

# Software Development II

Lecture 7– Introduction to OOP : Classes & Objects

*Reading List : Java for everyone Ch8*

# From Last Week

- Methods
- Variable Scope
- Method overloading
- Recursion

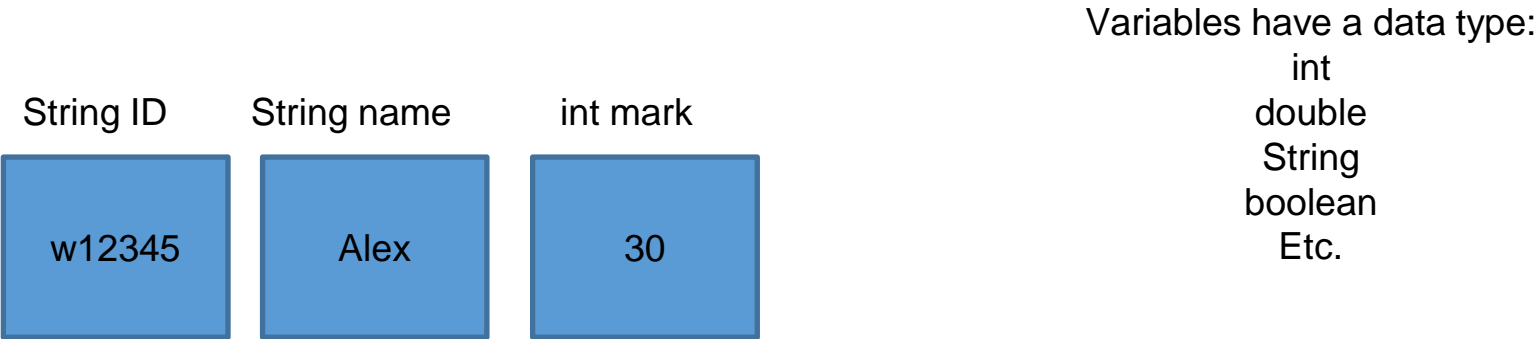
# This week

## Classes and Objects

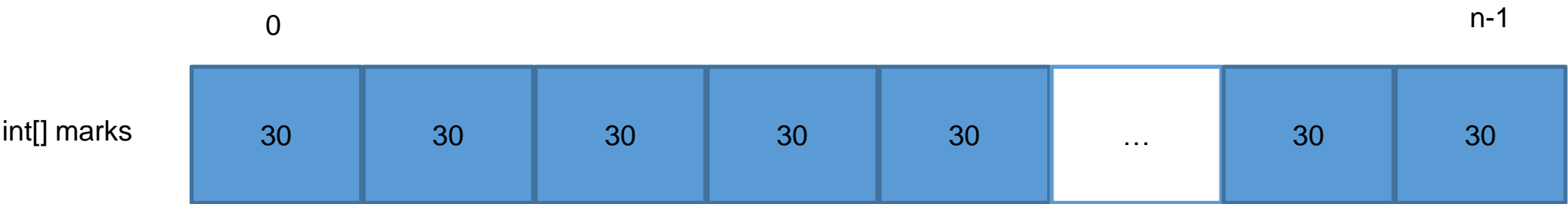
- State and behaviour
- Declaration and use
- Class constants and variables
- Constructors
- *this*
- Arrays with objects
- Class methods
- Methods declaration and use
- Public vs private
- Field modifiers
- Classes with other classes

# Introduction

In week 2 we learnt about variables:



In week 3 we learnt that when we have multiple variables representing the same thing, we can put them in an array of the same type:



# Limitations with arrays

- Let's imagine we want to create a program that stores all the information for all students in this module

	0	1	2	3	4	5		550
String[] ID	w12345	w12346	w12347	w12348	w12349	w12350	...	w12895
String[] name	Alex	Charlie	Taylor	Tony	Billy	Chris	...	Max
int[] marksICT	90	50	45		75	20	...	80
int[] marksCW	70		60	50	65	50	...	60

These students also attend other modules, how will you store their marks and info?  
What happens if a student does not have a mark?



**Very difficult to maintain**

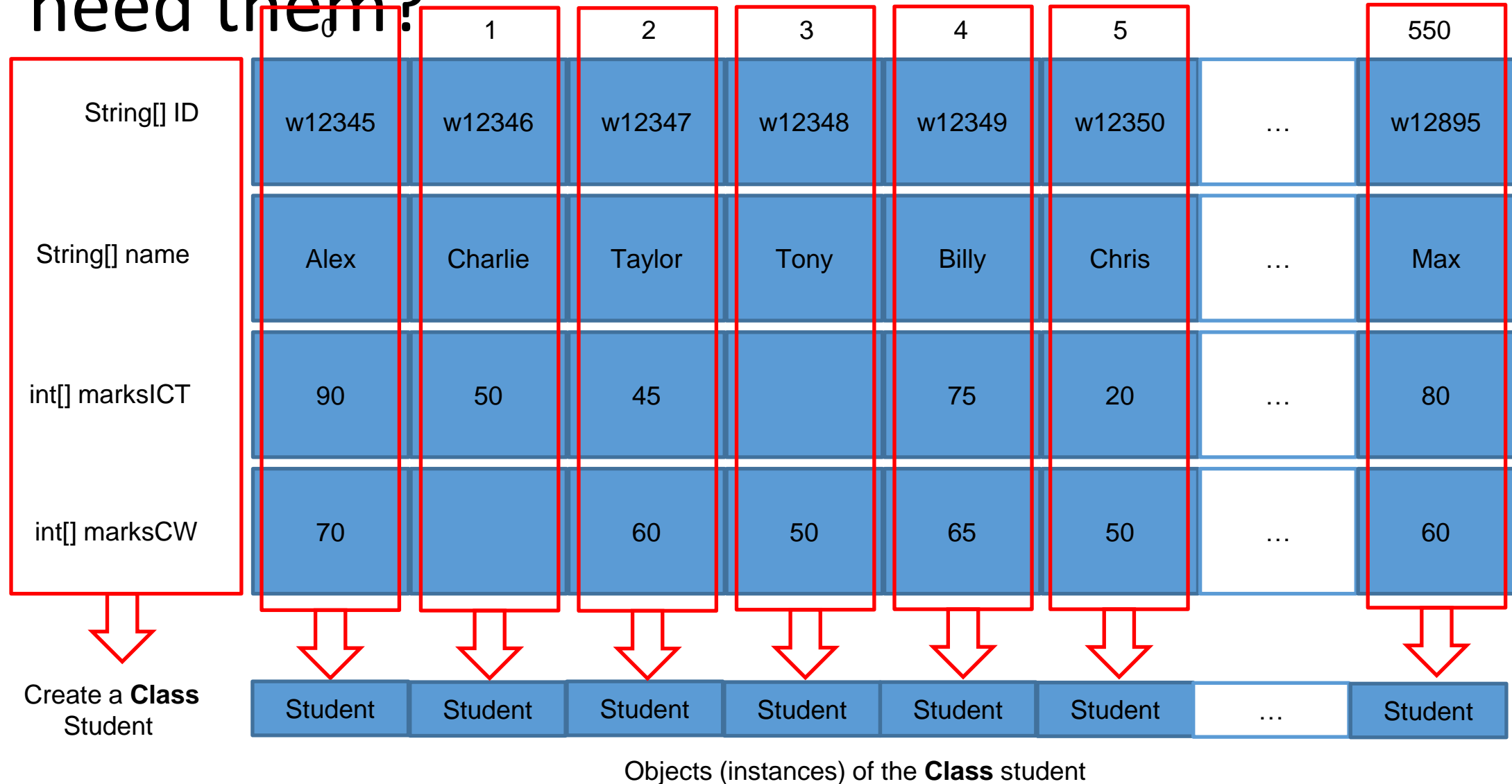
# Intro to Classes and Objects

How can we maintain large and complex systems like this?



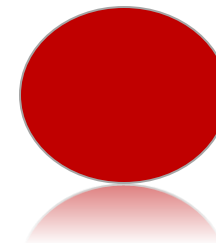
**Object oriented programming (OOP)**

# Intro to Classes and Objects: Why do we need them?



# What is an Object?

- An object represents an individual, identifiable item, or entity, either real or theoretical, with a well-defined role in the problem domain.





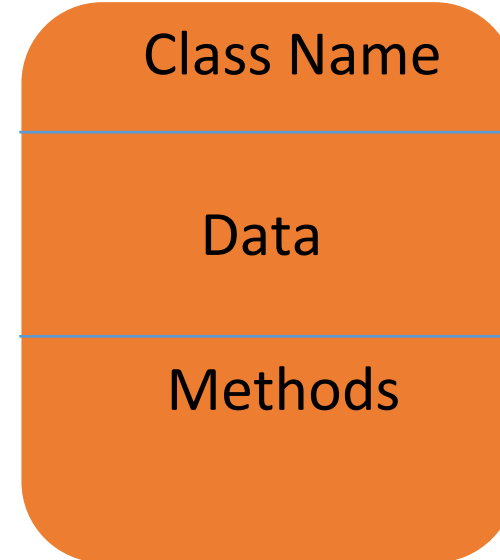
# Properties of an Object

- Identity – Who am I?
- Data/Attributes/State – What do I know?
- Methods/Behaviour/Operations – What can I do?

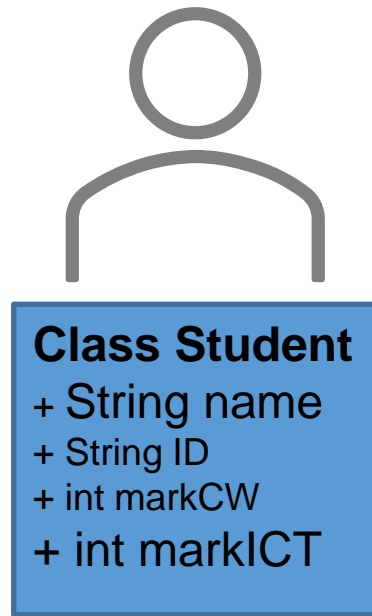


# What is a Class?

- A **class** presents a template or a blueprint of an object, its data and its methods.
- Structure of a **class** contains **Data** and **Methods**

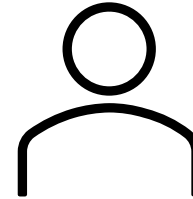


# Classes and Objects: state/data/attributes

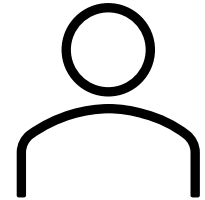


1 class definition

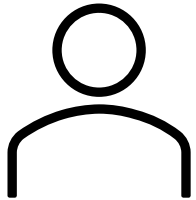
**Student s1**  
name: Alex  
ID: 12345  
markCW: 70  
markICT: 90



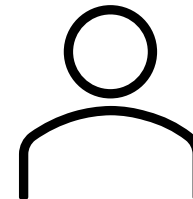
**Student s4**  
name: Tony  
ID: 12348  
markCW: 50  
markICT:



**Student s5**  
name: Billy  
ID: 12349  
markCW: 65  
markICT: 75

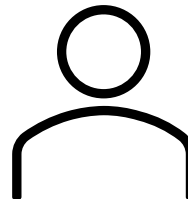


**Student s3**  
name: Taylor  
ID: 12347  
markCW: 60  
markICT: 45



This is not the name of the student, this is the name of a variable with type Student.

**Student s2**  
name: Charlie  
ID: 12346  
markCW: 50  
markICT:



**Instances (Objects)**

Multiple instances

# Classes and objects: more examples

Furniture store

**Furniture f\_1**  
price: 22  
width: 42  
height: 42  
name: KALLAX

**Furniture f\_2**  
price: 165  
width: 182  
height: 182  
name: KALLAX

**Furniture f\_3**  
price: 29  
width: 77  
height: 42  
name: KALLAX

## Class Furniture

+ double price  
+ double width  
+ double height  
+ String name



**KALLAX**  
Shelving unit, 42x42 cm  
**£22**



**KALLAX**  
Shelving unit, 182x182 cm  
**£165**



**KALLAX**  
Shelving unit, 77x42 cm  
**£29**

# Classes and objects: more examples

Book store / library

**Class Book**  
+ String title  
+ String author  
+ int year

**Book book\_1**

title: Java programming  
author: Rajkumar, K  
year: 2013

**Book book\_2**

title: Java programming  
author: Sarang, P.G  
year: 2011

**Book book\_3**

title: Java programming for  
android developers  
author: Burd, Barry  
year: 2017

**Book book\_4**

title: Practical Java programming for  
IOT, AI, and Blockchain  
author: Xiao, Perry  
year: 2019

The screenshot displays a library catalog interface with four book entries, each featuring a small thumbnail of the book cover on the left and text details on the right. The entries are numbered 3 through 6 in small boxes. Each entry includes the word 'BOOK' in all caps, the title, author, year, publisher information, and edition details. A link labeled 'Available Online' with a right-pointing arrow is present at the bottom of each entry.

Item Number	Book Title	Author	Year	Publisher / Edition
3	Java programming	Rajkumar, K.	2013	New Delhi, India : Pearson; 1st edition
4	Java programming	Sarang, P. G	2011	Place of publication not identified McGraw Hill Professional; 1st edition
5	Java programming for android developers	Burd, Barry, author.	2017	Hoboken, New Jersey : For Dummies; Second edition.
6	Practical Java programming for IOT, AI, and Blockchain	Xiao, Perry, author.	2019	Indianapolis, Indiana : Wiley; 1st edition

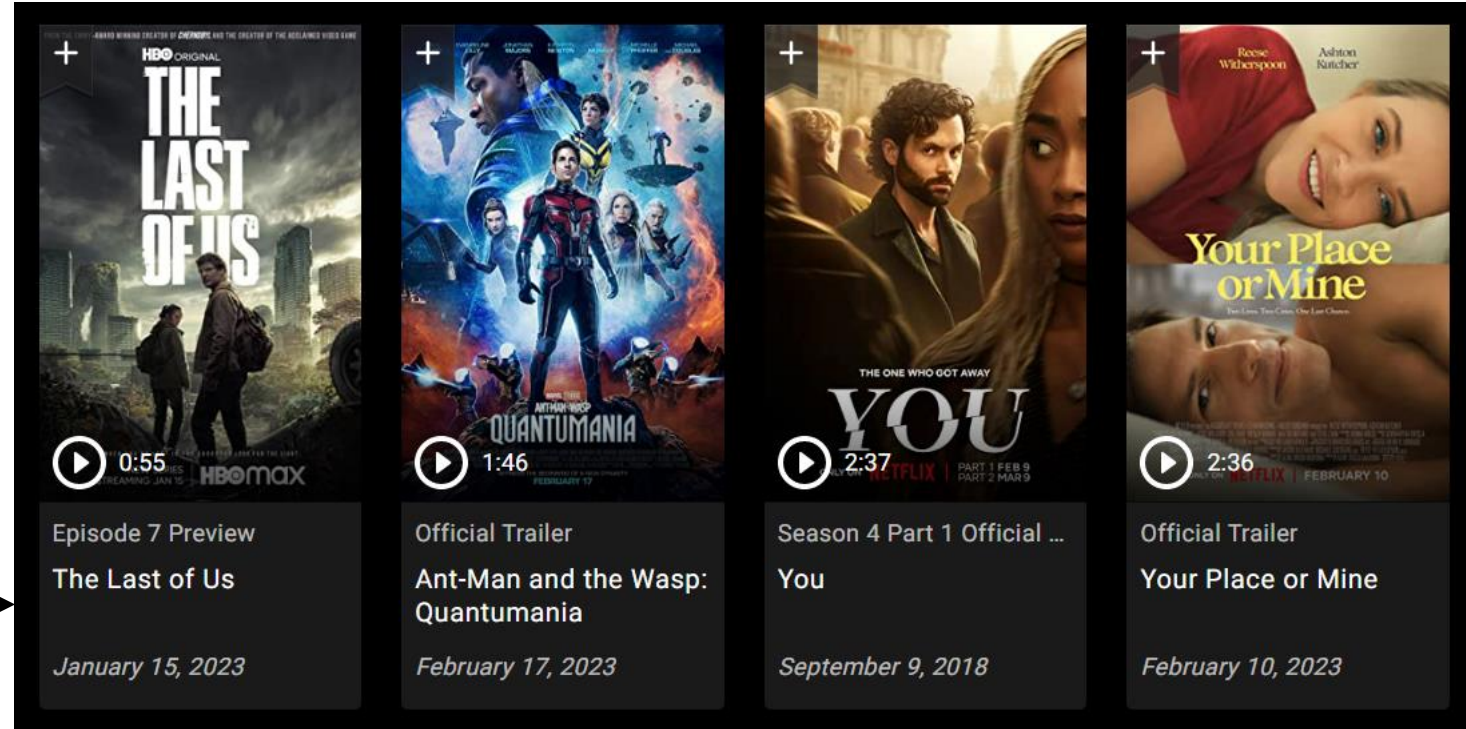
# Classes and objects: more examples

IMDb

Movie movie\_1

title: The Last of Us

release: January 15, 2023



**Class Movie**

+ String title

+ String release

Movie movie\_2

title: Ant-Man and the  
Wasp: Quantumania

release: February 17, 2023

Movie movie\_3

title: You

release: September 9, 2018

Movie movie\_4

title: Your Place or Mine

release: February 10, 2023

# You have been using Classes and Objects already

```
String input = new Scanner(System.in);
```

↑  
Class

↑  
Instance

↑  
Create a new instance of a Class

```
File file = new File("example.txt");
```

↑  
Class

↑  
Instance

↑  
Create a new instance of a Class

```
FileWriter writer = new FileWriter(file);
```

↑  
Class

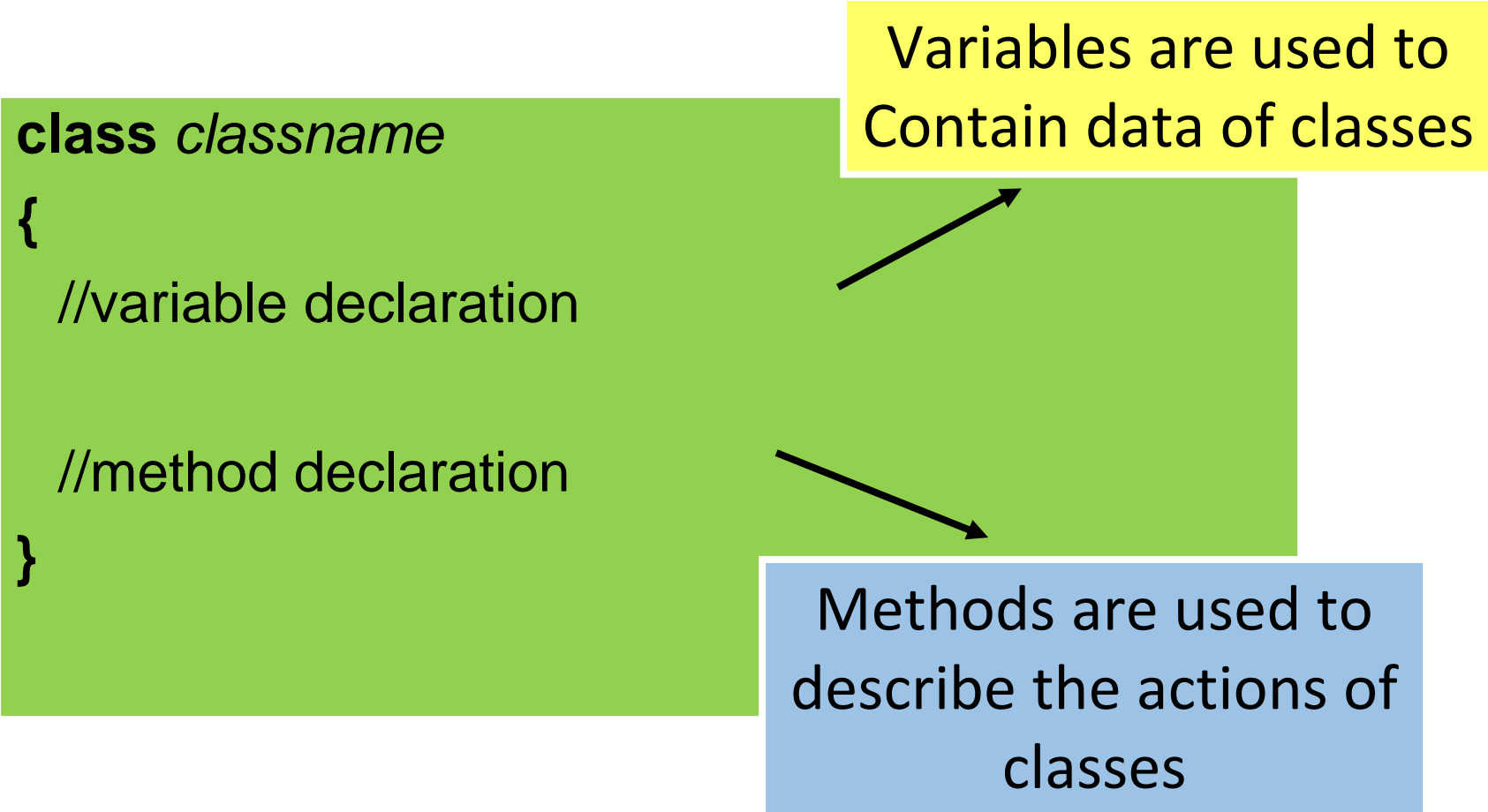
↑  
Instance

↑  
Create a new instance of a Class

# Classes in Java

```
class classname  
{  
    //variable declaration  
  
    //method declaration  
}
```

Variables are used to  
Contain data of classes



Methods are used to  
describe the actions of  
classes



# Adding Variables (Characteristics)

- These are known as **Instance variables** or **non-Static Fields** or **member variables**.
- These are created whenever an object of the class is instantiated.

```
class Rectangle
{
    int length;
    int width;
}
```

These variables are only declared - no storage space created in the memory YET

# Class declaration: Syntax-Review

```
public class Name_of_the_class{ ←————— Start of the class declaration

    // attributes (instance variables/constants) here

    public Name_of_the_class() {
        // this is a constructor (special method to create instances of this class)
    }
    public void method(){
        // this is a method
    }
} ←————— End of the class declaration
```

# Task One – Identifying Classes and Attributes

- You must develop a simple application to Santa's Helpers Toy Factory. This application should facilitate to add a new toy, update stock level of each toy and many more functions.
- Each toy has a name, but it is uniquely identified by its code number. Different toys have different prices. Some toys are specially made for little girls, some for little boys and some are general and the stock level for each toy should be maintained as well.



# Adding Methods (Actions)

- Adding methods to a class is necessary for manipulating the data contained in the class.
- The objects created by a class with only data fields cannot respond to any messages.

# Method Types: getters

Since attributes are private, we need a mechanism to get the values of the attributes.

We will use methods called **getters**:

- Getter methods are used to retrieve the values of private fields from a class.
- They typically have the prefix "get" followed by the name of the field they are accessing, but they can have any name.
- Getter methods are usually public to allow access from outside the class.
- They do not modify the state of the object; they only retrieve the values.

```
public class MyClass {  
    private int myField;  
  
    public int getMyField() {  
        return myField;  
    }  
}
```

# Method Types: setters

Since attributes are private, we need a mechanism to change the values of the attributes.

We will use methods called **setters**:

- Setter methods are used to modify the values of private fields in a class.
- They typically have the prefix "set" followed by the name of the field they are modifying, but they can have any name.
- Setter methods are usually public to allow modification from outside the class.
- They can include validation or other logic to ensure the integrity of the data being set.

```
public class MyClass {  
    private int myField;  
  
    public void setMyField(int value) {  
        if (value < 0) System.out.println("Invalid value");  
        else myField = value;  
    }  
}
```

# Example 1 contd...

```
class Rectangle {  
    int length;  
    int width;  
    String color;  
  
    void setData(int l, int w, String c) {  
        length = l;  
        width = w;  
        color = c;  
    }  
    int calcArea() {  
        int area = length * width;  
        return area;  
    }  
}
```

# Task Two – Identifying methods

When a new toy is introduced there should be a way to add it to the system. The stock level of each toy must be updated when toys been made or when toys are distributed to the outlets. If the raw material cost go high, sometimes the price of a toy can be increased. For reporting purpose there should be a way to view the details of each toy . Specially the name, code , category, price and the stock level.





# Method Types :Constructors

- **Constructor** is a special method that initialize object of a class.
- Always and only used with the **new** operator (keyword) to create an instance of a class.
- Example:

```
Rectangle r1 = new Rectangle();
```

# Why do we need Constructors?

- We have to define our own constructors to initialize the instance variables.
- If we don't define our own constructor, Java always calls the default constructor (no arguments).
- The default constructor initializes all class variables to default values.
- Example: **Rectangle ()** is the default constructor of Rectangle class.

# Characteristics of a Constructor

- Has the same name as the name of the class.
- Can take one or more parameters.
- Has NO return type, not even void.
- Default constructor – takes no parameters.
- More than one constructor can exist (method overloading).
- Declare as public (usually).

# Example 2-Defining a Constructor

```
class Rectangle {  
    int length, width;  
    String color;  
  
    public Rectangle(int l, int w, String c){  
        length = l;  
        width = w;  
        color = c;  
    }  
    void setData(int l, int w, String c) {  
        length = l;  
        width = w;  
        color = c;  
    }  
}
```

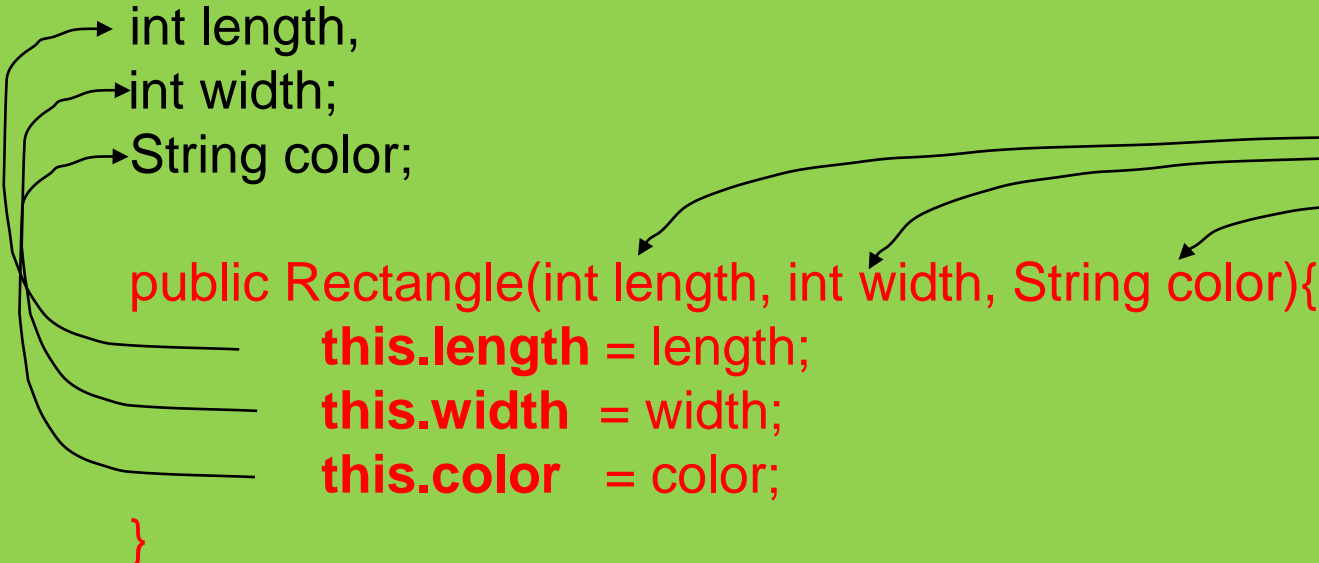
... ..

# Example 2-Invoke the Constructor

```
class RectTest {  
    public static void main(String arg[]){  
  
        Rectangle rect1 = new Rectangle(20,10,"blue");  
        rect1.setData(25,3,"green");  
  
    }  
}
```

# Constructor – Use of **this** keyword

```
class Rectangle {  
    int length,  
    int width;  
    String color;  
  
    public Rectangle(int length, int width, String color){  
        this.length = length;  
        this.width = width;  
        this.color = color;  
    }  
  
    void setData(int l, int w, String c) {  
        length = l;  
        width = w;  
        color = c;  
    }  
}
```



Rectangle rect1 = **new** Rectangle(30,10,"red");



# Creating Objects (Instantiating Objects)

- Objects in Java are created using the **new** Operator.
- The new operator creates objects of the specified class and returns a reference to that object.

```
Rectangle rect1 = new Rectangle( );
```

**OR**

```
Rectangle rect1;           //declare  
rect1 = new Rectangle( ); //instantiate
```

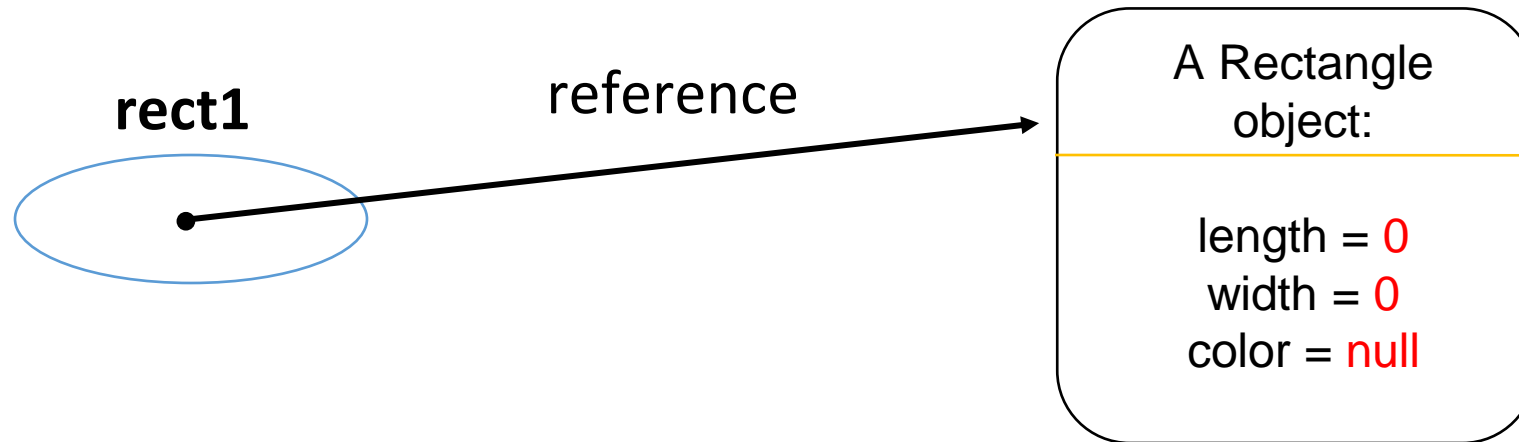
# Example 1

```
class RectTest {  
    public static void main(String arg[]){  
  
        Rectangle rect1 = new Rectangle();  
  
    }  
}
```



# What Really Happens...

```
Rectangle rect1 = new Rectangle( );
```



- **rect1** is a **reference variable**.
- The data type of **rect1** is "Rectangle".

# What Really Happens...

```
Rectangle rect1;
```

rect1  
null

- This declares a reference variable (to hold a object reference later).
- But currently it contains **null**.
- Because: default value for an object is **null**.

# Accessing Instance Variables

- Once an object is created:

**ObjectName.VariableName**

- Examples:

rect1.length = 15;

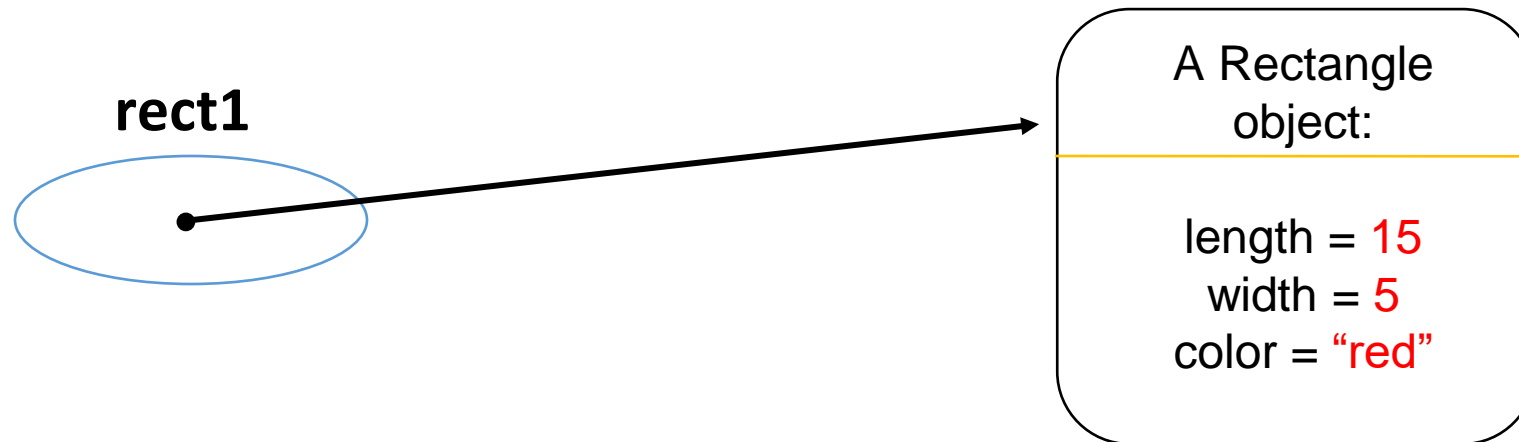
rect1.width = 5;

# Example 1 contd...

```
class RectTest {  
    public static void main(String arg[]){  
  
        Rectangle rect1 = new Rectangle();  
        rect1.length = 15;  
        rect1.width = 5;  
        rect1.color = "red";  
    }  
}
```

# What Really Happens...

```
rect1.length = 15;  
rect1.width = 5;  
rect1.color = "red";
```



# Accessing Instance Methods

- Once an object is created:

**ObjectName.MethodName(parameter-list)**

- Example:

```
rect1.setData(15,5,"red");
```

# Example 1 contd...

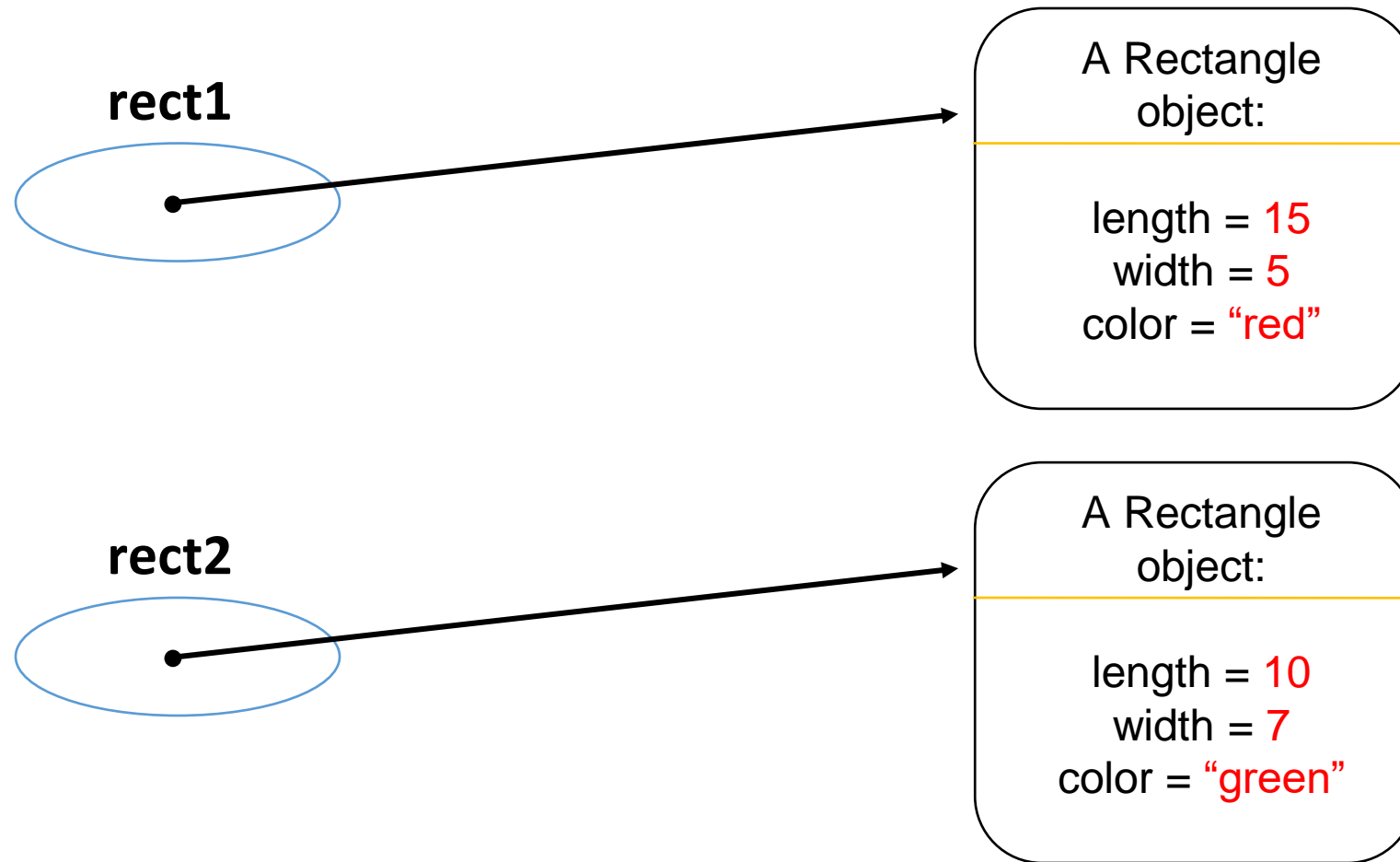
```
class RectTest {  
    public static void main(String arg[]){  
  
        Rectangle rect1 = new Rectangle();  
  
        rect1.setData(15,5,"red");  
    }  
}
```

# Creating More Objects

```
class RectTest {  
    public static void main(String arg[]){  
  
        Rectangle rect1 = new Rectangle();  
        rect1.setData(15,5,"red");  
        Rectangle rect2 = new Rectangle();  
        rect2.setData(10,7,"green");  
  
    }  
}
```



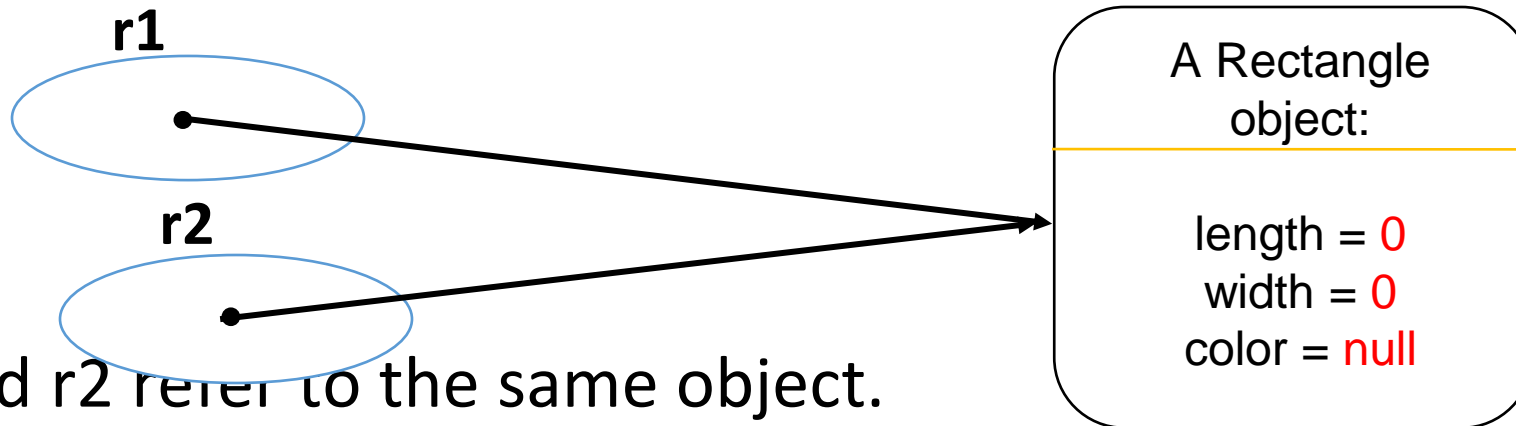
# What Really Happens...



# References to the Same Object

- We can assign values from other reference variables that are of the same type.

```
Rectangle r1 = new Rectangle( );  
Rectangle r2 = r1;
```



- Both r1 and r2 refer to the same object.

# Task Three



- Fine tune attributes and methods of Toy class with scope & visibility rules .
- Modify the Toy class by adding a constructor.
- Create a driver class called Factory. Write necessary code segments to do the following activities.
  - Add the following details.
    - Teddy bear code is TD0012 and the price is 1200/= . The stock level is 12000, General category.
    - Tea Set code is TE0123 and the price is 1000/= . The stock level is 22000, for girls.
    - Racing Car code is RC0342 and the price is 5000/= . The stock level is 7000, for boys.



# Task Three

- Factory issues 500 teddy bears to outlets, have to update the stock level.
- Factory produce 100 racing cars more, have to update the stock level.
- The price of the tea set change to 1200/=
- To view the new details of each toy, after the changes.



# Access Modifiers

- A way of controlling access to variables, methods and classes.
- Control access to a class or class members can be done through the keywords **public**, **protected** and **private**.
- If you don't specify an access modifier it is taken as default.
- **USUALLY**: data (instance variables) are declared private and methods are declared public.
- **NOTE**: using public data is uncommon and dangerous practice.

# Why you should NOT use public attributes



```
public class Student{  
    public String name;  
    public String ID;  
    int markICT;  
    int markCW;  
}
```

→ All public. Modifiable from outside (e.g., main).  
**NOT RECOMMENDED.**

```
Student s1 = new Student();  
s1.name = myemail@email.com;  
s1.ID = "Can't remember";  
s1.markICT = 205;  
s1.markICT = -12;
```

Do you think this is a good example of a Student object?  
With public attributes, any class can change the values and this may lead to the creation of object with no 'meaning'.



Make them private to have more control on how to modify them.  
We will see how to modify objects in the second part of this lecture

# Scope & Visibility Rules Table

	Scope	Visibility
public variables, methods & classes	public	Visible to all classes
private variables, methods & classes	private	Visible only within the class
protected variables, methods & classes	protected	Visible to all classes within the package & inherited classes to outside the package
default variables, methods & classes	treated as public within its package	Visible to all classes within the package

# Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier (Default)	Y	Y	N	N
private	Y	N	N	N

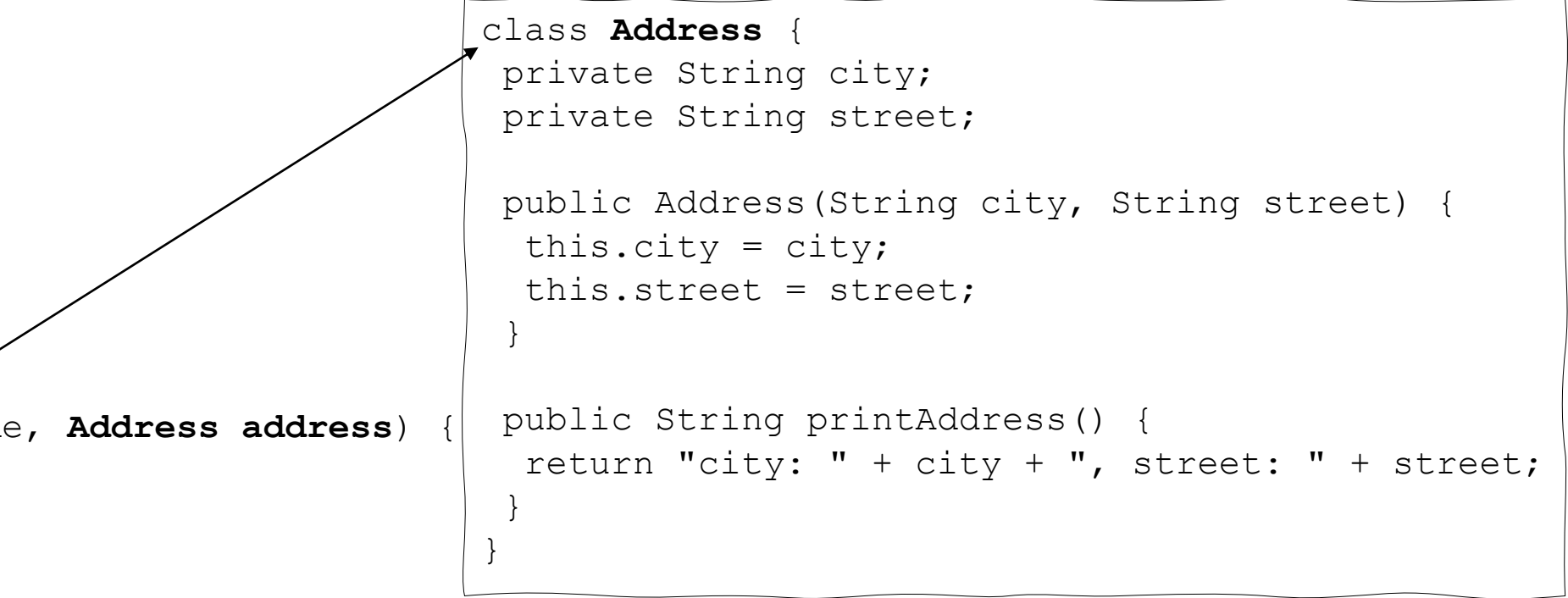


# Classes using another Class – Example I

- A class can also contain a variable (object) of another class.

```
public class Person {  
    private String name;  
    private Address address;
```

```
    public Person(String name, Address address) {  
        this.name = name;  
        this.address = address;  
    }  
  
    // All getters and setters here  
  
    public String printPersonInfo() {  
        return "Name: " + name + ", " + address.printAddress();  
    }  
}
```

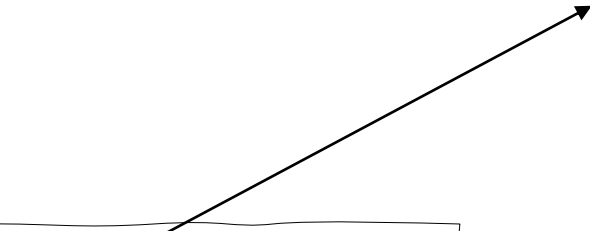


```
class Address {  
    private String city;  
    private String street;  
  
    public Address(String city, String street) {  
        this.city = city;  
        this.street = street;  
    }  
  
    public String printAddress() {  
        return "city: " + city + ", street: " + street;  
    }  
}
```

# Classes using another Class – Example II

A class can also contain an array of objects of another class type

```
public class Library {  
    private Book[] books;  
  
    public Library(Book[] books) {  
        this.books = books;  
    }  
  
    public Book[] getBooks() {  
        return books;  
    }  
  
    public void setBooks(Book[] books) {  
        this.books = books;  
    }  
}
```



```
class Book {  
    private String title;  
    private String author;  
  
    public Book(String title, String author) {  
        this.title = title;  
        this.author = author;  
    }  
    public String getTitle() {  
        return title;  
    }  
    public void setTitle(String title) {  
        this.title = title;  
    }  
    public String getAuthor() {  
        return author;  
    }  
    public void setAuthor(String author) {  
        this.author = author;  
    }  
}
```

# Self Study Example – Student Class

# Student Class

```
public class Student {  
    private String ID;  
    private int markICT;  
    private int markCW;
```

Private attributes, cannot be accessed from outside the class:

```
System.out.println(Student.ID);  
System.out.println(Student.getID());
```

```
    public Student(String ID, int markICT, int markCW) {  
        this.ID = ID;  
        this.markICT = markICT;  
        this.markCW = markCW;  
    }
```

Constructor

```
    public String getID() {  
        return this.ID;  
    }
```

ID is private, so to get the ID from outside the class can only be done by calling this method

```
    private double getFinalMark() {  
        return this.markICT*0.5 + this.markCW*0.5;
```

Private method that can **only** be called from the inside class.

```
    public boolean pass() {  
        return this.getFinalMark() >= 40;
```

Calls a private method from the same class:  
Student s1 = new Student("w123", 40, 60);  
s1.pass();

We can also use *this* with methods

# Class constructor - Student

```
public class Student {  
    String name;  
  
    public Student() {  
        this.name = "Not initialised";  
    }  
  
    public Student(String name) {  
        this.name = name;  
    }  
}
```

*Public only for demonstrating this example.  
It should be private.*

Student.java

```
public static void main(String[] args) {  
  
    Student s1 = new Student();  
    Student s2 = new Student("Alex");  
    // s1  
    System.out.println("Student: " + s1.name);  
    //s2  
    System.out.println("Student: " + s2.name);  
}
```

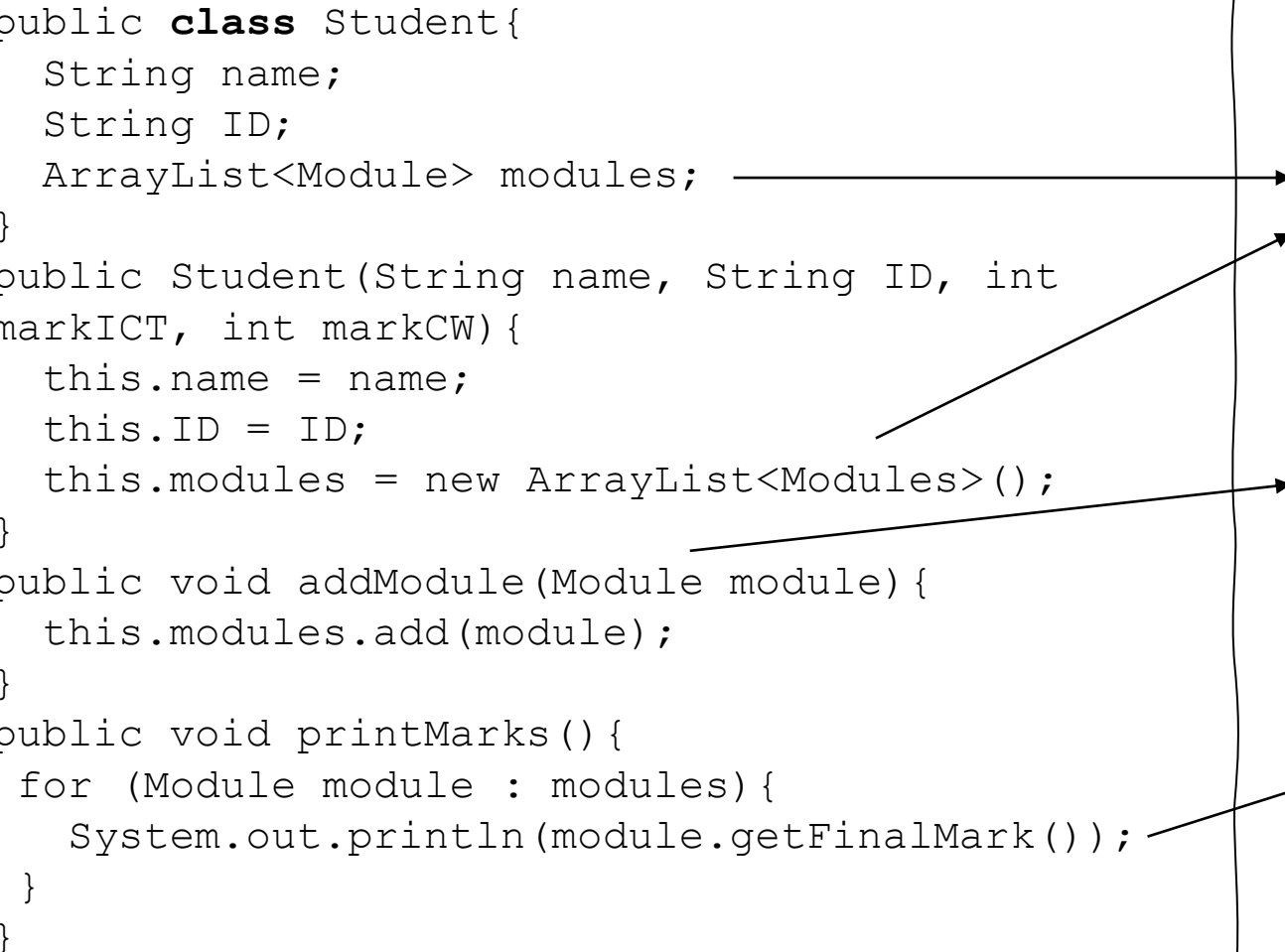
Main.java

Output:  
Student: Not initialised  
Student: Alex

# Classes containing other Classes

- A class can also contain a variable (object) of another class.

```
public class Student{  
    String name;  
    String ID;  
    ArrayList<Module> modules;  
}  
public Student(String name, String ID, int  
markICT, int markCW){  
    this.name = name;  
    this.ID = ID;  
    this.modules = new ArrayList<Modules>();  
}  
public void addModule(Module module){  
    this.modules.add(module);  
}  
public void printMarks(){  
    for (Module module : modules){  
        System.out.println(module.getFinalMark());  
    }  
}
```



```
public class Module{  
    String name;  
    String ID;  
    int markICT;  
    int markCW;  
}  
public Module(String name, String ID, int  
markICT, int markCW){  
    this.name = name;  
    this.ID = ID;  
    this.markICT = markICT;  
    this.markCW = markCW;  
}  
  
public double getFinalMark(){  
    double mark = markICT*0.5+markCW*0.5;  
    return mark;  
}
```

# Arrays with objects

- In the same way we have arrays of int, String, etc. we can have arrays of objects of a class.
- To create an array we simply declare an array of the Class type, and we assign instances of the class.
- We can use standard arrays and ArrayList:

```
public class Student{  
    private String name;  
    private String ID;  
    private int markICT;  
    private int markCW;  
}  
  
public Student(String name, String ID, int  
markICT, int markCW){  
    this.name = name;  
    this.ID = ID;  
    this.markICT = markICT;  
    this.markCW = markCW;  
}
```

```
Student[] list_students = new Student[4];  
Student student_1 = new Student("Alex", "w12345", 90, 70);  
Student student_2 = new Student("Billy", "w12349", 75, 65);  
list_students[0] = student_1;  
list_students[1] = student_2;  
Student s1 = list_students[0];
```

```
ArrayList<Student> arraylist_students = new ArrayList<Student>();  
arraylist_students.add(student_1);  
arraylist_students.add(student_2);  
Student s2 = arraylist_students.get(0);
```

# Class methods: example

```
public static void main(String[] args) {  
  
    Student student = new Student("Alex", "w12345", 90, 70);  
    double final_mark = student.getFinalMark();  
    System.out.println("Student: " + student.getName());  
    System.out.println("Final mark: " + final_mark);  
}
```

## Output:

Student: Alex

Final mark: 80.0



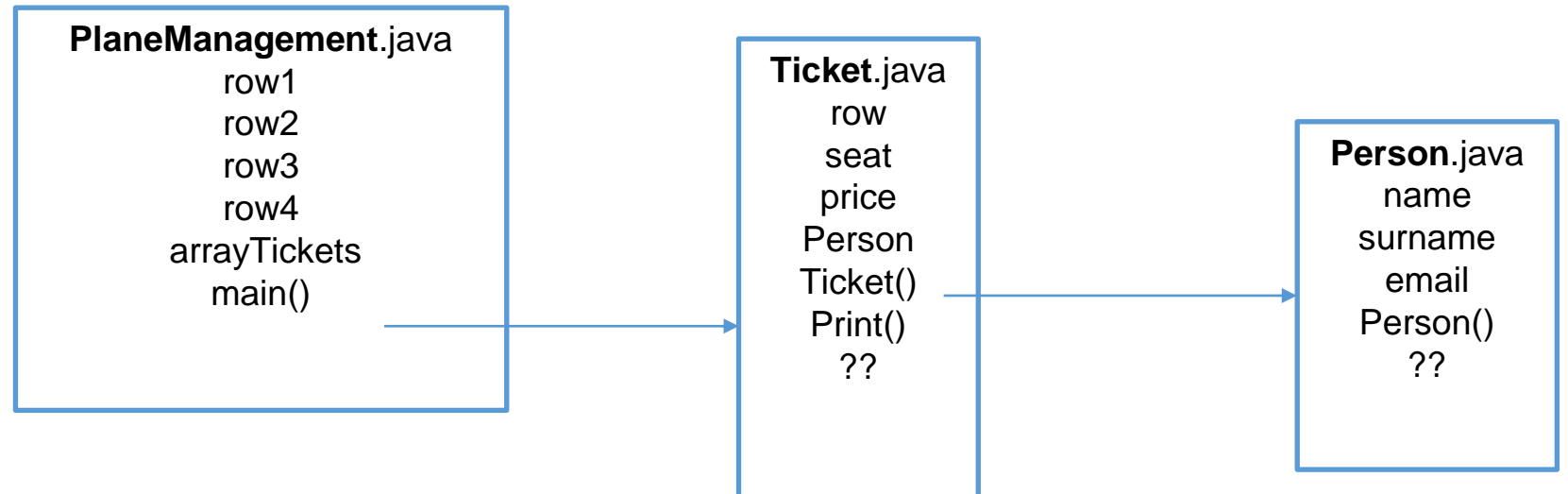
# Coursework

After this lecture, you should be able to implement:

## Part B: tasks 7-11.

- In part B you are asked to create two classes: Person and Ticket
- Use of public and private variables and methods.
- Search tickets: please use a searching **algorithm**.

## Work on Part C.



# Independent Work

- In Learning Resources > Week 7 you will find a formative test to get feedback on the content of this lecture.
- Solve the following exercises in HackerRank on exception handling:
  - Java Inheritance I (inheritance was not explained during the lecture but they provide an example)

# HackerRank: Interview questions on this topic

Solve the following exercises in HackerRank on exception handling:

- Java Inheritance I

Thank You !!