

```
Function equals

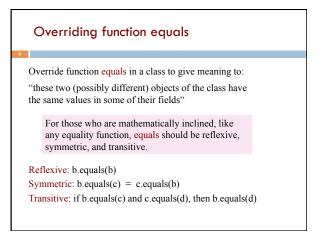
public class Object {
    /** Return true iff this object is
    * the same as ob */
    public boolean equals(Object b) {
        return this == b;
    }
}

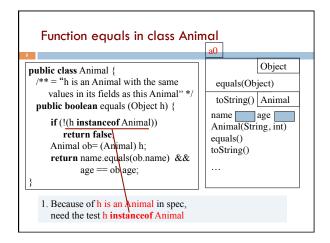
This gives a null-pointer exception:
    null equals(y)

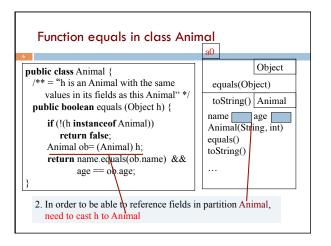
x = y
    except when x is null!

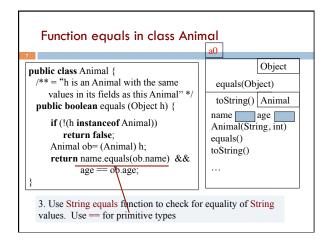
x ? Object

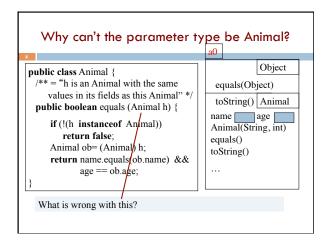
Object
```

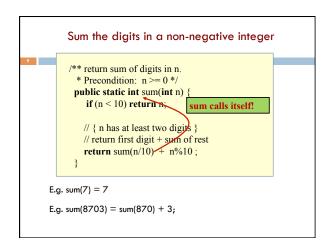


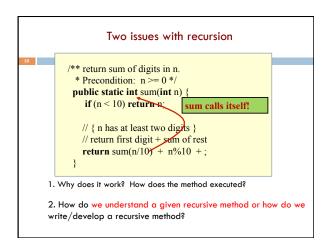


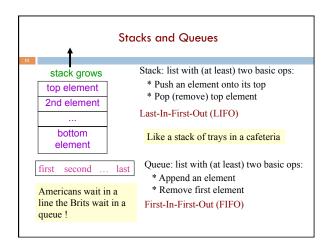


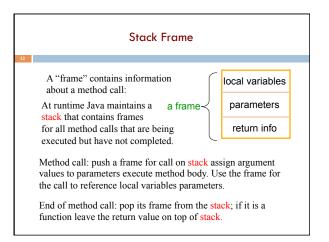












```
Frames for methods sum main method in the system
public static int sum(int n) {
   if (n \le 10) return n;
                                     frame:
   return sum(n/10) + n\%10;
                                              return info
public static void main(
    String[] args) {
                                              r ____ args
 int r = sum(824);
                                     frame:
                                              return info
 System.out.println(r);
  Frame for method in the system
                                     frame:
                                             return info
  that calls method main
```

```
Example: Sum the digits in a non-negative integer
public static int sum(int n) {
   if (n \le 10) return n;
   return sum(n/10) + n\%10;
public static void main(
    String[] args) {
 int r = sum(824);
 System.out.println(r);
                                                    args
                                    main
                                              return info
Frame for method in the system
that calls method main: main is
                                   system
                                             return info
then called
```

```
Example: Sum the digits in a non-negative integer
public static int sum(int n) {
   if (n \le 10) return n;
   return sum(n/10) + n\%10;
public static void main(
    String[] args) {
                                              n 824
 int r = sum(824);
                                              return info
 System.out.println(r);
                                              r ____ args _
                                    main
                                             return info
Method main calls sum:
                                   system
                                             return info
```

```
Example: Sum the digits in a non-negative integer
public static int sum(int n) {
   if (n \le 10) return n;
   return sum(n/10) + n\%10;
                                             n 82
public static void main(
                                             return info
    String[] args) {
                                             n 824
 int r = sum(824);
                                             return info
 System.out.println(r);
                                             r ____ args _
                                    main
                                             return info
n >= 10 sum calls sum:
                                   system
                                             return info
```

```
Example: Sum the digits in a non-negative integer
public static int sum(int n) {
   if (n < 10) return n;
                                               n_8_
   return sum(n/10) + n\%10;
                                              return info
                                               n <u>82</u>
public static void main(
                                              retultiti info
    String[] args) {
                                               n 824
 int r = sum(824);
                                              return info
 System.out.println(r);
                                               r ____ args
                                     main
                                              return info
n >= 10. sum calls sum:
                                    system
                                              return info
```

```
Example: Sum the digits in a non-negative integer
public static int sum(int n) {
   if (n < 10) return n;
                                              n_8_
   return sum(n/10) + n\%10;
                                              retu&n info
                                             n 82
public static void main(
                                             return info
    String[] args) {
                                              n 824
 int r = sum(824);
                                             return info
 System.out.println(r);
                                              ___ args
                                    main
                                             return info
n < 10 sum stops: frame is popped
and n is put on stack:
                                            return info
```

Example: Sum the digits in a non-negative integer public static int sum(int n) { if $(n \le 10)$ return n; **return** sum(n/10) + n%10; 8 n 82 public static void main(return info String[] args) { n 824 int r = sum(824); return info System.out.println(r); r ____ args main return info Using return value 8 stack computes 8 + 2 = 10 pops frame from stack puts return info return value 10 on stack

```
Example: Sum the digits in a non-negative integer
public static int sum(int n) {
   if (n \le 10) return n;
   return sum(n/10) + n\%10;
                                                 10
public static void main(
    String[] args) {
                                              n 824
 int r = sum(824);
                                              return info
 System.out.println(r);
                                              r ____ args _
                                    main
                                             return info
Using return value 10 stack computes
10 + 4 = 14 pops frame from stack
                                             return info
puts return value 14 on stack
```

Example: Sum the digits in a non-negative integer public static int sum(int n) { if (n < 10) return n; return sum(n/10) + n%10; } public static void main(String[] args) { int r= sum(824); System.out.println(r); } Using return value 14 main stores 14 in r and removes 14 from stack 14 r 14 raturn info return info

Summary of method call execution Memorize this! 1. A frame for a call contains parameters local variables and other information needed to properly execute a method call. 2. To execute a method call: push a frame for the call on the stack, assign values to parameters, execute method body, pop frame for call from stack, and (for a function) push returned value on stack When executing method body look in frame for call for parameters and local variables.

public static void m(...) { ... while (...) { int d = 5; ... } } In a call m() when is local variable d created and when is it destroyed? Which version of procedure m do you like better? Why?

```
Recursion is used extensively in math
                                    E.g. 3! = 3*2*1 = 6
Math definition of n factorial
   0! = 1
                                      Easy to make math definition
   n! = n * (n-1)! for n > 0
                                      into a Java function!
                                      public static int fact(int n) {
Math definition of b^c for c \ge 0
                                       if (n == 0) return 1;
  b^0 = 1
  b^c = b * b^{c-1} for c > 0
                                       return n * fact(n-1);
 Lots of things defined recursively:
 expression grammars trees ....
 We will see such things later
```

Two views of recursive methods

- How are calls on recursive methods executed?
 We saw that. Use this only to gain understanding / assurance that recursion works
- How do we understand a recursive method know that it satisfies its specification? How do we write a recursive method?

This requires a totally different approach. Thinking about how the method gets executed will confuse you completely! We now introduce this approach.

Understanding a recursive method

Step 1. Have a precise spec!

Step 2. Check that the method works in the base case(s): Cases where the parameter is small enough that the result can be computed simply and without recursive calls.

If n < 10 then n consists of a single digit. Looking at the spec we see that that digit is the required sum.

```
/** = sum of digits of n.
    * Precondition: n >= 0 */
public static int sum(int n) {
    if (n < 10) return n;

    // n has at least two digits
    return sum(n/10) + n%10;
}
```

Understanding a recursive method

Step 1. Have a precise spec!

Step 2. Check that the method works in the base case(s).

Step 3. Look at the recursive case(s). In your mind replace each recursive call by what it

 $\label{eq:condition:n} $$/** = $\sup of digits of n. $$ Precondition: $n >= 0 */$ $$ \begin{subarray}{c} \textbf{public static int } sum(\textbf{int } n) $ \\ \textbf{if } (n < 10) \begin{subarray}{c} \textbf{return } n; \end{subarray}$

// n has at least two digits return sum(n/10) + n%10;

does according to the method spec and verify that the correct result is then obtained.

```
return sum(n/10) + n%10;

return (sum of digits of n/10) + n%10; // e.g. n = 843
```

Understanding a recursive method

Step 1. Have a precise spec!

Step 2. Check that the method works in the base case(s).

Step 3. Look at the recursive case(s). In your mind replace each recursive call by what it does acc. to the spec and verify correctness.

/** = sum of digits of n.

* Precondition: n >= 0 */
public static int sum(int n) {
 if (n < 10) return n;

// n has at least two digits

orrectness.

return sum(n/10) + n%10;

Step 4. (No infinite recursion) Make sure that the args of recursive calls are in some sense smaller than the pars of the method.

n/10 < n

Understanding a recursive method

Step 1. Have a precise spec! Important! Can't do step 3 without it

Step 2. Check that the method works in the base case(s).

Step 3. Look at the recursive case(s). In your mind replace each recursive call by what it does according to the spec and verify correctness.

Once you get the hang of it this is what makes recursion easy! This way of thinking is based on math induction which we will see later in the course.

Step 4. (No infinite recursion) Make sure that the args of recursive calls are in some sense smaller than the pars of the method

Writing a recursive method

Step 1. Have a precise spec!

Step 2. Write the base case(s): Cases in which no recursive calls are needed Generally for "small" values of the parameters.

Step 3. Look at all other cases. See how to define these cases in terms of smaller problems of the same kind. Then implement those definitions using recursive calls for those smaller problems of the same kind. Done suitably point 4 is automatically satisfied.

Step 4. (No infinite recursion) Make sure that the args of recursive calls are in some sense smaller than the pars of the method

Examples of writing recursive functions

For the rest of the class we demo writing recursive functions using the approach outlined below. The java file we develop will be placed on the course webpage some time after the lecture.

- Step 1. Have a precise spec!
- Step 2. Write the base case(s).
- Step 3. Look at all other cases. See how to define these cases in terms of smaller problems of the same kind. Then implement those definitions using recursive calls for those smaller problems of the same kind.

The Fibonacci Function Mathematical definition: fib(0) = 0two base cases! fib(1) = 1fib(n) = fib(n-1) + fib(n-2) $n \ge 2$ Fibonacci sequence: 0 1 1 2 3 5 8 13 ... Fibonacci (Leonardo /** = fibonacci(n). Pre: $n \ge 0 */$ Pisano) 1170-1240? static int fib(int n) { if $(n \le 1)$ return n; Statue in Pisa Italy $// \{ 1 < n \}$ Giovanni Paganucci **return** fib(n-2) + fib(n-1);1863

Check palindrome-hood

A String palindrome is a String that reads the same backward and forward.

A String with at least two characters is a palindrome if

- (0) its first and last characters are equal and
- (1) chars between first & last form a palindrome:

e.g. AMANAPLANACANALPANAMA

- have to be the same -

have to be a palindrome

A recursive definition!

```
Example: Is a string a palindrome?

/** = "s is a palindrome" */
public static boolean isPal(String s) {
    if (s.length() <= 1)
        return true;

    // { s has at least 2 chars }
    int n = s.length()-1;
    return s.charAt(0) == s.charAt(n) && isPal(s.substring(1,n));
}

isPal("racecar") returns true
    isPal("pumpkin") returns false
```

A man a plan a caret a ban a myriad a sum a lac a liar a hoop a pint a catalpa a gas an oil a bird a yell a vat a caw a pax a wag a tax a nay a ram a cap a yam a gay a tsar a wall a car a luger a ward a bin a woman a vassal a wolf a tuna a nit a pall a fret a watt a bay a daub a tan a cab a datum a gall a hat a fag a zap a say a jaw a lay a wet a gallop a tug a trot a trap a tram a torr a caper a top a tonk a toll a ball a fair a sax a minim a tenor a bass a passer a capital a rut an amen a ted a cabal a tang a sun an ass a maw a sag a jam a dam a sub a salt an axon a sail an ad a wadi a radian a room a rood a rip a tad a pariah a revel a reel a reed a pool a plug a pin a peek a parabola a dog a pat a cud a nu a fan a pal a rum a nod an eta a lag an eel a batik a mug a mot a nap a maxim a mood a leek a grub a gob a gel a drab a citadel a total a cedar a tap a gag a rat a manor a bar a gal a cola a pap a yaw a tab a raj a gab a nag a pagan a bag a jar a bat a way a papa a local a gar a baron a mat a rag a gap a tar a decal a tot a led a tic a bard a leg a bog a burg a keel a doom a mix a map an atom a gum a kit a baleen a gala a ten a don a mural a pan a faun a ducat a pagoda a lob a rap a keep a nip a gulp a loop a deer a leer a lever a hair a pad a tapir a door a moor an aid a raid a wad an alias an ox an atlas a bus a madam a jag a saw a mass an anus a gnat a lab a cadet an em a natural a tip a caress a pass a baronet a minimax a sari a fall a ballot a knot a pot a rep a carrot a mart a part a tort a gut a poll a gateway a law a jay a sap a zag a fat a hall a gamut a dab a can a tabu a day a batt a waterfall a patina a nut a flow a lass a van a mow a nib a draw a regular a call a war a stay a gam a yap a cam a ray an ax a tag a wax a paw a cat a valley a drib a lion a saga a plat a catnip a pooh a rail a calamus a dairyman a bater a canal Panama