Object Oriented Programming using Java

Lab 2

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

- <u>For non-uniform 2D array:</u>

```
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			

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

Loops in Java

1. While:

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

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

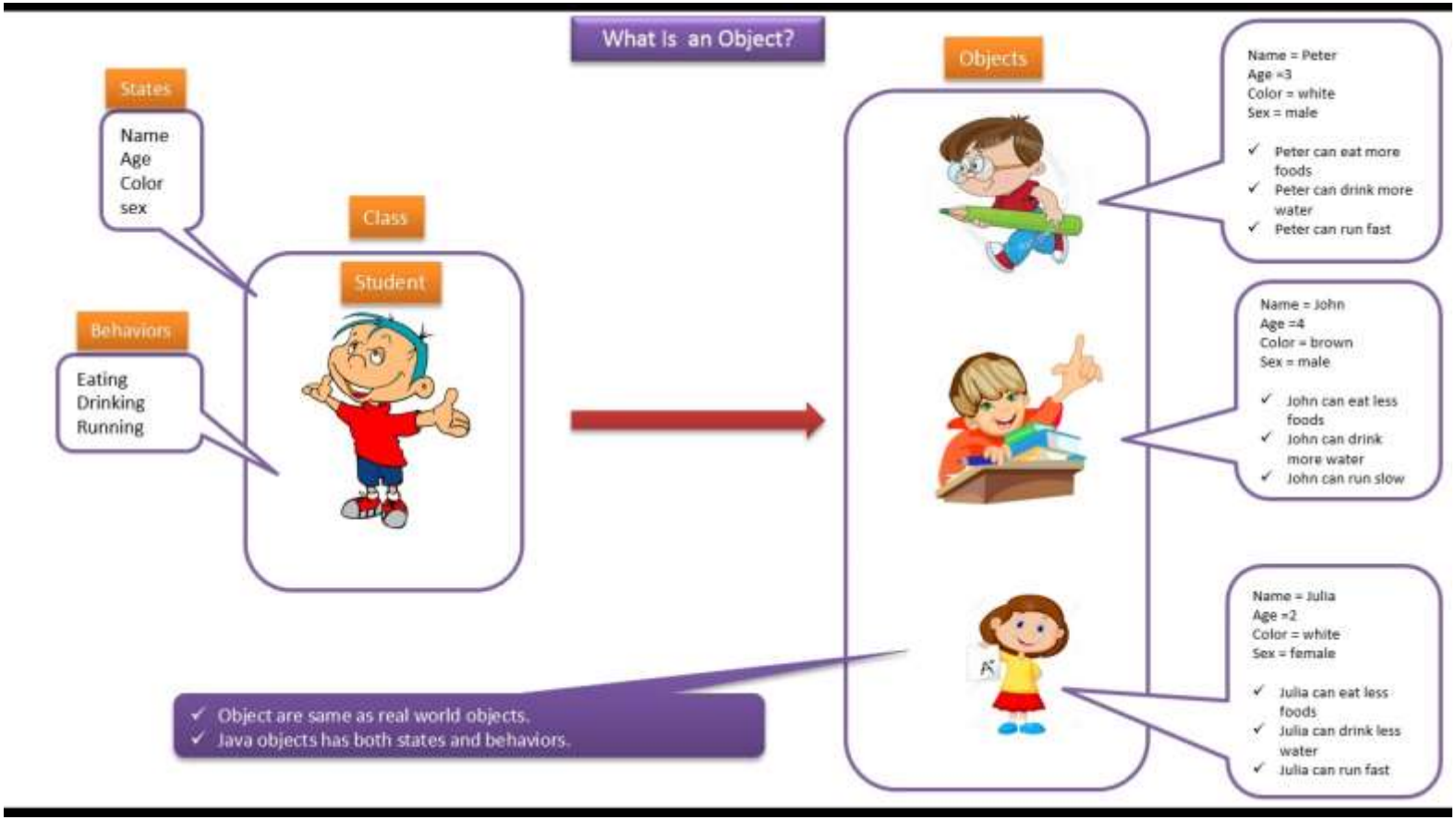
```
switch (Expression)
{
    case value1:            // Duplicate case values are not allowed
            statment1;
            break;          //optional, if omitted, then execution will
                            //continue on into the next case and execute
it                          //regardless of the expression value..

    case value2:
            statment2;
            break;

    default:                //optional
    statment3;
}
```

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!

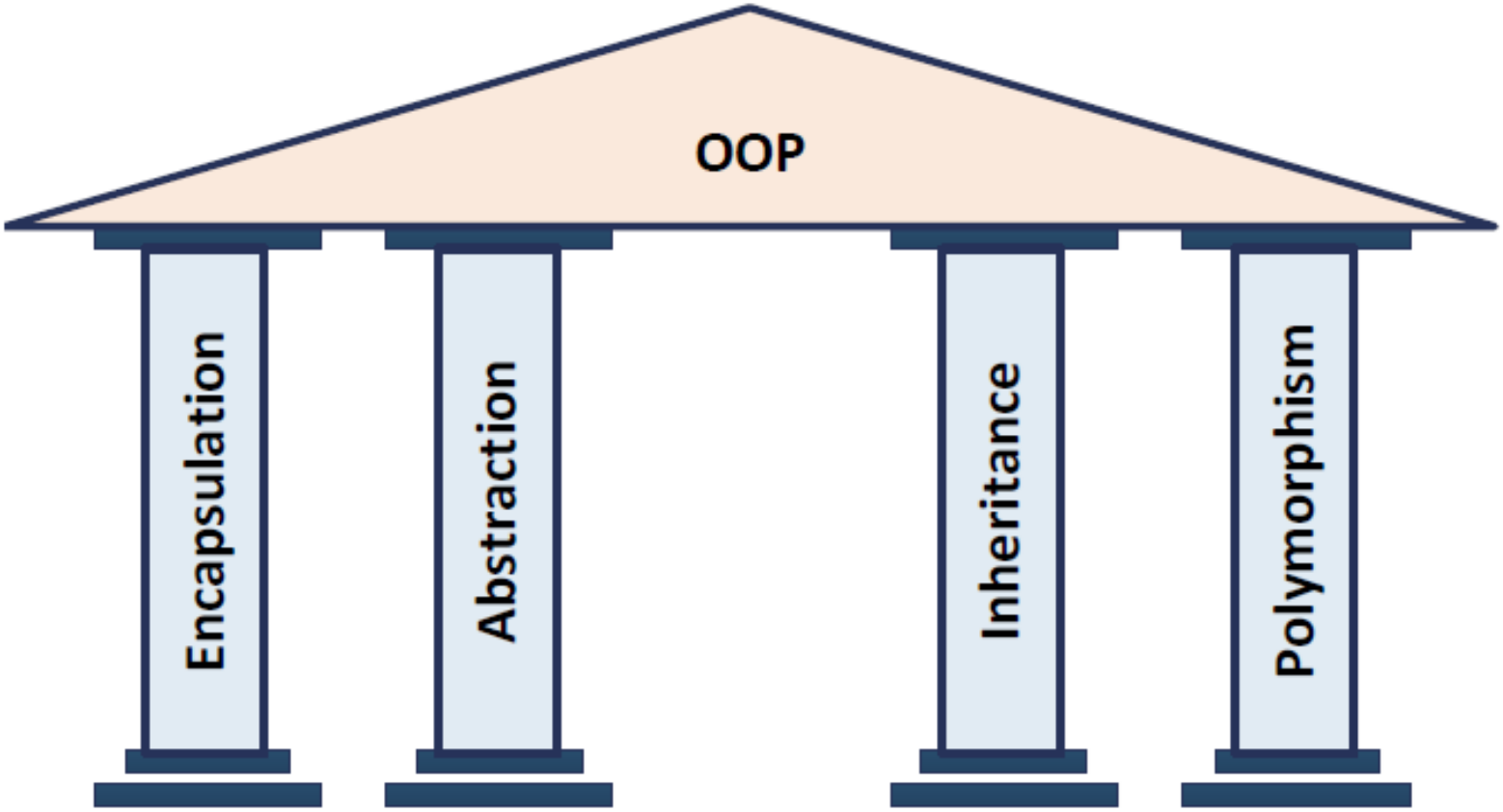
Classes and Objects



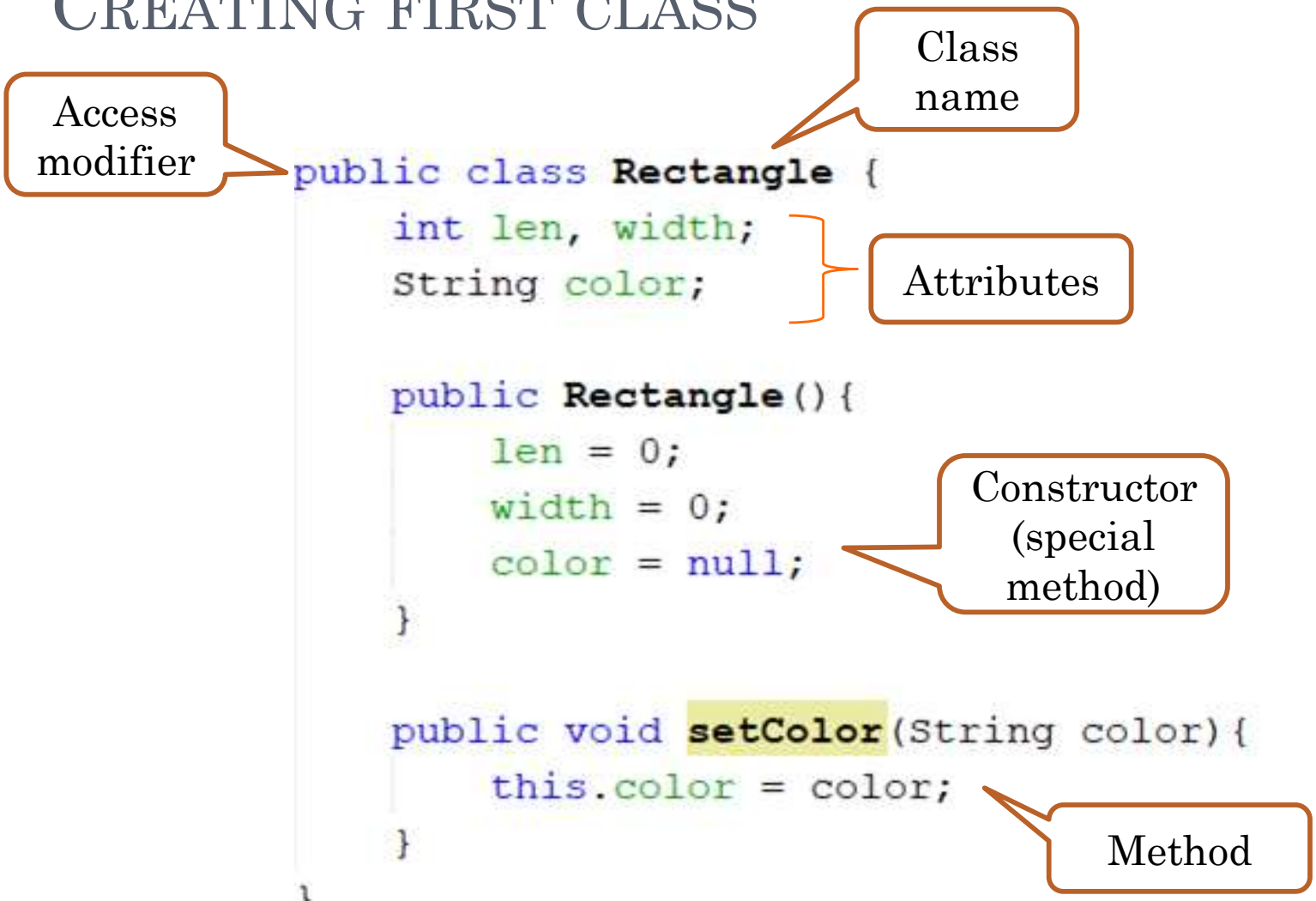
Classes and Objects

- 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.

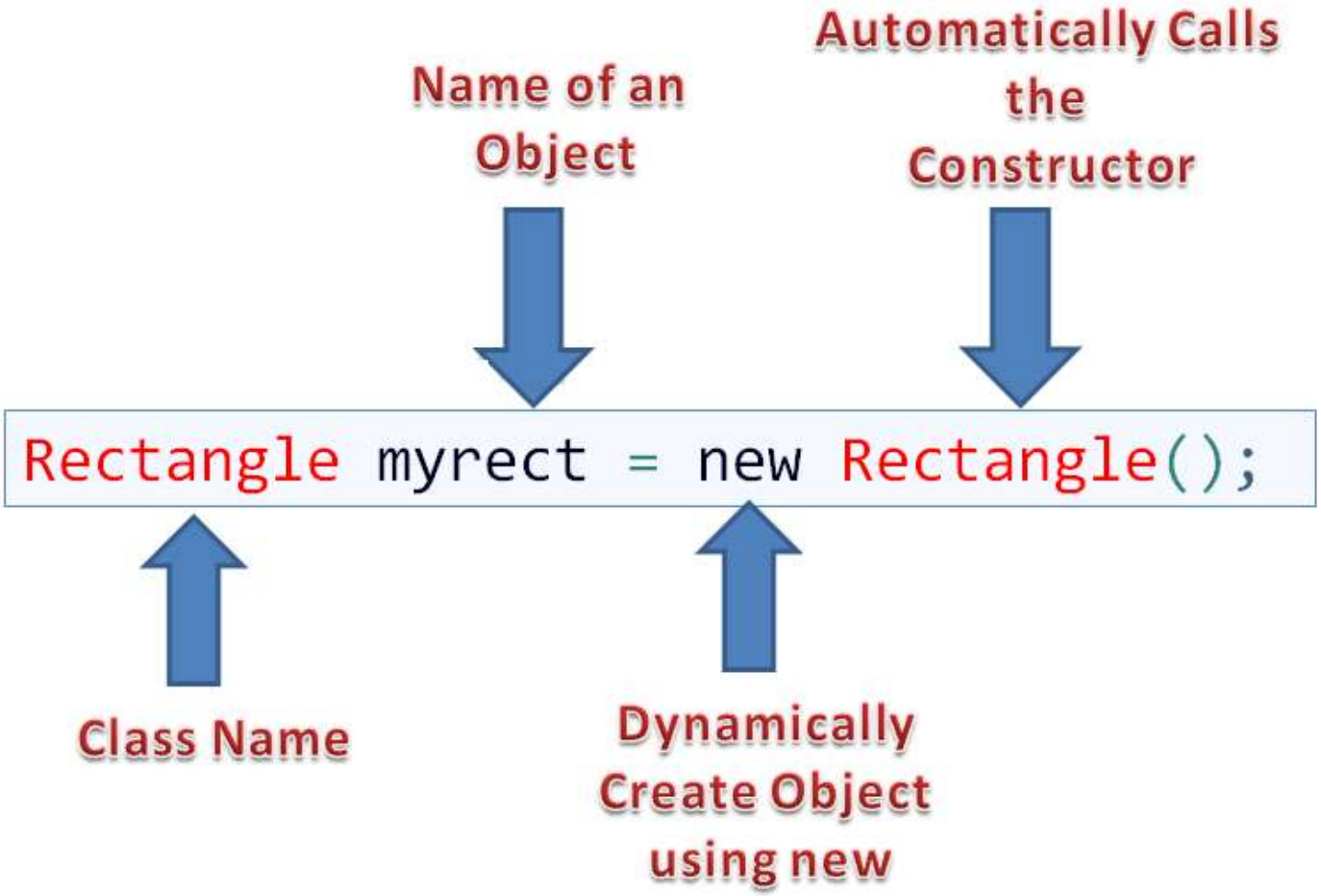
Principles of OOP



CREATING FIRST CLASS



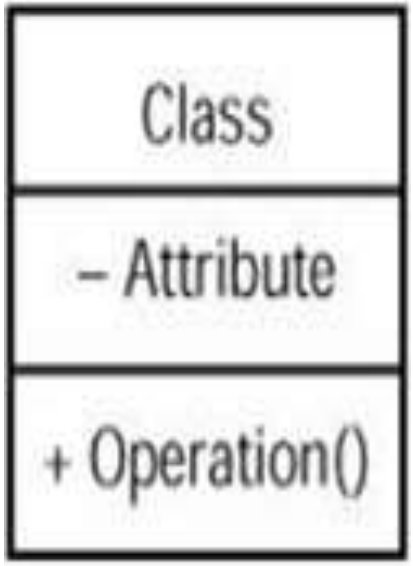
Making object from class



UML – Class Diagram

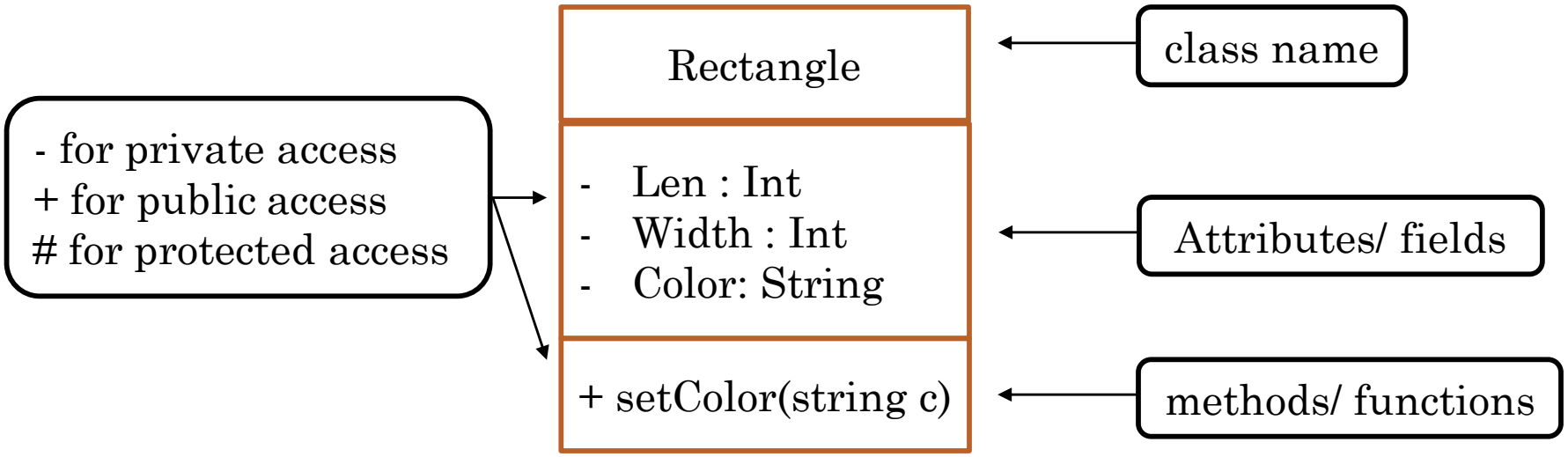
- In OOP, a class can contains member functions and member data(attributes).
- We can represent the class by graphical design called UML (Unified Modeling Language).
- Each class is represented by a rectangle subdivided into three compartments

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 time 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";                    //value assigned
and will                                                                  //never
change after that
        }
    }

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

    }
```

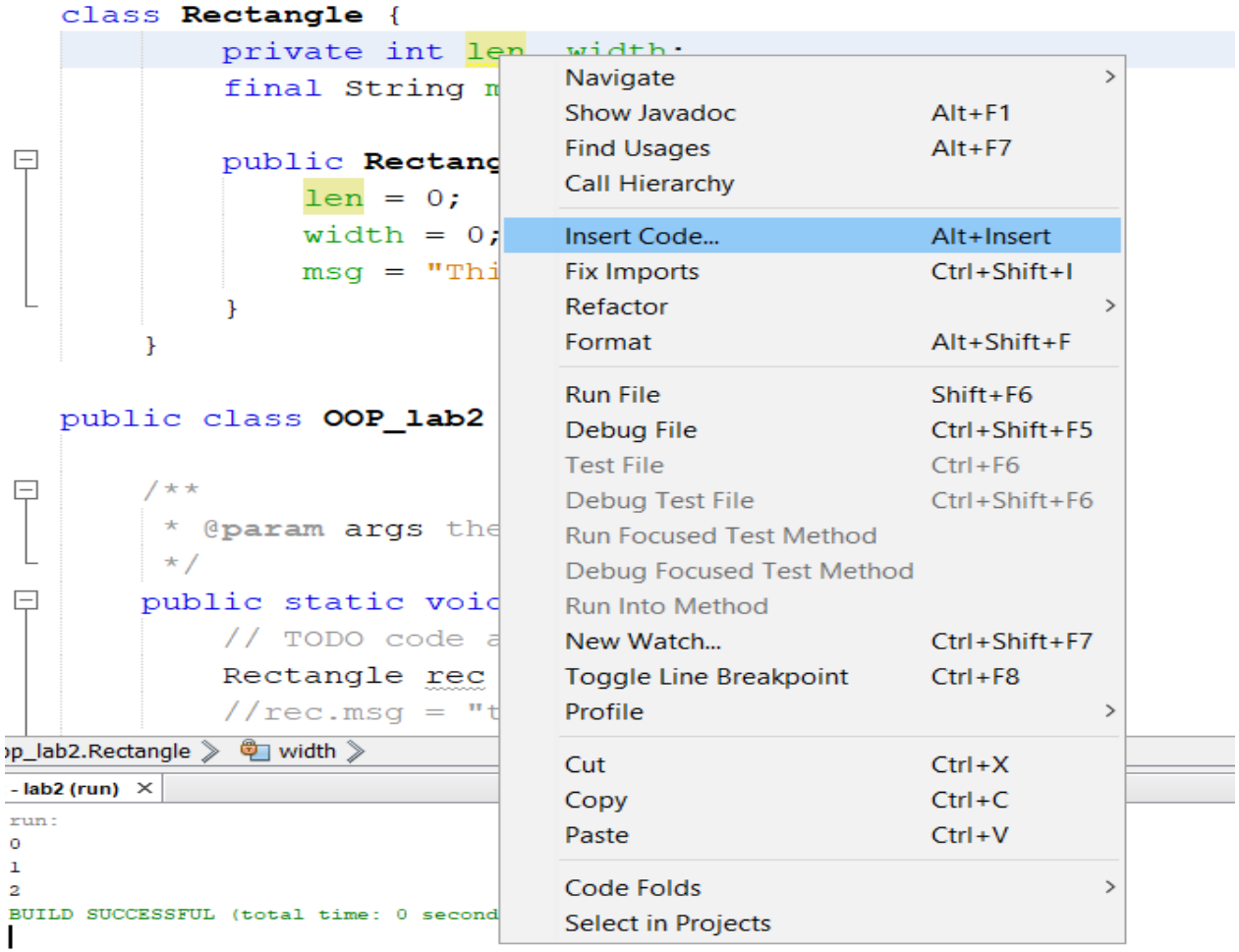
Getters and Setters

- 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.

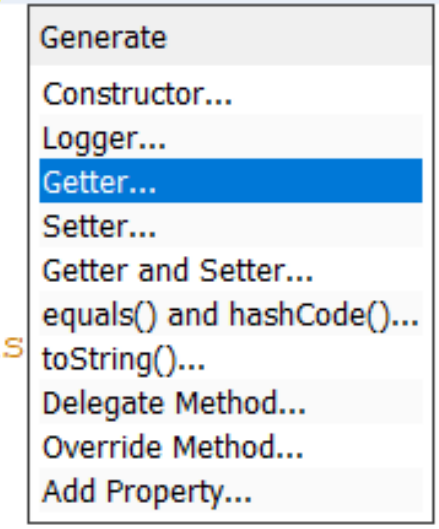
Getters and Setters



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

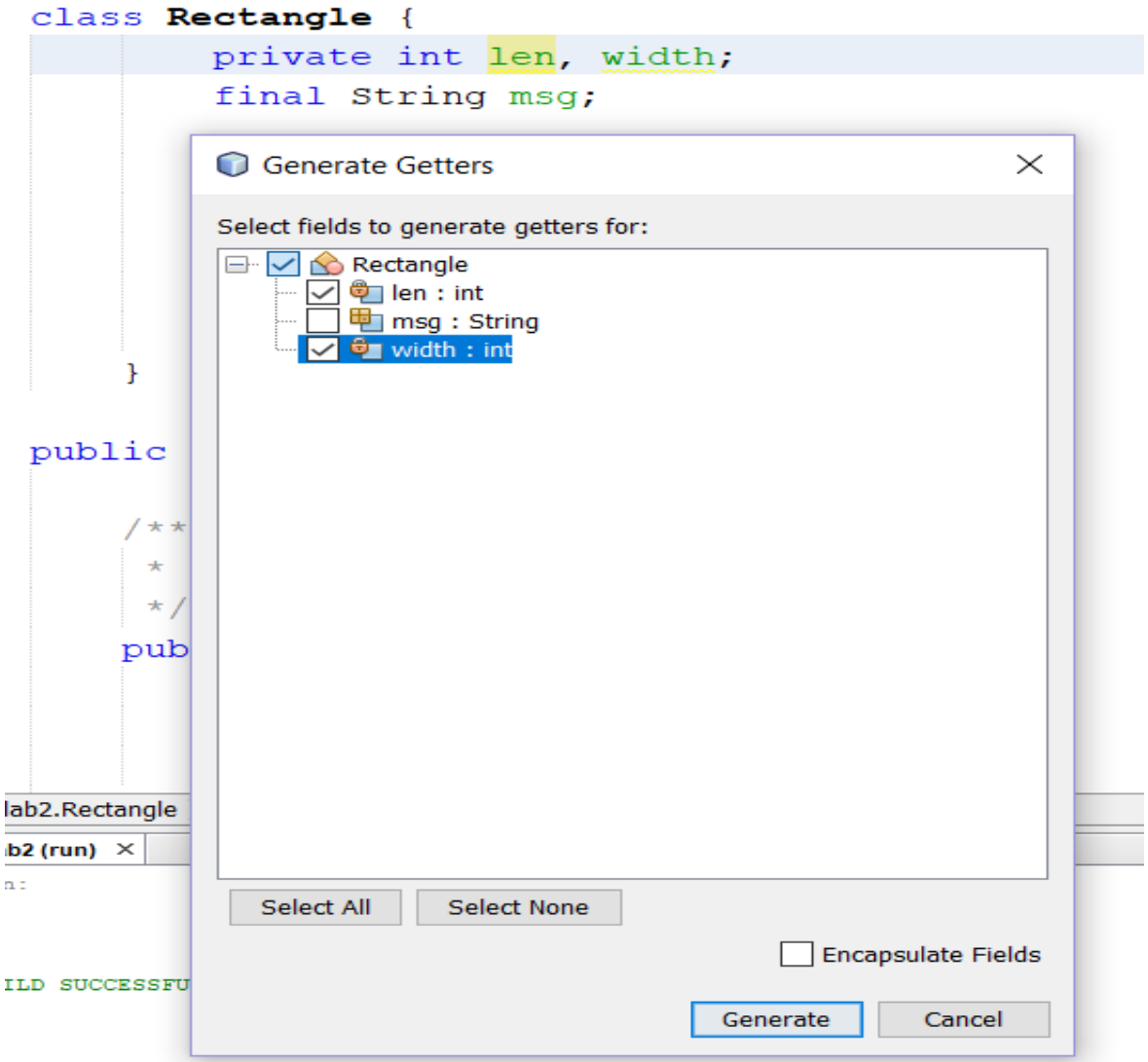
Getters and Setters

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

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



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

Getters and Setters



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;
    }
}
```

QUESTIONS