TA: Sherdil Niyaz

CS61B Extra Problems 2 Solutions

Git

Sometimes it can be unclear what state a file is in with Git.

- If you create a brand new file called hug.java in a repo (repository), what state is it in?

The file is **untracked** since it's brand new- Git knows nothing about it!

- If you then `add` that file, what state is it in?

The file is now **staged**. You've "shined a light" on the file and captured a snapshot of it that will be used during a commit.

- Great! You now edit hug.java and decide to `commit` the changes you made to that file. Now what state is hug.java in?

The file "resets" to a state called **unmodified** after the commit. This is different from untracked since the file is still being tracked by Git, but a file in a state of unmodified hasn't changed since the last commit.

- Finally, you take hug.java and edit it. If you `commit` now, will the changes be included? Why or why not?

No! This is because the file has been **modified**, but the changes have not been staged with the `add` command. You need to use `add hug.java` and then commit to include these changes in the commit.

If you had trouble with this question, check out this Stack Overflow post:

Git essentially has 4 main statuses for the files in your local repo:

***untracked**: The file is new, Git knows nothing about it. If you git add <file>, it becomes:*

***staged**: Now Git knows the file (tracked), but also made it part of the next commit batch (called the index). If you git commit, it becomes:*

***unmodified**: The file has not changed since its last commit. If you modify it, it becomes:*

***modified** (also called **unstaged**): Modified but not part of the next commit yet. You can stage it again with git add*

http://stackoverflow.com/questions/7564841/concept-of-git-tracking-and-git-staging

TA: Sherdil Niyaz

Git: Detached Head

Oh no! You got a `detached head error`! What will you do?

http://cs61b.ug/sp16/materials/guides/git-wtfs.html (second item)

Box and Pointer

Note: Adapted from Jonathan Shewchuck

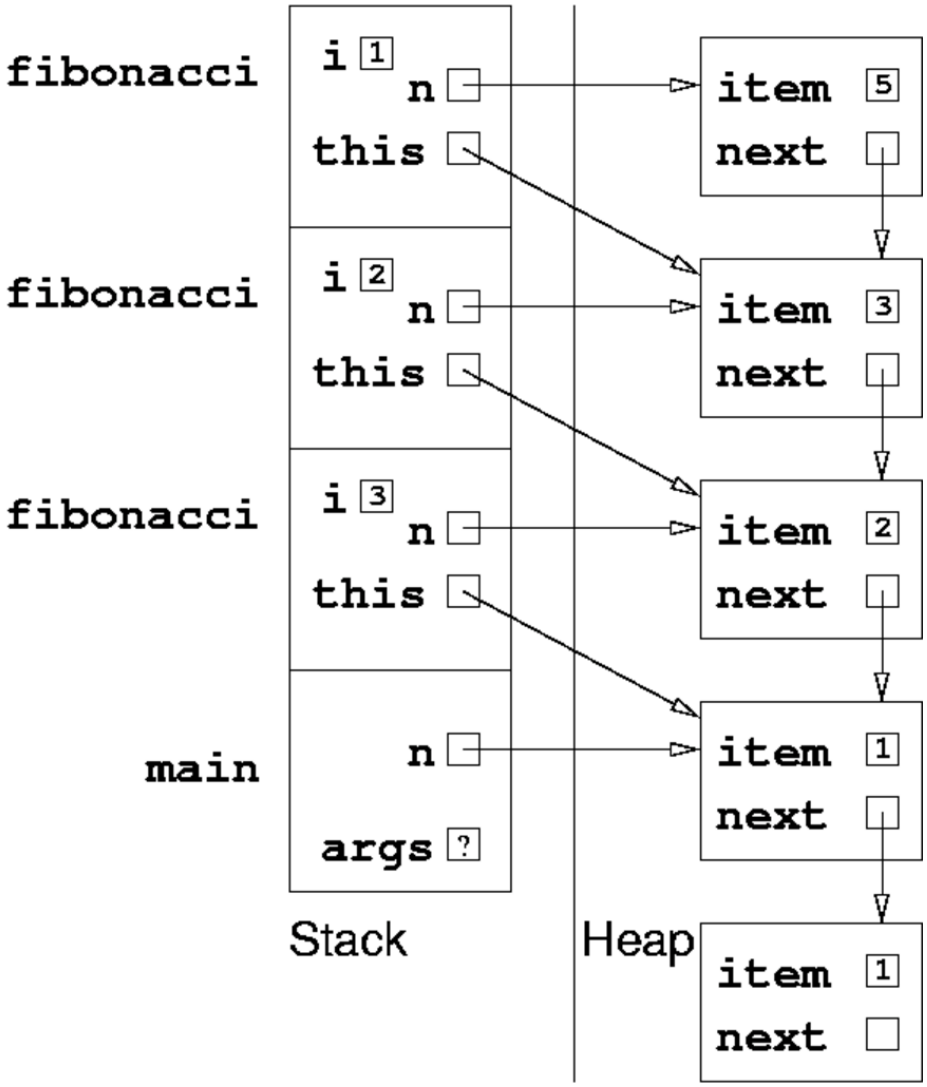
(CS61B Fall 2006, Midterm 1)

The following code creates a linked list representing the Fibonacci series in reverse order. (Don't worry if you don't know what that is.) Suppose we execute main. Draw the stack and heap at the moment when the topmost fibonacci call on the stack is about to "return n". Specifically, draw a box-and-pointer diagram with a box for every stack frame, local variable, object, and field in memory at that moment. Include the stack frames for all methods in progress, and illustrate which entities are inside those stack frames. Don't forget "this". Entities on the stack should be on the left-hand side of the page, and entities on the heap (aka objects) should be on the right-hand side.

```
public class ListNode {
  public int item;
  public ListNode next;

  public ListNode(int i, ListNode n) {
    item = i;
    next = n;
  }

  public ListNode fibonacci(int i) {    /* Appends i more terms to series. */
    /* Next number is sum of previous two numbers in series. */
    ListNode n = new ListNode(item + next.item, this);
    if (i <= 1) {
      // Draw the stack and the heap when execution reaches this point.
      return n;
    } else {
      return n.fibonacci(i - 1);
    }
  }

  public static void main (String[] args) {
    /* First 2 numbers in series are 1. */
    ListNode n = new ListNode(1, new ListNode(1, null));
    n = n.fibonacci(3);                          /* Append 3 more terms. */
  }
}
```

TA: Sherdil Niyaz

Solution:



This is a good check that you understand how Java actually treats objects and method calls.

Note: In Hug's version of 61B we usually don't include "this" like the above solution, so don't get too weirded out if you forgot it :) Read the following note if you're confused on what "this" is.

Also, the Java visualizer labels the Stack as "Frames" and the Heap as "Objects". Don't let that trip you up!