Recitation 3. Debugging. Start on Exceptions

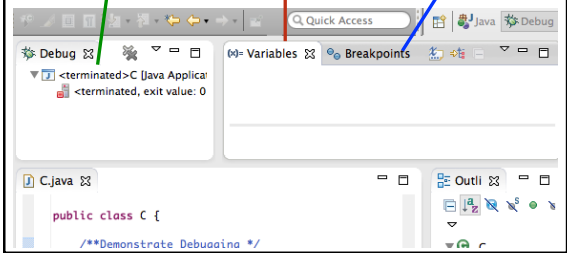
Debug perspective: Click on this icon.
In pop-window, select
Debug. Debug perspective opens (next slide)

Debug perspective

Use pane to see breakpoints (next slide)

Use pane to see variables/values

Debug pane –see later

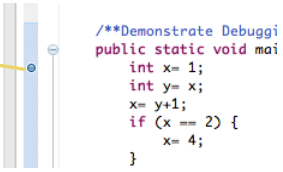


Setting a breakpoint

Breakpoint: a line of program that is set so that execution will stop when it is reached during debugging.

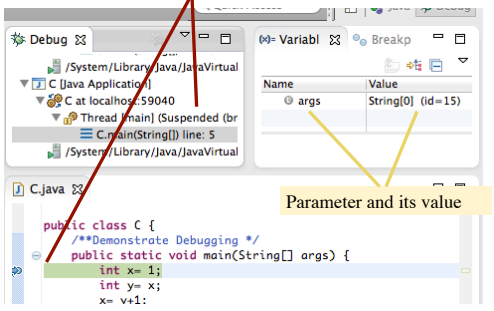
Put mouse here,
right click,
select Toggle Breakpoint
from contextual menu.
The breakpoint is set and
the dot appears.

If it was already set, it is unset.



Running the debugger

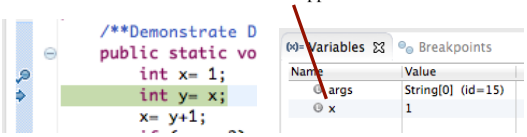
Choose **Run > Debug.** Execution starts but stops at the first line with a breakpoint set



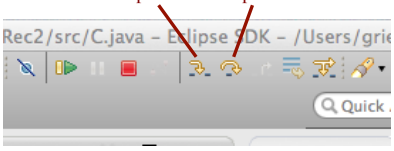
Click icon Step Over to have statement executed.

Statement has been executed

x appears with its value



Step Into Step Over



Step into and **Step over** do the same thing except for executing a call.

Step over executes a call in one step.

Step into makes the first statement in body of method being called the next statement to execute, so you can see detailed execution of method body.

Recitation 3. Exceptions

```
public static void main(String[] args) {
    int b= 3/0;
}
```
This is line 7

Division by 0 causes an "Exception to be thrown". program stops with output:

```
Exception in thread "main"
        java.lang.ArithmeticException: / by zero
    at C.main(C.java:7)
```

Happened in main, line 7

The "Exception" that is "thrown"

parseInt throws a NumberFormatException

```
/** Parse s as a signed decimal integer and return
    the integer. If s does not contain a signed decimal
    integer, throw a NumberFormatException. */
public static int parseInt(String s)
```

parseInt, when it find an error, does not know what caused the error and hence cannot do anything intelligent about it. So it "throws the exception" to the calling method. The normal execution sequence stops! *See next slide*

```
public static void main(String[] args) {
    … code to store a string in s —expected to be an int
    int b= Integer.parseInt(s);
}
```

parseInt throws a NumberFormatException

```
public static void main(String[] args) {
    int b= Integer.parseInt("3.2");
}
```
3.2 not an int

Output is:

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "3.2"
  at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
  at java.lang.Integer.parseInt(Integer.java:458)
  at java.lang.Integer.parseInt(Integer.java:499)
  at C.main(C.java:6)
```

called from C.main, line 6

called from line 499

Found error on line 458

We see stack of calls that are not completed!

Exceptions and Errors

In Java, there is a class Throwable:

Throwable@x1

detailMessage "/ by zero"

getMessage()
Throwable()
Throwable(String)

When some kind of error occurs, an Exception is "thrown" —you'll see what this means later.

An Exception is an instance of class Throwable

(or one of its subclasses)

Two constructors in class Throwable. Second one stores its String parameter in field detailMessage.

Exceptions and Errors

So many different kind of exceptions that we have to organize them.

Do nothing with these

Throwable@x1

Throwable() Throwable(String)

detailMessage "/ by zero"
getMessage()

Exception

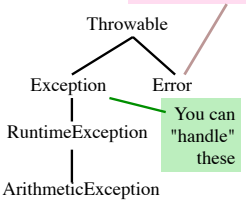
Exc…() Exc…(..)

RuntimeException

RunTimeE…() RunTimeE…(…)

ArithmeticException

Arith…E…() Arith…E…(…)



Subclass: 2 constructors, no other methods, no fields. Constructor calls superclass constructor

Creating and throwing and Exception

Class:

Call

Output

```
Ex.main();
ArithmeticException: / by zero
 at Ex.third(Ex.java:13)
 at Ex.second(Ex.java:9)
 at Ex.main(Ex.java:5)
 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
 at sun.reflect.NativeMethodAccessorImpl.invoke(…)
 at sun.reflect.DelegatingMethodAccessorImpl.invoke(…)
 at java.lang.reflect.Method.invoke(Method.java:585)
```

```
03 public class Ex {
04     public static void main(…) {
05         second();
06     }
07
08     public static void second() {
09         third();
10     }
11
12     public static void third() {
13         int x= 5 / 0;
14     }
15 }
```

a0 AE

a0 AE

a0 AE

throw statement

Class:

Call

Output

```
Ex.main();
ArithmeticException: I threw it
 at Ex.third(Ex.java:14)
 at Ex.second(Ex.java:9)
 at Ex.main(Ex.java:5)
 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
 at sun.reflect.NativeMethodAccessorImpl.invoke(…)
 at sun.reflect.DelegatingMethodAccessorImpl.invoke(…)
 at java.lang.reflect.Method.invoke(Method.java:585)
```

```
public class Ex {
  public static void main(…) {
    second();
  }
  public static void second() {
    third();
  }
  public static void third() {
    throw new ArithmeticException("I threw it");
  }
}
```

a0 AE

a0 AE

a0 AE

How to write an exception class

```
/** An instance is an exception */
public class OurException extends Exception {

   /** Constructor: an instance with message m*/
   public OurException(String m) {
      super(m);
   }

   /** Constructor: an instance with no message */
   public OurException() {
      super();
   }
}
```

Won't compile. Needs a "throws clause, see next slide

If a method throws an Exception that is not a subclass of RuntimeException, the method needs a throws clause.

Don't be concerned with this issue. Just write your method and, if Java says it needs a throws clause, put one in

```
/** Illustrate exception handling */
public class Ex {
   public static void main() {
      second();
   }
   public static void second() {
     third();
   }
   public static void third() {
      throw new
         OurException("mine");
   }
}
```

The "throws" clause

```
/** Class to illustrate exception handling */
public class Ex {
   public static void main() throws OurException {
      second();
   }
   public static void second() throws OurException {
      third();
   }
   public static void third() throws OurException {
      throw new OurException("mine");
   }
```

If Java asks for it, insert the throws clause. Otherwise, don't be concerned with it.

Try statement: catching a thrown exception

```
public class Ex1 {
   public static void main() throws MyExeception{
      try {
         second();
      }
      catch (MyException ae) {
         System.out.println
           ("Caught MyException: " + ae);
      }
      System.out.println
           ("procedure first is done");
   }
   public static void second() throws MyException {
      third();
   }
   public static void third() throws MyException {
      throw new MyException("yours");
   }
}
```

Execute the try-block. If it finishes without throwing anything, fine.

If it throws a MyException object, catch it (execute the catch block); else throw it out further.

```
/** Input line supposed to contain an int. (whitespace on either side OK).
    Read line, return the int. If line doesn't contain int, ask user again
   */
public static int readLineInt() {
     String input= readString().trim();
     // inv: input contains last input line read; previous
     // lines did not contain a recognizable integer.
     while (true) {
        try {
           return Integer.valueOf(input).intValue();
        }
        catch (NumberFormatException e) {
           System.out.println( "Input not int. Must be an int like");
           System.out.println("43 or -20. Try again: enter an int:");
           input= readString().trim();
        }
     }
  }
```

Useful example of catching thrown object

readLineInt continues to read a line from keyboard until user types and integer