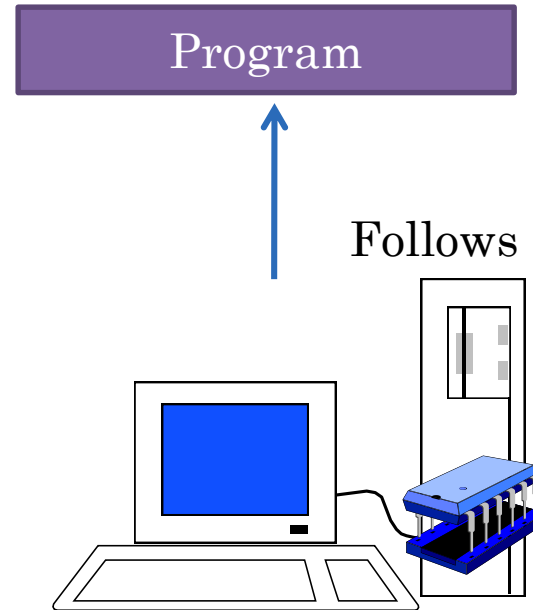
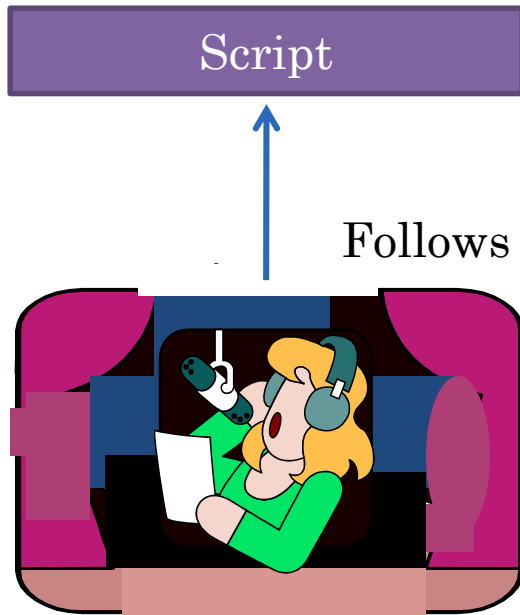


COMP 401: THE DUAL ROLES OF A CLASS

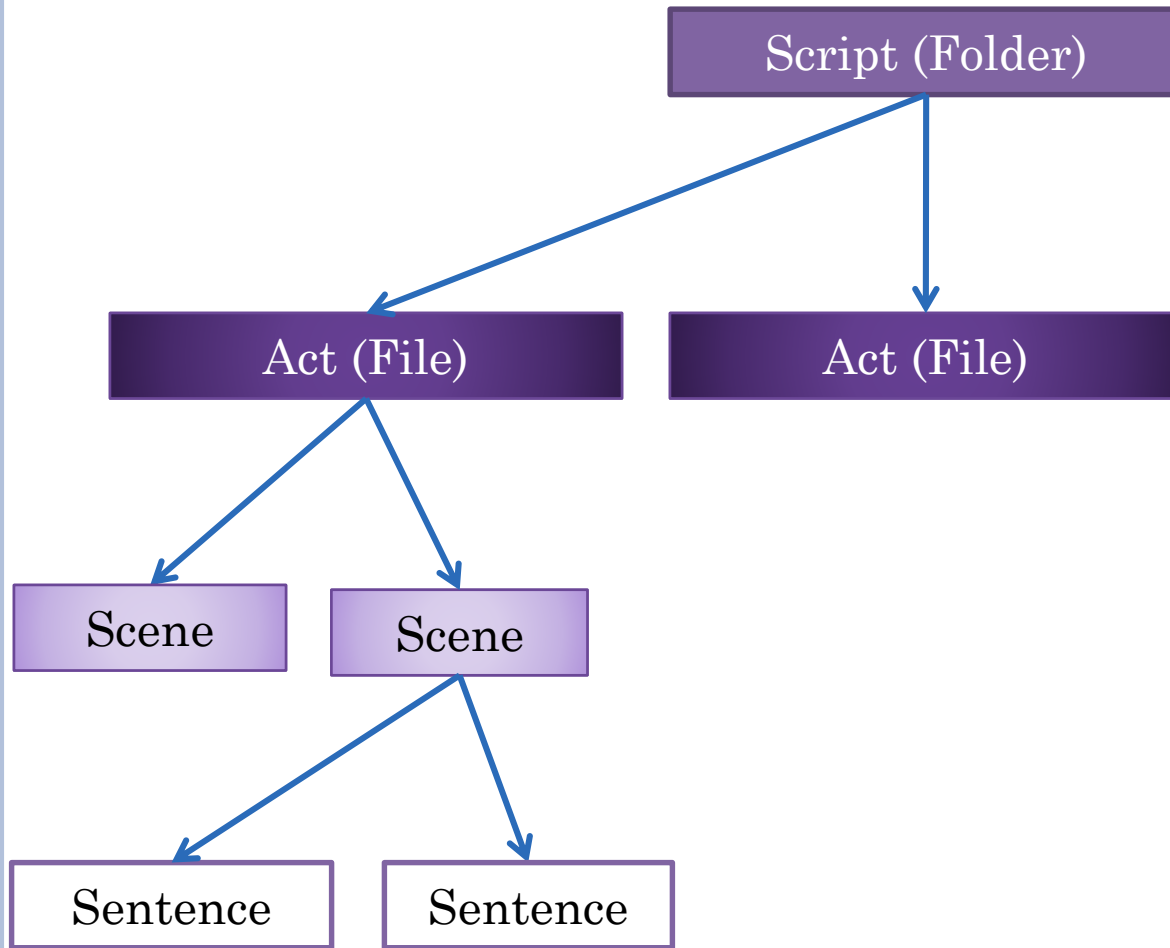
Instructor: Prasun Dewan (FB 150, dewan@unc.edu)



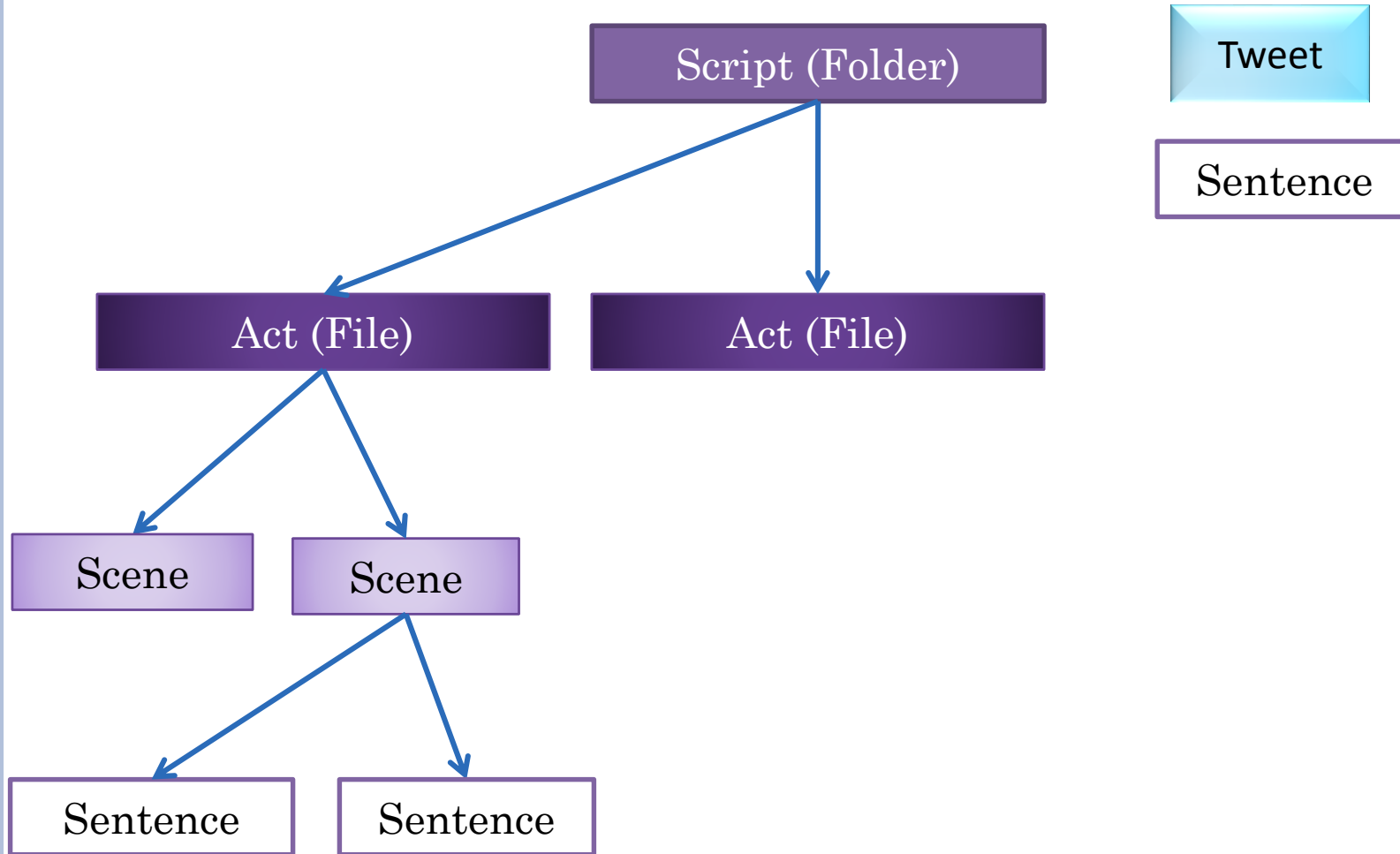
SCRIPTS ANALOGY



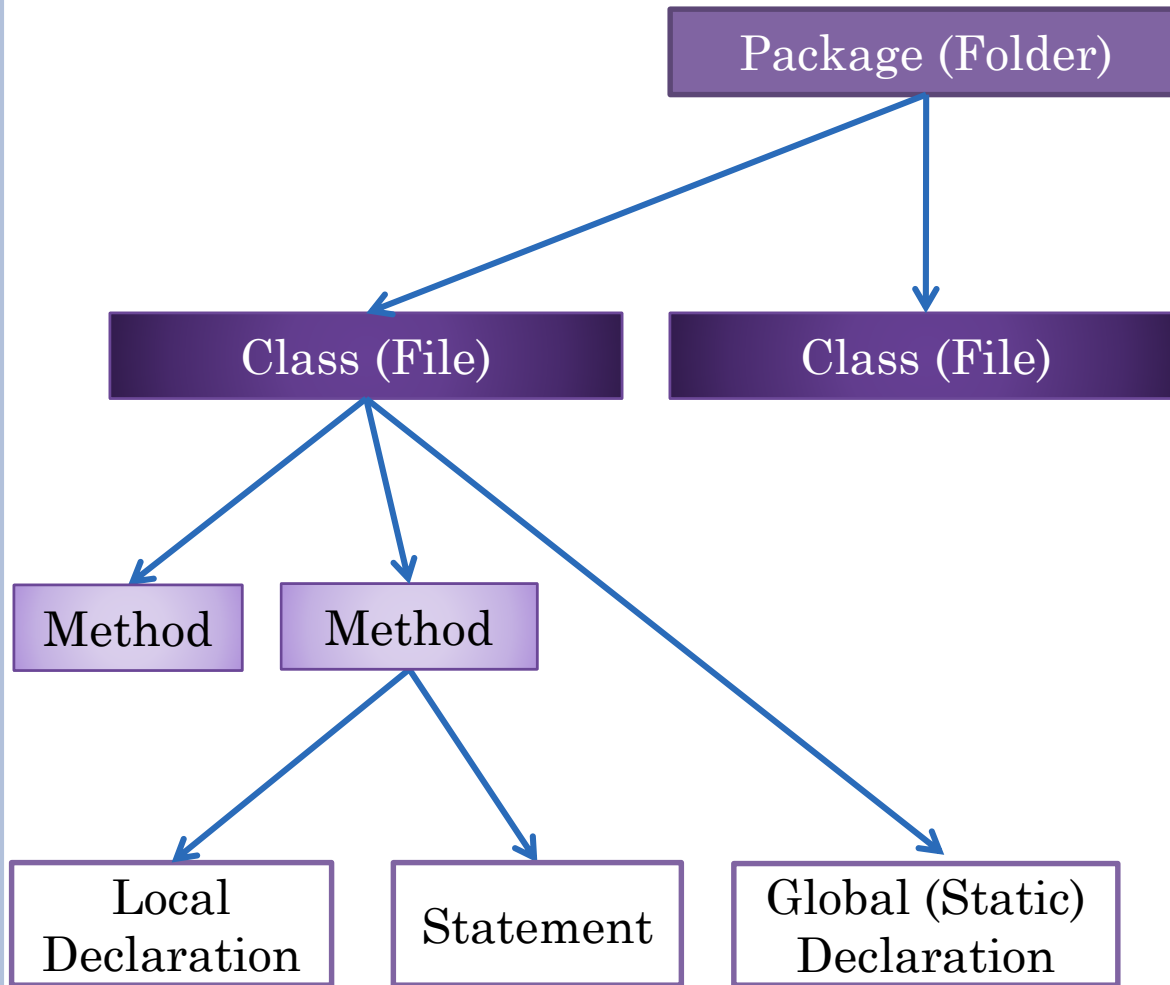
STRUCTURING IN SCRIPTS



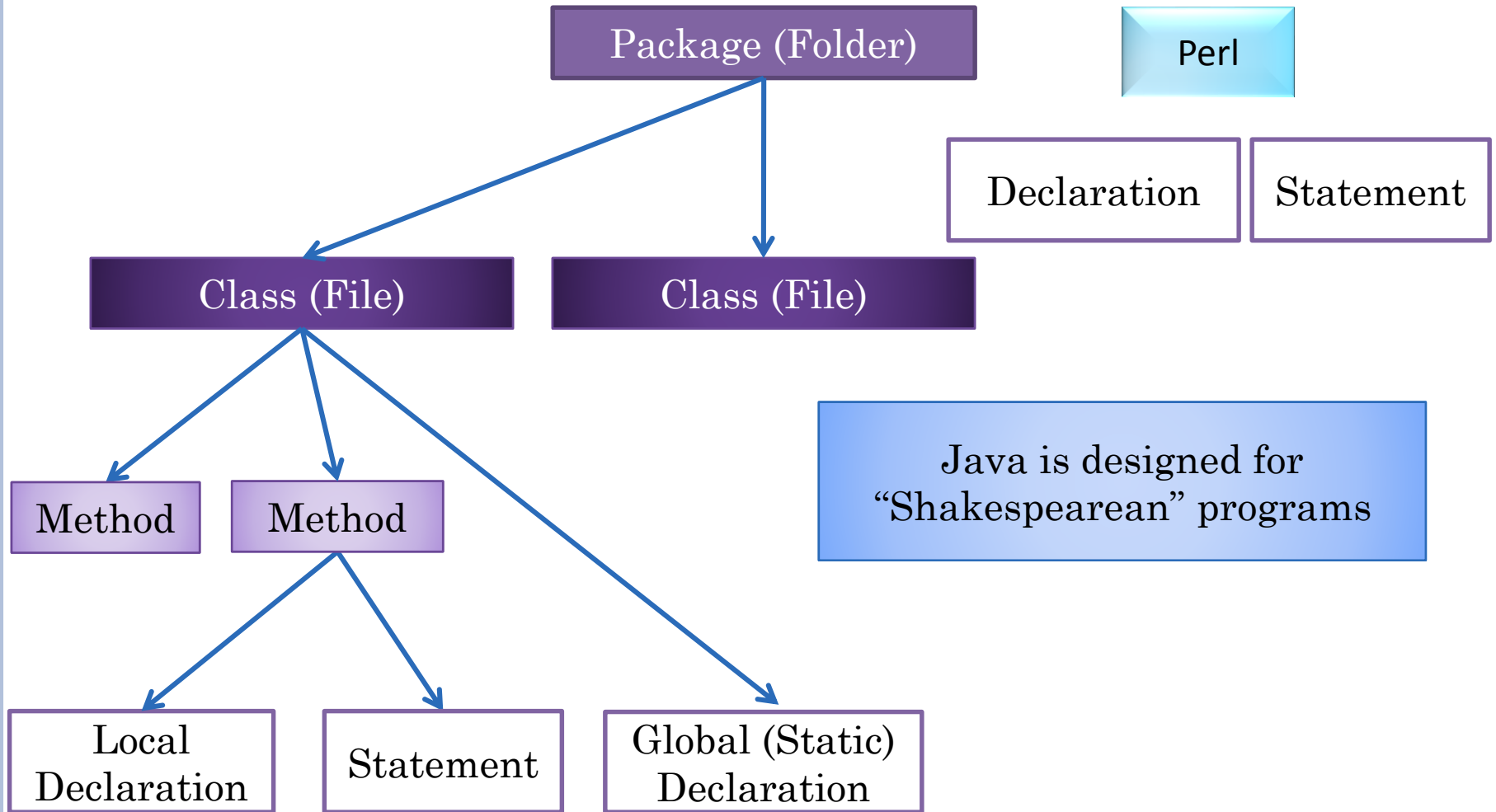
SHAKESPEARE PLAYS VS. TWEETS



STRUCTURING IN JAVA PROGRAMS



JAVA VS. SCRIPTING LANGUAGES



LOOP AND VARIABLE

```
long product = 1;  
while (n > 0) {  
    product *= n;  
    n -= 1;  
}
```



METHOD ENCAPSULATION

```
public static long loopingFactorial(int n) {  
    long product = 1;  
    while (n > 0) {  
        product *= n;  
        n -= 1;  
    }  
    return product;  
}
```



CLASS AND PACKAGE ENCAPSULATION

```
package lectures.java_basics_overview;  
public class Factorials {  
    public static long loopingFactorial(int n) {  
        long product = 1;  
        while (n > 0) {  
            product *= n;  
            n -= 1;  
        }  
        return product;  
    }  
}
```

Package declaration is optional

Every class should be in a
named package

Static?



FACTORIAL USER

```
package lectures.java_basics_overview;  
public class Permutations{  
    public static long permutations(int n, int r) {  
        return Factorials.loopingfactorial(n) /  
            Factorials.loopingFactorial(n-r);  
    }  
}
```

Class

Static (class) method,
invoked on a (static) class



REMEMBERING FACTORIALS

Cannot be
accessed from
arbitrary classes
in other packages

Can be accessed
from arbitrary
classes in other
packages

```
package lectures.java_basics_overview;
public class StaticLoopingFactorialSpreadsheet {
    static int number;
    static long factorial;
    public static int getNumber() {
        return number;
    }
    public static void setNumber(int newVal) {
        number = newVal ;
        factorial = Factorials.LoopinFactorial(number);
    }
    public static long getFactorial() {
        return factorial;
    }
}
```

Computes the factorial of a stored number ~ grade
spreadsheet computing the grade of stored scores



CLASSES AS MODULES

Classes are modules that put walls around the methods and global variables in a program

These walls make only certain methods and variables visible outside

Class (File)

```
graph TD; A[Class (File)] --> B[Method]; A --> C[Method]; C --> D[Local Declaration]; C --> E[Statement]; C --> F[Global (Static) Declaration];
```

Method

Method

Local
Declaration

Statement

Global (Static)
Declaration

Principles about what should
be visible outside?



REMEMBERING FACTORIALS (REVIEW)

Cannot be
accessed from
arbitrary classes
in other packages

Can be accessed
from arbitrary
classes in other
packages

```
package lectures.java_basics_overview;
public class StaticLoopingFactorialSpreadsheet {
    static int number;
    static long factorial;
    public static int getNumber() {
        return number;
    }
    public static void setNumber(int newVal) {
        number = newVal ;
        factorial = Factorials.LoopinFactorial(number);
    }
    public static long getFactorial() {
        return factorial;
    }
}
```

Computes the factorial of a stored number ~ grade
spreadsheet computing the grade of stored scores



CLASSES AS MODULES

Classes are modules that put walls around the methods and global variables in a program

These walls make only certain methods and variables visible outside

Class (File)

```
graph TD; A[Class (File)] --> B[Method]; A --> C[Method]; C --> D[Local Declaration]; C --> E[Statement]; C --> F[Global (Static) Declaration];
```

Method

Method

Local
Declaration

Statement

Global (Static)
Declaration

Principles about what should
be visible outside?



LEAST PRIVILEGE/NEED TO KNOW /ENCAPSULATION/ABSTRACTION

Do not make non
final variable
public

Do not give user
of some code
more rights than
it needs

Export state
through
properties

```
package lectures.java_basics_overview;
public class StaticLoopingFactorialSpreadsheet {
    static int number;
    static long factorial;
    public static int getNumber() {
        return number;
    }
    public static void setNumber(int newVal) {
        number = newVal ;
        factorial = Factorials.LoopinFactorial(number);
    }
    public static long getFactorial() {
        return factorial;
    }
}
```



CLASSIFYING METHODS

Function

Getter

Procedure

Setter

Function

```
package lectures.java_basics_overview;
public class StaticLoopingFactorialSpreadsheet {
    static int number;
    static long factorial;
    public static int getNumber() {
        return number;
    }
    public static void setNumber(int newVal) {
        number = newVal ;
        factorial = Factorials.LoopinFactorial(number);
    }
    public static long getFactorial() {
        return factorial;
    }
}
```



PROPERTIES

Number: int

Factorial: int

```
package lectures.java_basics_overview;
public class StaticLoopingFactorialSpreadsheet {
    static int number;
    static long factorial;
    public static int getNumber() {
        return number;
    }
    public static void setNumber(int newVal) {
        number = newVal ;
        factorial = Factorials.LoopinFactorial(number);
    }
    public static long getFactorial() {
        return factorial;
    }
}
```



PROPERTY WITH NO ASSOCIATED VARIABLE?

Number: int

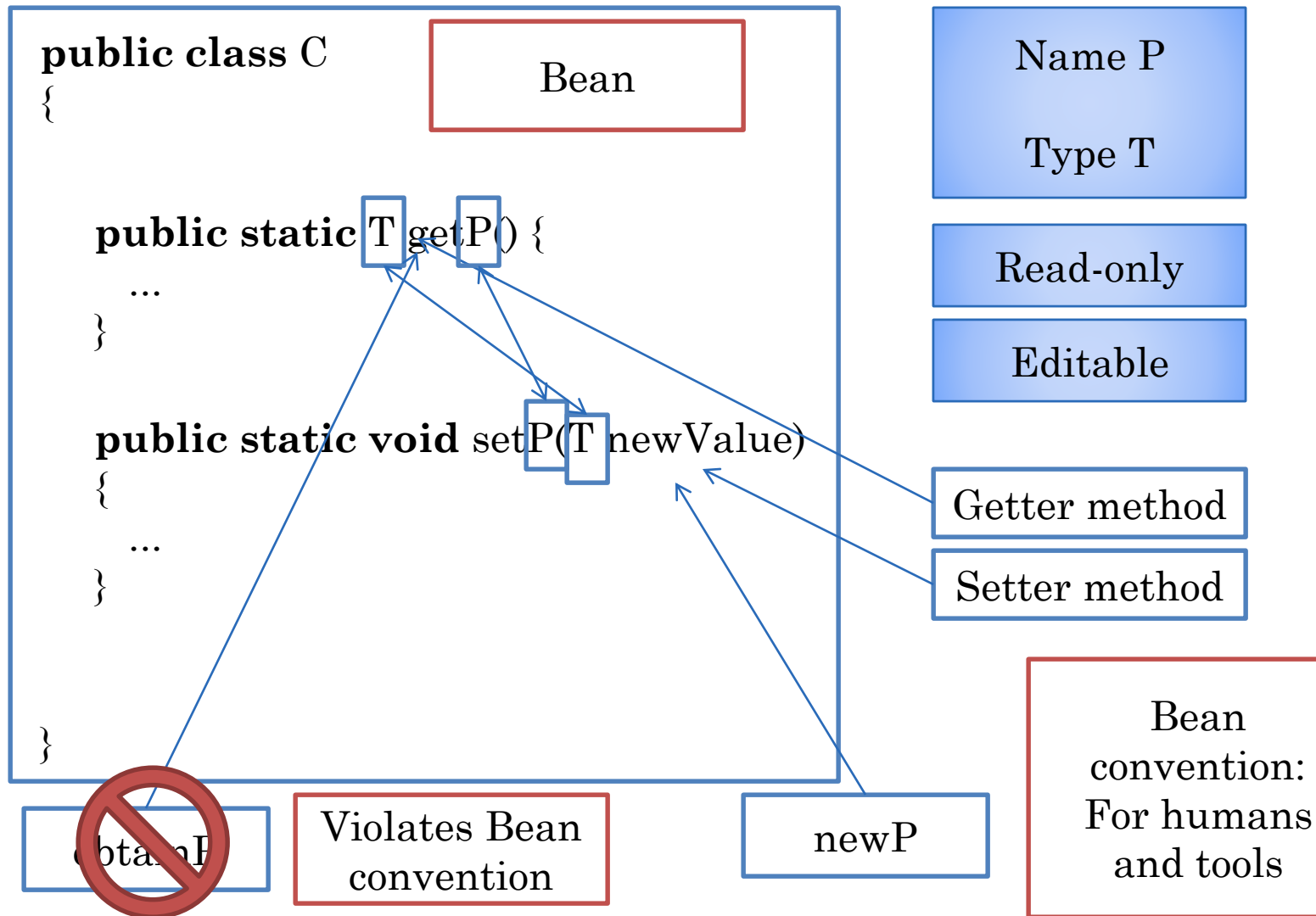
Factorial: int

```
package lectures.java_basics_overview;
public class StaticLoopingFactorialSpreadsheet {
    static int number;
    public static int getNumber() {
        return number;
    }
    public static void setNumber(int newVal) {
        number = newVal ;
    }
    public static long getFactorial() {
        return Factorials.LoopingFactorial(number);
    }
}
```



READ-ONLY AND EDITABLE STATIC PROPERTIES

Typed, Named Unit of Exported Class State



READONLY AND EDITABLE PROPERTIES

Number

editable

Factorial

readonly

```
package lectures.java_basics_overview;
public class StaticLoopingFactorialSpreadsheet {
    static int number;
    static long factorial;
    public static int getNumber() {
        return number;
    }
    public static void setNumber(int newVal) {
        number = newVal ;
        factorial = Factorials.LoopinFactorial(number);
    }
    public static long getFactorial() {
        return factorial;
    }
}
```



DEPENDENT AND INDEPENDENT

Number

Independent

Factorial

Dependent

```
package lectures.java_basics_overview;
public class StaticLoopingFactorialSpreadsheet {
    static int number;
    static long factorial;
    public static int getNumber() {
        return number;
    }
    public static void setNumber(int newVal) {
        number = newVal ;
        factorial = Factorials.LoopinFactorial(number);
    }
    public static long getFactorial() {
        return factorial;
    }
}
```



STORED AND COMPUTED

Number

Stored

Factorial

Computed

```
package lectures.java_basics_overview;
public class StaticLoopingFactorialSpreadsheet {
    static int number;
    public static int getNumber() {
        return number;
    }
    public static void setNumber(int newVal) {
        number = newVal ;
    }
    public static long getFactorial() {
        return Factorials.LoopingFactorial(number);
    }
}
```



MY FACTORIAL

```
package lectures.java_basics_overview;
public class StaticLoopingFactorialSpreadsheet {
    static int number;
    static long factorial;
    public static int getNumber() {
        return number;
    }
    public static void setNumber(int newVal) {
        number = newVal ;
        factorial = Factorials.LoopinFactorial(number);
    }
    public static long getFactorial() {
        return factorial;
    }
}
```

A second
spreadsheet?



REUSE THROUGH COPY AND PASTE

```
package lectures.java_basics_overview;
public class StaticLoopingFactorialSpreadsheet2 {
    static int number;
    static long factorial;
    public static int getNumber() {
        return number;
    }
    public static void setNumber(int newVal) {
        number = newVal ;
        factorial = Factorials.LoopinFactorial(number);
    }
    public static long getFactorial() {
```

```
StaticLoopingFactorialSpreadsheet.setNumber(2);
```

```
StaticLoopingFactorialSpreadsheet2.setNumber(4);
```

Code
duplication



Initial work

Maintenance



CLASSES AS MODULES

Classes are modules that put walls around the methods and global variables in a program

These walls make only certain methods and variables visible outside

Class (File)

```
graph TD; A[Class (File)] --> B[Method]; A --> C[Method]; A --> D[Global (Static) Declaration]; B --> E[Local Declaration]; B --> F[Statement]; C --> F; C --> D;
```

Method

Method

Local
Declaration

Statement

Global (Static)
Declaration



CLASSES AS TYPES

Classes are ways for programmers to define their own types

Values of these types are created by instantiating the classes and share the behavior defined by the type. These values have copies of the dynamic variables defined by the class

Class (File)

Method

Method

Local
Declaration



Statement

Global (Static)
Declaration

A class is a template or blue print for generating instances



CLASSES AS TYPES

```
public class ALoopingFactorialSpreadsheet {
    int number;
    
    
    return number;
}
public void setNumber(int newVal) {
    number = newVal ;
    factorial = Factorials.LoopingFactorial(number);
}
public long getFactorial() {
    return factorial;
}
}
```

No static: instance variables and instance methods

A new set of instance variables dynamically created each time a new instance is created

```
factorial1 =  
    new ALoopingFactorialSpreadsheet ();  
ALoopingFactorialSpreadsheet factorial2 =  
    new ALoopingFactorialSpreadsheet ();  
factorial1.setNumber(2);  
factorial2.setNumber(4);
```

Instance is essentially
an extra parameter to
instance method

Class as type

Dynamically creating a Class instance



JAVA INSTANCEOF BOOLEAN OPERATOR

(new ALoopingFactorialSpreasheet()) instanceof
ALoopingFactorialSpreasheet

true

(new ALoopingFactorialSpreasheet()) instanceof Factorials

false

(new ALoopingFactorialSpreasheet()) instanceof String

false

true and false are values of Java type boolean

instanceof is Java keyword

If class of object o is T then o instanceof T returns true



DOUBLE ROLE OF CLASS

Classes are modules that puts walls around the methods and variables in a program

These walls make only certain methods and variables visible outside

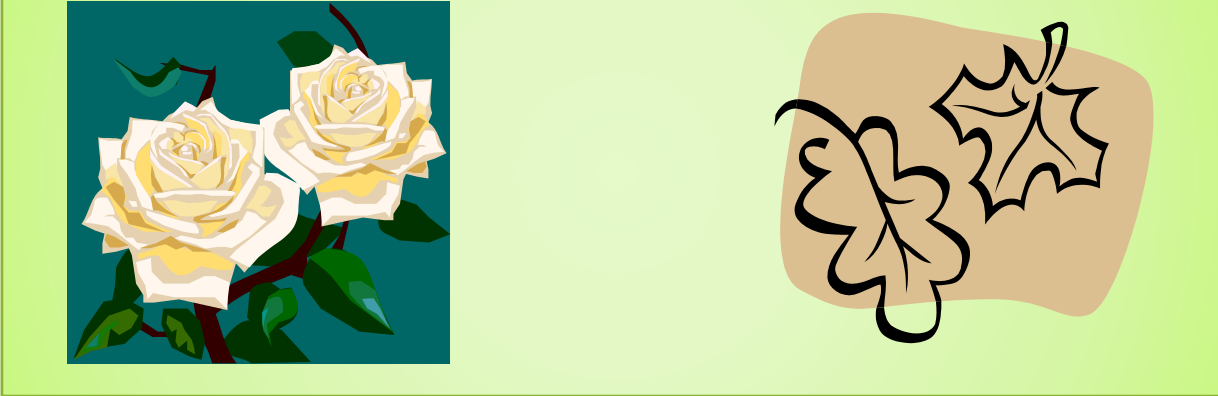
Classes are ways for programmers to define their own types

Values of these types are created by instantiating the classes



PROGRAM OBJECTS~ PHYSICAL OBJECTS

Natural Objects



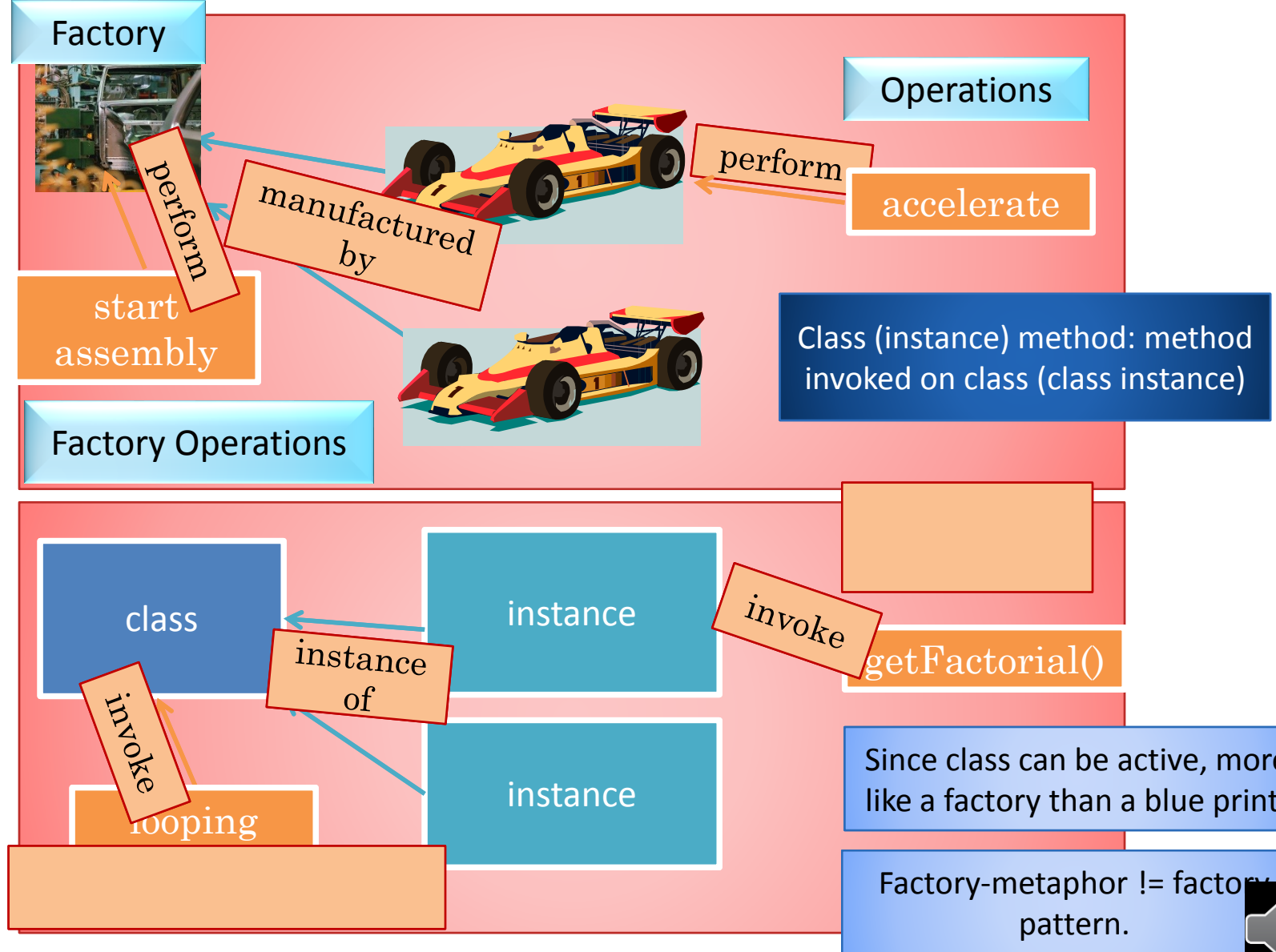
Manufactured Objects



~ Program Components



FACTORY ANALOGY FOR BOTH ROLES



PROPERTIES OF STATIC SPREADSHEET

Number

Factorial

```
package lectures.java_basics_overview;
public class StaticLoopingFactorialSpreadsheet {
    static int number;
    static long factorial;
    public static int getNumber() {
        return number;
    }
    public static void setNumber(int newVal) {
        number = newVal ;
        factorial = Factorials.LoopinFactorial(number);
    }
    public static long getFactorial() {
        return factorial;
    }
}
```



PROPERTIES OF DYNAMIC SPREADSHEET?

Number

Factorial

```
public class ALoopingFactorialSpreadsheet {  
    int number;  
    long factorial;  
    public int getNumber() {  
        return number;  
    }  
    public void setNumber(int newVal) {  
        number = newVal ;  
        factorial = Factorials.LoopinFactorial(number);  
    }  
    public long getFactorial() {  
        return factorial;  
    }  
}
```



READ-ONLY AND EDITABLE STATIC PROPERTIES

Typed, Named Unit of Exported Class State

```
public class C
{
    Bean

    public static T getP() {
        ...
    }

    public static void setP(T newValue)
    {
        ...
    }
}
```

Name P

Type T

Read-only

Editable



READ-ONLY AND EDITABLE INSTANCE PROPERTIES

Typed, Named Unit of Exported Class State

```
public class C  
{
```

Bean

```
    public getP() {  
        ...  
    }
```

```
    public void setP(T newValue) {  
        ...  
    }
```

```
}
```

Name P

Type T

Read-only

Editable

No static



STATIC GLOBAL VARIABLES

Global variable declared outside a method

Static (global) variable created in memory without creating an object

Static methods can access only static global variables as they are invoked on classes and would not know which instance's variables should be accessed

```
static int number = 1;
```

```
static void setNumber (int newVal)
{
    number = newVal;
}
```

Global variable

Local variable



INSTANCE GLOBAL VARIABLES

```
int number = 1;
```

Global
variable

```
void setNumber (int newVal)  
{  
    number = newVal;  
}
```

Local
variable

Instance (global) variable
created in memory as
part of a specific object

Instance methods are invoked on
a specific instance and can access
that instance's variables and also
static variables of the class

Static variables are more global
than instance variables as they
can be accessed by both instance
and static methods



MORE ON INSTANCE VS. STATIC

110 and 401	Object-first Introduction to Programming (8/27, 8/29)	PowerPoint	PDF	Objects Chapter			lectures.objects Package
-------------	--	----------------------------	---------------------	-------------------------------------	--	--	--

110 and 401	State and Properties (9/5, 9/10)	PowerPoint	PDF	State Chapter	Video		lectures.state.properties Package
-------------	---	----------------------------	---------------------	-----------------------------------	-----------------------	--	---

110 (401 Prerequisite)	Static State	PowerPoint	PDF	Static Chapter			lectures.static_state P
-------------------------------	--------------	----------------------------	---------------------	------------------------------------	--	--	---



DOUBLE ROLE OF CLASS

Classes are modules that puts walls around the methods and variables in a program

These walls make only certain methods and variables visible outside

Classes are ways for programmers to define their own types

Values of these types are created by instantiating the classes

In either case they are used to create reusable code

Reusable code is often packaged as libraries to be used by other projects