



OBJECT ORIENTED PROGRAMMING USING JAVA

Lab 2

1

CONTENT

- ? Arrays in Java
- ? Loops in Java
- ? Decision Making in Java: if – else, switch, break and continue.
- ? Introduction to OOP
 - Classes and objects
 - Principles of OOP
 - Class constructor
 - Access modifiers and field modifiers
 - ? public, protected, default, private
 - ? Static (field and method), final field
- ? UML diagrams.

ARRAYS IN JAVA

? In Java, all arrays are dynamically allocated, means you can read the size as an input from the user and then allocate the array.

? Obtaining an array is a two-step process:

1. Variable declaration:

```
int myArr[ ]; or int[ ] myArr;
```

```
int multiArr[ ][ ]; or int [ ][ ] multiArr;
```

1. Memory allocation (Instantiating an Array):

```
myArr = new int[3];
```

```
multiArr = new int[2][3];
```

ARRAYS IN JAVA

? Can make them in one step:

```
int myArr[ ] = new int[3];
```

```
int myArr[ ] = {10,11,12};
```

```
int[ ][ ] multiArr = new int[2][3];
```

```
int[ ][ ] multiArr = {{1,2}, {3,4}};
```

Row	Col	0	1	2
0		-	-	-
1		-	-	-

Row	Col	0	1
0		1	2
1		3	4

Row	Col	0	1	2
0		-	-	-
1				
2				

? For non-uniform 2D array:

```
int[ ][ ] multD = new int[3][ ];
```

```
multD[0] = new int[3];
```

```
multD[1] = new int[2];
```

```
int[ ][ ] arr2 = {{1,2}, {4},{1,1,1}};
```

Row	Col	0	1	2
0		1	2	
1		4		
2		1	1	1

LOOPS IN JAVA

1. While:

```
while (x_IsTrue)
{ // do these statements }
```

2. Do while:

```
do
{ // do these statements
} while (x_IsTrue);
```

3. For:

```
for(int i=0; i<N; i++)
{ //do these statements }
```

IF - ELSE

```
if (condition1) //Boolean expression
{
    statement1;
    statement2;
}
else if (condition2)
{
    statement3;
    statement4;
}
else
    statement5;
```

SWITCH CASE

`switch` (Expression)

{

`case` value1:

statement1;

`break`;

it

// Duplicate case values are not allowed

//optional, if omitted, then execution will
//continue on into the next case and execute
//regardless of the *expression* value..

`case` value2:

statement2;

`break`;

`default`:

statement3;

//optional

}

INTRODUCTION TO OOP

- ? Object oriented programming is an approach that provides a way of modularizing programs by creating partitioned memory area of both data and functions that can be used as templates for creating copies of such modules on demand.
- ? Think about everything as objects of classes!

CLASSES AND OBJECTS

What Is an Object?

Class

Student



Objects



Name = Peter
Age = 3
Color = white
Sex = male

- ✓ Peter can eat more foods
- ✓ Peter can drink more water
- ✓ Peter can run fast



Name = John
Age = 4
Color = brown
Sex = male

- ✓ John can eat less foods
- ✓ John can drink more water
- ✓ John can run slow



Name = Julia
Age = 2
Color = white
Sex = female

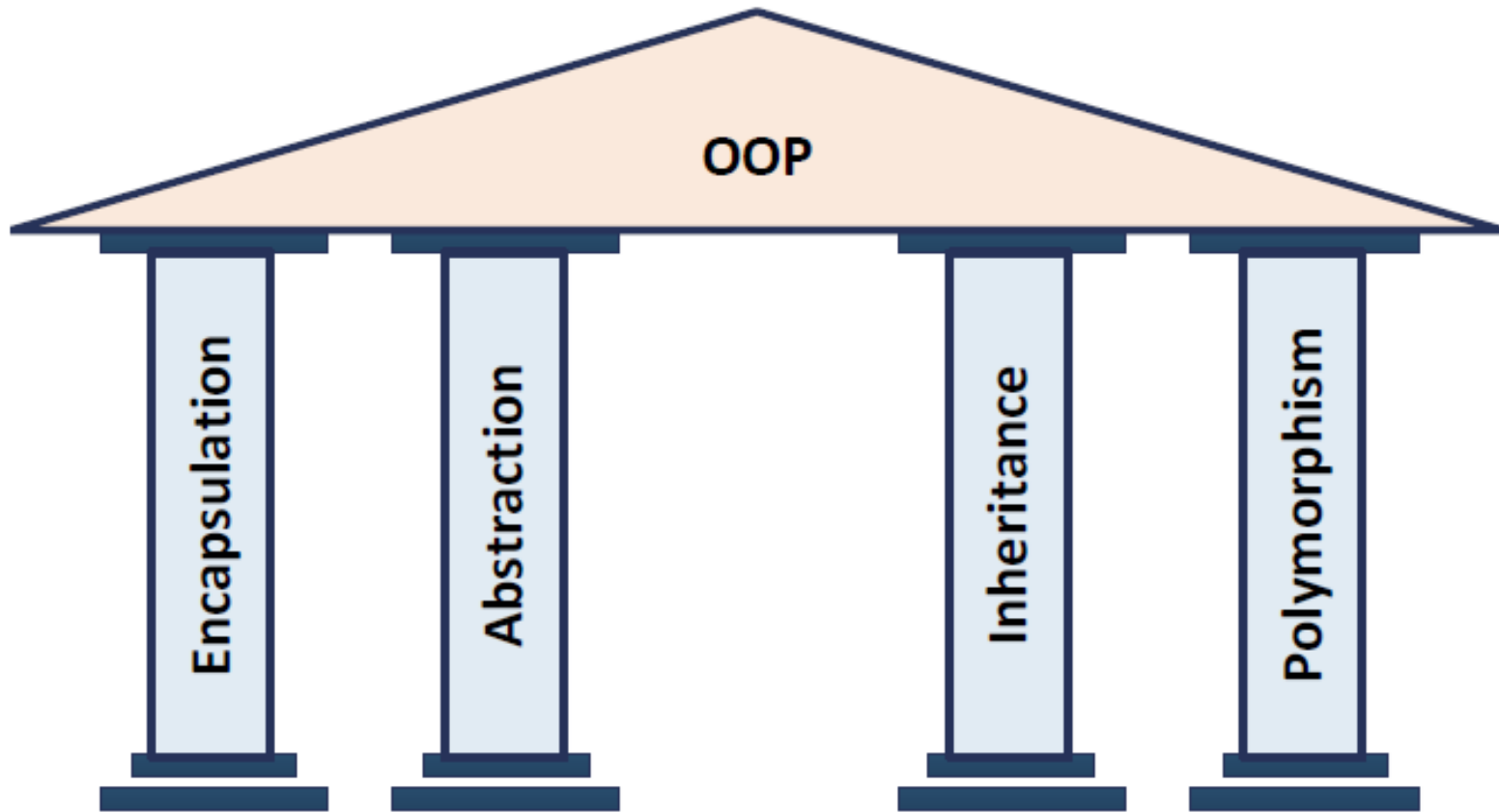
- ✓ Julia can eat less foods
- ✓ Julia can drink less water
- ✓ Julia can run fast

- ✓ Object are same as real world objects.
- ✓ Java objects has both states and behaviors.

CLASSES AND OBJECTS

- ? A class is a kind of data type that you can define yourself.
- ? The class defines the characteristics of the object and the operations that are performed on/by the object.
- ? Objects are variable of type class.
- ? Every object belongs to (is an instance of) a class.
- ? A program is a set of objects telling each other what to do, by sending messages.

PRINCIPLES OF OOP



CREATING FIRST CLASS

Access
modifier

Class
name

```
public class Rectangle {
```

```
    int len, width;  
    String color;
```

Attributes

```
    public Rectangle() {
```

```
        len = 0;  
        width = 0;  
        color = null;
```

Constructor
(special
method)

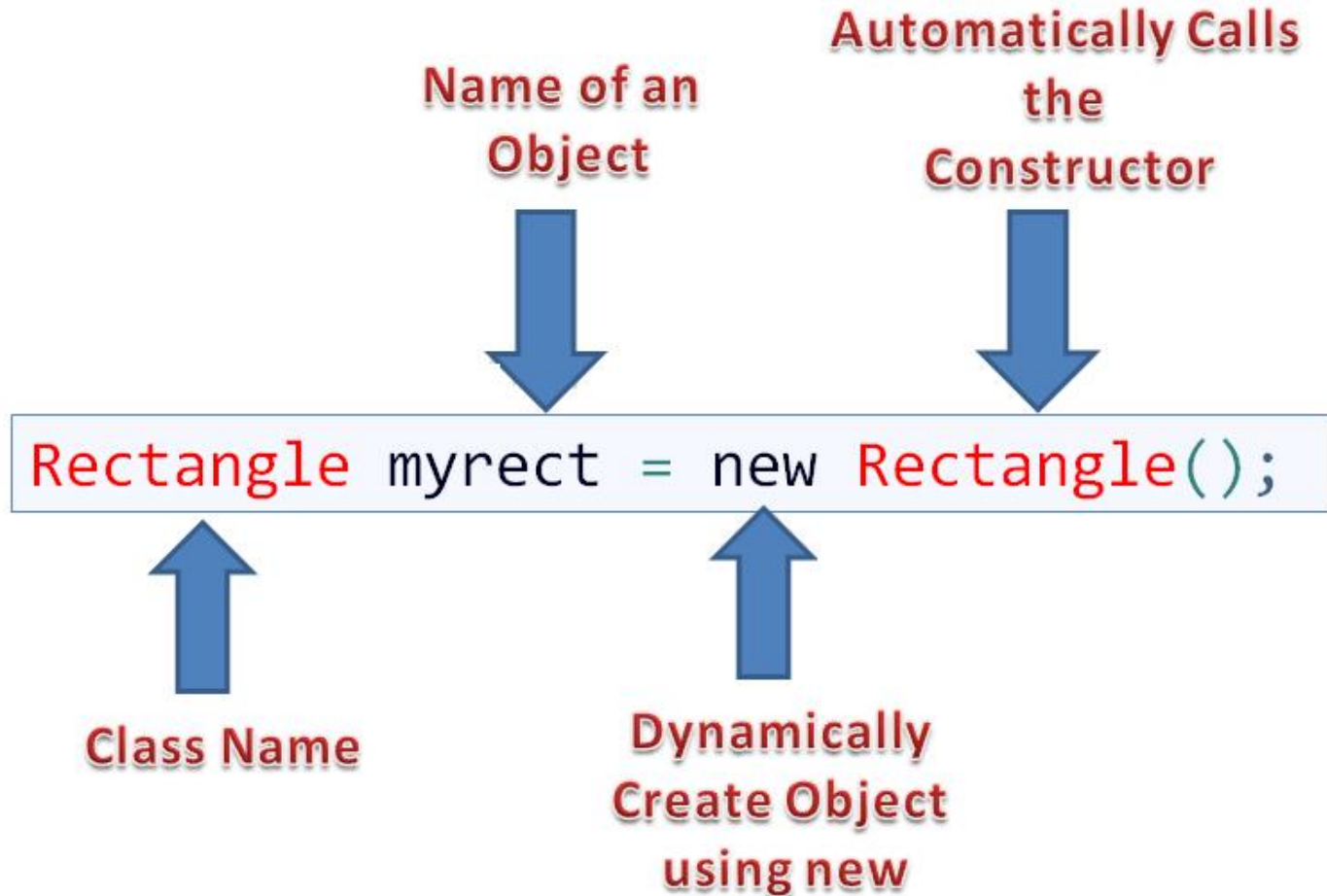
```
    public void setColor(String color) {
```

```
        this.color = color;
```

Method

```
}
```

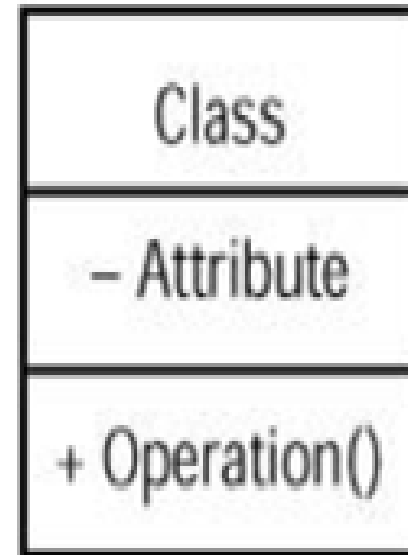
MAKING OBJECT FROM CLASS



UML – CLASS DIAGRAM

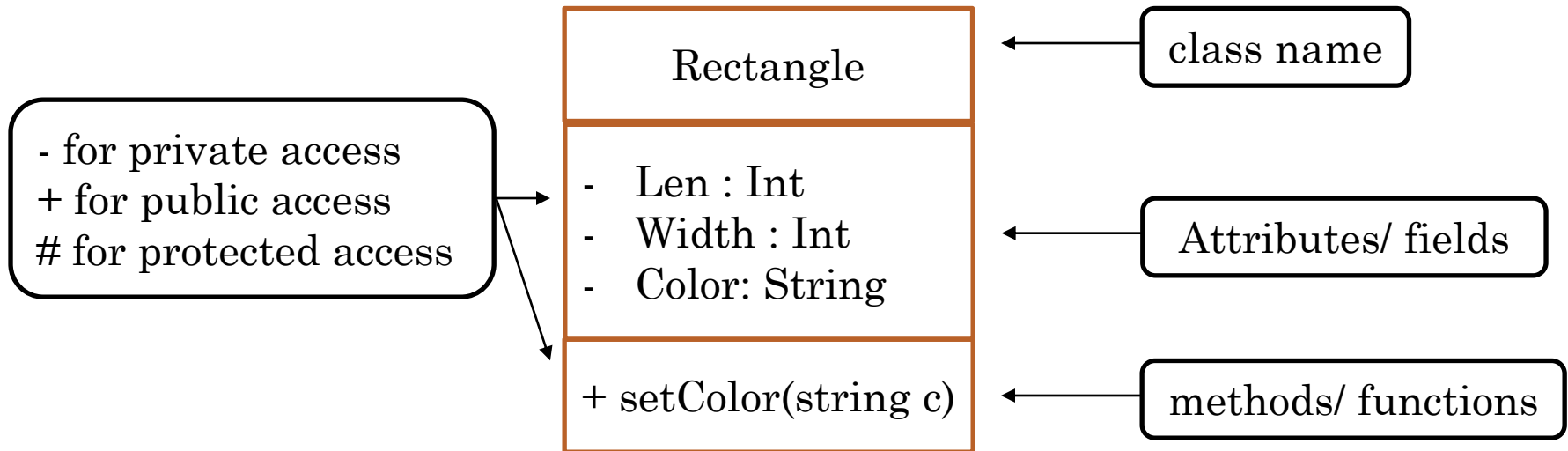
- ? In OOP, a class contains member functions and member data(attributes).
- ? We can represent the class by graphical design called UML (Unified Modeling Language).
- ? Each class is represented by a rectangle subdivided into three compartments

1. Class name in top box.
Unique Name, No spaces, Short
Use italics for an abstract class name.
2. Attributes (optional) in middle box.
Should include all fields of the object.
3. Operations / Methods (optional) in bottom box.
May omit trivial (get/set) methods.
Should not include inherited methods.



UML – CLASS DIAGRAM

? Represent the rectangle class in UML



? There is a plugin for NetBeans that can be used to generate a class diagram for your code and vice versa.

CONSTRUCTOR

- ? Is a member function with the **same name** as the class, **can take arguments** but has **no return type (not even void)**.
- ? A special method that is used to **initialize a newly created object** and is called automatically just after the memory is allocated for the object.
- ? It can be used to **initialize** the objects to **required or default values** at the time of object creation.
- ? A class can have **any number of constructors** that differ in parameter lists (*Constructor overloading*).
- ? **Public constructors** enable any other class to create object from it. While **Private constructors** prevent other classes from making object using this constructor.

EXAMPLE ON CONSTRUCTORS

```
? public Rectangle() {  
    len = 0;  
    width = 0;  
    color = null;  
}
```

Default constructor, takes no parameters.
Public means other classes will be able to
create object from the class.

```
? public Rectangle(int l, int w) {  
    len = l;  
    width = w;  
    color = null;  
}
```

Parametrized constructor. Takes any
number of parameters.

```
? public Rectangle(Rectangle r) {  
    len = r.len;  
    width = r.width;  
    color = r.color;  
}
```

Copy constructor. Takes only one parameter
which is of the same type of the class and set
all the values of data members to make copy
of the object.

Unlike C++, java doesn't create a copy
constructor if you don't write your own.

NOTES ON CONSTRUCTORS

- ? If you do not define ANY constructor, then the compiler will provide a default constructor and initializes the member variables with their default values.
- ? If you defined any other type of constructors (parameterized or copy); then the compiler will NOT provide any default constructors.

```
public class Rectangle {  
    int len, width;  
    public Rectangle(int l, int w){           %parametrized constructor  
        len = l;  
        width = w;  
    }  
  
    public static void main(String[] args) {  
        Rectangle rec = new Rectangle( ); %Error  
    }  
}
```

ACCESS MODIFIERS

- ? Java provides several access modifiers to set access levels for classes, variables, methods, and **constructors**. The four access levels are:
1. **No modifiers** are needed to be Visible to the package, this is default.
 2. **Private:** Visible to the class only.
 3. **Public:** Visible to the world, anywhere.
 4. **Protected:** Visible to the package and all subclasses.

EXAMPLE USING PRIVATE MODIFIER

```
Package myPack {
    class A{ //No constructors are written then a default one will be
            //generated automatically with public access
        private int data = 40;
        private void msg( ){
            System.out.println("Hello java");
        }
    }
}

public class Simple{
    public static void main(String args[]){
        A obj=new A(); //will call the default constructor that
                       //was generated automatically
        System.out.println(obj.data); //Compile Time Error as "data"
                                       //attribute is private
        obj.msg(); //Compile Time Error as msg() function is private
    }
}
}
```

FIELD MODIFIERS: STATIC

- ? A **static member** has only one copy of instance variables that is shared among all the objects of the class whereas a **non-static member** has its own copy of instance variable at each object.
- ? Static fields are be accessed by the class name but also can be accessed with object name.
- ? One of the most common use for static fields is to count number of instantiated objects created from the class.

FIELD MODIFIERS: STATIC

```
public class Item {  
    static int numOfItems= 0;  
    public Item() {  
        numOfItems ++;  
    }  
  
    public static void main(String[] args) {  
        Item item1=new Item();    // numOfItems=1  
        Item item2=new Item();    // numOfItems=2  
        Item.numOfItems++;        // numOfItems=3  
    }  
}
```

FILED MODIFIERS: FINAL

- ? Final modifier is used to declare a **constant** attribute that takes its value once at the constructor or while declaring this variable, and then can not be changed after that at all.

Ex. Math.PI and Math.E (These fields are declared in class Math with the modifiers public, final and static.)

FILED MODIFIERS: FINAL

```
package oop_lab2;
class Rectangle {
    int len, width;
    final String msg;

    public Rectangle(){
        len = 0;
        width = 0;
        msg = "This is constant string";
    }
}
```

and will
change after that

//value assigned
//never

```
public class OOP_lab2 {
    public static void main(String[] args) {
        Rectangle rec = new Rectangle();
        rec.msg = "try to change!";
    }
}
```

//Error: cannot assign
//value to final variable

HANDS-ON 1... 20 MINUTES

- ? Generate a Package with two public classes: **Clinic class and Doctor class**.
- ? **Doctor class** should contain a string for name, that must be constant (final), a string for working day (ex: Saturday, Sunday ..etc.) and a static counter to count number of generated Doctors objects.
- ? Implement a parametrized constructor that sets the name and working day.
- ? **Clinic class** should contain (1) a private array of 3 doctors . (2) A function Insert doctor , that accepts a doctor object and adds it in the array , and also prints the value of the counter after each insertion.(3) A function display_doctors that prints all the doctors names that work on a given day.
- ? Create a class with the main function in another package.
- ? In main generate a Clinic object and scan data of the 3 doctors and display all doctors who works on saturday.

SOLUTION

```
package my_system;

public class Doctor {
    final String name;
    static int counter = 0;
    String working_day;

    public Doctor(String n,
String day )
    {
        name = n;
        working_day=day;
        counter++;
    }
}
```

```
Package my_system;
public class Clinic
{
    Private Doctor[]doctors=new Doctor[3];
    Public void insert_doctor (Doctor d , int i)
    {
        doctors[i]=d;
        System.out.println(Doctor.counter);
    }
    Public Display_docs( String day)
    {
        for ( int i=0;i<3; i++)
        {
            if(doctors[i].working_day.equals(day))
                System.out.println(doctors[i].name)
        }
    }
}
```

Note that: The doctor object is created outside the Clinic and passed for insertion... Aggregation relationship

SOLUTION CONT...

```
package main_pack;
import java.util.Scanner;
import my_system.*;
public class my_main {

public static void main(String[] args) {
Clinic my_clinic=new Clinic();
    Scanner input=new Scanner (System.in);

for ( int i=0;i<3; i++)
{
Doctor d;
System.out.println("Please enter the doctor's name and working day");
String Name=input.next();
String Day=input.next();
d=new Doctor(Name,Day);
my_clinic.insert_doctor(d,i);
}
my_clinic.Display_docs("Saturday");
```

GETTERS AND SETTERS

- ? The class's private members can not be accessed by other classes (but we can access them using their getters and setters).
- ? Why using getters and setters?
 - Main problem with making field public instead of getter and setter is that it violates **Encapsulation** by exposing internals of a class.
 - Once you exposed internals of class you can not change internal representation or make it better until making change in all client code.
- ? You can write them using a wizard on the IDE.

GETTERS AND SETTERS

```
class Rectangle {
    private int len, width;
    final String msg;

    public Rectangle() {
        len = 0;
        width = 0;
        msg = "This is a rectangle";
    }
}

public class OOP_lab2 {
    /**
     * @param args the
     */
    public static void main(String[] args) {
        // TODO code a
        Rectangle rec = new Rectangle();
        //rec.msg = "t
    }
}

p_lab2.Rectangle > width >
-lab2 (run) x
run:
0
1
2
BUILD SUCCESSFUL (total time: 0 second
```

Right click on the line you want to place your generated code. Chose insert code.

GETTERS AND SETTERS

```
class Rectangle {  
    private int len, width;  
    final String msg;  
  
    public Rectangle () {  
        len = 0;  
        width = 0;  
        msg = "This is cons  
    }  
}
```

- Generate
- Constructor...
- Logger...
- Getter...
- Setter...
- Getter and Setter...
- equals() and hashCode()...
- toString()...
- Delegate Method...
- Override Method...
- Add Property...

Choose which function you need, and it will be written for you.

GETTERS AND SETTERS

```
class Rectangle {  
    private int len, width;  
    final String msg;  
}
```

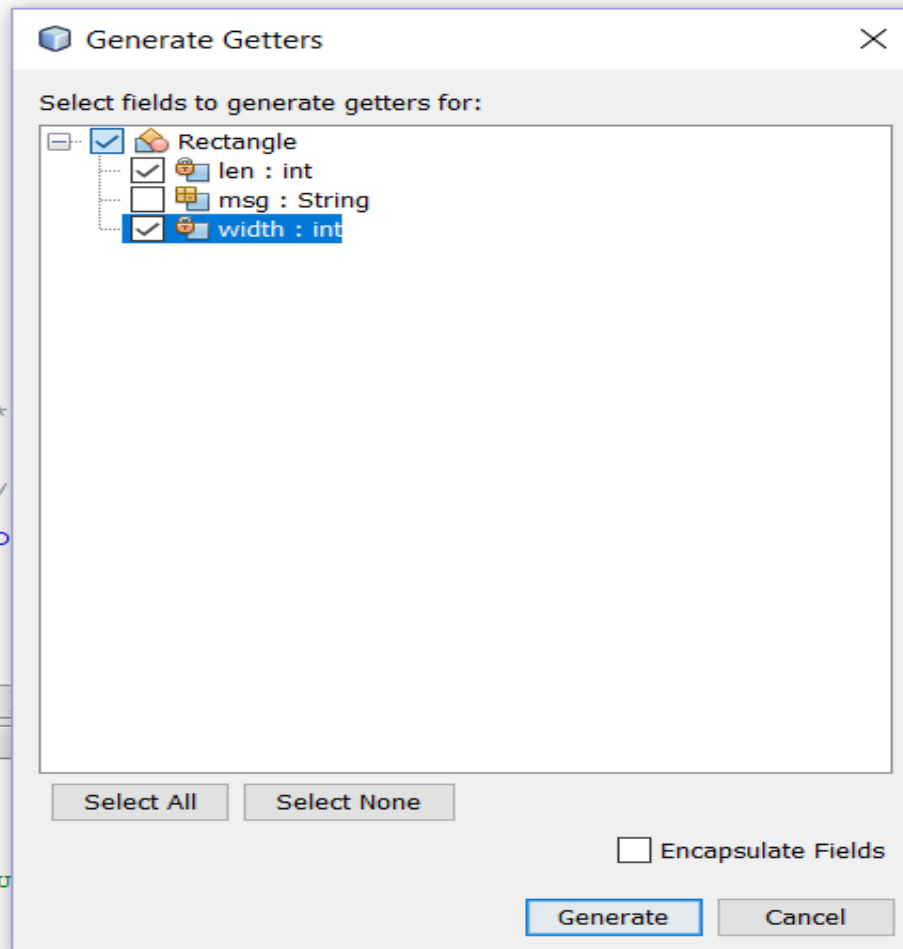
```
public  
/**  
 *  
 */  
pub
```

lab2.Rectangle

b2 (run) x

n:

ILD SUCCESSFU



Select the attributes you want to generate code for them.

Click “Generate”

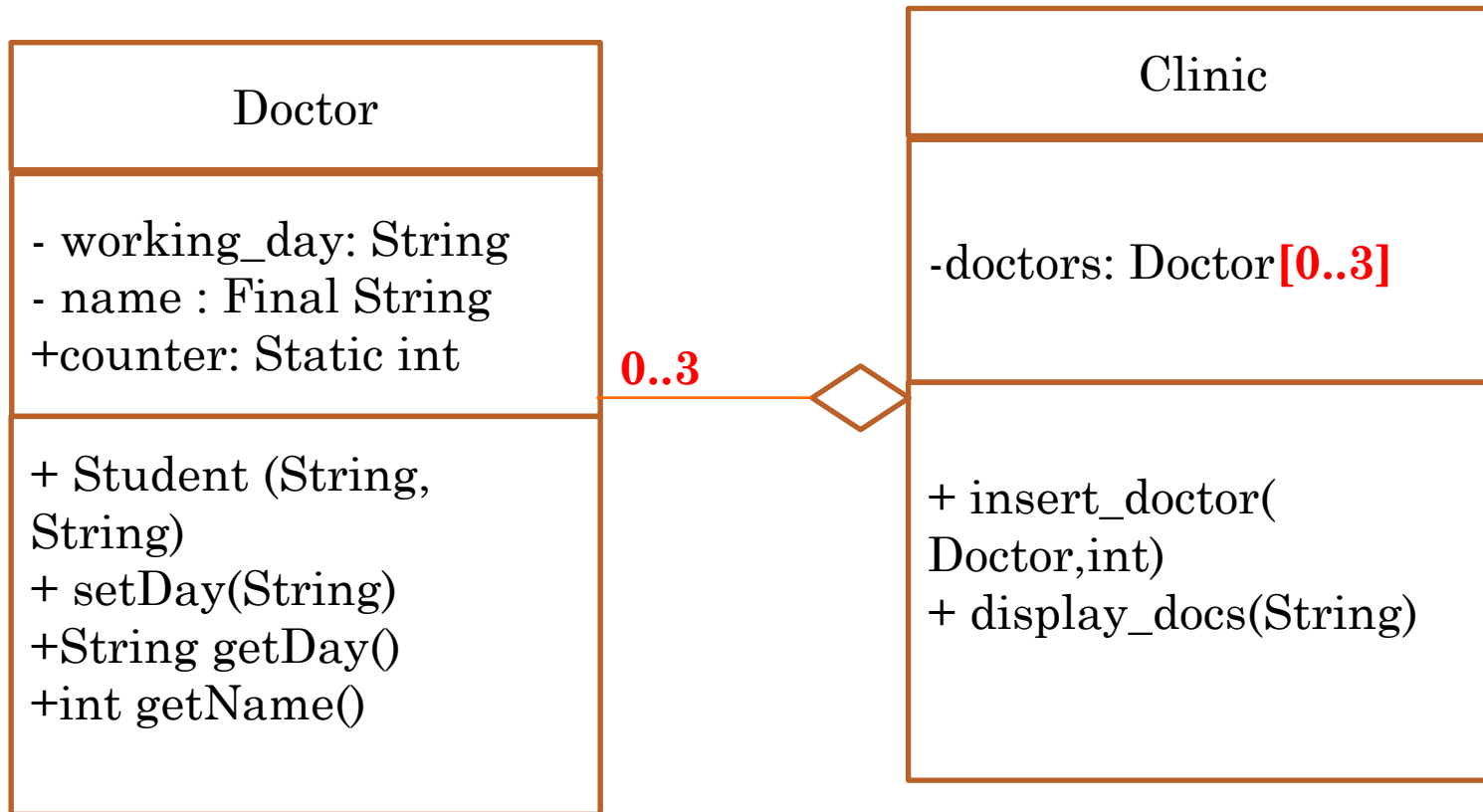
GETTERS AND SETTERS

```
class Rectangle {  
    private int len, width;  
  
    public void setLen(int len) {  
        this.len = len;  
    }  
  
    public void setWidth(int width) {  
        this.width = width;  
    }  
  
    public int getLen() {  
        return len;  
    }  
  
    public int getWidth() {  
        return width;  
    }  
}
```


HANDS-ON 2..... 10 MINUTES

- ? Modify last program so that the name and working_day fields **in Doctor class** are private.
- ? Write a getter method for name and working_day.
- ? Write a setter method for working_day. (name is a final attribute, so it takes its value once in the constructor only).
- ? Draw a UML diagram for your system.

UML - SOLUTION



Aggregation Relation

QUESTIONS

