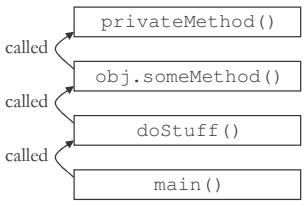
The Call Stack

The Call Stack

- When function/method calls are made, the programming language must keep track of who called who.
  - … and where to jump back to when the functions are finished.
- This is done with the "call stack".
  - A list of the functions that have been called to get to the current code.

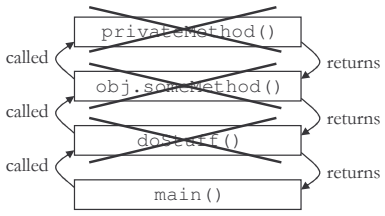
The Call Stack

- If a method `privateMethod()` is running now, the stack might look like this:



Returning Values

- When functions return a value, it is passed back to the calling function & the call is removed.

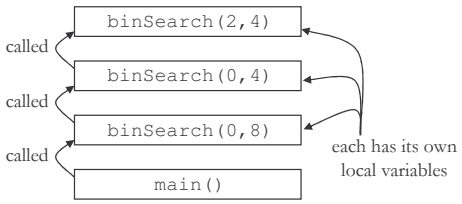


Storing

- The call stack is stored in memory by the Java virtual machine.
- All of the method's parameters and local variables and parameters are part of its entry on the stack.
  - This is why recursive calls all run separately: different calls & variables on the call stack.

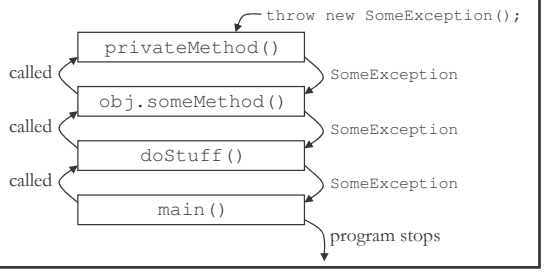
Recursion on the Stack

- Since local variables & parameters are stored on the stack, recursive calls work properly:



Exceptions and the Stack

- When a function throws an exception, that gets passed up the stack, instead of the return value:



Stack Display

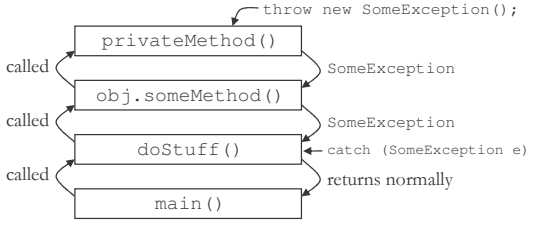
- When an exception halts the program, the output is a representation of the call stack.
  - eg.

```
Exception in thread "main" SomeException
        at MyClass.privateMethod(MyClass.java:8)
        at MyClass.someMethod(MyClass.java:56)
        at MainProg.doStuff(MainProg.java:31)
        at MainProg.main(MainProg.java:127)
```

- This stack trace corresponds to the previous example.

Catching Exceptions

- If an exception is caught, its propagation stops and the function returns as usual:



Designing with Exceptions

- An exception should occur if some uncommon condition occurs that the function can't handle.
  - Shouldn't happen if all goes well.
- An exception should be used in "exceptional" circumstances where the function can't recover by itself.
- The function should throw an exception.

```
throw new TypeOfException("error message");
```

Designing with Exceptions

- Some function up the call stack should `catch` the exception: `catch(TypeOfException e) {…}`
  - Whatever function can actually handle the problem should do so.
- e.g.
  - user input function throws `InputMismatchException`, menu function catches it and asks for another choice
  - network IO function can't read web page, main program pops up error message