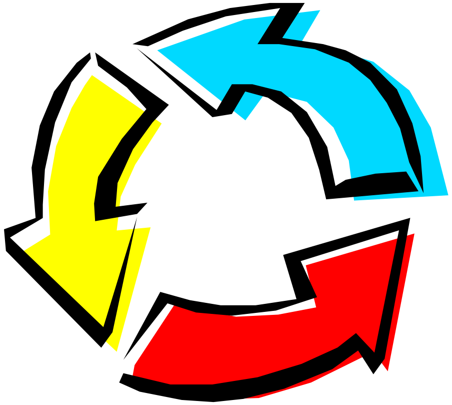
Recursion

Lab 21

Recursion



Recursion occurs when a method calls itself.

Recursion

```
public class RecursionOne
{
  public void run(int x)
  {
    out.println(x);
    run(x+1);
  }
  public static void main(String args[]  )
  {
    RecursionOne test = new RecursionOne();
    test.run(1);
  }
}
```

Will it stop?

OUTPUT

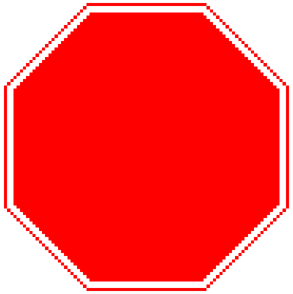
```
1
2
3
4
5
. . . . .
stack overflow
```

open recursionone.java

Base Case

A recursive method must have a stop condition/ base case.

Recursive calls will continue until the stop condition is met.



Recursion 2

```
public class RecursionTwo
{
  public void run(int x )
  {
    out.println(x);
    if(x<5)
      run(x+1);
  }
  public static void main(String args[]  )
  {
    RecursionTwo test = new RecursionTwo();
    test.run(1);
  }
}
```

base case
It will stop!

OUTPUT

```
1
2
3
4
5
```

Recursion 3

```
public class RecursionThree
{
  public void run(int x )
  {
    if(x<5)
      run(x+1);
    out.println(x);
  }
  public static void main(String args[]  )
  {
    RecursionThree test = new RecursionThree ();
    test.run(1);
  }
}
```

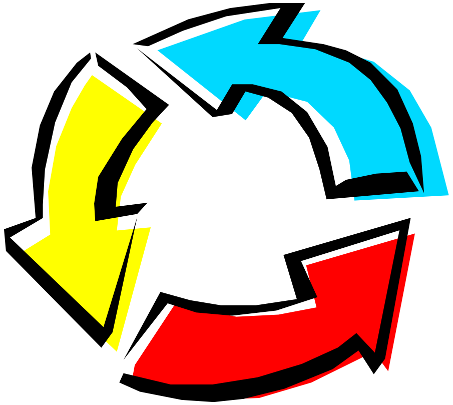
base case

OUTPUT

```
5
4
3
2
1
```

open
recursiontwo.java
recursionthree.java

Recursion



Recursion is basically a loop that is created using method calls.

do while

```
class DoWhile
{
   public void run( )
   {
      int x=0;
      do{
         x++;
         out.println(x);
      }while(x<10);          //condition
   }
   public static void main(String args[]  )
   {
      DoWhile test = new DoWhile();
      test.run( );
   }
}
```

open dowhile.java

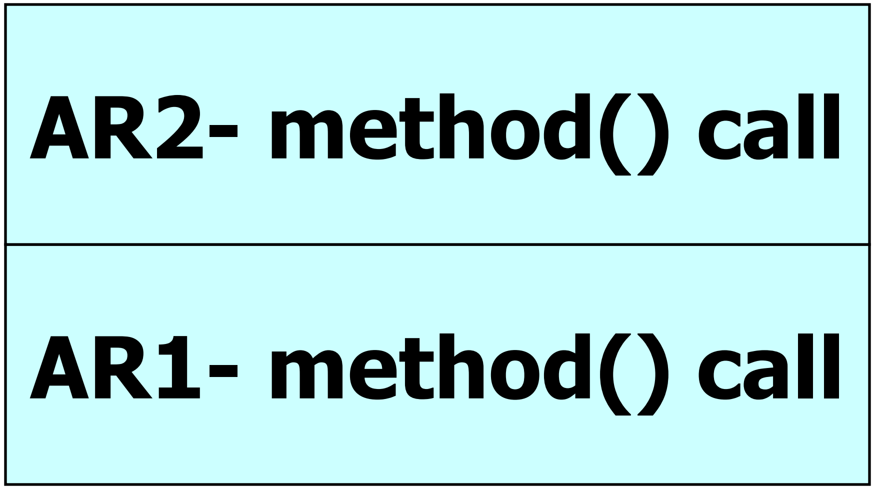
The Stack

When you call a method, an activation record for that method call is put on the stack with spots for all parameters/arguments being passed.

The Stack

AR1- method() call

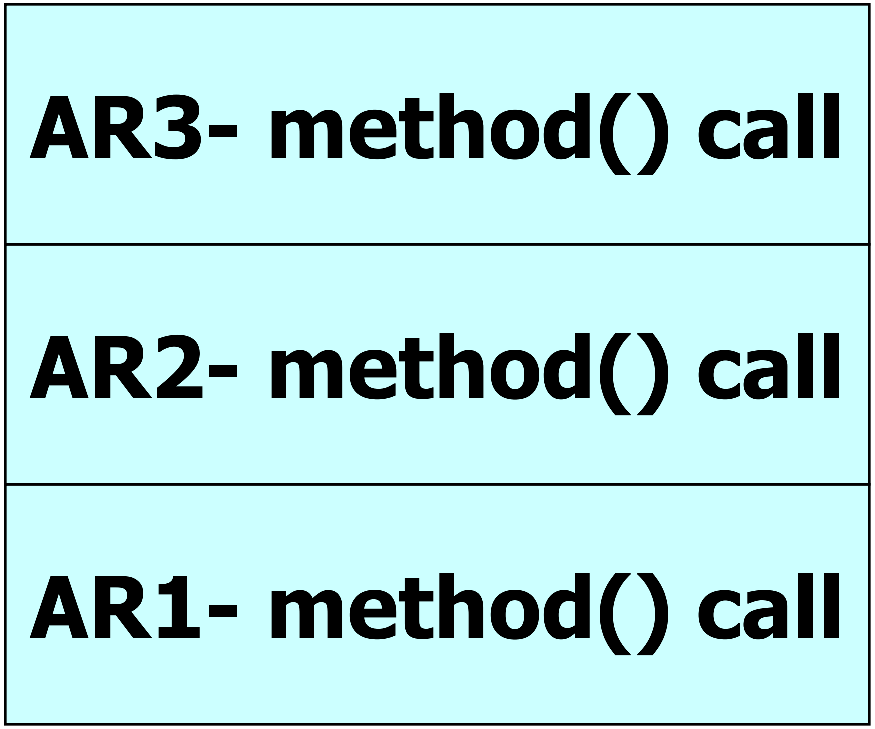
The Stack



AR2- method() call

AR1- method() call

The Stack

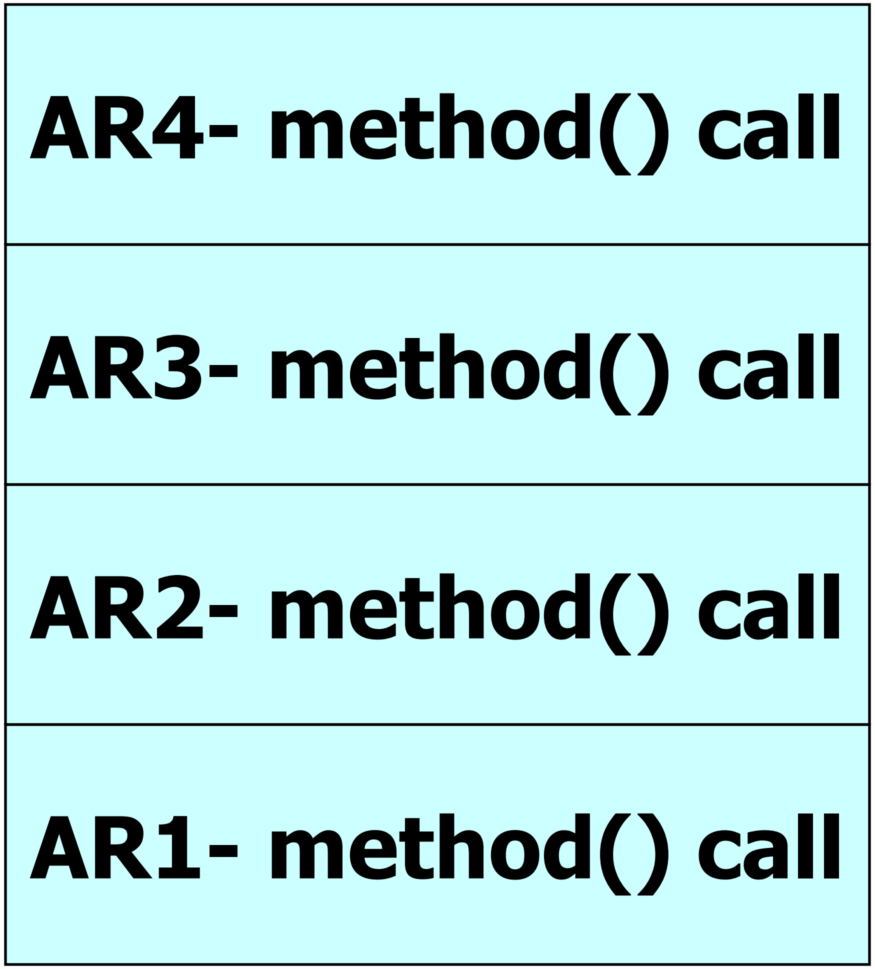


AR3- method() call

AR2- method() call

AR1- method() call

The Stack



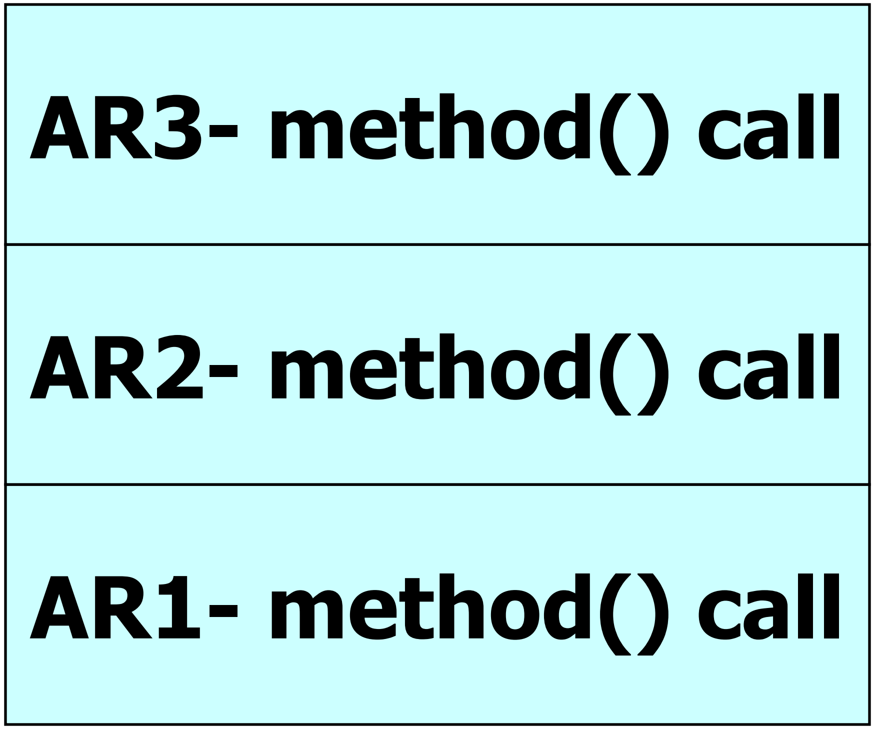
AR4- method() call

AR3- method() call

AR2- method() call

AR1- method() call

The Stack

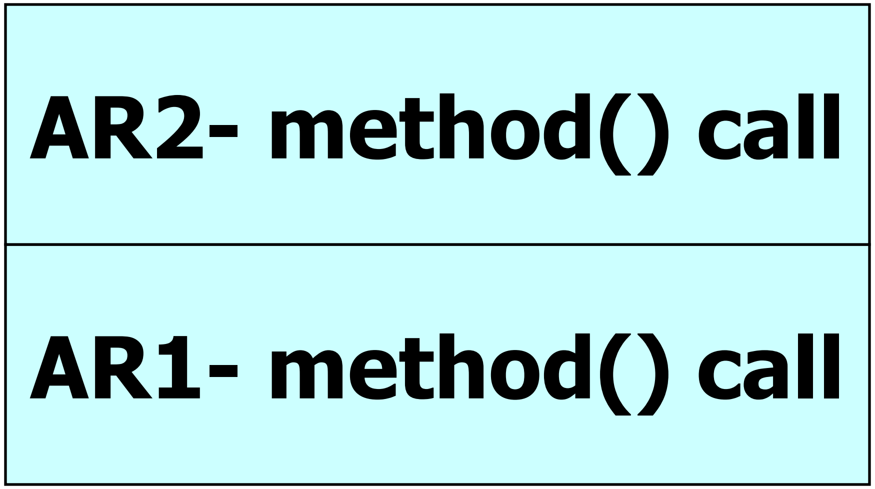


AR3- method() call

AR2- method() call

AR1- method() call

The Stack

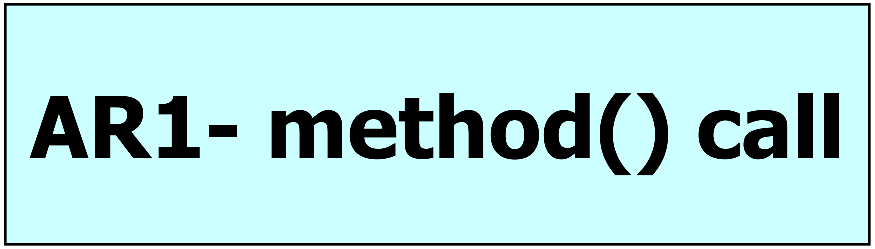


AR2- method() call

AR1- method() call

The Stack

As each call to the method completes, the instance of that method is removed from the stack.



AR1- method() call

Recursion 2

```
public class RecursionTwo
{
  public void run(int x )
  {
    out.println(x);
    if(x<5)
      run(x+1);
  }
  public static void main(String args[]  )
  {
    RecursionTwo test = new RecursionTwo();
    test.run(1);
  }
}
```

base case
It will stop!

OUTPUT

```
1
2
3
4
5
```

Recursion 3

```
public class RecursionThree
{
  public void run(int x )
  {
    if(x<5)
      run(x+1);
    out.println(x);
  }
  public static void main(String args[]  )
  {
    RecursionThree test = new RecursionThree();
    test.run(1);
  }
}
```

base case

OUTPUT

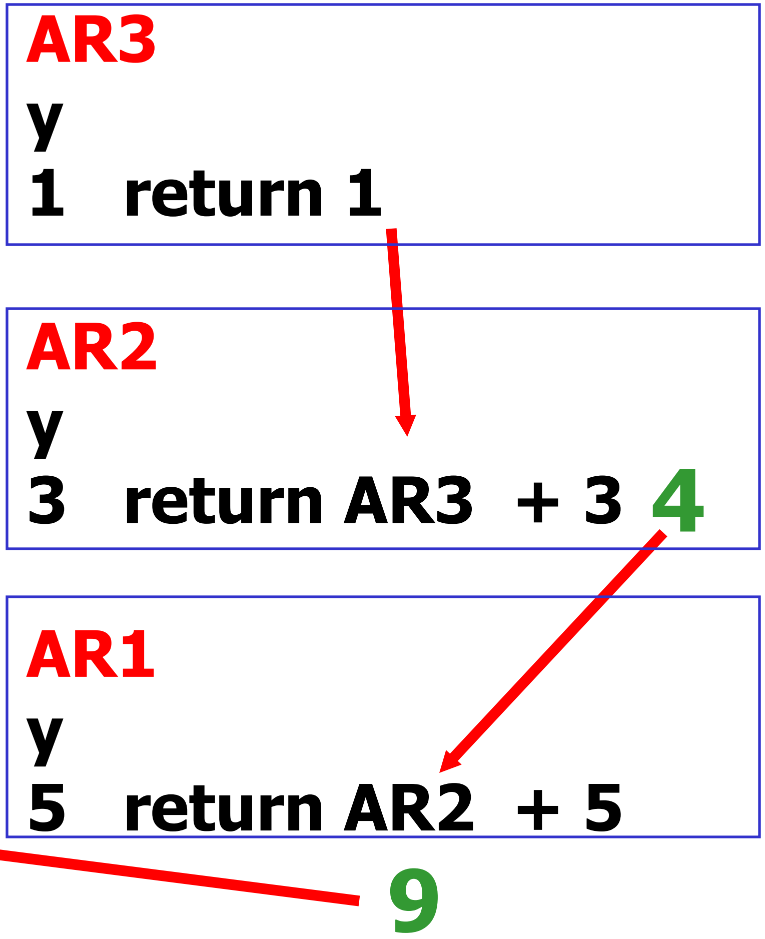
```
5
4
3
2
1
```

Why does this output differ from recur2?

Tracing Recursive Code

```
int fun(int y)
{
   if(y<=1)
      return 1;
   else
      return fun(y-2) + y;
}

//test code in client class
out.println(test.fun(5));
```



AR3
y
1   return 1

AR2
y
3   return AR3 + 3 4

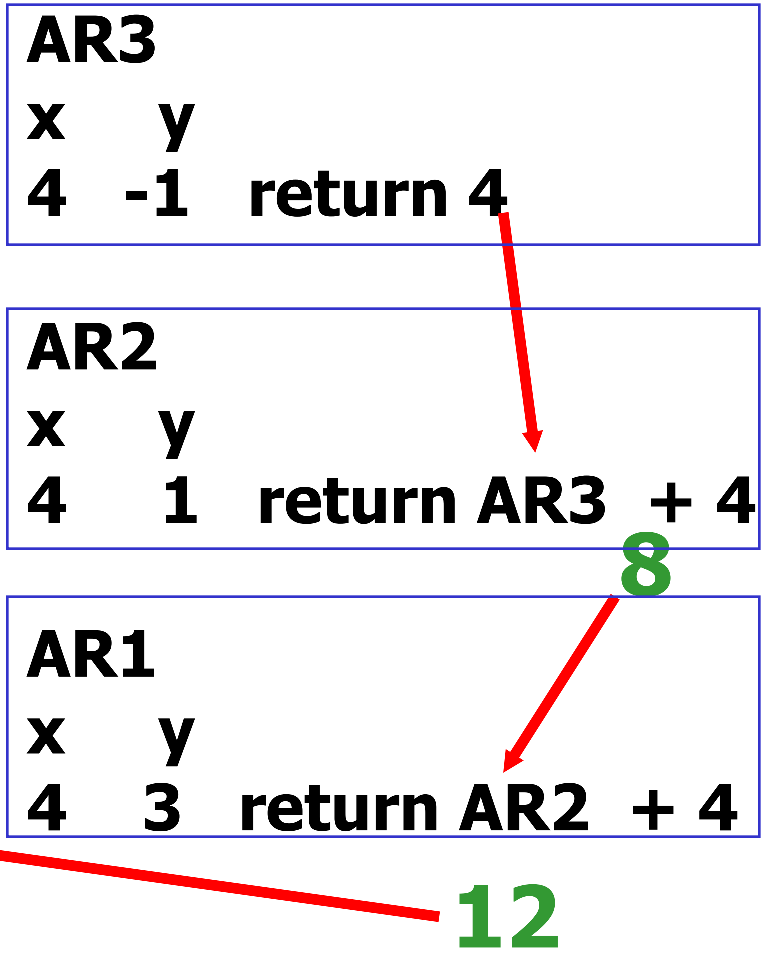
AR1
y
5   return AR2 + 5

9

Tracing Recursive Code

```
int fun( int x, int y)
{
 if( y < 1)
    return x;
 else
    return fun( x, y - 2) + x;
}

//test code in client class
out.println(test.fun(4,3));
```



AR3

x y

4 -1 return 4

AR2

x y

4 1 return AR3 + 4

8

AR1

x y

4 3 return AR2 + 4

12

open

recursionfour.java

recursionfive.java

Recursive Fun

```
int fun(int x, int y)
{
  if ( x == 0 )
    return x;
  else
    return x+fun(y-1,x);
}
```

What would fun(4,4) return?

OUTPUT

```
16
```

open recursionsix.java

split recursion

tail recursion

```
public String recur(String s)
{
  int len = s.length();
  if(len>0)
      return recur(s.substring(0,len-1)) +
                                             s.charAt(len-1);
  return "";
}
```

split recursion

tail recursion

```
public String recur(String s)
{
  int len = s.length();
  if(len>0)
       return s.charAt(len-1) +
                              recur(s.substring(0,len-1));
  return "";
}
```

open

recursionseven.java

recursioneight.java

split recursion

tail recursion

The Stack

call out.println(recur("abc"))

```
public String recur(String s)
{
  int len = s.length();
  if(len>0)
      return recur(s.substring(0,len-1)) +
                                          s.charAt(len-1);
  return "";
}
```

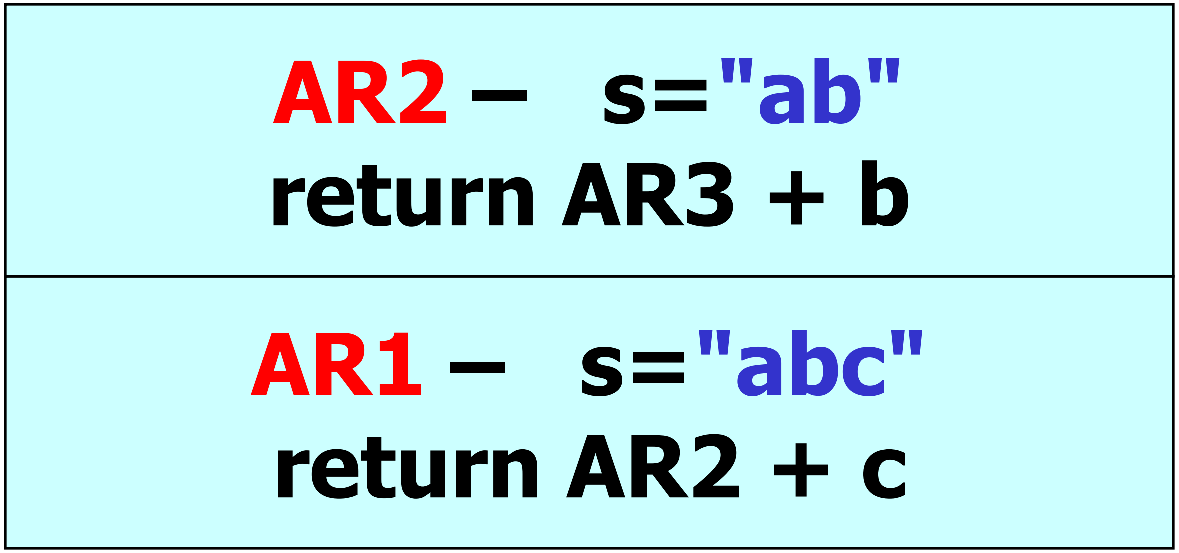
The Stack

call out.println(recur("abc"))

AR stands for activation record. An AR is placed on the stack every time a method is called.

AR1 – s="abc"
return AR2 + c

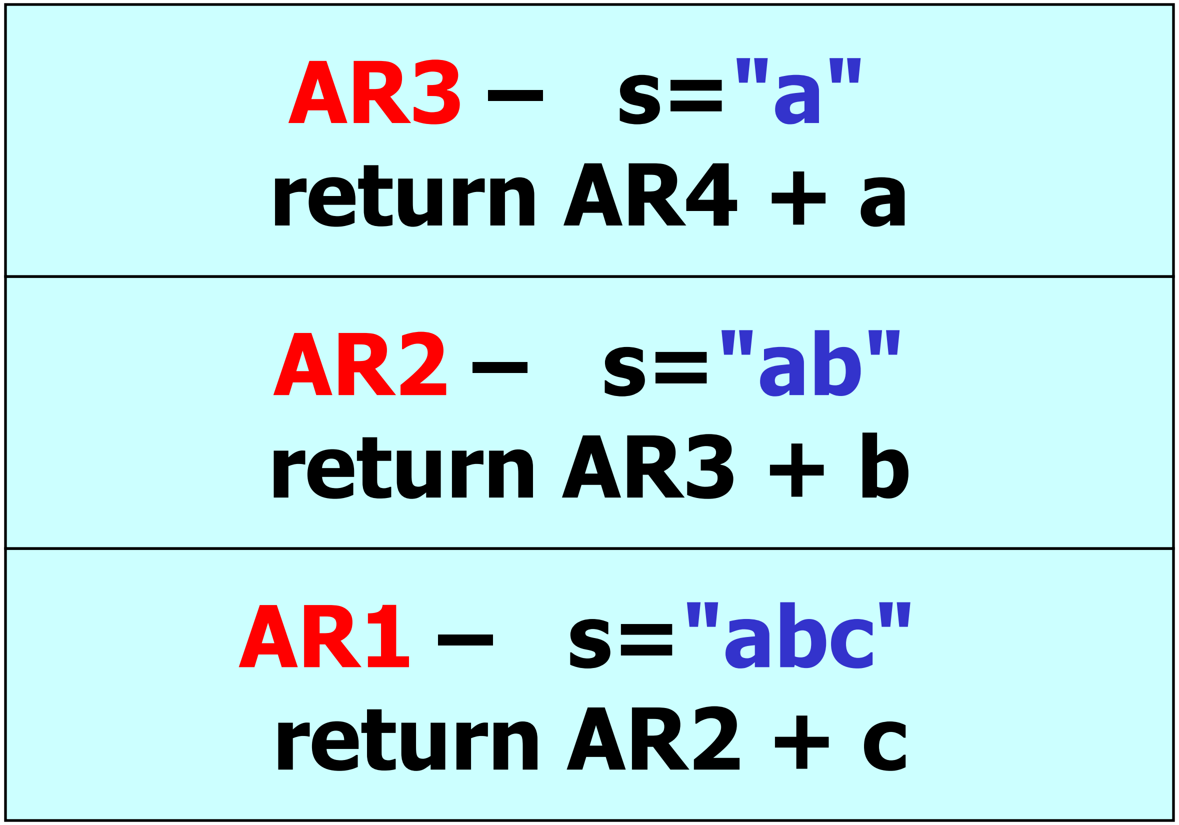
The Stack



AR2 – s="ab"
return AR3 + b

AR1 – s="abc"
return AR2 + c

The Stack

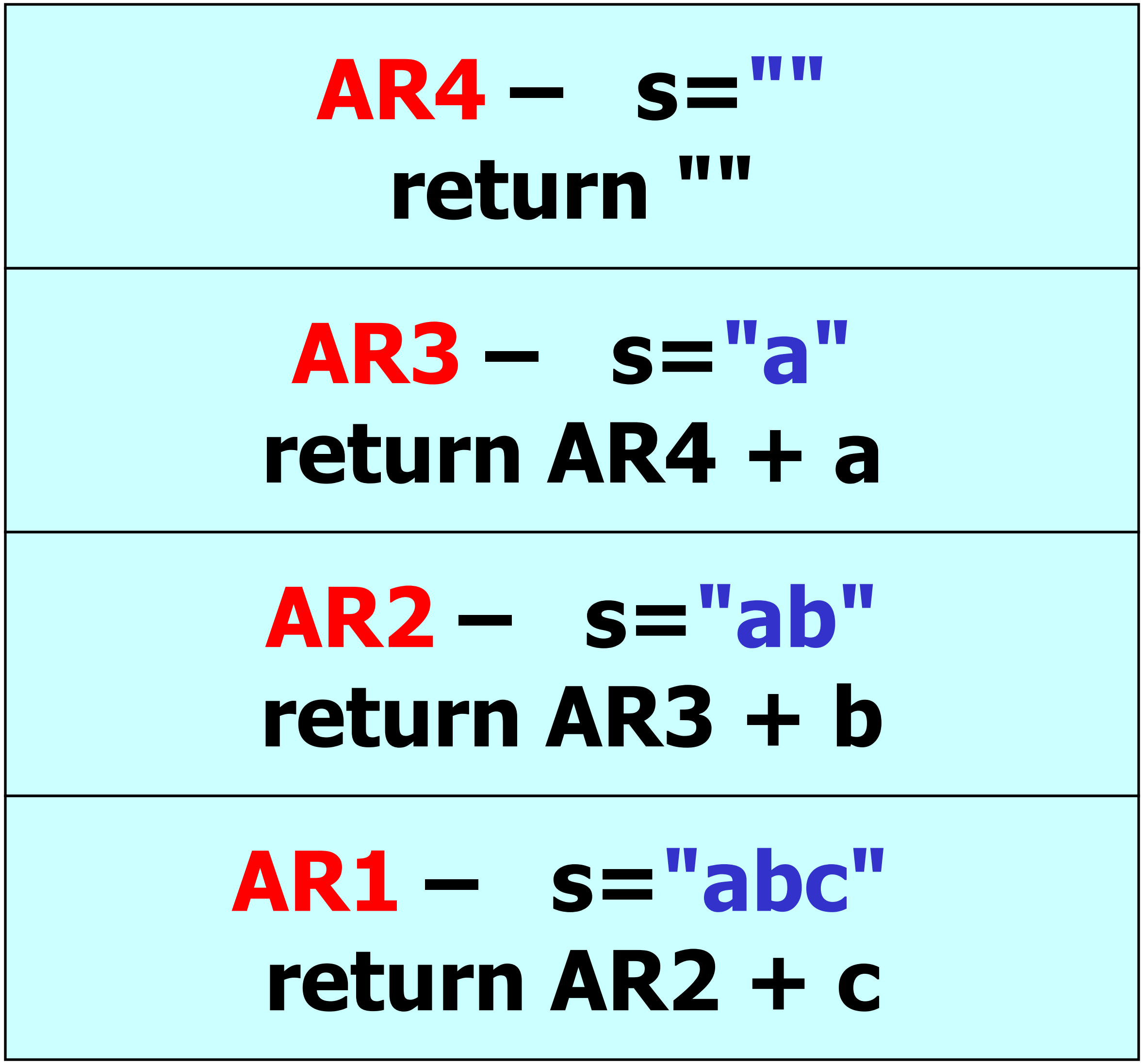


AR3 – s="a"
return AR4 + a

AR2 – s="ab"
return AR3 + b

AR1 – s="abc"
return AR2 + c

The Stack



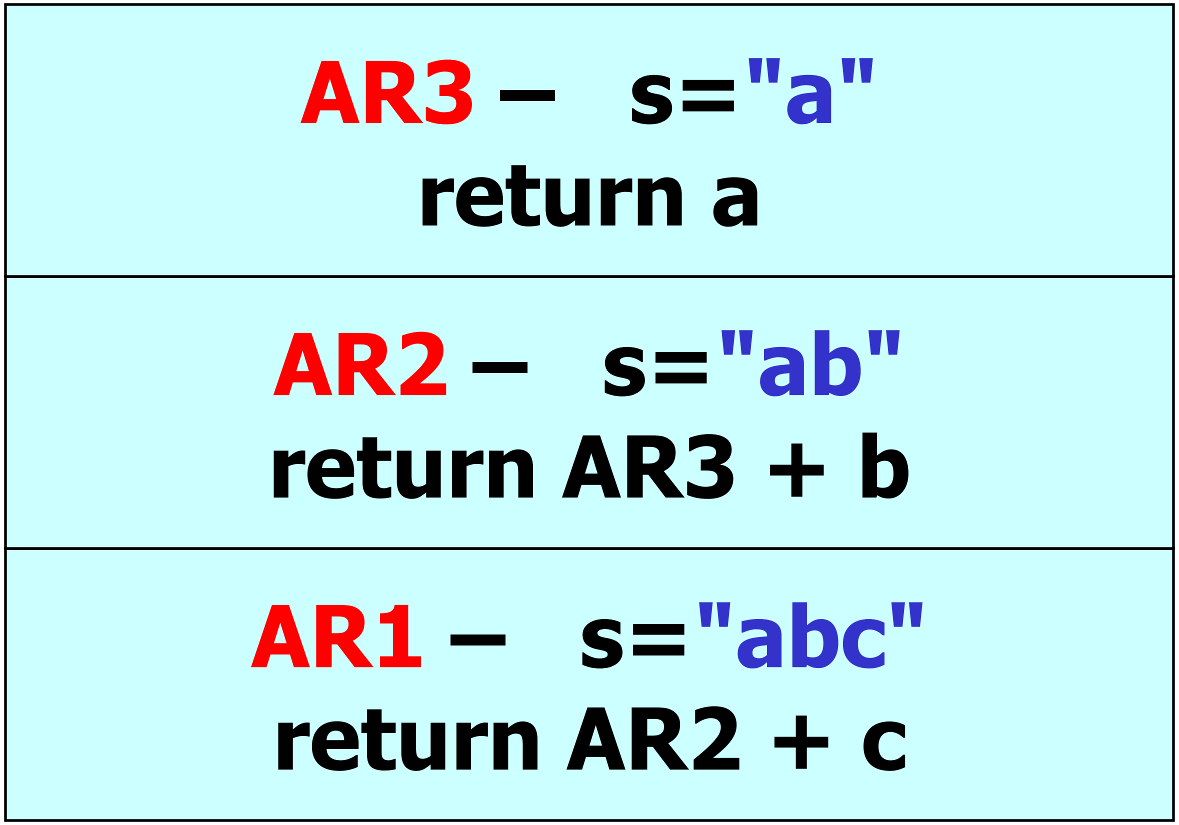
AR4 – s=""
return ""

AR3 – s="a"
return AR4 + a

AR2 – s="ab"
return AR3 + b

AR1 – s="abc"
return AR2 + c

The Stack

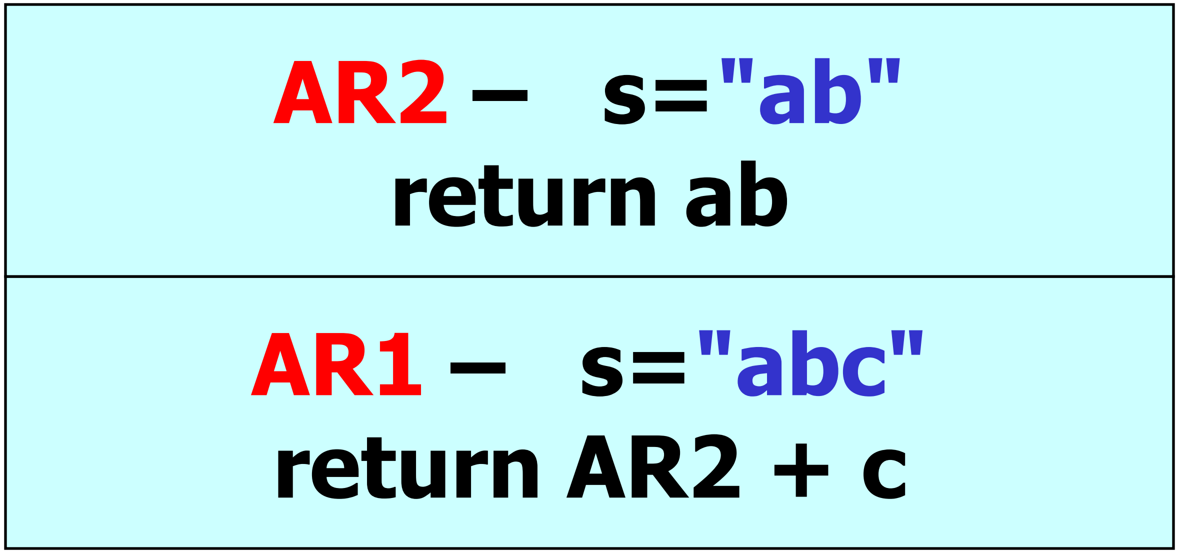


AR3 – s="a"
return a

AR2 – s="ab"
return AR3 + b

AR1 – s="abc"
return AR2 + c

The Stack



AR2 – s="ab"
return ab

AR1 – s="abc"
return AR2 + c

The Stack

call out.println(recur("abc"))

OUTPUT

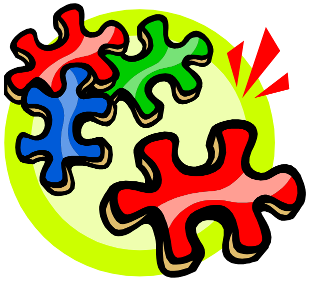
abc

AR1 – s="abc"
return abc

What is the point?

If recursion is just a loop, why would you just not use a loop?

Recursion is a way to take a block of code and spawn copies of that block over and over again. This helps break a large problem down into smaller pieces.



Counting Spots

If checking 0 0, you would find 5 @s are connected.

```
@ - @ - - @ - @ @ @
@ @ @ - @ @ - @ - @
- - - - - - - @ @ @
- @ @ @ @ @ - @ - @
- @ - @ - @ - @ - @
@ @ @ @ @ @ - @ @ @
- @ - @ - @ - - - @
- @ @ @ - @ - - - -
- @ - @ - @ - @ @ @
- @ @ @ @ @ - @ @ @
```

@ at spot [0,0]
@ at spot [0,2]
@ at spot [1,0]
@ at spot [1,1]
@ at spot [1,2]

The exact same checks are made at each spot.

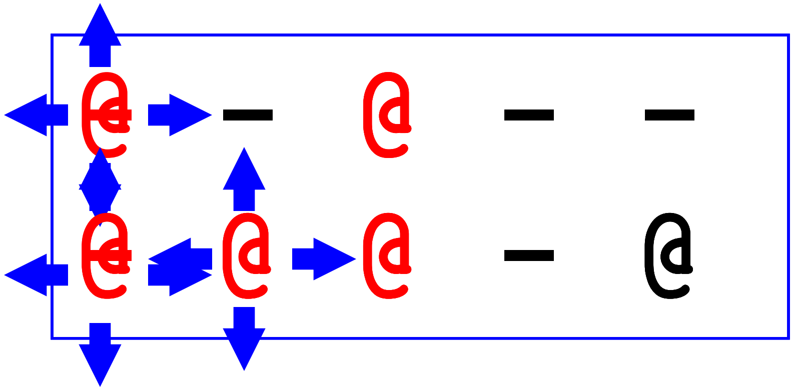
Counting Spots

```
if ( r and c are in bounds and
                          current spot is a @ )
   mark spot as visited
   bump up current count by one
   recur up
   recur down
   recur left
   recur right
```

This same block of code is recreated with each recursive call. The exact same code is used to check many different locations.

Counting Spots

```
if ( r and c are in bounds and
                    current spot is a @ )
   mark spot as visited
   bump up current count by one
   recur up
   recur down
   recur left
   recur right
```



```
@ - @ - -
@ @ @ - @
```

Start work on Lab 21