Lecture Set #8: Debugging

1. Complete Class Summary
2. The Eclipse Debugger
3. Common Error – Privacy Leaks

Putting the pieces together

Constructors
- default constructor
- constructors with parameters
- copy constructors

Data
- data members: instance/static and public/private
- local variables
- stack and heap
- null references

Methods
- instance/static and public/private
- overloading: toString and others

Libraries
- importing and using methods from the library (the API)

JUnit Testing

Exceptions
- Throwing, trying, catching

The problem

Problem

JUnit can only tell if that passes or fails and where

Need a way to be able to see what is in memory (variables) at every step to be able to do complete trace [like that call stack examples we have been doing]

Solution

The debugger gives the ability to go through the code – displaying additional information similar to the by-hand call stack that we have been doing

Terminology

Break Point

drop a marker into the code so when it runs the execution will stop at that point

allows you to not have to go step by step through things you believe are correct

Step Over

takes one step in the current method

if that step is a method call, it performs that whole method call and steps to the next line in the current method

Step Into

takes one step in the current method

if that step is a method call, it steps into that method so that you can then step through it before getting to the next line in the method you were in

Eclipse

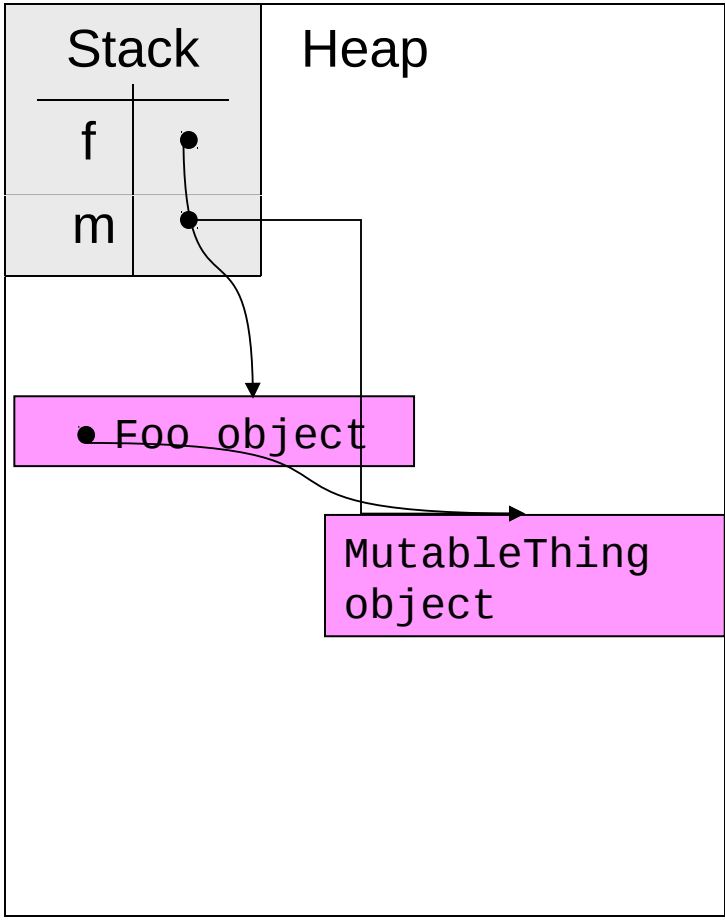
Perspective
- Debug Perspective
- Java Perspective

Run
- Debug As…
- Run As…

Know if it is still running
- Watch the red square – click it to kill

Privacy Leaks

```
public class MutableThing {
    …
    public void mutateMe() {…};
}
public class Foo {
    private MutableThing q
                    = new MutableThing();
    …
public MutableThing getQ(){
    return q;
    }
}
```



Consider following code

```
Foo f = new Foo ();
MutableThing m = f.getQ();
m.mutateMe();
```

After this executes, what happens?

This phenomenon is called a privacy leak

- Private instance variables can be modified outside class
- Behavior is due to aliasing

Fixing Privacy Leaks

Return copies of objects referenced by instance variables

To fix `getQ` method in `Foo`:

```
MutableThing getQ(){
        return new MutableThing(q);
}
```

This returns a copy of q

Changes made to this copy will not affect original