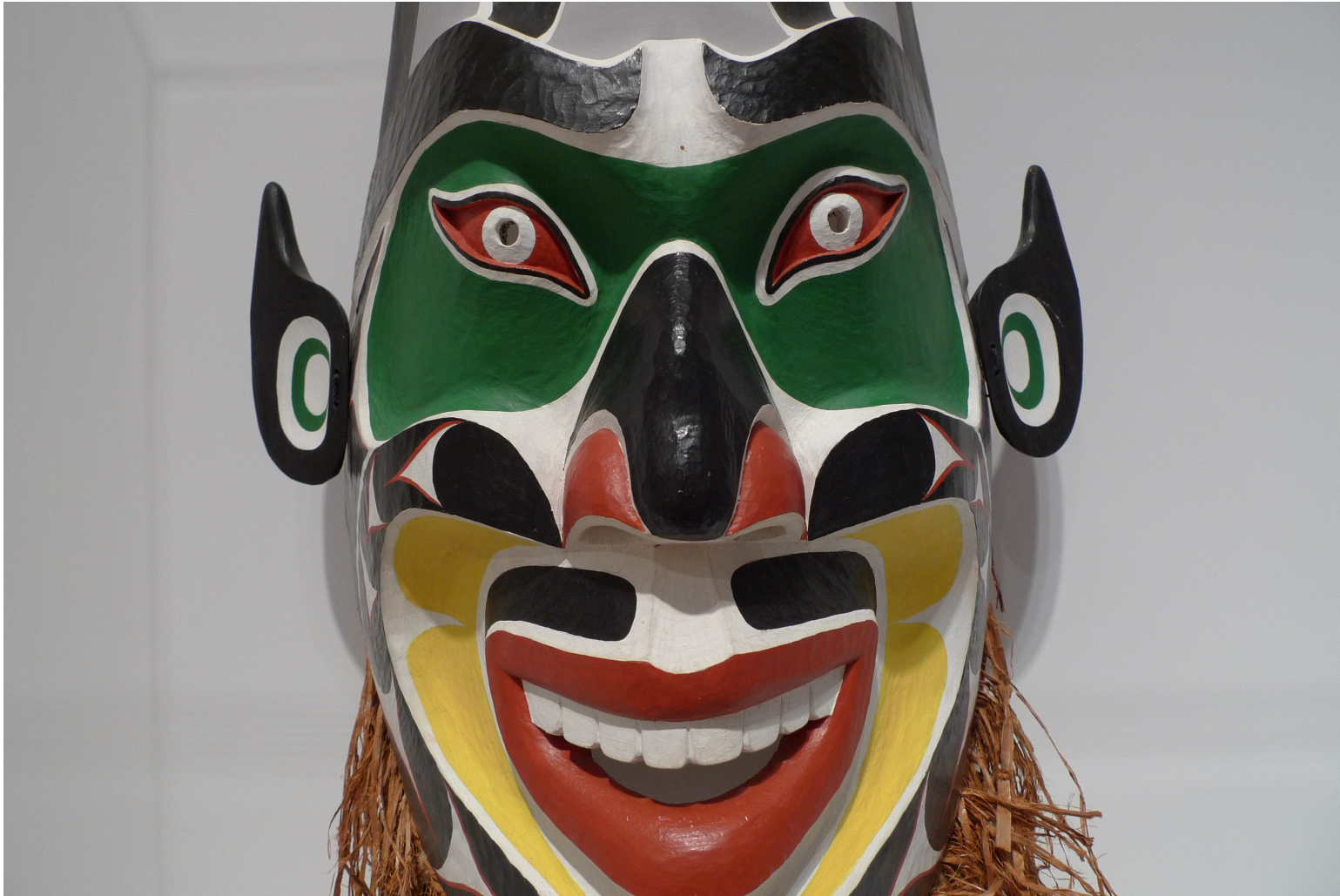


Recursion

Understanding “static”



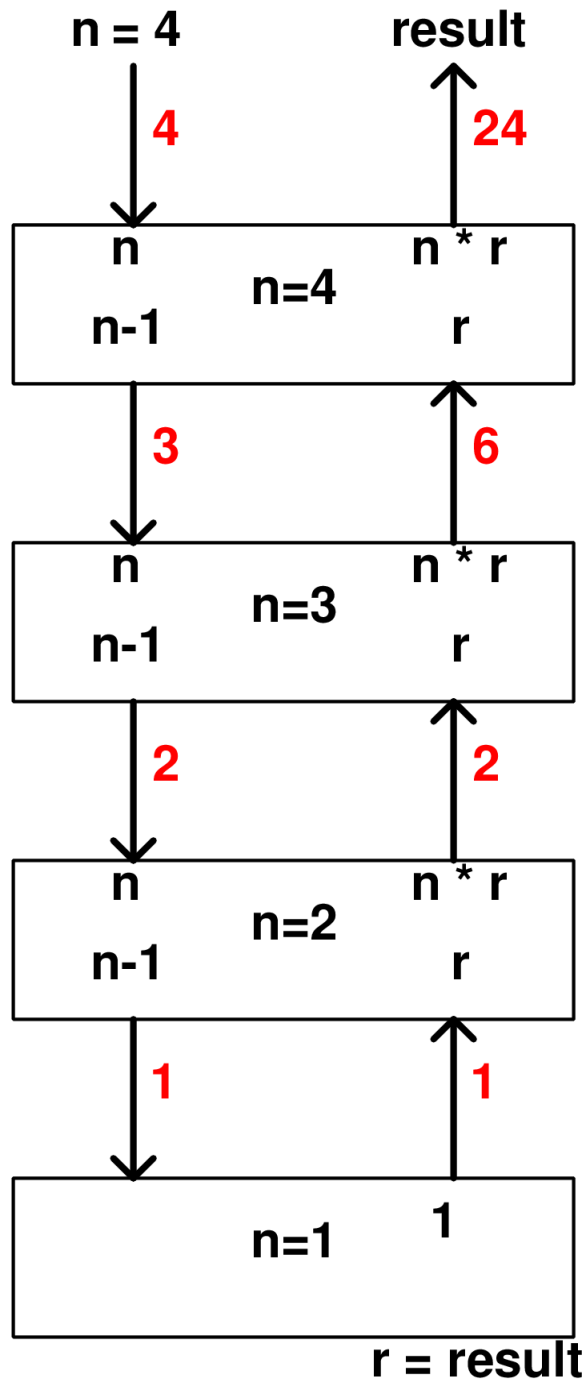
Recursion

- A method can call itself, that is recursion
- Ex: find factorial of N
 where factorial is the product of integers 1 to N
 For $N = 3$, factorial of 3 = $1 \times 2 \times 3 = 6$
- Factorial can be calculated without recursion
 using an iterative approach based on a for loop
- An alternative is to use recursion

```
class Factorial {                // A recursive function
    int factR(int n) {
        int result;

        if(n==1) return 1;
        result = factR(n-1) * n;
        return result;
    }
}
class Recursion {
    public static void main(String args[ ]) {
        Factorial f = new Factorial( );

        System.out.println("Factorial of 4 is " + f.factR(4));
    }
}
```



```

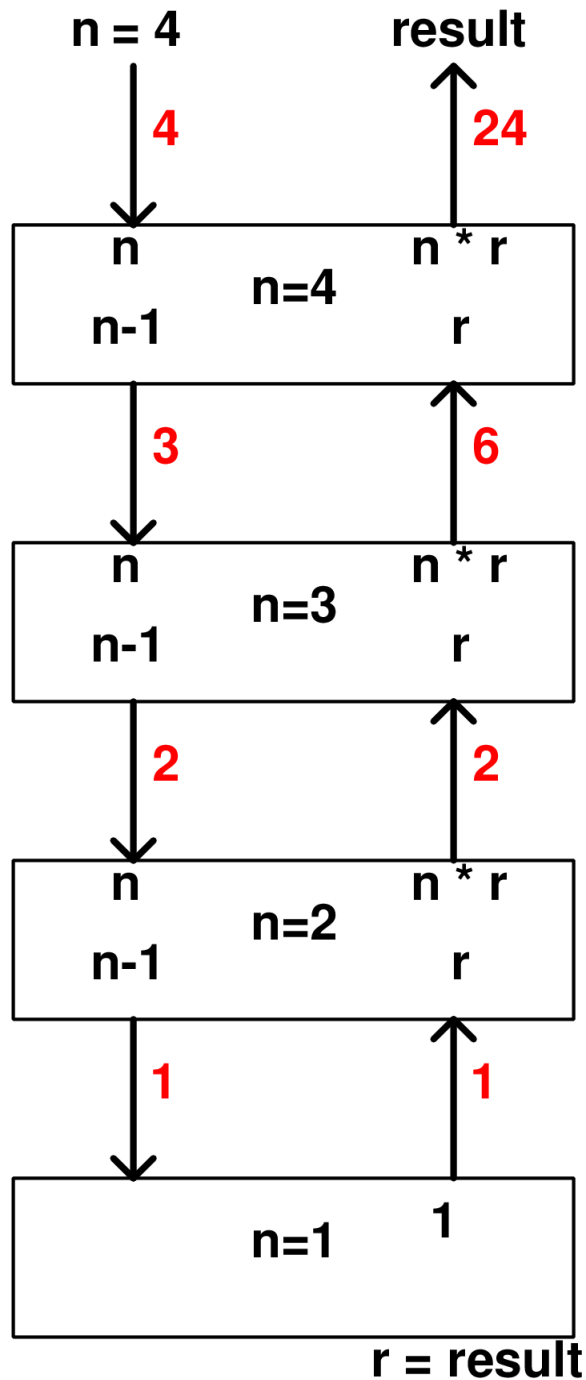
if(n==1) return 1;
result = factR(n-1) * n;
return result;

```

```

if(n==1) return 1;
result = factR(n-1) * n;
return result;

```



Stack grows

```

class Factorial {
  int factR(int n) {
    int result;

    if(n==1) return 1;
    result = factR(n-1) * n;
    return result;
  }
}

```

static members

- Placing the keyword `static` ahead of a variable or method basically creates a global member of a class.
- The static member is shared among all objects that belong to its class, i.e. only one copy exists even though there may be many objects
- The static member can be used or called before any objects of its class are created
- Note that `main` is declared `static`. It is called by the Java virtual machine when a program begins

Consider a class with two variables, one static and one not static. One method is defined that adds these two variables and returns sum:

```
class StaticDemo {  
    int x;          // a normal instance variable  
    static int y;  // a static variable  
  
    int sum( ) {  
        return x + y;  
    }  
}
```

(keep this class definition in mind)


```
class SDemo {
    public static void main(String args[]) {
        StaticDemo.y = 5;          // y is referenced before
                                   // object creation

        StaticDemo ob1 = new StaticDemo();
        StaticDemo ob2 = new StaticDemo();

        ob1.x = 10;
        ob2.x = 20;

        System.out.println( ob1.sum," and ", ob2.sum);
    }
}
```

resulting in ?

15 and 25

```
class SDemo {  
    public static void main(String args[]) {  
  
        StaticDemo ob1 = new StaticDemo();  
        StaticDemo ob2 = new StaticDemo();  
  
        ob1.x = 10;  
        ob2.x = 20;  
        ob1.y = 25;           // y assigned value via ob1  
                             // (for demo only)  
        System.out.println( ob1.sum," and ", ob2.sum);  
    }  
}
```

resulting in ?

```
class SDemo {  
    public static void main(String args[]) {  
  
        StaticDemo ob1 = new StaticDemo();  
        StaticDemo ob2 = new StaticDemo();  
  
        ob1.x = 10;  
        ob2.x = 20;  
        ob1.y = 25;           // y assigned value via ob1  
                               // (for demo only)  
        System.out.println( ob1.sum," and ", ob2.sum);  
    }  
}
```

resulting in ?

35 and 45

```
class SDemo {  
    public static void main(String args[]) {  
  
        StaticDemo ob1 = new StaticDemo();  
        StaticDemo ob2 = new StaticDemo();  
  
        ob1.x = 10;  
        ob2.x = 20;  
        ob2.y = 25;           // y assigned value via ob2  
                               // (for demo only)  
        System.out.println( ob1.sum," and ", ob2.sum);  
    }  
}
```

resulting in ?

```
class SDemo {  
    public static void main(String args[]) {  
  
        StaticDemo ob1 = new StaticDemo();  
        StaticDemo ob2 = new StaticDemo();  
  
        ob1.x = 10;  
        ob2.x = 20;  
        ob2.y = 25;           // y assigned value via ob2  
                               // (for demo only)  
        System.out.println( ob1.sum," and ", ob2.sum);  
    }  
}
```

resulting in ?

35 and 45

```
class SDemo {  
    public static void main(String args[]) {  
  
        StaticDemo ob1 = new StaticDemo();  
        StaticDemo ob2 = new StaticDemo();  
  
        ob1.x = 10;  
        ob2.x = 20;  
        ob2.y = 25;           // y assigned value via ob2  
                               // (for demo only)  
        System.out.println( ob1.sum," and ", ob2.sum);  
    }  
}
```

resulting in ?

35 and 45

Same result assigning to either ob1.y or ob2.y

Note: In the prior slides a value was assigned to `y` using `ob1.y` or `ob2.y` which means `y` was being accessed via an object. Since `y` was declared static normally it should be accessed via the class name, i.e. `StaticDemo.y` as was done in the first slide where a value was assigned to `y`.

Static Blocks

- A static block can be defined in a class
- The static block is executed when the class is first loaded, prior to when constructors or methods are executed
- Can be thought of as initialization code


```
class StaticBlock {
    static double rootOf2;
    static double rootOf3;

    static {           // executes when class is loaded
        System.out.println("Inside static block.");
        rootOf2 = Math.sqrt(2.0);
        rootOf3 = Math.sqrt(3.0);
    }

    StaticBlock(String msg) {
        System.out.println(msg);
    }
}
```





```
class SDemo3 {  
    public static void main(String args[ ]) {  
        StaticBlock ob = new StaticBlock("Inside Constructor");  
  
        System.out.println("Square root of 2 is " +  
                            StaticBlock.rootOf2);  
        System.out.println("Square root of 3 is " +  
                            StaticBlock.rootOf3);  
    }  
}
```

(Note that since the two variables were declared static they are being referred to using the name of the class)

```
class SDemo3 {
    public static void main(String args[ ]) {
        StaticBlock ob = new StaticBlock("Inside Constructor");

        System.out.println("Square root of 2 is " +
            StaticBlock.rootOf2);
        System.out.println("Square root of 3 is " +
            StaticBlock.rootOf3);
    }
}
```

Output is:

Inside static block.  Printed before object was constructed
Inside Constructor.  Printed when object was constructed
Square root of 2 is 1.4142135623730951
Square root of 3 is 1.7320508077688772

Operating on Strings

The **String** class contains several methods that operate on strings. Here are the general forms for a few:

<code>boolean equals(<i>str</i>)</code>	Returns true if the invoking string contains the same character sequence as <i>str</i> .
<code>int length()</code>	Obtains the length of a string.
<code>char charAt(<i>index</i>)</code>	Obtains the character at the index specified by <i>index</i> .
<code>int compareTo(<i>str</i>)</code>	Returns less than zero if the invoking string is less than <i>str</i> , greater than zero if the invoking string is greater than <i>str</i> , and zero if the strings are equal.
<code>int indexOf(<i>str</i>)</code>	Searches the invoking string for the substring specified by <i>str</i> . Returns the index of the first match or <code>-1</code> on failure.
<code>int lastIndexOf(<i>str</i>)</code>	Searches the invoking string for the substring specified by <i>str</i> . Returns the index of the last match or <code>-1</code> on failure.

`string substring(int startIndex, int stopIndex)`