class Fibber {
    public static int FibFun(int x) {
        if (x == 0) {
            return 0;
        } else if (x == 1) {
            return 1;
        }
        return FibFun(x - 1) + FibFun(x - 2);
    }

    public static void main(String a[]) {
        System.out.println(FibFun(4));
    }
}
Suppose \texttt{FibFun(3)} is called. The blue circled numbers tell what happens in order of time.

\begin{verbatim}
class Fibber {
  public static int FibFun(int x) {
    if (x == 0) {
      return 0;
    } else if (x == 1) {
      return 1;
    } return FibFun(x - 1) + FibFun(x - 2);}
public static void main(String a[]) {
  System.out.println(FibFun(4));
}
\end{verbatim}

Copy line 4 of the Java program next to line 0 of the Java Virtual Machine program as shown, because the LineNumberTable says line 4 in the Java program corresponds to Virtual Machine instructions beginning at position 0.

You copy four more Java program lines next to corresponding code positions in the same manner.

Start Length Slot Name Signature
0 27 0 x I

Code:
\begin{verbatim}
  load from slot 0
  ifne 6
  ireturn
  iload_0
  iconst_0
  if_icmpne
  iconst_1
  iinc 1 1
  ireturn
  iload_0
  iconst_1
  isub
  invokespecial // Method FibFun(1)
  12
  invokespecial // Method FibFun(1)
  22
  iadd
  ireturn
  FiberFun(3)  returns 2 (the return value is left at the top of the stack)
\end{verbatim}

Some JVM memories are in a stack.

What if \texttt{ifne 6} does when \texttt{x=3}:

1. Tests if 3==0.
2. Pops the stack.
3. Make the JVM Loy to the instruction starting at 6:

First Recursive CALL
\begin{verbatim}
  if (x == 0) {
    return 0;
  } return FibFun(x - 1) + FibFun(x - 2);
\end{verbatim}

return 1 12
FibFun(2)
FibFun(3)
FibFun(1)
FibFun(0)

Second Recursive CALL
\begin{verbatim}
  add
  ireturn
\end{verbatim}

return 1 12
FibFun(3)
Java Virtual Machine instructions

`iload_0`  integer load of local variable in slot 0
It pushes (creates) a "memory cell" at the top of the stack and copies the value of that local variable into that cell.

`ifne 6`  if not equal
(1) Tests if the cell on top of the stack contains 0
(2) (In all cases) pops (removes) the top stack cell.
(3) If not equal to 0, makes JVM continue at ("go to") instructions from position 6.
If equal to 0, "Fall through" to the next instruction.

`iconst_0`  integer constant 0 (`iconst_1`, `iconst_2` are similar)
Push a cell containing the value 0 on top of the stack.

`ireturn`  integer return
Do the return "dance" (Return to caller)
The return value, an integer, should be on top of the stack.

`if_icmpne 13`  if integer compare not equal
(1) Tests if the top two stack cells contain equal ints.
(2) (In all cases) pop the top two stack cells.
(3) If not equal, go to instruction at pos. 6 and beyond.
(4) If equal, "Fall through"
isub, iadd, int2add, int2sub
(1) For subract, subract the top stack entry
    from the one below it. For add, add them.
(2) In all cases, pop both operands
(3) Push the difference or sum.
So, the result in one cell replaces the operands
in two cells

invokestatic #2
Invoke (i.e. call) the static method specified
by constant #2 in the class file's constant pool.
The arguments should be on top of the stack.
The method call should leave any return value
on top of the stack

So with stack $[2, 1]$ after invokestatic #2 (FibFun),
the stack looks like $[1, 1]$ because FibFun(2) returns 1.
Compiled from "Fibber.java"
class Fibber extends java.lang.Object
    SourceFile: "Fibber.java"
    minor version: 0
    major version: 50
Constant pool:
  const #1 = Method
  const #2 = Method
  const #3 = Field
  const #4 = Method
  const #5 = class
  const #6 = class
  const #7 = Asciz
  const #8 = Asciz
  const #9 = Asciz
  const #10 = Asciz
  const #11 = Asciz
  const #12 = Asciz
  const #13 = Asciz
  const #14 = Asciz
  const #15 = Asciz
  const #16 = Asciz
  const #17 = Asciz
  const #18 = Asciz
  const #19 = Asciz
  const #20 = Asciz
  const #21 = Asciz
  const #22 = Asciz
  const #23 = Asciz
  const #24 = Asciz
  const #25 = NameAndType
  const #26 = NameAndType
  const #27 = class
  const #28 = NameAndType
  const #29 = class
  const #30 = NameAndType
  const #31 = Asciz
  const #32 = Asciz
  const #33 = Asciz
  const #34 = Asciz
  const #35 = Asciz
  const #36 = Asciz
  const #37 = Asciz
  const #38 = Asciz

{ Fibber();
  LineNumberTable:
    line 1: 0

LocalVariableTable:
    Start  Length  Slot  Name    Signature
    0      5      0    this    Ljava/lang/Object;