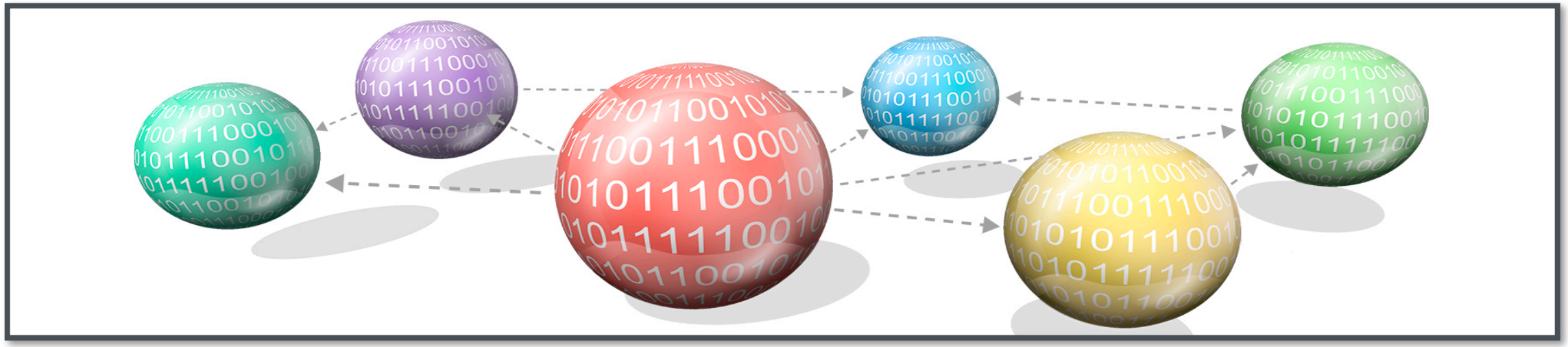
NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

# Chapter 2: Class and Object

CE2002 Object Oriented Design & Programming

Dr Zhang Jie
Assoc Prof, School of Computer Science and Engineering (SCSE)



# Learning Objectives

After the completion of this chapter, you should be able to:

- Describe classes and objects in Object-Oriented Programming (OOP).
- Define class in OOP.
- Explain the concept of message sending.
- Explain copying objects.
- Explain '*this*' keyword.
- Describe Accessors and Mutators.
- Describe static keyword.
- Differentiate between static and instance method.
- Differentiate between encapsulation and information hiding.
- Describe object composition.
- Explain '*string*' class.

# Additional Reading

- Class Definition:
  - Textbook: Chapter 4, Page 75-86
- Constructors:
  - Textbook: Chapter 3, Page 53-60
- Java Programming:
  - Lecture Slides: Blackboard/ NTU Learn -> Content
  - Recorded Lectures: Blackboard/ NTU Learn
- C vs. Java
- Java Quick Guide
  - Blackboard/ NTU Learn-> Content -> Course -> Additional Resources

# Topic 1: Classes and Objects

# What is an Object?

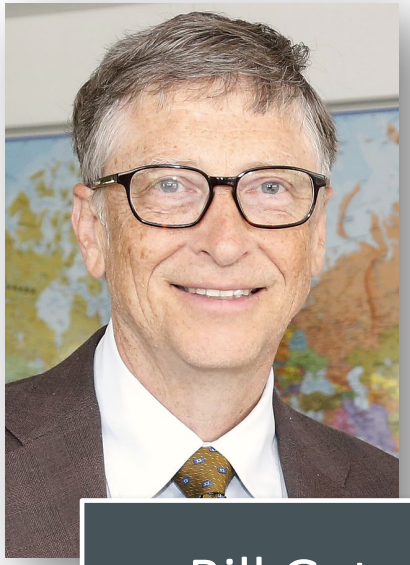
Objects are everywhere!

Characteristics:

- Identity
- State
- Behaviour

# Object Identity

Each object has its own **unique** identity (or Object Identifier (OID)).



Bill Gates

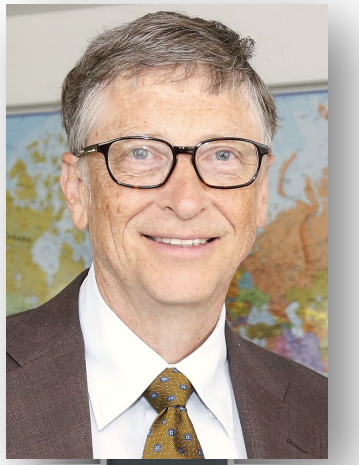
John Hui

**Bill Gates**. Retrieved November 11, 2016 from Wikimedia Commons https://commons.wikimedia.org/wiki/File:Bill_Gates_July_2014.jpg.
**John Hui**. Retrieved November 11, 2016 from Wikimedia Commons https://en.wikipedia.org/wiki/Lap_Shun_Hui

# Object State (Data Members)

- Made up of **attribute(s)** (or property)
- Example: A person has properties

- Gender
- Age
- Monthly Salary



**Bill Gates**

- Male
- 57 years old
- >>> $72,000,000

**John Hui**

- Male
- 47 years old
- ≈ $7,000

# Object Behaviour

Behaviour refers to the **things that an object can do**.

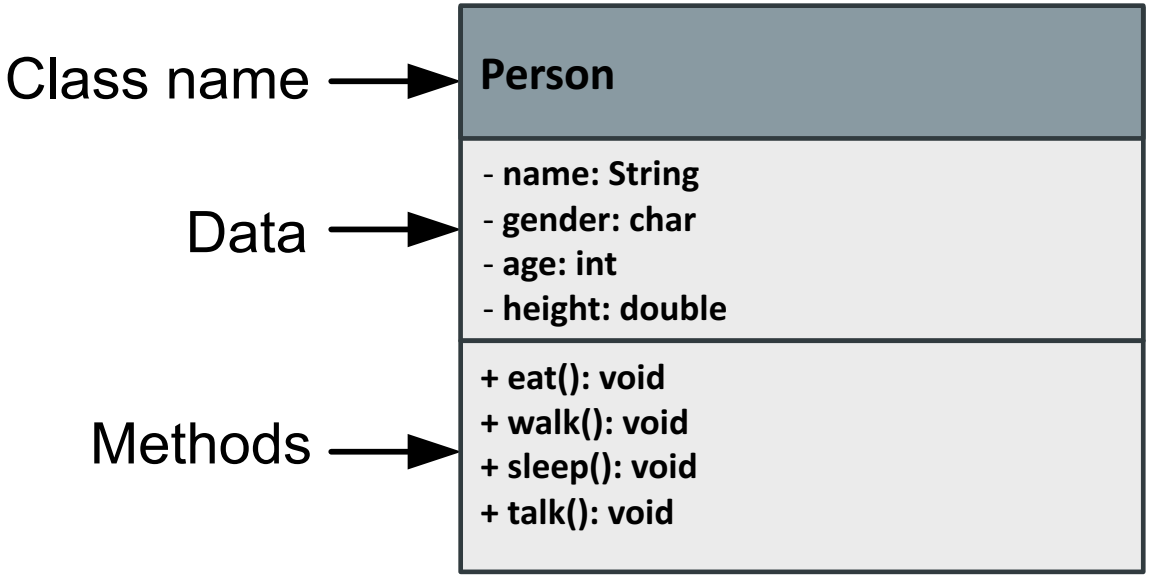
Example: What can a person do?

- Eat
- Walk
- Sleep
- Talk
- Study

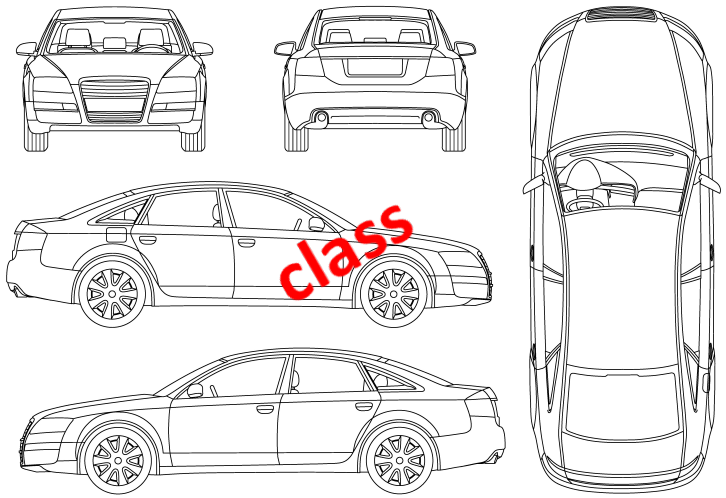
These operations that can be <u>acted upon (performed) by an object</u> are referred to as methods.

# What is a Class?

- A class defines the **blueprint/ structure** for creating objects. In Java, a class contains:
  - **Data properties** (or attributes)
  - **Methods**
- To create an object, we need to define a class for it.
- UML (Unified Modelling Language) diagram:

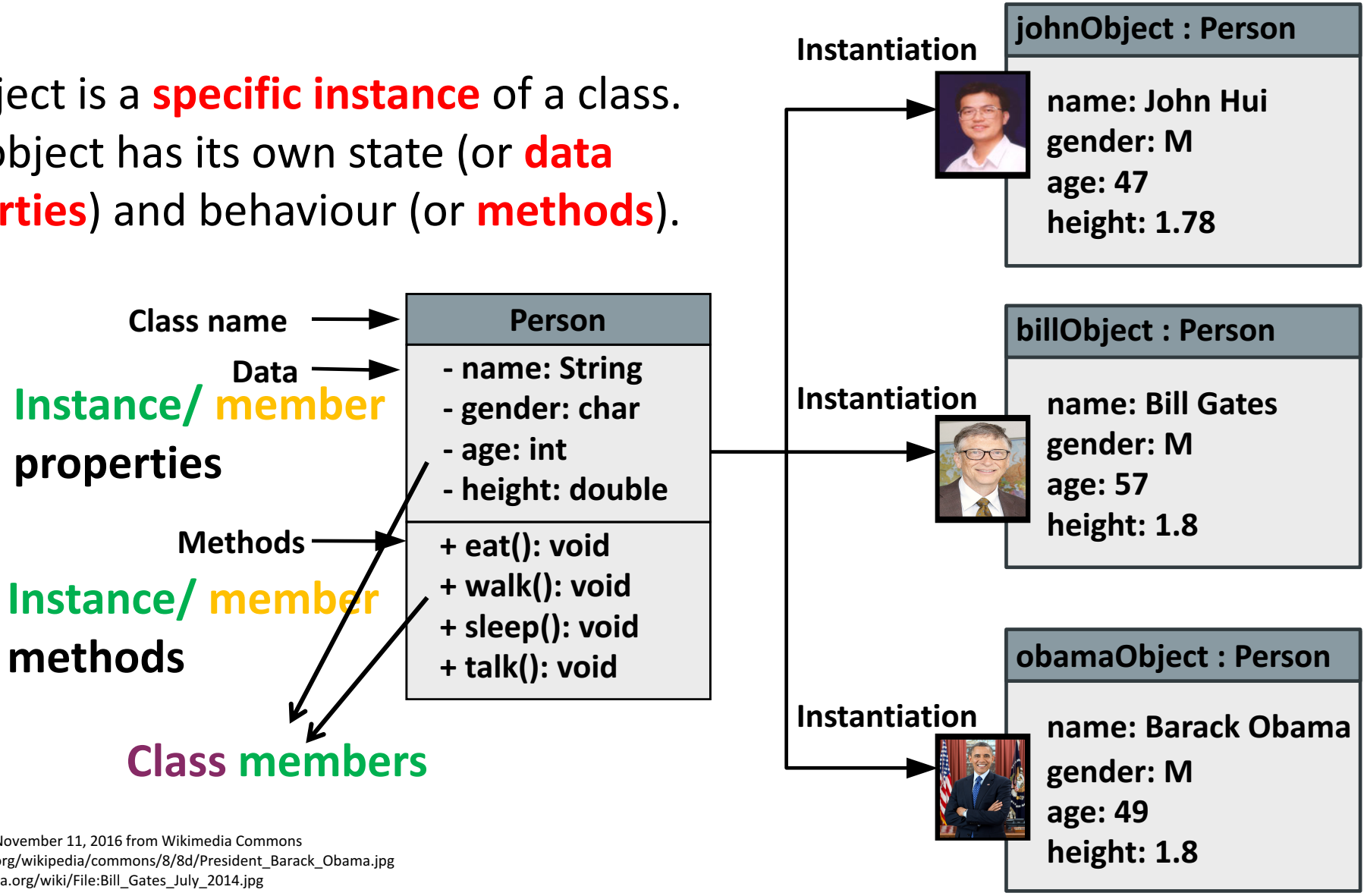



**Class Diagram**



# Classes and Objects

- An object is a **specific instance** of a class.
- Each object has its own state (or **data properties**) and behaviour (or **methods**).




**Barack Obama**. Retrieved November 11, 2016 from Wikimedia Commons
https://upload.wikimedia.org/wikipedia/commons/8/8d/President_Barack_Obama.jpg
https://commons.wikimedia.org/wiki/File:Bill_Gates_July_2014.jpg

# Topic 2: Class Definition

# Class Definition (in Java)

The syntax for defining a class:

**Keyword**

```
public class Class_Name {
    Instance_Variable_Declaration_1;
    …
    Instance_Variable_Declaration_n;
    Method_Definition_1;
    …
    Method_Definition_n;
}
```

**Defining Data**

**Defining Methods**

**Example:**

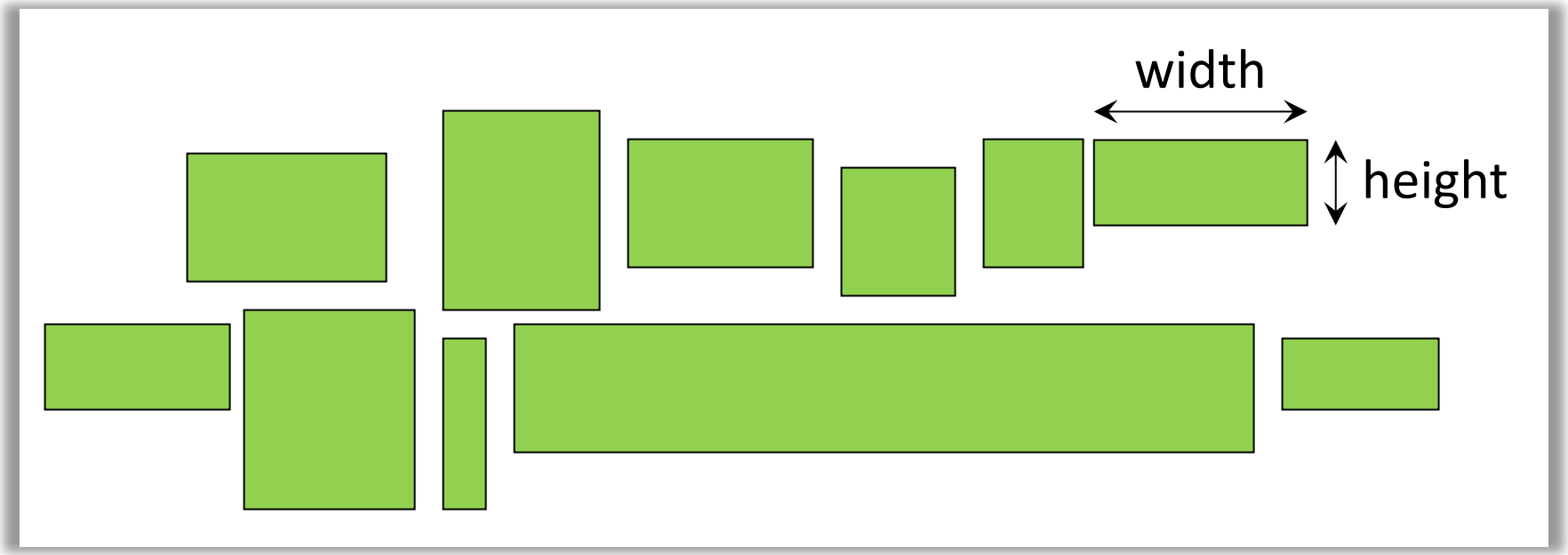
```
public class Rectangle {
    // instance variables
    // instance methods
}
```

# Class Definition Example: The Rectangle

- There are an infinite number of rectangles.
- What do all have in common? Any attribute?
  - Width
  - Height

**OO: abstraction**

- What common operations we would like to perform?
  - Find area
  - Find Perimeter



# Class Definition (Data Abstraction)

- Defining Data Attribute – syntax:

  private Type Instance_Variable_Name;

- Defining Method – syntax:

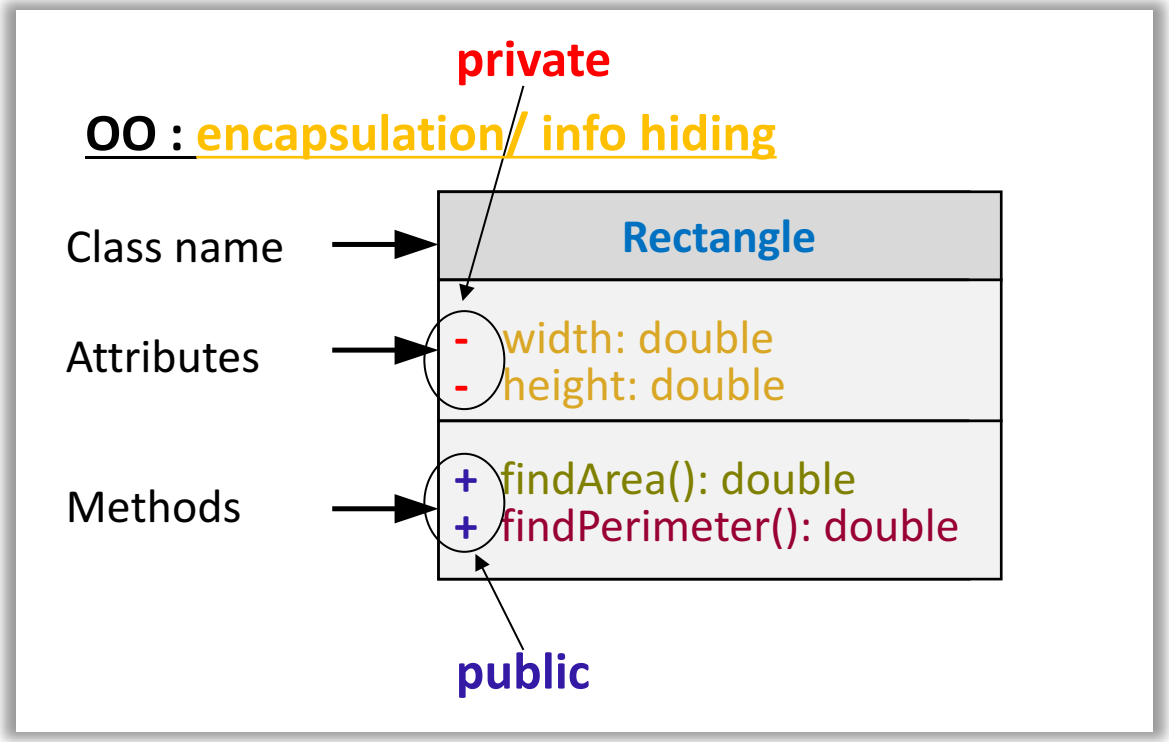
  public Return_Type Method_Name( Parameter_List ) {

  Method_body

  }

**Example:**

```
public class Rectangle {
  // instance variables
  private double width  ;
  private double height ;
  // instance methods
  public double findArea() {
    return width * height ;
  }
  public double findPerimeter() {
    return ( width + height ) * 2;
  }
}
```



# Method Header

***Return data*** **General format** ***Accept input data***

**public <static>** **Return_Type** **Method_Name (Parameter_List)**

***1. public <static>***

reserved keywords

***2. Return_Type***

can be int, double, float, char,…, void

Class type

***(Abstract Data Type)***

***3. Parameter_List***

**Example**

public static **int** successor(*double num*) { …… }

public **boolean** *savetoDB(String name, int age, char gender)*
{……}

# Creating Objects

**2 steps**:

**Object Identifier**

1. Create a variable for the object:

**declaration** Class_Name Object_Variable_Name ;

2. Allocate memory:

**creation** Object_Variable_Name = Class_Name() ;

keyword

**Example**:

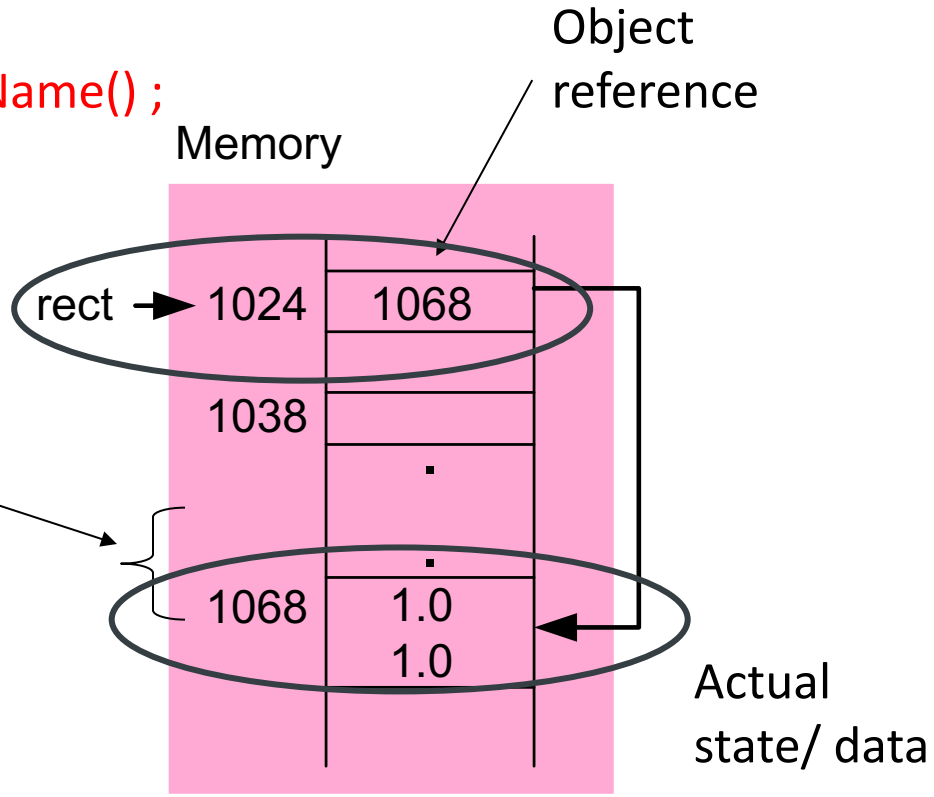
1. **Rectangle rect ;**
2. **rect** = new **Rectangle**() ;

Memory space is allocated to store the object

**Or**

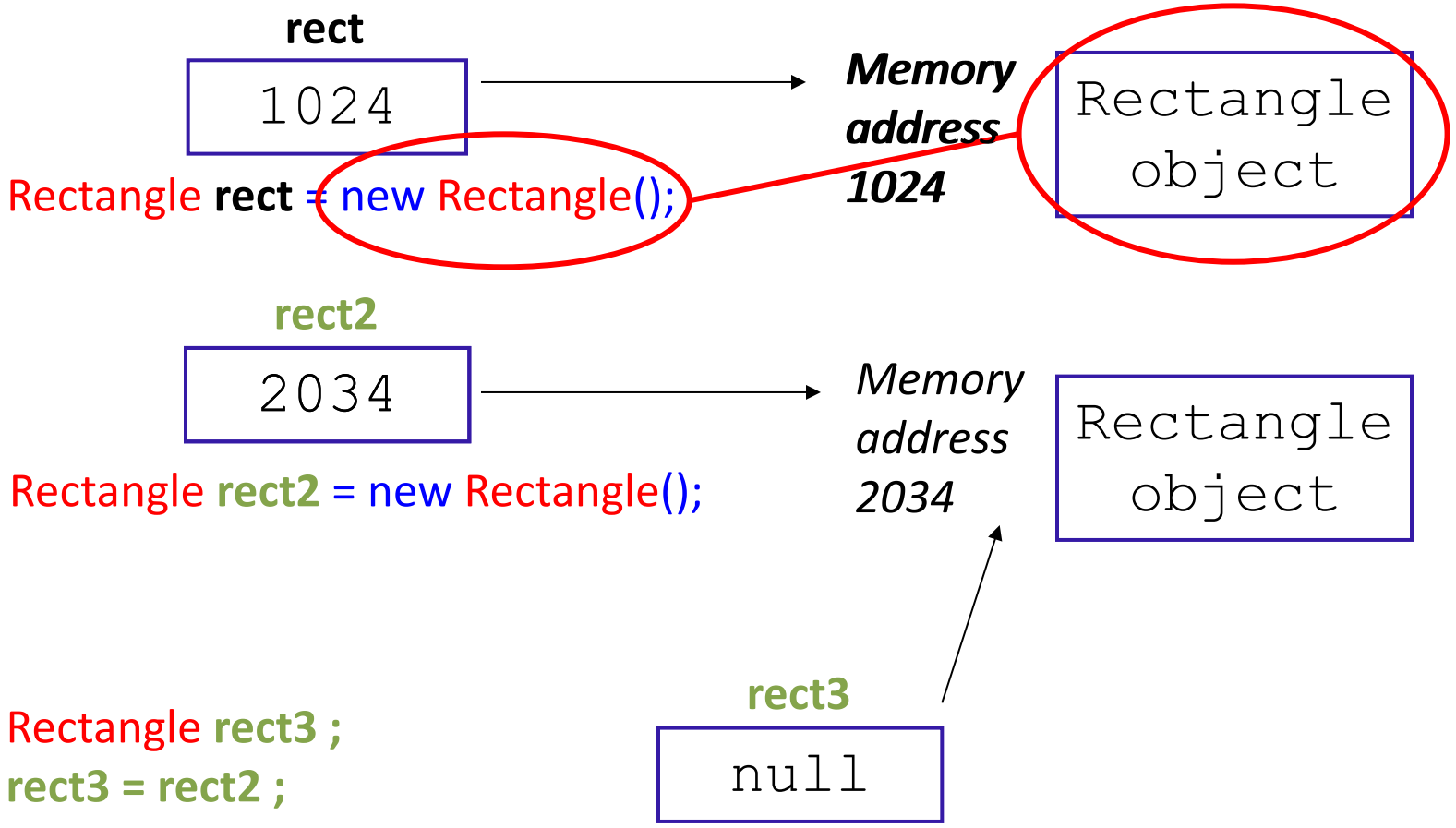
```
Rectangle rect = new Rectangle() ;
```

```
int [] array = new int[10] ;
```



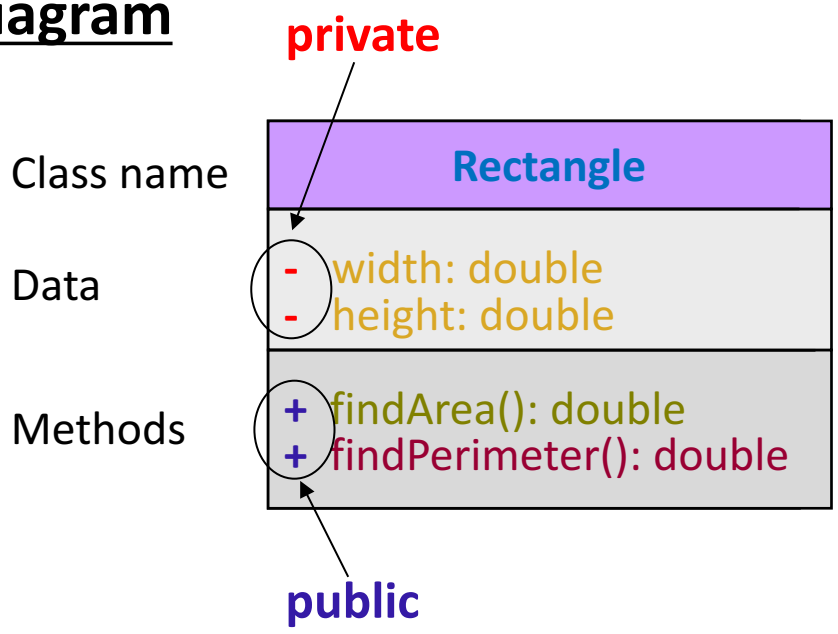
# Object Reference and Object

The **object reference**, **rect**, does not store the 'Rectangle' object; instead it stores the memory address of the 'Rectangle' object.



# Defining Class

## Class Diagram




```
public class Rectangle {
  // instance variables
  private double width  ;
  private double height ;
  // instance methods
  public double findArea() {
    return width * height ;
  }
  public double findPerimeter() {
    return ( width + height ) * 2;
  }
}
```

## Creating Objects (Instantiating Object)

**Rectangle rect** ;
**rect** = new **Rectangle**() ;

**OR** Rectangle rect = new Rectangle();

# Constructors

- Constructors are used for initialising object data (Constructing obj).
- If no constructors are written by programmers, Java has a default constructor without argument; the default constructor does nothing.

Example: Rectangle rect = **new** Rectangle();

Constructor

Default Constructor
public Rectangle( ) { }

- To create objects with different initial values, we can create constructors using the following syntax (basically, **overloading**):

```
public                    ( Parameter_List ) {
    // initialise instance variables
}
```

**Examples:**

```
public Rectangle( ) {
    width = 1.0 ; height = 1.0 ;
}
```

No parameter

Initialise data properties/ attributes

```
public Rectangle( double w , double h ) {
    width = w ;
    height = h ;
}
```

With parameters

Initialise data properties using parameter values

# Constructors

**Rectangle**

- width: double
- height: double

+ Rectangle()
+ Rectangle(w: double, h : height)
+ findArea(): double
+ findPerimeter(): double

```
public class Rectangle {
  private double width ;
  private double height ;
  public Rectangle() {
     width = 1.0 ;
     height = 1.0 ;
  }
  public Rectangle( double w , double h ) {
     width = w ;
     height = h ;
  }
  public double findArea()     // methods
        { return width * height; }
  public double findPerimeter() {
      return( width + height ) * 2 ;
  }
}
```

# Creating Rectangle Objects

## Calling the two constructors:

1. Using default values for width and height:

   Rectangle rect1 = new Rectangle();

```
public Rectangle() {
     width = 1.0 ;
    height = 1.0 ;
 }
```

2. Using argument values 10.0 and 20.0 for width and height:

   Rectangle rect2 = new Rectangle( 10.0 , 20.0 );

width height

```
public Rectangle( double w , double h ){
     width = w ;
     height = h ;
 }
```

# Finalizer

Now you know about a **Constructor!**

Is there also a **Destructor ?**

*In **Java**, it is called* **finalizer**

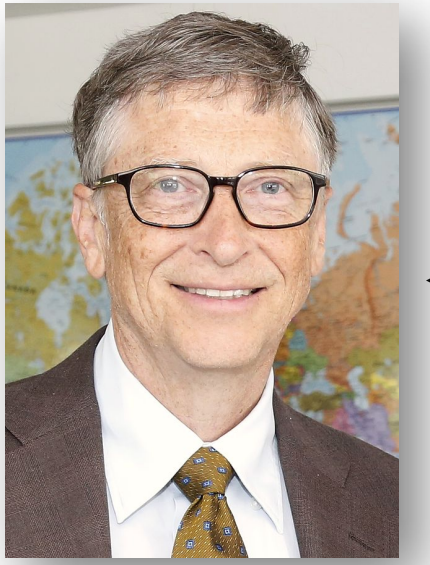
- **finalize( ) method**
    - **release resources**

**Garbage Collector**

# Topic 3: Message Sending

# Message Sending

When I want to ask the student **Bill** to come down here, I will send a voice message to him by saying.



"Bill, please come down!"

**Ask an object to do something.**

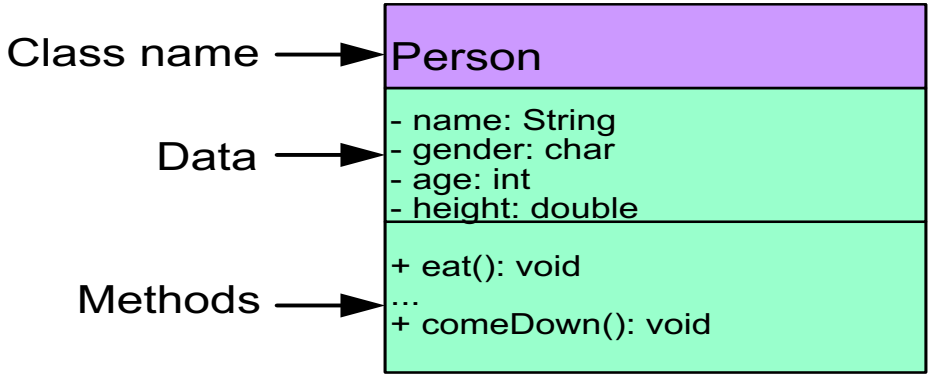
Walking down

# Message Sending

**Syntax:**

**Object_Variable_Name.Instance_Variable**
**Object_Variable_Name.Instance_Method( Argument_List );**

**Example:**




```
Person bill ;

bill.comeDown( );
   // make sure bill reference an object that exists.
```

**NullPointerException Error!!**

Object reference | dot operator | Method name | Argument(s) to the method

**Example:**

```
(1) public class Rectangle {
        private double width = 5.0 ;
        private double height = 10.0 ;

        public double findArea() {
          return width * height ;
        }
        public double findPerimeter() {
          return ( width + height ) * 2;
        }
    }
```

Data properties

Methods

# Testing with Application Class

```
(2) public class RectangleApp {
      public static void main( String[] args ){
        Rectangle rect = new Rectangle();   // instantiation
        System.out.println(
          "The area of rectangle is "
          + rect.findArea() );
        System.out.println(
          "The perimeter of rectangle is "
          + rect.findPerimeter() );
      }
    }
```

Object reference

Method name

```
Program Input and Output
The area of rectangle is 50.0
The perimeter of rectangle is 30.0
```

# Topic 4: Copying Objects

# Copying Objects

- Consider the following code:

```
public class CopyObjectsApp {
   public static void main(String[] args){
        Rectangle rect1, rect2;
        rect1 = new Rectangle( 10.0 , 20.0 );
        rect2 = rect1;
   }
}
```

➡ **This code will create only one object with two object references rect1 and rect2 both pointing to it.**

- **copyObjects()** method:

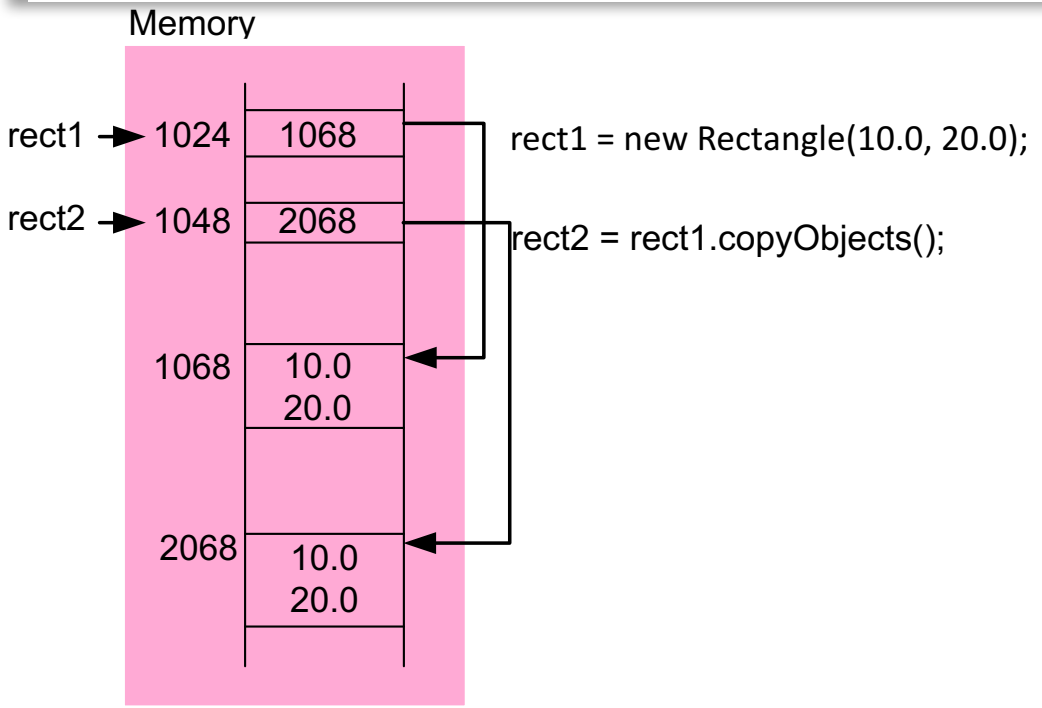
```
public class Rectangle {
   // instance variables and methods
   ...
   public Rectangle copyObjects() {
        Rectangle rect = new Rectangle( width , height );
        return rect;
   }
}
```

➡ **This method will create an object using the data properties and pointed at by reference rect.**

# Application Class

```
public class CopyObjectsApp {
   public static void main(String[] args){
      Rectangle rect1, rect2;
        rect1 = new Rectangle( 10.0 , 20.0 );
        rect2 = rect1.copyObjects();
   }
}
```

This method will create two separate objects with the same data properties and pointed at by references rect1 and rect2 respectively.



# Topic 5: The Keyword ‘this’

# The Keyword 'this'

- The keyword **this** refers to the **receiver object** with which you **call the method (or access the object's variable)**.
- Actually, **this** is the **object reference** that stores the receiver object.
- If the receiver object is not specified when a method is invoked, Java will use **this** automatically.
- The syntax is:

**this**.Method_Name;

```
public class Rectangle {
……
 public void print() {
    System.out.println("The area of rectangle is "
                       + this.findArea() );
    System.out.println("The perimeter of rectangle is "
                       + this.findPerimeter() );
 }
}
```

Current instance of this class

# The Keyword 'this'

```
public class Rectangle {
  private double width ;   // data properties
  private double height ;
  public Rectangle () {    // constructors
     width = 1.0 ;
     height = 1.0 ;
  }
  public Rectangle ( double w , double h ) {
     width = w ;
     height = h ;
  }
  public double findArea()     // methods
        { return width * height; }
  public double findPerimeter() {
      return( width + height ) * 2 ;
  } . . .
  public void print() {
    System.out.println("The area of rectangle is "
                       + this.findArea() );
    System.out.println("The perimeter of rectangle is "
                       + this.findPerimeter() );
 }
}
```

# The Keyword 'this'

- The '**this**' keyword can also be used for **instance variables**.
- The syntax:

```
this.Instance_Variable;
```

```
E.g.
  public class Rectangle {
  . . .
    public double findArea() {
      return this.width * this.height ;
    }
  }
In the statement:
  double area1 = rect1.findArea();
  double area2 = rect2.findArea();
```

- The keyword '**this**' above contains the value (reference) of rect1 when calling rect1.findArea() in the first statement.

# The Keyword 'this'

- **This** is commonly used in **constructors**.
- Note that the names of **parameters** or variables declared within constructors or methods must not be the same as the names for the instance variables.

```
public Rectangle(double width, double height){
    width  = width  ;
    height = height ;
}
```

To solve this problem, we can use **this**:

```
public Rectangle(double width, double height)
{
    this.width  = width  ;
    this.height = height ;
}
```

- The syntax form: **this(…);** refers to constructor. E.g. this(10.0, 20.0); activates the Rectangle (double width, double height) constructor.

# The Keyword 'this'

- **This** is co
- Note tha
  construc
  instance

```
public R
    width
    heigh
}
```

To solve th

```
public Rectangle(double width, double height)
{
    this.width  = width  ;
    this.height = height ;
}
```

```
public class Rectangle {
  private double width  ;      // data properties
  private double height ;
  public Rectangle() {
      width  = 1.0 ;
      height = 1.0 ;
  }
  public Rectangle( double width , double height ) {
      this.width  = width ;
      this.height = height ;
  }
……..................................

}
```

- The syntax form: **this(…);** refers to constructor. E.g. this(10.0, 20.0); activates the Rectangle (double width, double height) constructor.

# The Keyword 'this'

- **This** is co
- Note tha
  construc
  instance

```
public R
    width
    heigl
}
```

To solve th

```
public Rectan
{
    this.widtl
    this.heigl
}
```

- The syntax fo
  activates the

```
public class Rectangle {
  private double width  ;      // data properties
  private double height ;
  public Rectangle() {
     width  = 1.0 ;
     height = 1.0 ;
  }
  public Rectangle( double width , double height ) {

  }
.............
}
```

```
public class Rectangle {
  private double width  ; // data properties
  private double height ;
  public Rectangle () {        // constructors
     this(1.0, 1.0);
  }
  public Rectangle( double w , double h ) {
     width  = w ;
     height = h ;
  }
.................................
}
```

# Topic 6: Accessors and Mutators

# Accessors and Mutators

- Question: Can we change the values of the **data properties** (state) of an object by another object or function **directly**?
  - NO!!!
  - Well, OK, yes, but it is considered a very bad OO design.

```
public double height ;
```

- Two common types of methods:
  - **Accessors** (or **get** methods): Responsible for returning the value of a data property.
  - **Mutators** (or **set** methods): Responsible for changing the values of a data property (Help to maintain <u>data consistency</u>).

# Accessors and Mutators

```
public class Rectangle {
    private double width  ;
    private double height ;
    public Rectangle() {
        this(1.0, 1.0);
    }
    public Rectangle(double width, double height) {
        this.width  = width ;
        this.height = height ;
    }
    public void setWidth(  double w ) { width  = w ; }
    public void setHeight( double h ) { height = h ; }
    public double getWidth()  { return width  ; }
    public double getHeight() { return height ; }
    public double findArea() { return width * height ; }
    public double findPerimeter() {
        return ( width + height ) * 2; }
}
```

1

2

3

4

Set methods

Get methods

# Accessors and Mutators

```
public class GetAndSetApp {
   public static void main( String[] args ) {
      Rectangle rect= new Rectangle(5.0,10.0); print(rect);
      rect.setWidth(  20.0 ); print( rect );
      rect.setHeight( 40.0 ); print( rect );
   }
   public static void print(Rectangle r) {
     System.out.println( "Width = "    + r.getWidth()
                       + "; Height = " + r.getHeight() );
     System.out.println( "The area of rectangle is "
                       + r.findArea() );
     System.out.println( "The perimeter of rectangle is "
                       + r.findPerimeter() );
   }
}
```

**Program Output**

```
Width = 5.0; Height = 10.0
The area of rectangle is 50.0
The perimeter of rectangle is 30.0
Width = 20.0; Height = 10.0
The area of rectangle is 200.0
The perimeter of rectangle is 60.0
Width = 20.0; Height = 40.0
The area of rectangle is 800.0
The perimeter of rectangle is 120.0
```

# Topic 7: The Keyword ‘static’

# The Keyword 'static'

- The **static** keyword can be used in the declaration of an instance variable or method.
- Applied to the **whole class** instead of individual objects.
- Static variables - class variables
- Static methods - class methods

**1. Static Variables (per class)**

- Count number of created **Rectangle** objects:

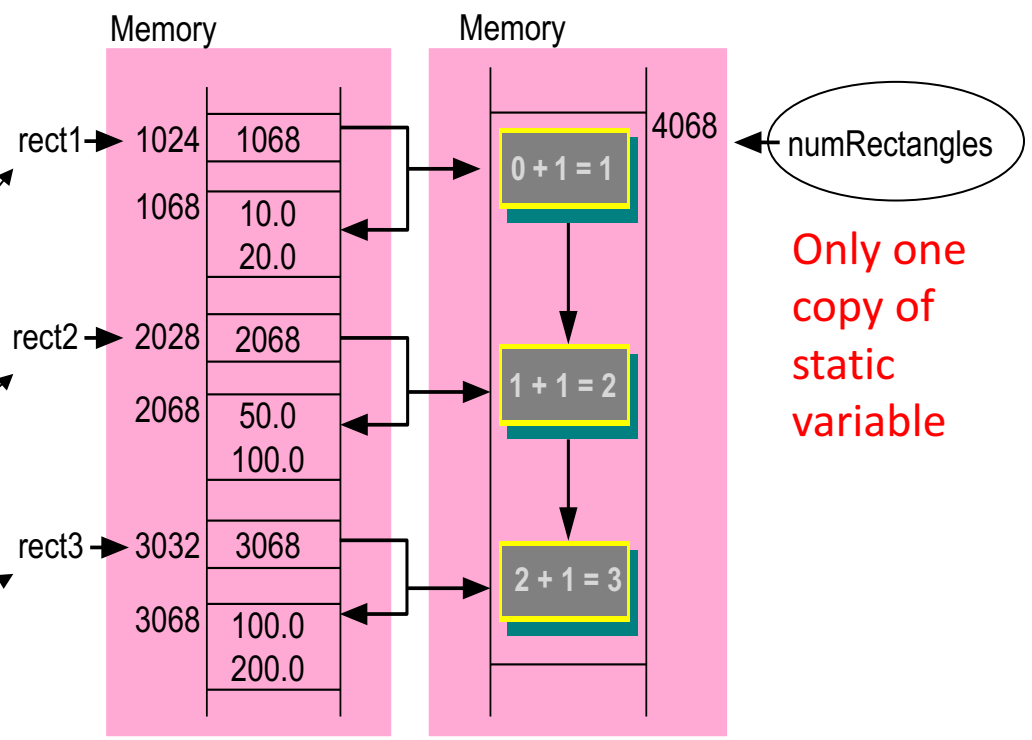
```
private static int numRectangles ;
```

- A **common** separate memory location is allocated to store the value for the static instance variable, and only a **single copy** of the static instance variable is allocated for all objects. i.e. the static variable numRectangles is **shared** by **all objects of the class Rectangle**.

# The Keyword 'static'

To count the number of **Rectangle** objects that are created, we can modify the **constructors**:

```
public Rectangle()
   {
     width  = 1.0 ;
     height = 1.0 ;
     numRectangles++;
   }
   public Rectangle
   (double w , double h)
   {
     width  = w ;
     height = h ;
     numRectangles++;
   }
```




Only one copy of static variable

```
Rectangle rect1 = new Rectangle(  10.0 ,  20.0 );
Rectangle rect2 = new Rectangle(  50.0 , 100.0 );
Rectangle rect3 = new Rectangle( 100.0 , 200.0 );
```

# Static Methods (Methods belong to the Class)

- The **static** keyword can also be applied to any **methods** in the class Rectangle.

```
public class Rectangle {
  // instance variables and methods
  private static int numRectangles = 0;
      ...
  // class methods
  public static void setNumRectangles(int number){
     numRectangles = number;
  }
  public static int getNumRectangles() {
     return numRectangles;
  }
}
```

Initialise static variable (class data)

Set method

Get method

- Since these two methods **do not** manipulate any of the **Rectangle** non-static object data, we define them as **class methods**.

# Static Methods (Methods belong to the Class)

- To access a class method outside its class, we use:

```
Class_Name.Method_Name();
```

```
Application Class:
   public class UsingStaticApp {
        public static void main(String[] args){
          Rectangle rect1 = new Rectangle( 5.0, 10.0);
          Rectangle rect2 = new Rectangle(50.0,100.0);
          System.out.println("Number of rectangles = "
            + Rectangle.getNumRectangles());
        }
   }
```

Class name

Method name

The Math class is an excellent example of calling “class methods”.

# Static Methods (Methods belong to the Class)

- Also notice that **class methods** can reference **class variables and methods**, but **not** **instance** variables and methods in the class.

```
public class Rectangle {
  // instance variables and methods
  ...
  public double findArea() {return width * height; }
  public static void countArea() {
     findArea();    // an error
  }
}
```

- This will give an error as the receiver object for the method findArea() is the object reference this. This violates the use of class method as it does not apply to individual object's methods.

➡ Can we just add **static** in front of findArea() and make it a class method???

# Class Constants (Class Variables that are Constant)

- Use keywords **static final**.

```
public static final double MAX_WIDTH  = 100;
public static final double MAX_HEIGHT = 100;

public void setWidth( double w ) {
   if ( w     >= MAX_WIDTH )
        width  = MAX_WIDTH ;    // data consistency
   else
         width  = w ;
}
public void setHeight( double h ) {
   if (  h     >= MAX_HEIGHT )
         height = MAX_HEIGHT ;
   else
         height = h ;
}
```

- **Advantages:** (1) Easy to change to new value.

  (2) Symbolic names are easier to understand.

# Complete Class Example

```
public class Rectangle {
   public  static final double MAX_WIDTH  = 100;
   public  static final double MAX_HEIGHT = 100;
   private double width  ;
   private double height ;
   // class variable
   private static int numRectangles=0;
   public Rectangle() {
      width  = 1.0;
      height = 1.0;
      numRectangles++;
   }
   public Rectangle( double w , double h ) {
      width  = w ;
      heigth = h ;
      numRectangles++;
   }
   public void setWidth( double w ) {
      if ( w     >= MAX_WIDTH )
           width  = MAX_WIDTH ;
      else
           width  = w ;
   }
```

Class constant (per class)

Instance variables (per object)

Static variable (per class)

Class constant

# Complete Class Example

```
    public void setHeight( double h ) {
       if ( h       >= MAX_HEIGHT )
            height  = MAX_HEIGHT ;
       else
            height = h ;
    }

    // instance methods
    public double getWidth()      { return width  ; }
    public double getHeight()     { return height ; }
    public double findArea()      { return  width * height ; }
    public double findPerimeter(){ return (width + height) * 2; }

    // class methods
    public static void setNumRectangles(int number) {
       numRectangles = number ;
    }
    public static int getNumRectangles() {
       return numRectangles ;
    }
}
```

Class constant

Instance methods

Static methods

# Application Class

```
public class UsingStaticApp {
   public static void main( String[] args ) {
      Rectangle rect1 = new Rectangle( 5.0 , 10.0 );
      print(rect1);
      rect1.setWidth(  40.0 );
      rect1.setHeight( 10.0 );
      print(rect1);
      Rectangle rect2 = new Rectangle( 50.0 , 100.0 );
    System.out.println( "Number of rectangles = " +
      Rectangle.getNumRectangles());
   }

   public static void print( Rectangle r ){
      System.out.println( "Width = "  + r.getWidth()
                        + "Height = " + r.getHeight() );
      System.out.println( "The area of rectangle is "
                        + r.findArea() );
      System.out.println( "The perimeter of rectangle is "
                        + r.findPerimeter() );
   }
}
```

static/class methods are called by class name

Calling instance methods

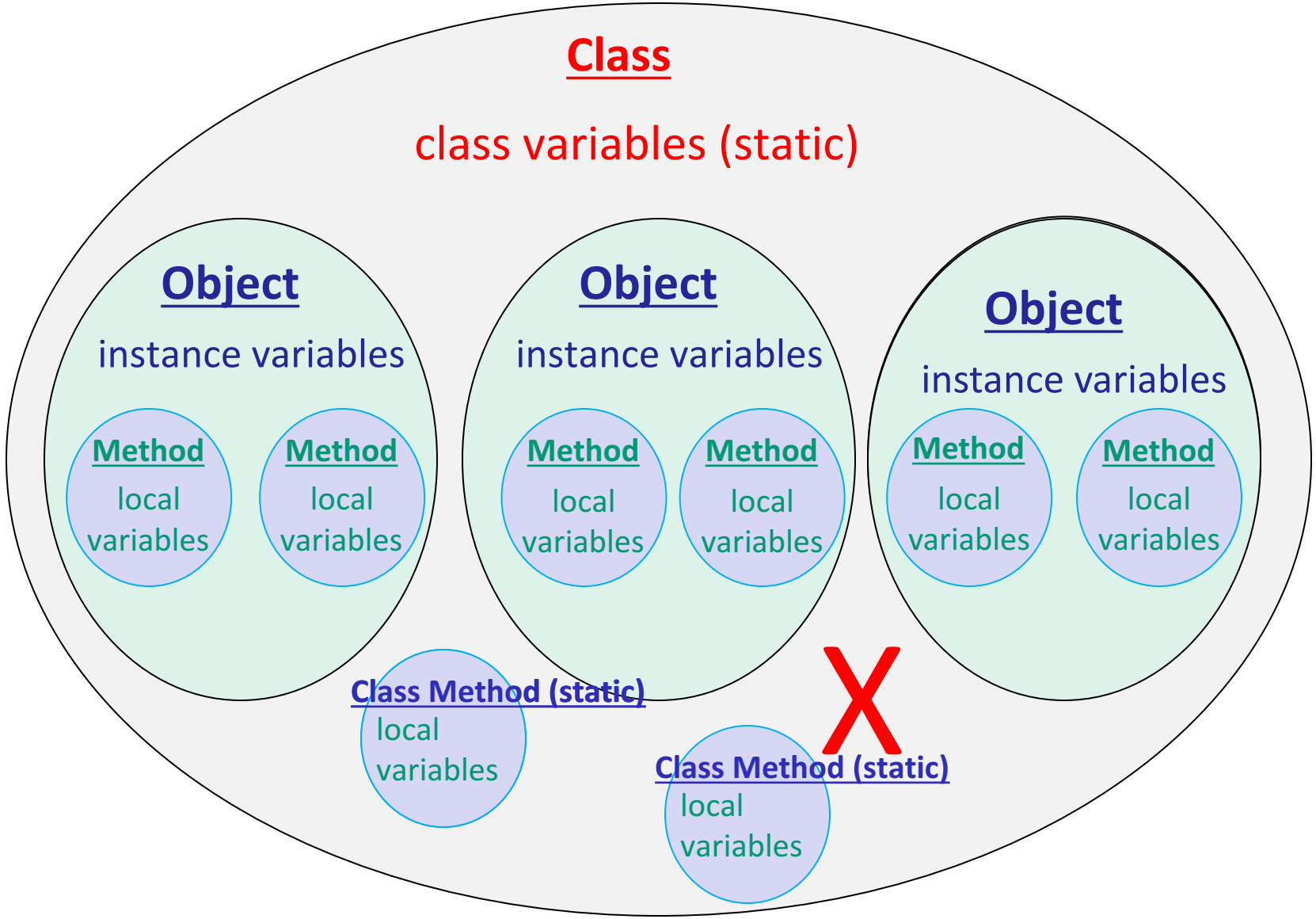
# Program Output

**Program Output:**

```
Width = 5.0; Height = 10.0
The area of rectangle is 50.0
The perimeter of rectangle is 30.0
Width = 40.0; Height = 10.0
The area of the rectangle is 400.0
The perimeter of rectangle is 100.0
Number of rectangles = 2
```

# Topic 8: Static vs. Instance methods

# Class


Class
class variables (static)
Object
instance variables
Method
local variables
Method
local variables
Object
instance variables
Method
local variables
Method
local variables
Object
instance variables
Method
local variables
Method
local variables
Class Method (static)
local variables
Class Method (static)
local variables

# Class

```
public class Person{
  double height  ;
  double weight ;

  public Person(double h, double w)
    height = h ;
    weight = w ;
  }
  public double getHeight() {
    return height ;
  }
  public double getWeight() {
    return weight ;
  }
  public double calculateBMI() {
    calculateBMI( height,  weight) ;
  }
  public static double calculateBMI(double height,
                                    double weight) {
    return (weight/(height * height));
  }
}
```

| Category | BMI |
|---|---|
| Very severely underweight | < 15 |
| Severely underweight | 15 – 16 |
| Underweight | 16 – 18.5 |
| Normal (healthy weight) | 18.5 – 25 |
| Overweight | 25 – 30 |
| Obese Class I | 30 – 35 |
| Obese Class II | 35 – 40 |
| Obese Class III | > 40 |

# Class

```
public class PersonApp{
  public static void main(String[] args) {
    Person bill = new Person(1.80 , 70);
    double bmi = bill .calculateBMI(); //instance method

    System.out.println("BMI is " + bmi);

    bmi =  bill .calculateBMI(bill.getHeight(), bill.getWeight());

    System.out.println("BMI is " + bmi);

    bmi = Person.calculateBMI (1.90, 90);

    System.out.println("BMI is " + bmi);
  }
}
```

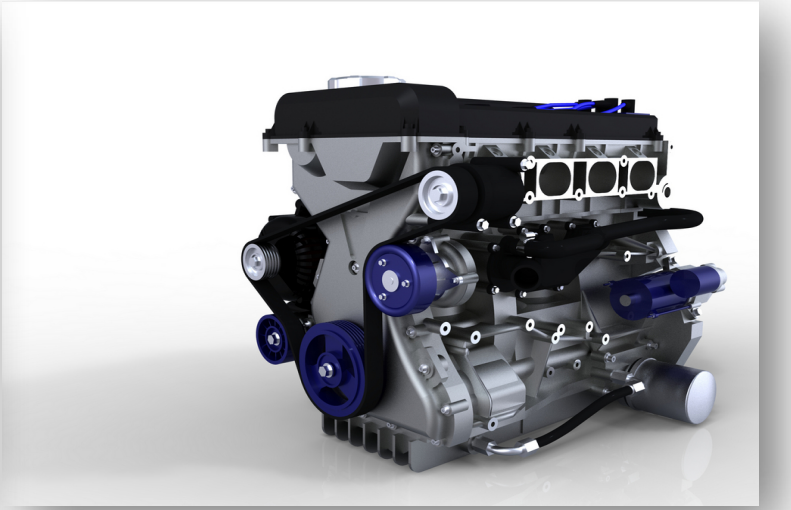
```
BMI is 21.604938271604937
BMI is 21.604938271604937
BMI is 24.930747922437675
```

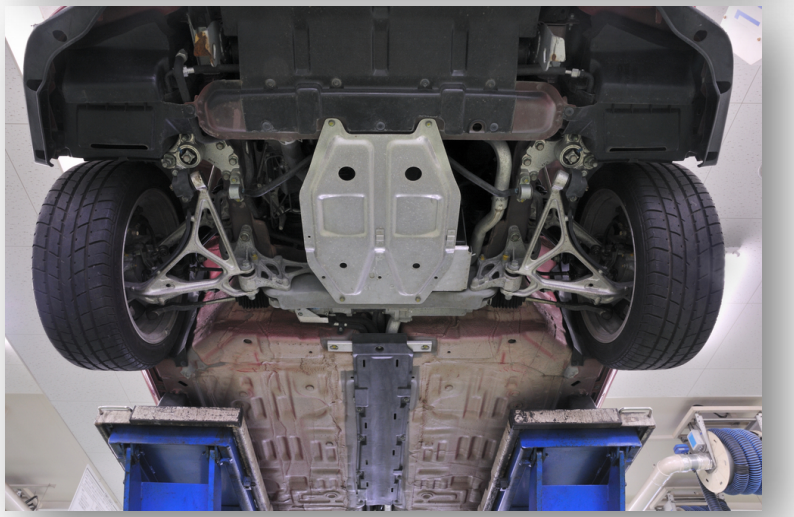
# Topic 9: Encapsulation and Information Hiding

# Encapsulation and Information Hiding

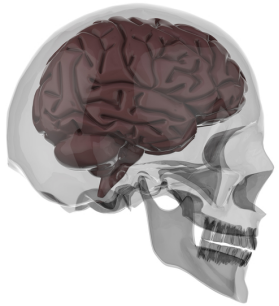
- **Encapsulation** – Builds a barrier to protect an object's private data. Access to private data can be done through the public methods of the object's class (e.g. get and set methods).
- **Information hiding** – Hides the details or implementation of the class from users.
  - With encapsulation, the users of a class only need to know **what** a class does and **how** to call the methods to perform the tasks.
  - The users **do not** need to know the **implementation** details for the methods.

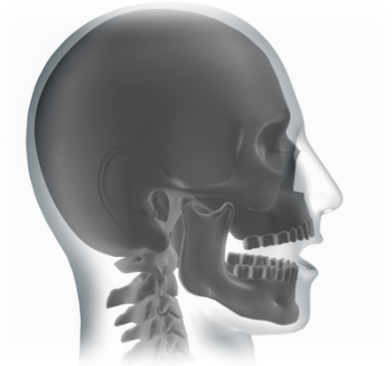
# Encapsulation and Information Hiding

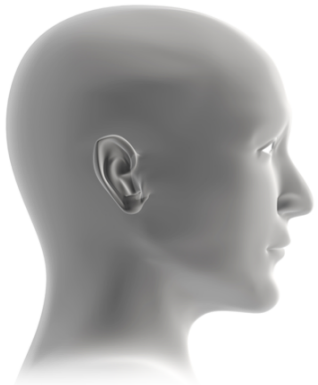




My name is Shadow, I am 48 years old and weigh 70 kg.





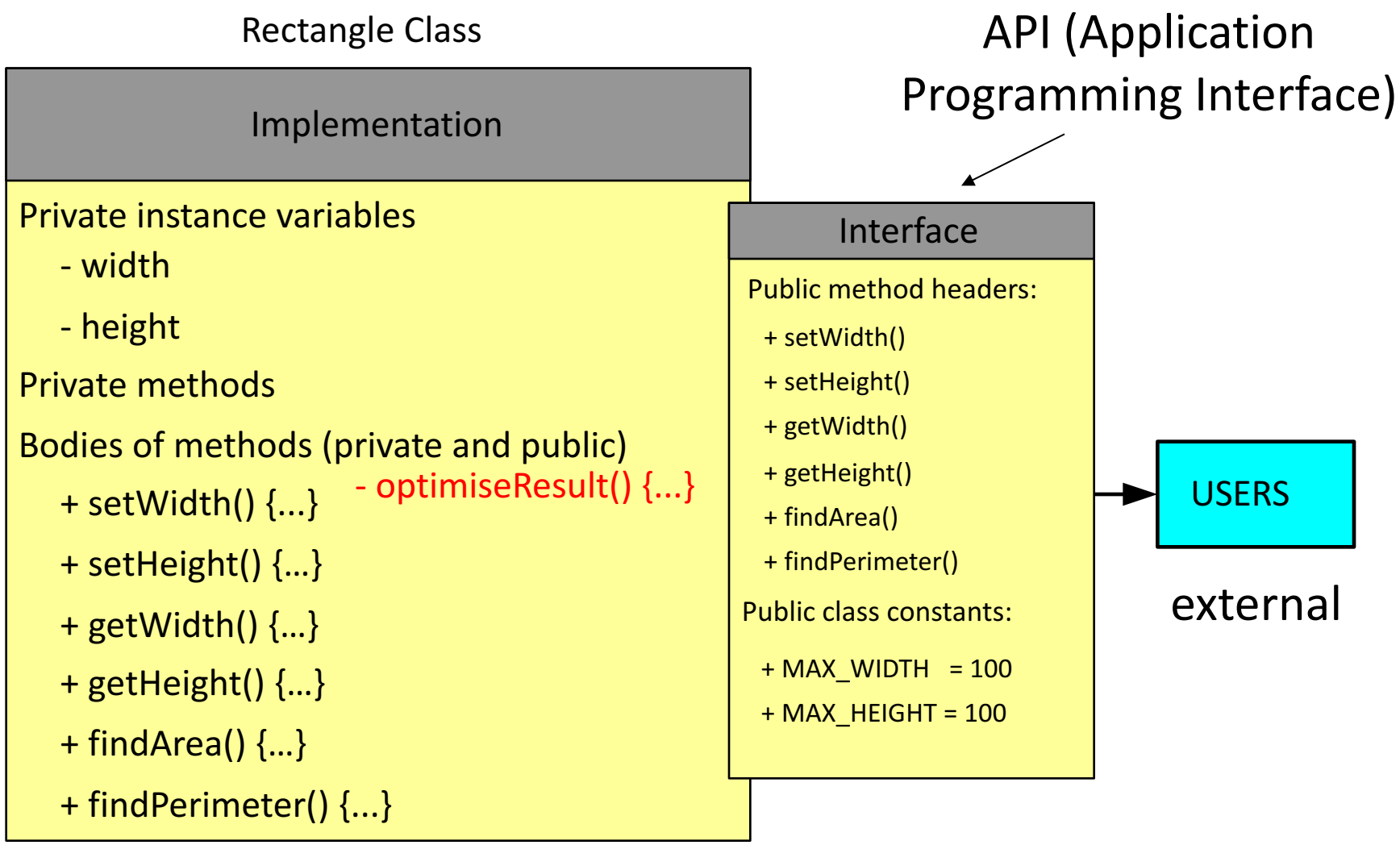


# Encapsulation and Information Hiding

Encapsulation and Information Hiding help to achieve the following:

- Protect what you have
- Protect your data

# Example of Information Hiding



# Topic 10: Object Composition

# Object Composition

- An object can include other objects as its **data** member. This is called **object composition**.
- A class contains object references from other classes as its instance variables.
- Object references are of reference data types.
- In OO, this is called the "has-a" relationship.
- Example: a student *has a* course.

**Student object**

studentId: 89076

gender: m

height: 1.77

age: 25

course

```
public class Student {
    private int    studentId;
    private char   gender;
    private double height;
    private int    age;
    private Course course;
```

# Object Composition

```
public class Course {
    private int courseId;
    private int year;
    private int semester;
    private int score;
    public Course( int courseId , int year  ,
                   int semester , int score ) {
        this.courseId = courseId ;
        this.year     = year     ;
        this.semester = semester ;
        this.score    = score    ;
    }
    public void setCourseId(int courseId) {
        this.courseId = courseId; }
    public void setYear(int year) {
        this.year     = year;     }
    public void setSemester(int semester) {
        this.semester = semester; }
    public void setScore(int score) {
        this.score    = score;    }
```

=> Define the Course class

=> Constructor

=> Set methods

# Object Composition

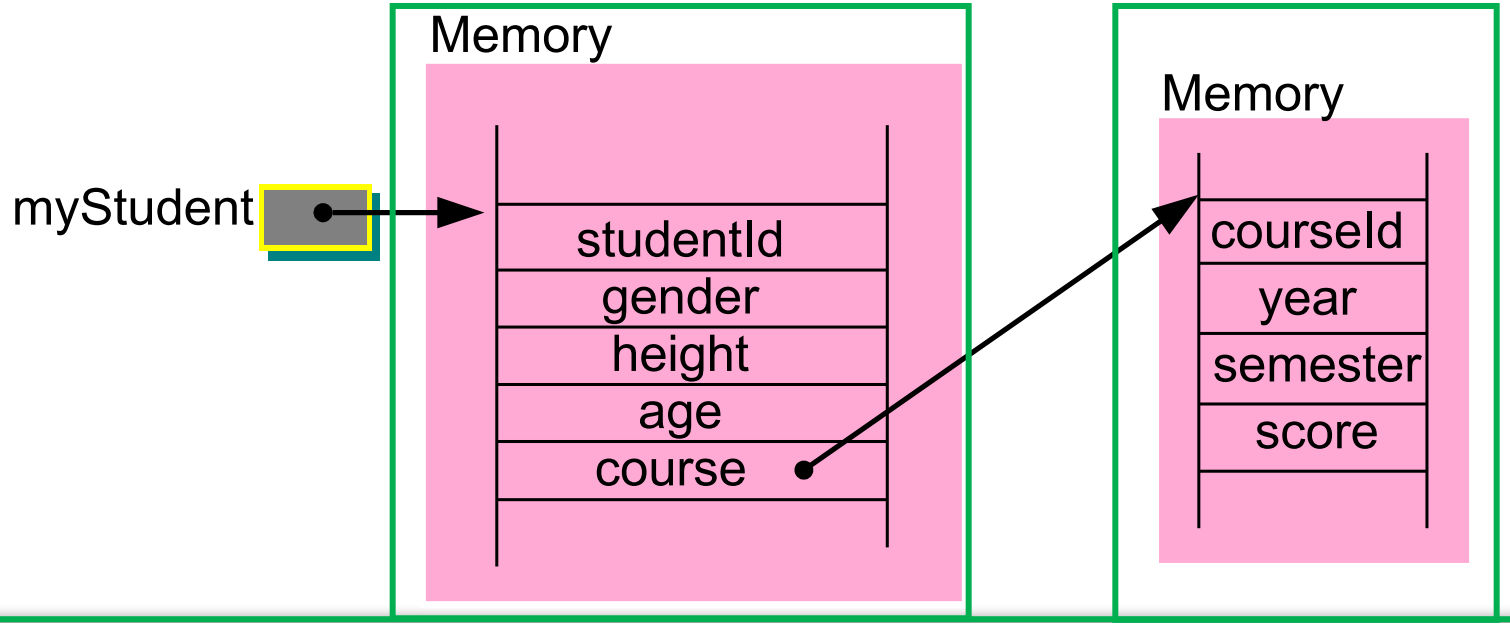
```
    public int getCourseId(){
        return courseId;
    }
    public int getYear(){
        return year;
    }
    public int getSemester(){
        return semester;
    }
    public int getScore(){
        return score;
    }
}
```

=> Get methods

```
public class Student {
    private int    studentId;
    private char   gender;
    private double height;
    private int    age;
    private Course course;
```

=> Define an object as an instance variable (data member)

# Object Composition




```
public Student(int studId, char sex, double
   studHeight, int studAge, Course courseIn)
{
     studentId = studId;
     gender    = sex;
     height    = studHeight;
     age       = studAge;
     course    = courseIn;
}
public int    getStudentId() { return studentId ; }
public double getHeight()    { return height    ; }
```

=> In the constructor

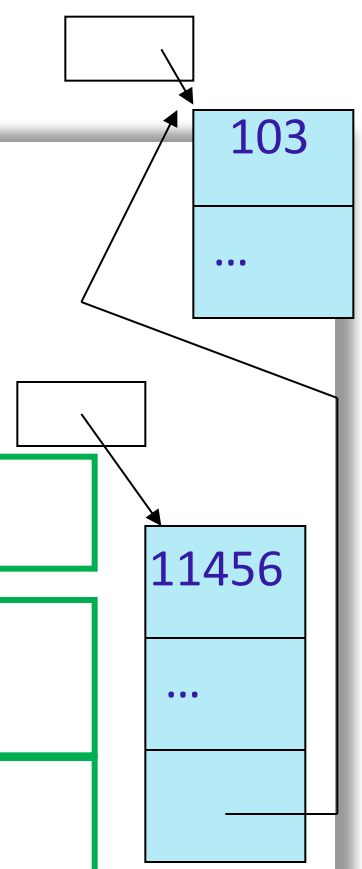
# Object Composition

```
    public int getAge() { return age ; }
    public void setStudentId( int studId ) {
        studentId = studId ;
    }
    public void setHeight( double studHeight ) {
        height = studHeight ;
    }
    public void setAge( int studAge ) { age = studAge ; }
    public void printInfo() {
        System.out.println( "=== Student Info ==="     );
        System.out.println( "Student ID: " + studentId );
        System.out.println( "Gender: "     + gender    );
        System.out.println( "Height: "     + height    );
        System.out.println( "Age: "        + age       );
        System.out.println( "Course: "
                            + course.getCourseId());
        System.out.println( "Year: " + course.getYear());
        System.out.println( "Semester: "
                            + course.getSemester());
        System.out.println( "Score: " + course.getScore());
    }
}
```

# Object Composition

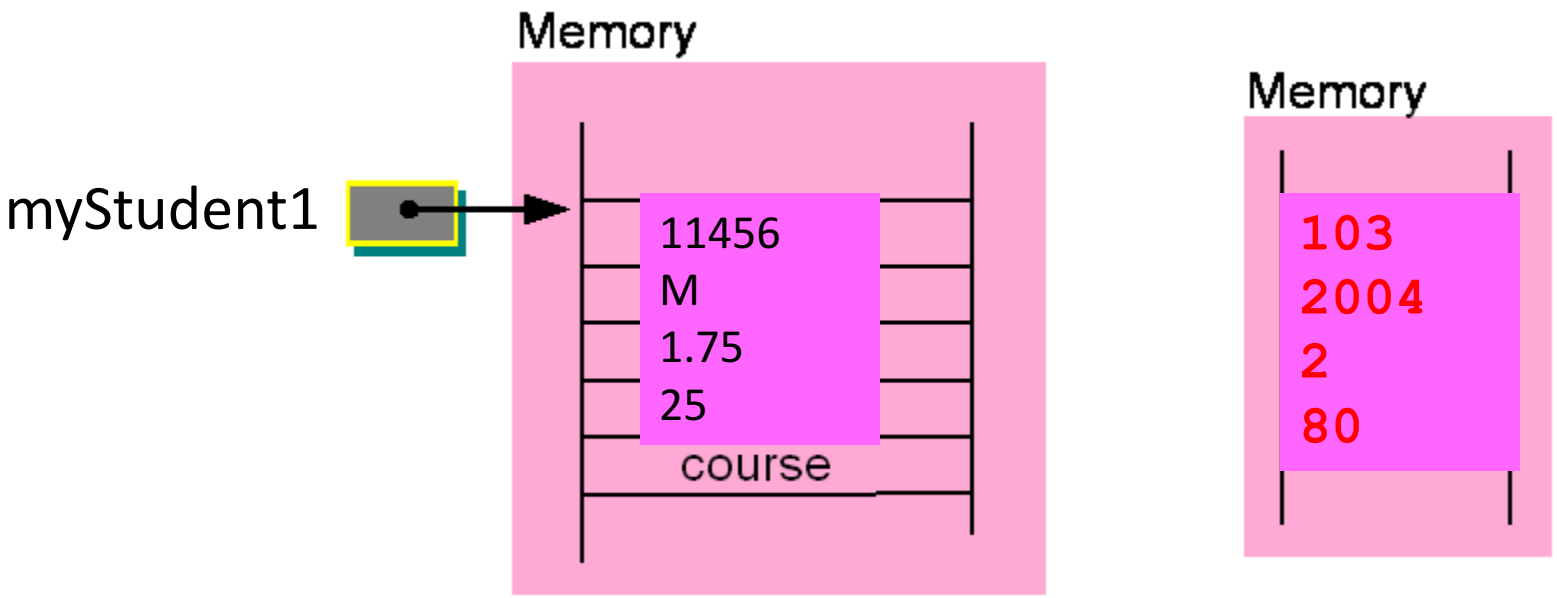
=> Define the application class

```
public class StudentApp {
  public static void main(String[] args){
   Course course1 = new Course(103,2004,2,80);

   StudentInfo myStudent1 =
        new Student(11456,'M', 1.75,25,course1);

    Course course2 = new Course(104,2004,1,60);
   StudentInfo myStudent2 =
        new Student(11457,'F', 1.55,21,course2);
   Course course3 = new Course(105,2004,1,90);
   StudentInfo myStudent3 =
        new Student(11458,'F', 1.85,28,course3);
   myStudent1.printInfo();
   myStudent2.printInfo();
   myStudent3.printInfo();
 }   }
```



# Object Composition

```
Course course1 = new Course(103,2004,2,80);
StudentInfo myStudent1 =
    new Student(11456,'M', 1.75,25,course1);
```

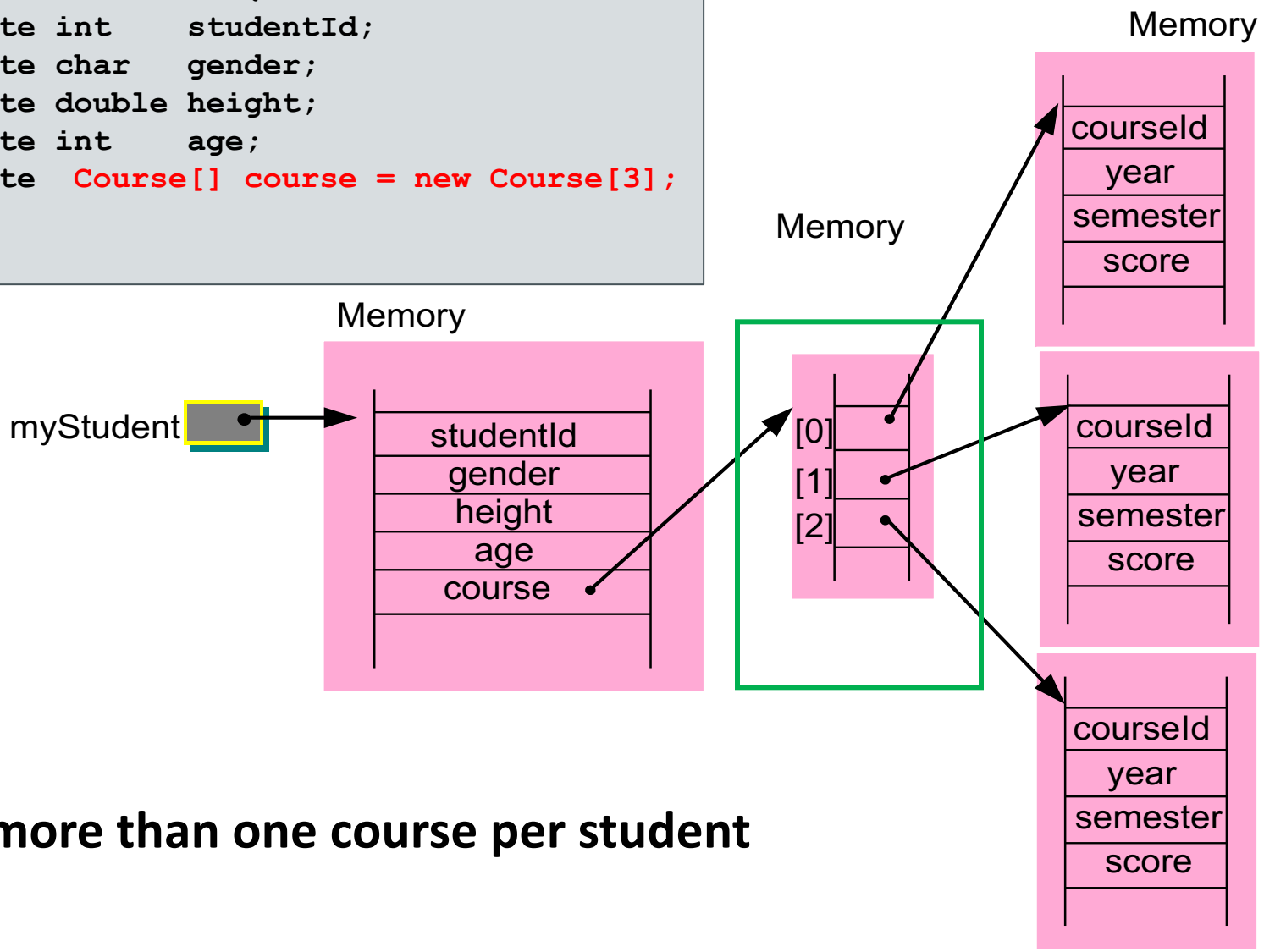


# Program Output

```
=== Student Info ===
Student ID: 11456
Gender: M
Height: 1.75
Age: 25
Course: 103
Year: 2004
Semester: 2
Score: 80
=== Student Info ===
Student ID: 11457
Gender: F
Height: 1.55
Age: 21
Course: 104
Year: 2004
Semester: 1
Score: 60
=== Student Info ===
Student ID: 11458
Gender: F
Height: 1.85
Age: 28
Course: 105
Year: 2004
Semester: 1
Score: 90
```

# Object Composition with Array of Objects

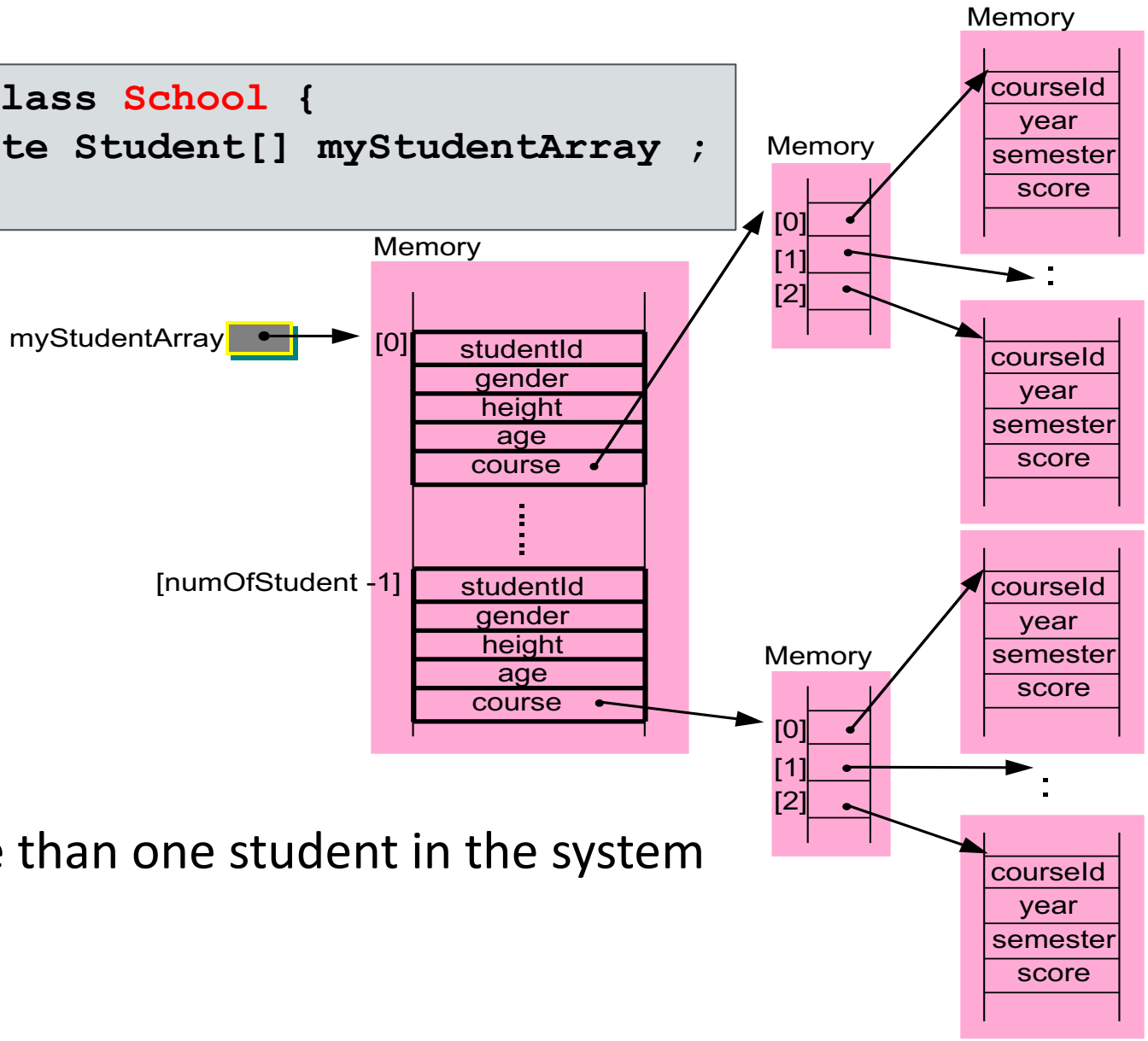
```
public class Student {
    private int    studentId;
    private char   gender;
    private double height;
    private int    age;
    private  Course[] course = new Course[3];
…
}
```




**Having more than one course per student**

# A Database of Composite Objects

```
public class School {
    private Student[] myStudentArray ;
    .........
```

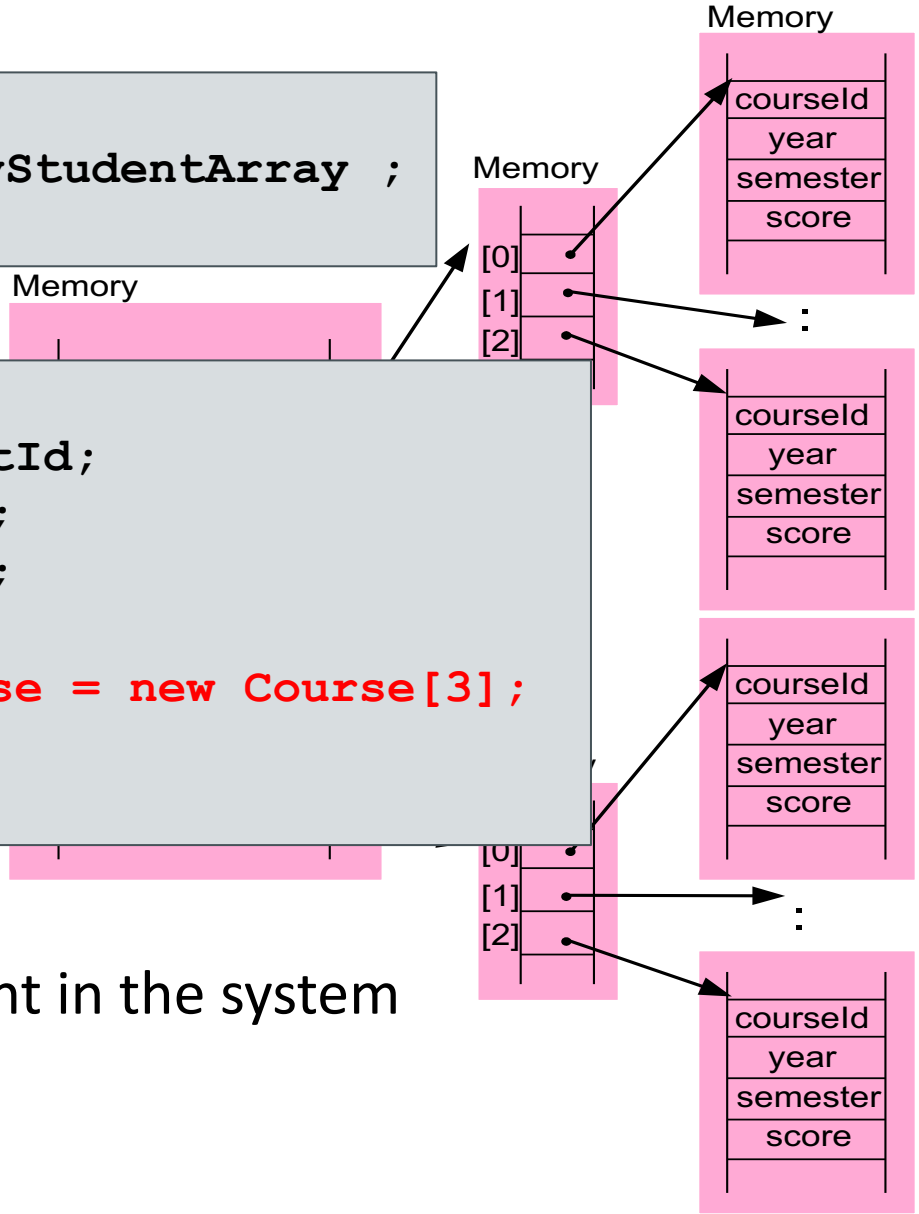



Having more than one student in the system

# A Database of Composite Objects

```
public class School {
    private Student[] myStudentArray ;
    .........
```

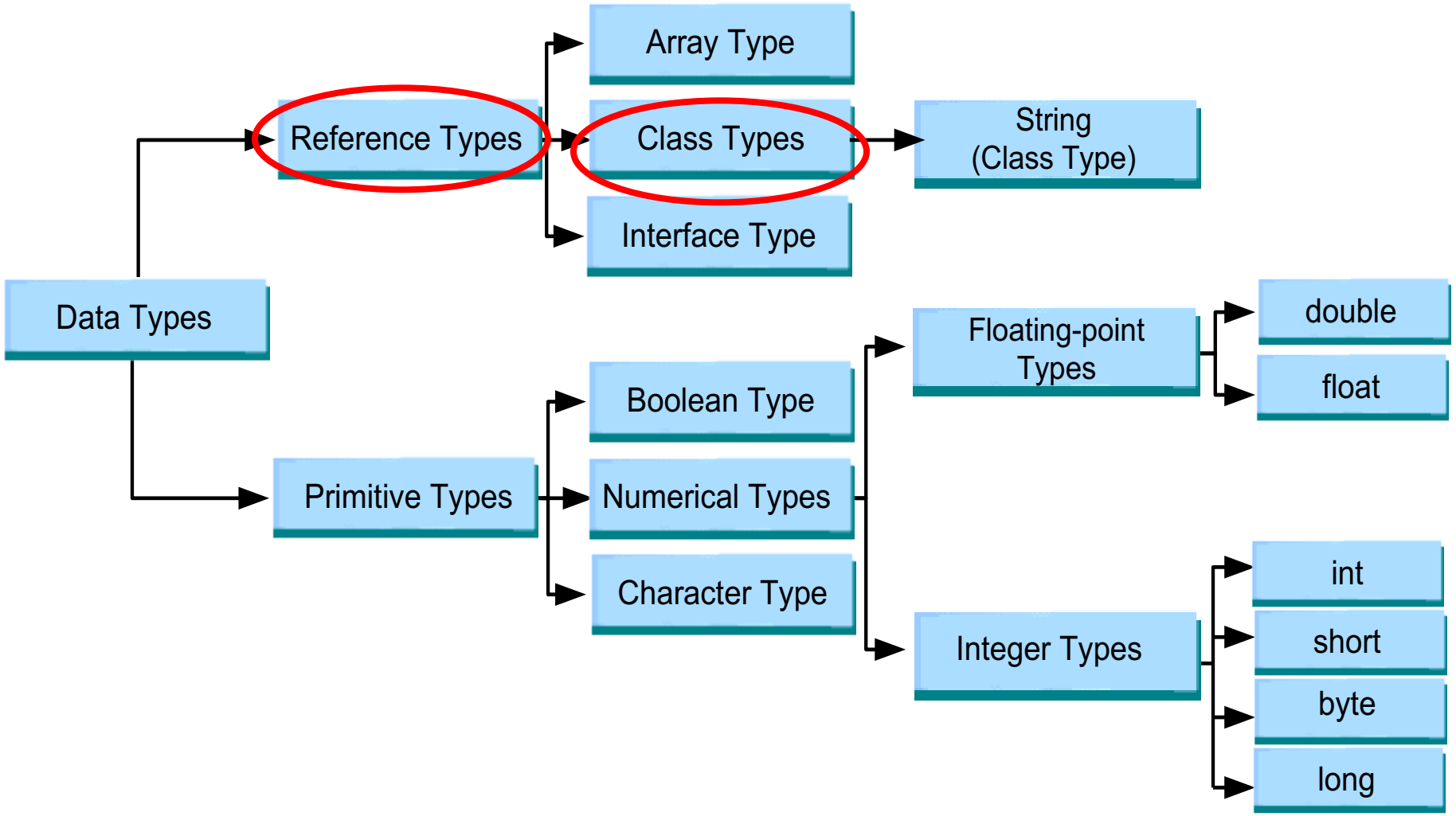
```
public class Student {
    private int     studentId;
    private char    gender;
    private double height;
    private int     age;
    private Course[] course = new Course[3];
…..
}
```




Having more than one student in the system

# Topic 11: String Class

# Data Types


Data Types
Reference Types
Array Type
Class Types
Interface Type
String
(Class Type)
Primitive Types
Boolean Type
Numerical Types
Character Type
Floating-point
Types
double
float
Integer Types
int
short
byte
long

# String Class

- java.lang.String is a class representing strings.
- A string or string constant is a series of characters in double quotes, e.g. “Java Programming”.
- In Java, strings are always created as objects.
- Syntax to declare a string:
  String Variable_Name;

- Since it is an object, memory will only be allocated when:
  Variable_Name = new String( String_Value );

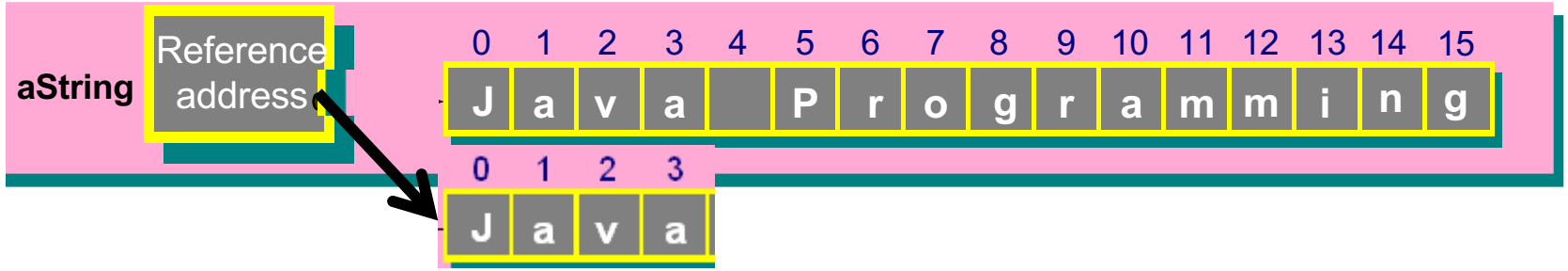
  Combined into one:
  String Variable_Name = new String( String_Value ) ;
  or
  String Variable_Name = String_Value ;

# Examples

**String is Immutable**

String aString = new String( “Java Programming” ); OR
String aString = “Java Programming” ;

aString = “Java” ;




The variable aString contains reference address of the string object
The length of the string is also stored in the string object’s storage.

String can be used as arguments for System.out.println()

```
System.out.println( “Java Programming” ); OR
System.out.println( aString );      aString.charAt(0)
```

# Java Naming Convention

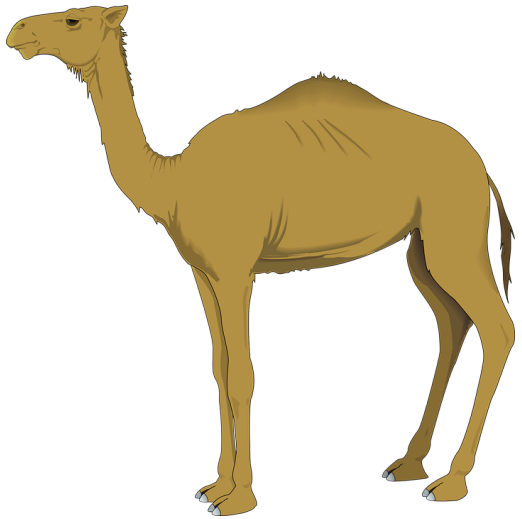
Using the right letter case is the key to following a naming convention:

1. **Lowercase** is where all the letters in a word are written without any capitalisation (e.g., packagename ). **For package name.**

2. **Uppercase** is where all the letters in a word are written in capitals. When there are more than two words in the name use underscores to separate them (e.g., MAX_HOURS, FIRST_DAY_OF_WEEK). **For constants, enums.**

3. **CamelCase** (also known as *Upper* CamelCase) is where each new word begins with a capital letter (e.g., CamelCase, CustomerAccount, PlayingCard). **For Classes and Interfaces.**

4. **Mixed case** (also known as *Lower* CamelCase) is the same as CamelCase except the first letter of the name is in lowercase (e.g., **h**as**C**hildren, **c**ustomerFirstName, For **methods, variables**.



# Summary

Key points from this chapter:

- Objects have states and behaviors. For example, a dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.

- A Class can be defined as a template/ blueprint that describes the behavior/ state that the object of its type support.

# Additional Reading

- Class Definition:
  - Text Book: Chapter 4, Page 75-86
- Constructors:
  - Text Book: Chapter 3, Page 53-60
- Java Programming:
  - Lecture Slides: Blackboard/ NTU Learn -> Content
  - Recorded Lectures: Blackboard/ NTU Learn
- C vs. Java
- Java Quick Guide
  - Blackboard/ NTU Learn-> Content -> Course -> Additional Resources