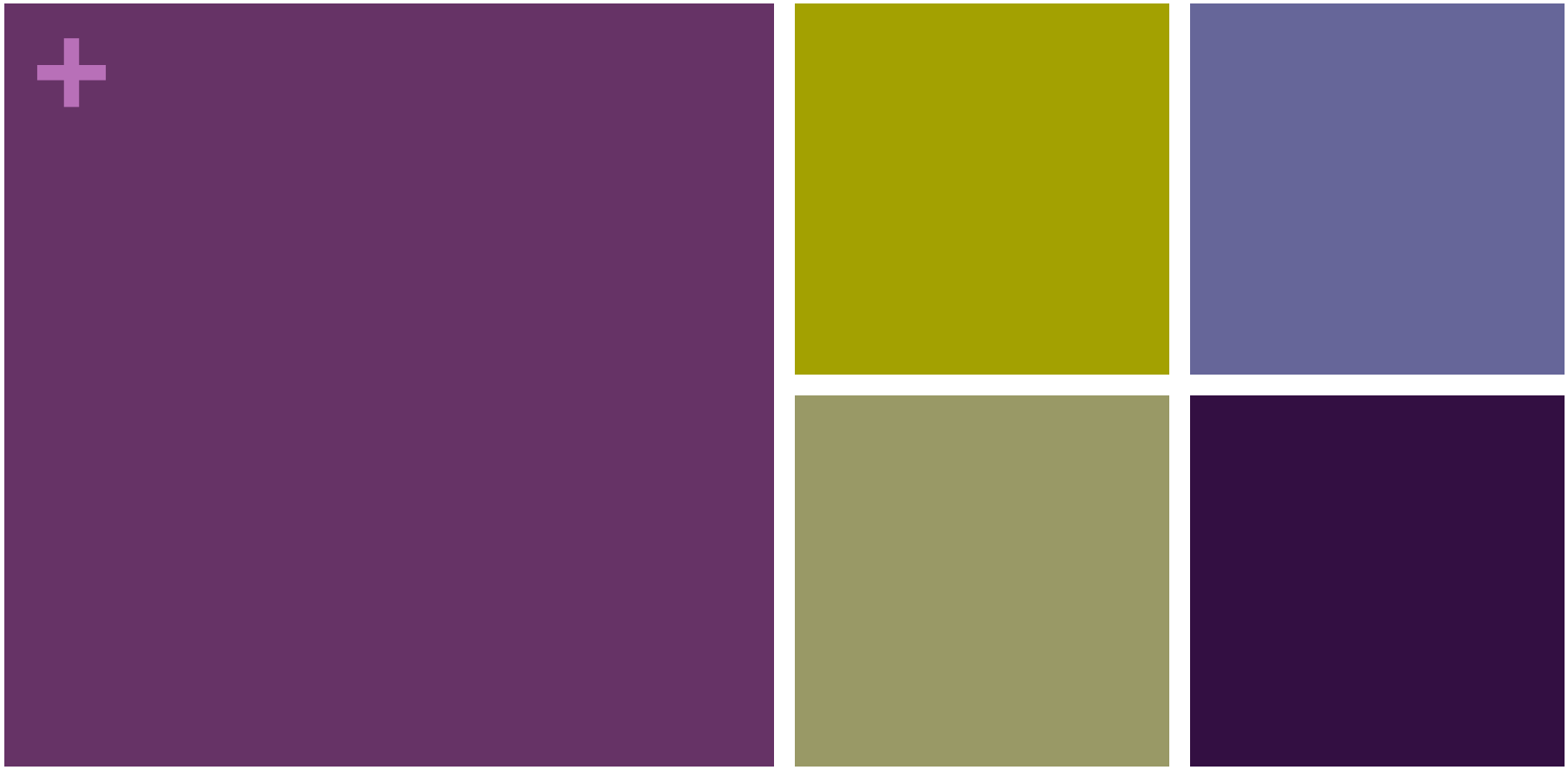


Debugging with Eclipse

Thanks to Carly Orr

Debugging

- **Benefits** of using a source code debugger
- **Basic services** of a source code debugger
- **Terminology** used with source code debuggers
- How to **step** **through** a program
- When and how to use a **breakpoint**
- **Look at variables** as a program is executing
- **Look at the call stack** for a program

Run-Time Errors – Common Types

Null pointer exception

is there a method call made to an object that is undefined?

```
24      publc class OperateCar() {
25        private Car speedy;
26        public moveForward() { System.out.println ("vroom!");}
27        public void drive() {
28          speedy.moveForward();
29        }
30      }
```

Array index out of bounds exception

is the index negative, or out of range?

```
13      Car[] listOfCars = new Car[5];
14      for (int i=0; i<=5; i++){
15        listOfCars[i].drive();
16      }
```

Stack overflow?

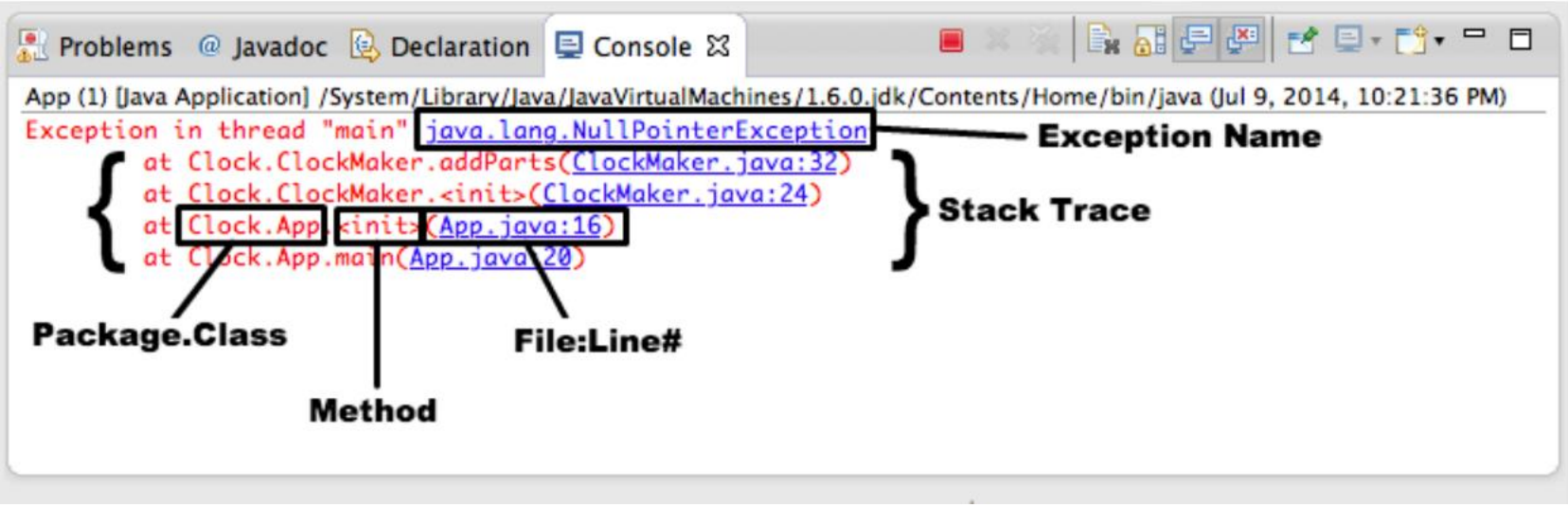
is there a method you are calling infinite number of times?

```
19      Boolean alwaysTrue = true;
20      while (alwaysTrue) {
21          myCar.drive();
22      }
```

Google: "java runtime exceptions" (unchecked)

Traceback AKA Stack Trace

- In Java, a runtime error results in a traceback
- Traceback tells where the exception occurred
  - Each line gives the statement that called the former line
- Look for code that you wrote to find the error
  - Often the first line that references your code is the culprit



Defensive Programming

- Write robust programs
  - Think of situations that might happen, check for them.

    Examples: files that don't exist, bad input data
- Generate appropriate error messages, allow the user to re-enter the data or exit from the program
- Throw exceptions – covered after midterm
  - Helps to find errors or avoid errors
  - Example: invalid arguments (IllegalArgumentException)

```
//A void method
public void sample()
{
   //Statements
   //if (somethingWrong) then
   IOException e = new IOException();
   throw e;
   //More Statements
 }
```

```
MyClass obj =  new MyClass();
try{
      obj.sample();
}catch(IOException ioe)
 {
      //Your error Message here
      System.out.println(ioe);
  }
```

Debug: Traditional Techniques

Strategically placing print statements to show control flow and values of key variables

To narrow down problem area.

Shotgun approach.

```
01   System.out.println(x >= 0);
02   System.out.println(y == 7);
03   System.out.println(z < x + y);
04   if (x >= 0 && y == 7 || z < x + y)
05   { //...
06   }
```

Run IDE in Debug Mode

Two development modes in Eclipse:

1. **Run mode** (compile and execute as if using command-line)
2. **Debug mode** (Eclipse controls execution of program)

- Debugger sends compiled instructions to the CPU one at a time:
  - Fetch next instruction
  - Decode next instruction
  - Send instruction to CPU for execution
- Debugger needs access to:
  - Source code, and Symbol information (Language construct info)
  - If missing, it skips over this part when stepping

Breakpoint

- Run up to some point, and stop.
- The **breakpoint is persistent** in the file. **Every time the debugger reaches the breakpoint it stops**
  - Before the line with the breakpoint
- Breakpoint shown at the “left margin” (beside line number)
- Can have many breakpoints.
- When debugger stops at a breakpoint you can examine variables, etc.

Stepping through code

- After you stop (at a breakpoint), how do you continue?
- Some common definitions:

Run	Start executing from current point. • run until program ends or a breakpoint is reached.
Step Into	If next instruction is a function call, step into it. If next instruction is not a function, just execute it.
Step Over	If next instruction is a function call, step over it. If next instruction is not a function, just execute it.
Step out of	Step out of the current function. Execute until the end of the current function, stop after caller line.

More on the Call Stack

- A *stack* is a data structure that has 2 operations
  - Push – put something on top
  - Pop – take something off of the top
- Last data stored is first data out (called LIFO)
- When a program calls a method the current position data is pushed onto a call stack
  - Allows program flow to return to the caller when call is done
- Debug mode lets us see the call stack.
- Call stack is typically displayed as separate window/tab.

Examining variables

The most important thing you want to do when debugging is:

- **Stop execution** (either by setting a breakpoint, or stepping)
- **Look at the values** stored in variable

There are usually a few different ways to see values in variables

- Using your mouse, hover over the variable in the code window
- Look in the variables window
- For object or data structures, you may need to un-twirl

Inspecting expressions

Another valuable capability is to be able to see what a compound expression evaluates to

Eg: what is (a + b || c && !d < f)

Or maybe you want to see what the result of a function call will be – without stepping into it

Use the expressions (or console) view

It is a window that lets you type expressions to be evaluated – relative to the current position in the program

Get to this window in Eclipse with

Window >> Show View >> Expressions

Watchpoints

- ***Watchpoints*** triggers **on variable**, not line in code
  - Debugger stops when a variable is read or changed
- Double-click on the left of a *class-level* variable declaration
- Right-click on the *watchpoint* icon beside the variable and select Breakpoint Properties….
  - Defines if break happens during read access (Field Access)
  - or during write access (Field Modification)

Using "HIT" count

- Hit count can be set for any breakpoint
  - Right click on breakpoint icon and select breakpoint properties

- If you set **Hit Count** to ***n***, the debugger will stop the ***n-th*** time the breakpoint is reached

Change variables

- A very useful capability of most debuggers is the ability to change values on the fly
- How?
  - Set a breakpoint
  - Run or step to the breakpoint
  - In the variables window
    - Right click on the variable you want to change, select change value

Tips: Where to add breakpoint or print statement?

1. Just before the line with error
2. Beginning of methods
3. At the end of methods
4. Inside loops
5. After initializing variables

Tips:

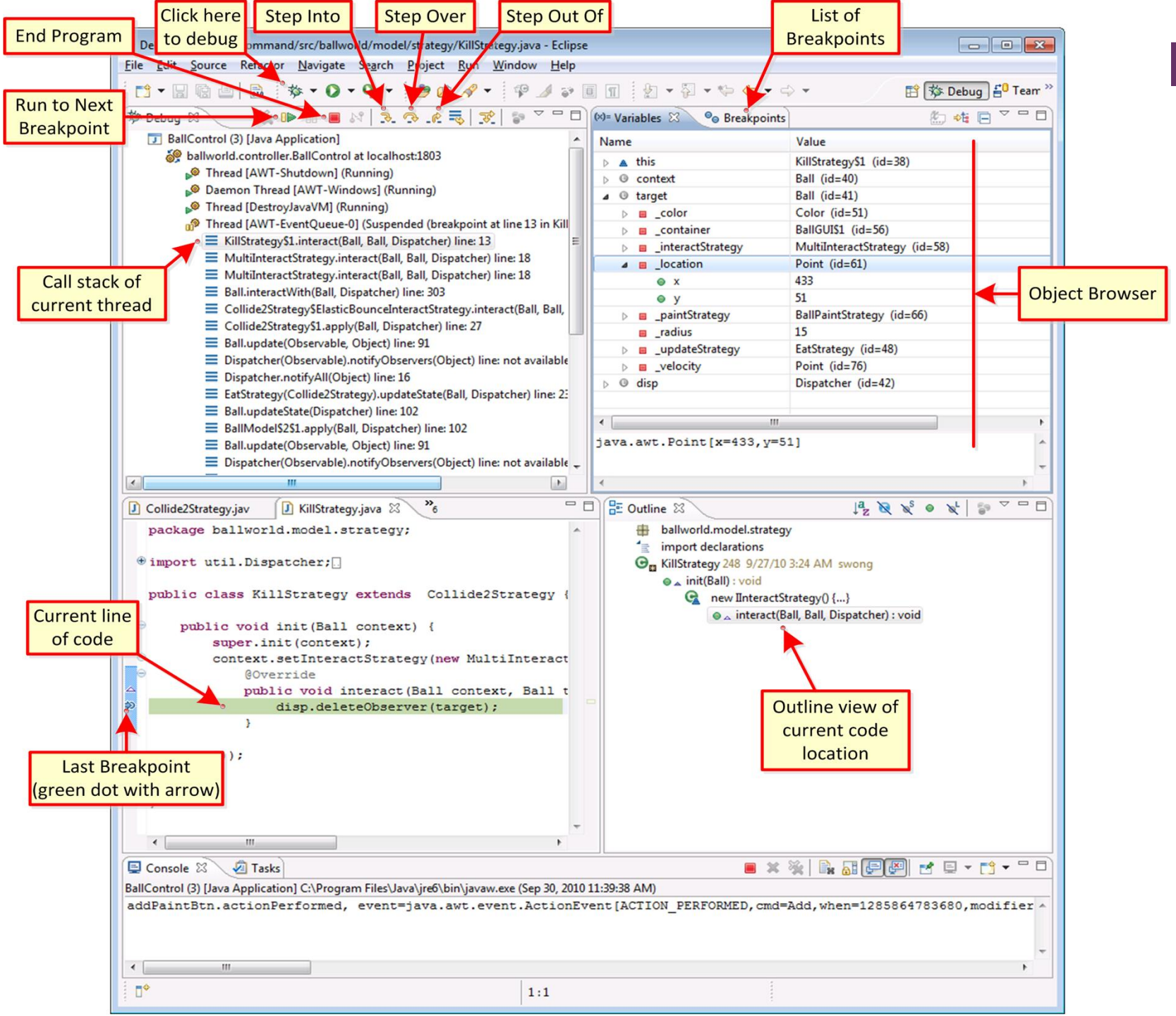
- Turn on Line Numbers:

  Go to Windows → Preferences → General → Text Editors → **Show** Line Numbers

- After you fix your code, don't forget to "terminate" the current Debug Run, so you can start Debug from the beginning again.

- You can flip back and forth between Perspectives: "Java Browsing" and "Debug".

- Eclipse will break if an exception is thrown
  - If not check Preferences>Java>Debug>Suspend execution on uncaught exceptions


End Program
Click here to debug
Step Into
Step Over
Step Out Of
List of Breakpoints
Run to Next Breakpoint
Call stack of current thread
Object Browser
Current line of code
Last Breakpoint (green dot with arrow)
Outline view of current code location
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Team
BallControl (3) [Java Application]
ballworld.controller.BallControl at localhost:1803
Thread [AWT-Shutdown] (Running)
Daemon Thread [AWT-Windows] (Running)
Thread [DestroyJavaVM] (Running)
Thread [AWT-EventQueue-0] (Suspended (breakpoint at line 13 in Kill
KillStrategy$1.interact(Ball, Ball, Dispatcher) line: 13
MultiInteractStrategy.interact(Ball, Ball, Dispatcher) line: 18
MultiInteractStrategy.interact(Ball, Ball, Dispatcher) line: 18
Ball.interactWith(Ball, Dispatcher) line: 303
Collide2Strategy$ElasticBounceInteractStrategy.interact(Ball, Ball,
Collide2Strategy$1.apply(Ball, Dispatcher) line: 27
Ball.update(Observable, Object) line: 91
Dispatcher(Observable).notifyObservers(Object) line: not available
Dispatcher.notifyAll(Object) line: 16
EatStrategy(Collide2Strategy).updateState(Ball, Dispatcher) line: 2
Ball.updateState(Dispatcher) line: 102
BallModel$2$1.apply(Ball, Dispatcher) line: 102
Ball.update(Observable, Object) line: 91
Dispatcher(Observable).notifyObservers(Object) line: not available
Variables
Breakpoints
Name
Value
this KillStrategy$1 (id=38)
context Ball (id=40)
target Ball (id=41)
_color Color (id=51)
_container BallGUI$1 (id=56)
_interactStrategy MultiInteractStrategy (id=58)
_location Point (id=61)
x 433
y 51
_paintStrategy BallPaintStrategy (id=66)
_radius 15
_updateStrategy EatStrategy (id=48)
_velocity Point (id=76)
disp Dispatcher (id=42)
java.awt.Point[x=433,y=51]
Collide2Strategy.jav
KillStrategy.java
package ballworld.model.strategy;
import util.Dispatcher;
public class KillStrategy extends Collide2Strategy {
public void init(Ball context) {
super.init(context);
context.setInteractStrategy(new MultiInteract
@Override
public void interact(Ball context, Ball t
disp.deleteObserver(target);
}
);
Outline
ballworld.model.strategy
import declarations
KillStrategy 248 9/27/10 3:24 AM swong
init(Ball) : void
new IInteractStrategy() {...}
interact(Ball, Ball, Dispatcher) : void
Console
Tasks
BallControl (3) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Sep 30, 2010 11:39:38 AM)
addPaintBtn.actionPerformed, event=java.awt.event.ActionEvent[ACTION_PERFORMED,cmd=Add,when=1285864783680,modifier
1:1