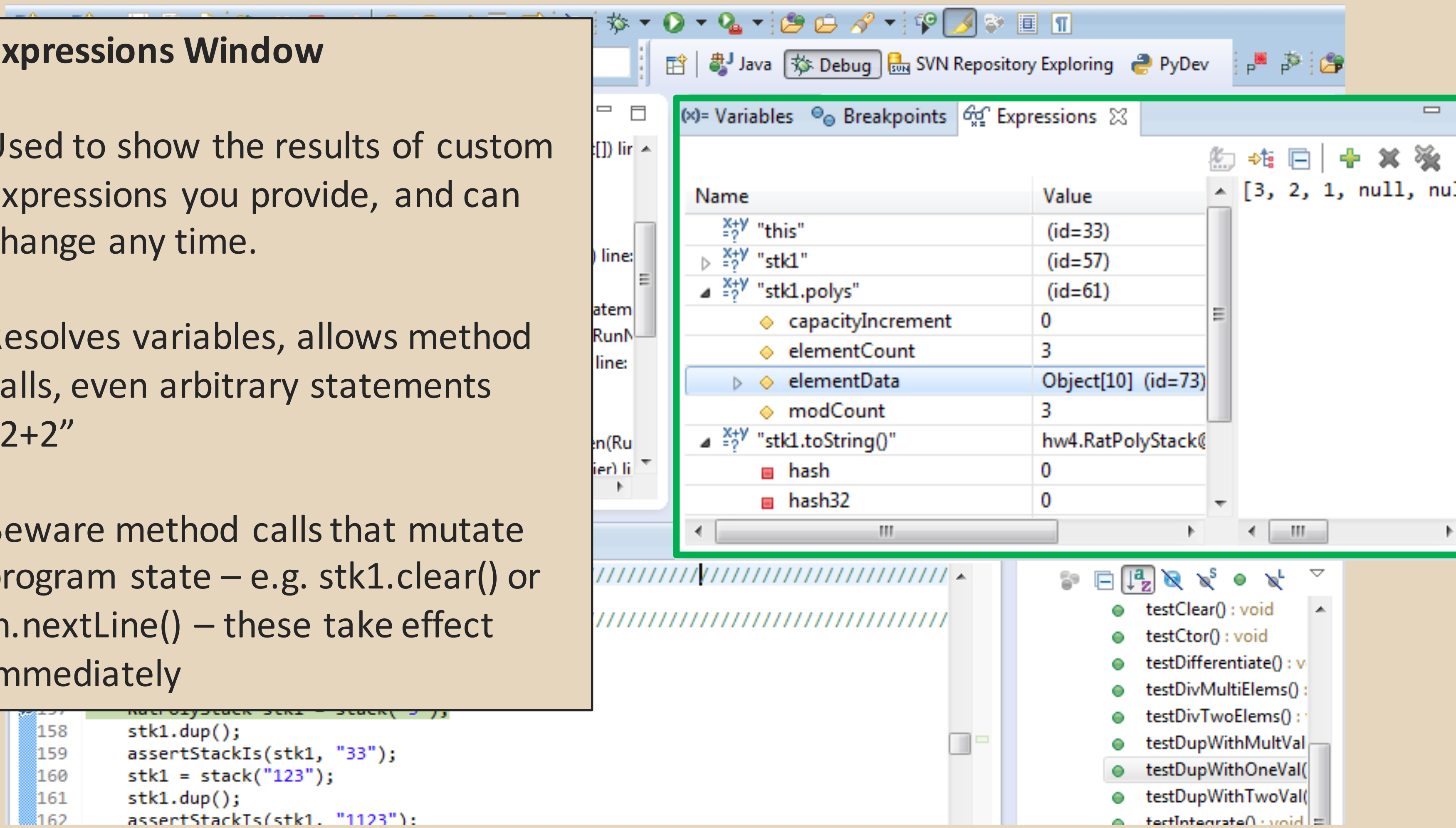
ECLIPSE DEBUGGING

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



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

```
[3, 2, 1, null, nu
```

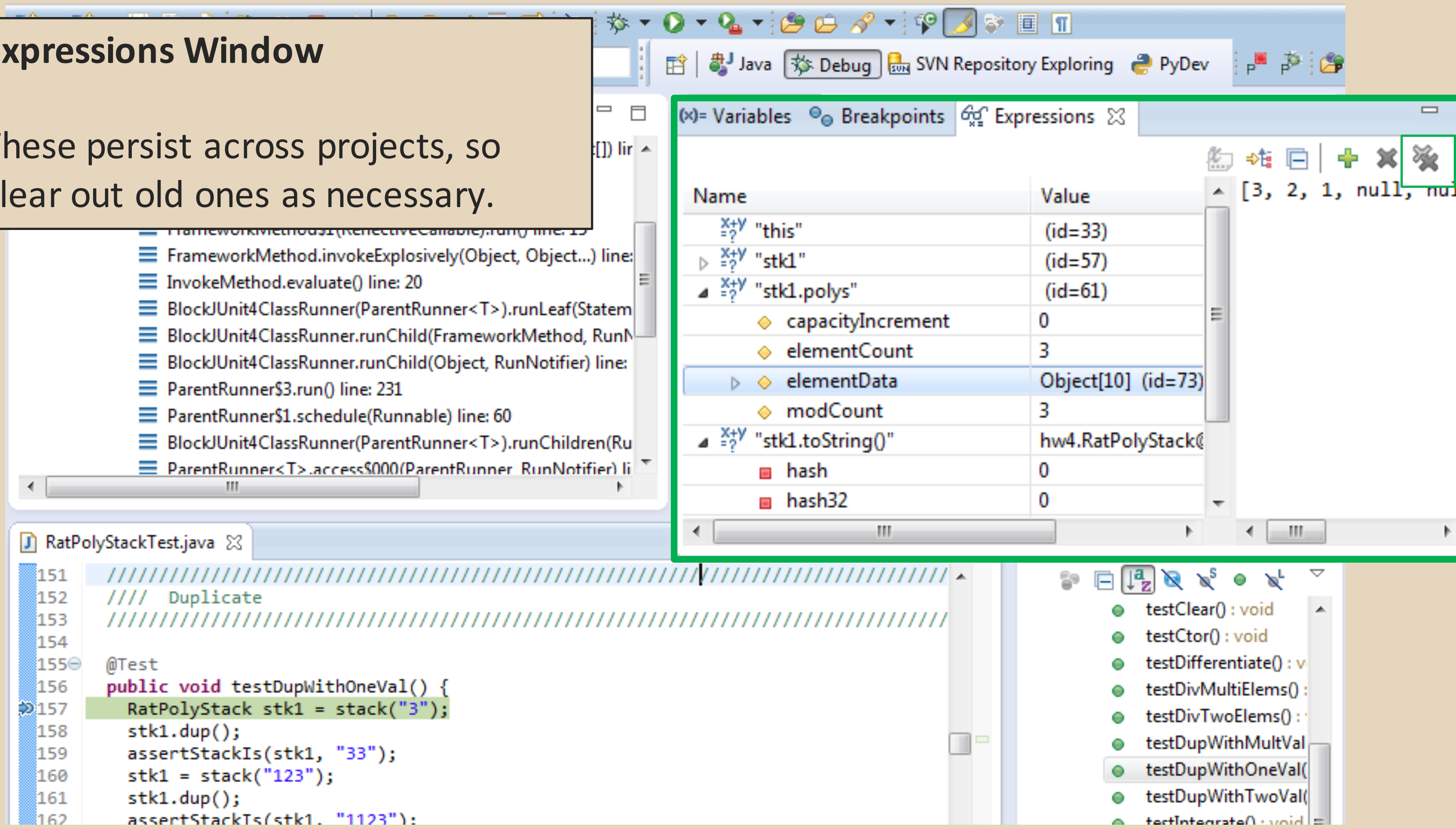
```
        RatPolyStack stk1 = stack("3");
        stk1.dup();
        assertStackIs(stk1, "33");
        stk1 = stack("123");
        stk1.dup();
        assertStackIs(stk1, "1123");
```

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

ECLIPSE DEBUGGING

Expressions Window

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



```
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
```

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

```
[3, 2, 1, null, nu
```

```
//////////////////////////////////////////////////////////////////////////////
////  Duplicate
//////////////////////////////////////////////////////////////////////////////

@Test
public void testDupWithOneVal() {
  RatPolyStack stk1 = stack("3");
  stk1.dup();
  assertStackIs(stk1, "33");
  stk1 = stack("123");
  stk1.dup();
  assertStackIs(stk1, "1123");
```

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

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