Object Oriented Programming with Java II

Dr. Mohamed K. Hussein

Associate Prof., Computer Science department,

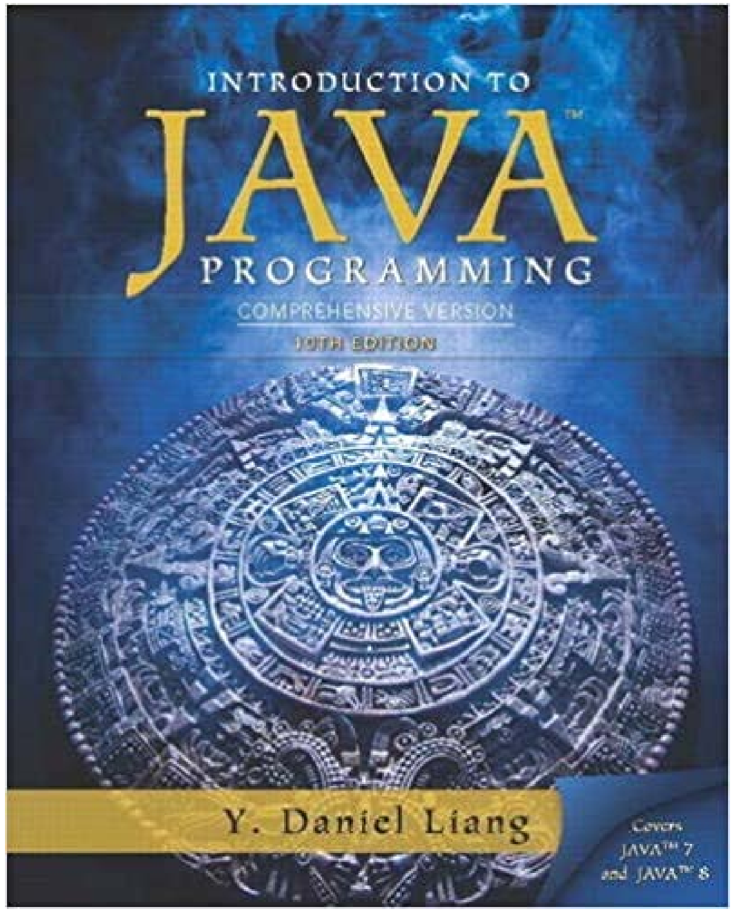
Faculty of Computers & Information Science,

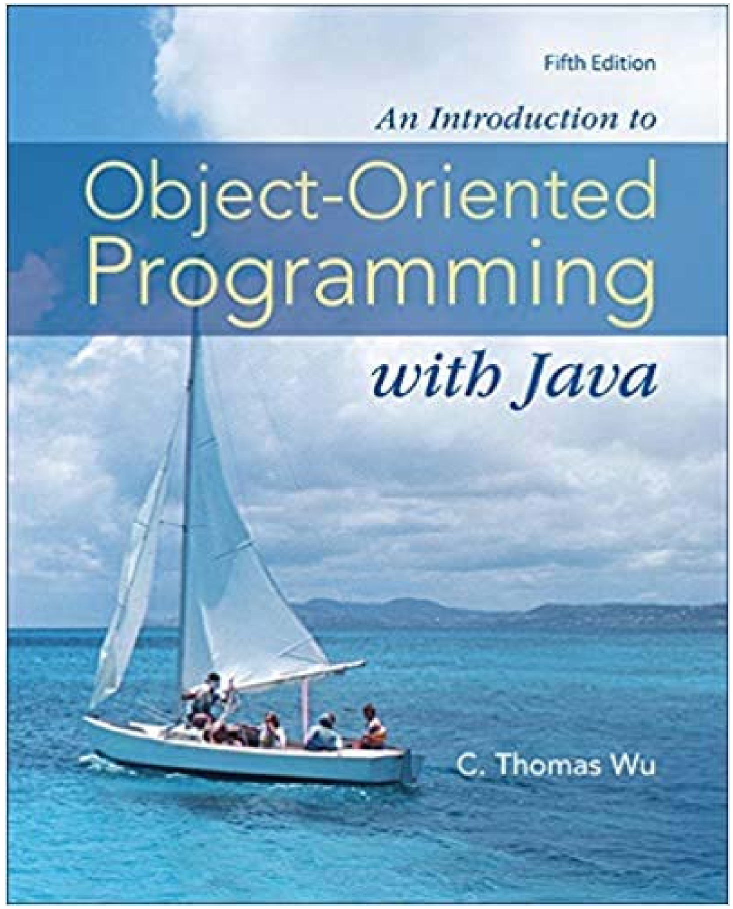
Suez Canal University

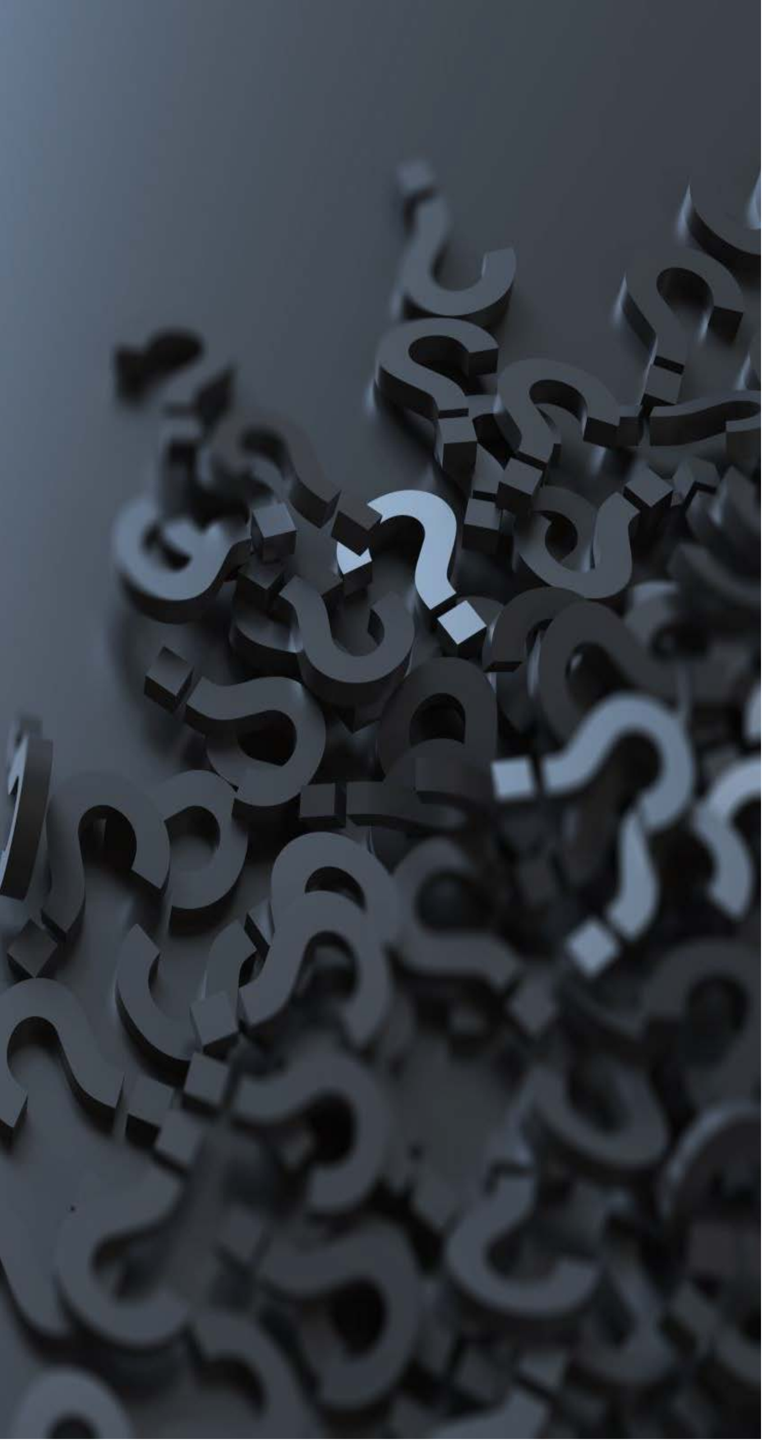
Email: m_khamis@ci.suez.edu.eg

Books

- **Intro to Java Programming**, Comprehensive Version, 10th Edition By Y. Daniel Liang Published by Prentice Hall, 2015.
- **An Introduction to Object-Oriented Programming with Java**, 5th Edition, 2009.







Lecture outcomes

- Introduction
  - Structured programming versus object-oriented programming.
- Classes versus objects
- How to define classes, instantiate objects and access different part of an object.
  - Class attributes
  - Class methods
- What is a constructor and how is it defined and used.
- Objects methods calling.
- Encapsulation/information hiding

Introduction

- A ***program*** is a list of instructions.
- A ***procedural program*** is divided into a number of functions.
  - Each function has a defined purpose and a defined interface to the other functions in the program.
  - Dividing a program into functions is called ***structured programming***

Disadvantage of Structured programming

- Large programs become excessively complex.
- There is *no data protection* where functions have unrestricted access to global data.
  - The restricted access is only provided to local variables.
  - A change in a single global data item may necessitate modifying all the functions that access that global data item.
  - That makes large programs very difficult to modify.

Dr. Mohamed K. Hussein

Disadvantage of Structured programming

- A poor model of the real world where functions and data are not related.
- In the physical world we deal with objects such as people, university, and cars.
  - Complex real-world objects have both attributes and behavior.
- ***Object-oriented Programming*** is to encapsulate both data and the functions that operate on that data in a single unit, called an ***object***.

Introduction to Objects

- An object represents something with which we can interact in a program.
  - consists of *member data and a group of methods*.
  - An object provides a collection of services that we can tell it to perform for us
    - The services are defined by methods in a class that defines the object
- A class represents a concept,
  - An object represents an instance of a class.
  - A class can be used to create multiple objects

Class versus Object Example

A class
(the concept)

Bank Account

An object
(the realization)

Adel’s Bank Account
Balance: $5,257

Ahmed’s Bank Account
Balance: $1,245,069

Multiple objects
from the same class

Mariam’s Bank Account
Balance: $16,833

OOP characteristics – Encapsulation

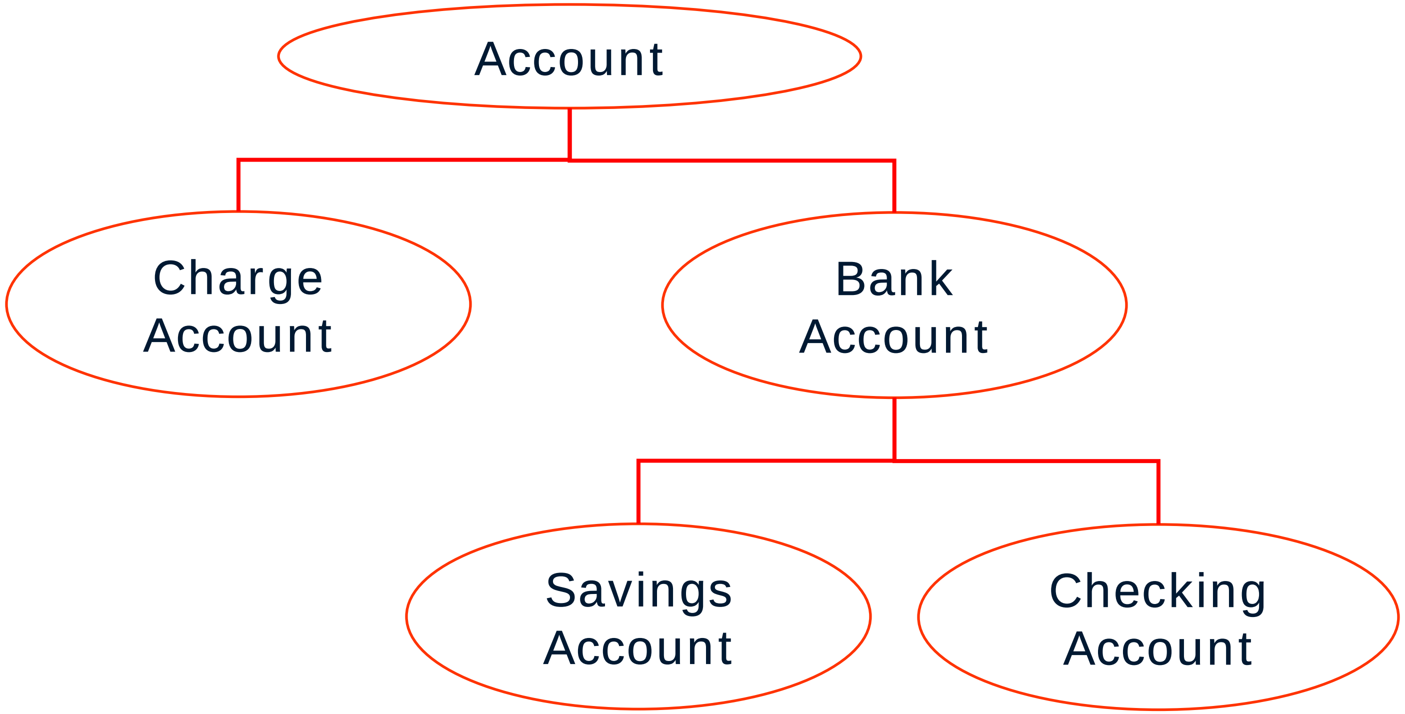
- *Data and its functions are encapsulated into a single entity.*
- ***Data encapsulation*** and ***data hiding***.
  - If you want to modify a certain data in an object, you simply call the object's method which interacts with it.

This simplifies writing and modifying large complex programs.

OOP characteristics – Inheritance

- One class can be used to derive another via *inheritance*

- Classes can be organized into inheritance hierarchies.



Class Vs Object

- A class is a data type that allows programmers to create objects.
  - A class provides a definition for an object,
    - Describing an object's attributes (data) and methods (operations).
  - A class serves as a plan, or ***blueprint***.
- Defining the class doesn't create any objects.
  - Acts as the existence of data types int, long, float, and double doesn't create any variables.

Class

- A class is a new type of variable.
- The class definition specifies:

  1. What descriptive data is needed?

     attributes = data

  2. What are the possible set of actions?

     methods = actions

     - A method is the Object-Oriented equivalent of a function.

Format:

```
  public class <name of class> {
       attributes
       methods
  }
```

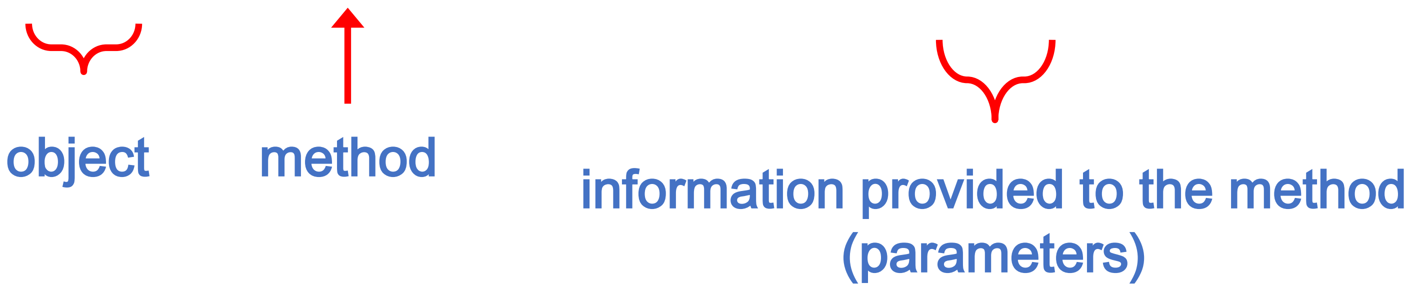
```
public class Person {
    private int age;             // Attribute
    public void sayHello() { // Method
          System.out.println(“Hi there”);
      }
}
```

Each class definition must occur its own “.java” file).

Using Objects

- The System.out object represents a destination to which we can send output
- The following statement, invokes the println method of the System.out object:

```
System.out.println ("Whatever you are, be a good one.");
```



object

method

information provided to the method
(parameters)

Using Objects

- A specific example or instance of a class.
- Objects have all the attributes specified in the class definition

```
public class MainClass {
  public static void main(String [] args){
        Person ahmed = new Person();
        ahmed.sayHello();
  }
}
```

```
Hi there ........
```

Attributes of a Class

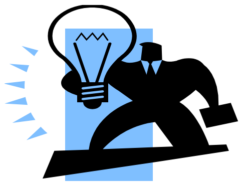
- Attributes: Data that describes each instance or example of a class.
  - Attributes can be variable or constant
  - Constants (preceded by the 'final' keyword).
- Different objects have the same attributes but the values of those attributes can vary
  - The class definition specifies the attributes and methods for *all objects*.
- The current value of an object's attribute's determines it's state.

Format:

```
<access modifier> <type of the attribute> <name
  of the attribute>;
```

Example:

```
public class Person {
        private int age;
        private int weight;
}
```



Age: 35
Weight: 192

Age: 50
Weight: 125

Age: 1
Weight: 7

Methods

- Possible behaviors or actions for each instance (example) of a class.

Walk()
Talk()

Walk()
Talk()

Fly()

Swim()

Methods of a Class

Method Format:

```
<access modifier> <return type>
 <method name> (<p1 type> <p1
 name>, <p2 type> <p2 name>…) {
     <Body of the method>
 }
```

Example:

```
public class Person {
    public int age;
    // Method definition
    public void sayAge() {
        age = 20;
        System.out.println("My age is " + age);
    }
}
```

Parameter Passing & Return values

Parameter type	Format	Example
Simple types	<*method*>(<*type*> <*name*>)	`method(int x, char y) { ... }`
Objects	<*method*>(<*class*> <*name*>)	`method(Person p) { ... }`
Arrays	<*method*>(<*type*> []… <*name*>)	`method(Map [][] m) { ... }`

Return type	Format	Example
Simple types	<*type*> <*method*>()	`int method() { return(0); }`

Objects	<class> <*method*>()	`Person method() {`
		`Person p = new Person();`
		`return(p);`
		`}`

Arrays	<*type*>[]... <*method*>()	`Person [] method() {`
		`Person [] p = new`
		`Person[3];`
		`return(p);`
		`}`

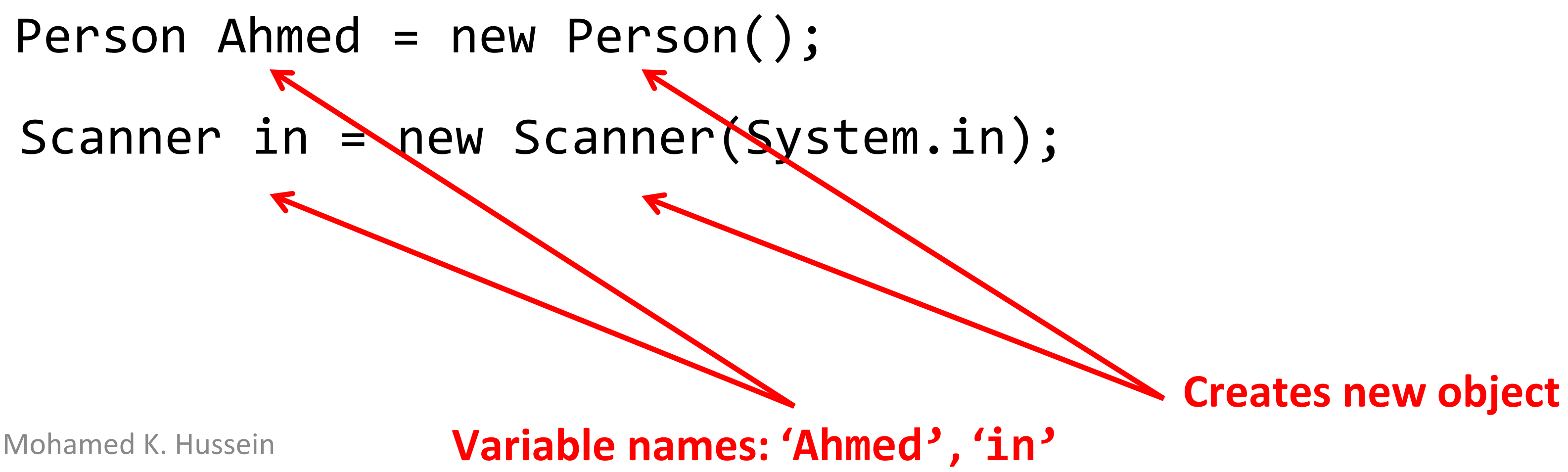
Object Instantiation

- Instantiation is creating a new instance or example of a class.
- Instances of a class are referred to as *objects*.

Format:

```
<class name> <instance name> = new <class name>();
```

Examples:



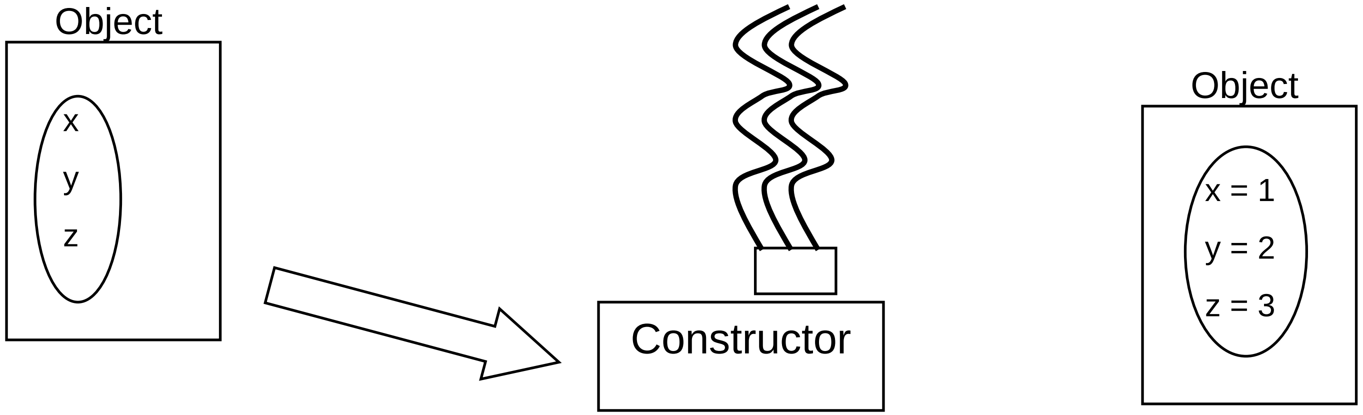
```
Person Ahmed = new Person();

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

Creates new object

Variable names: ‘Ahmed’, ‘in’

Constructor

- A special method to initialize the attributes of an object as the objects are instantiated (created).



Constructor

- The constructor is automatically invoked whenever an instance of the class is created.

  Person maryiam = new Person();

  Call to constructor (creates something 'new')

- Constructors can take parameters but **never** have a return type.

```
class Person {
    // Constructor
    public Person() {
        …
    }
}
```

Default Constructor

- Takes no parameters.
- If no constructors are defined for a class then a default constructor comes ‘built-into’ the Java language.

```
class MainClass {
    main() {
        Person aly = new Person();
    }
}

class Person {
    private int age;
}
```

Calling Methods:
Outside The Class You’ve Defined

- ➢Calling a method outside the body of the class (i.e., in another class definition)
- ➢The method must be prefaced by a variable (actually a reference to an object).

```
public class MyClass {
  public static void main(String [] args){
        Person ahmed = new Person();
        Person mona = new Person();
        // Incorrect! Who ages?
        SayHello();

        // Correct. Happy birthday Bart!
        ahmed.SayHello();
    }
}
```

Calling Methods: Inside the Class

- Calling a method inside the body of the class (where the method has been defined)
  - You can just directly refer to the method (or attribute).

```
public class Person {
  private int age;

  public void SayHi() {
    SayHello();  // access a method
  }

 public void SayHello() {
    System.out.println(“Hi there”);
 }
}
```

Example

```
import java.util.Scanner;
public class Person {
   private int age;

  public Person() {
       age = 0;
  }
  public void getAge() {
       Scanner in = new Scanner(System.in);
       System.out.print("Enter age: ");
       age = in.nextInt();
   }
   public void sayAge() {
       System.out.println("My age is " + age);
  }
}
```

Class Person

```
public class MainClass {

    public static void main(String [] args){

        Person aly = new Person();

        aly.getAge();

        aly.sayAge();

    }

}
```

Class MainClass

Private Keyword

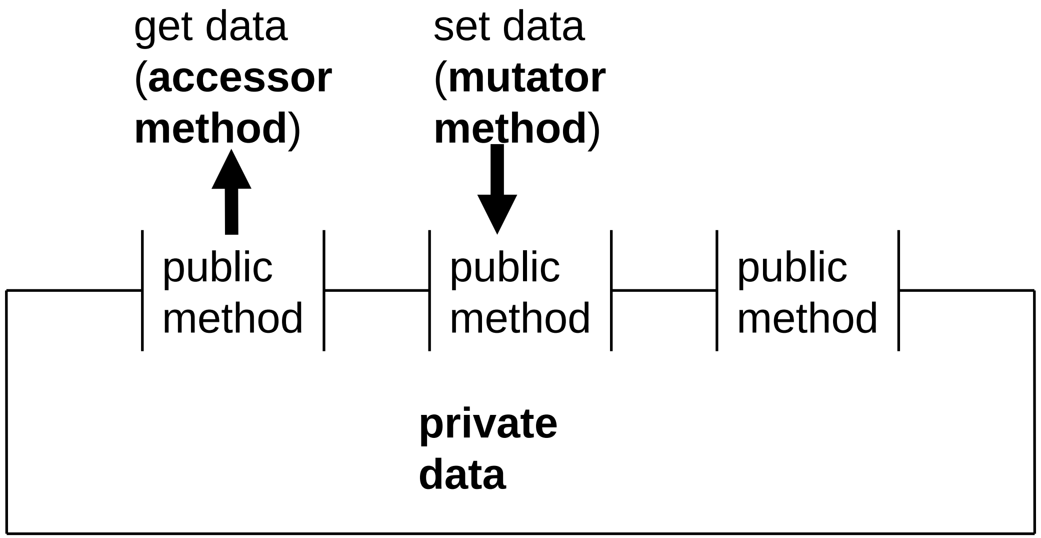
- It syntactically means this part of the class cannot be accessed outside of the class definition.
- You should always do this for variable attributes, very rarely do this for methods.

```
public class Person {
     private int age;
     public Person() {
       age = 12;
       //OK – access allowed here
    }
}
```

```
public class MainClass {
     public static void main(String [] args) {
          Person aPerson = new Person();
          aPerson.age = 12;
         // Syntax error: program won’t compile!
     }
}
```

Encapsulation/Information Hiding

- Protects the inner-workings (data) of a class.
- Only allow access to the core of an object in a controlled fashion (use the *public* parts to access the *private* sections).
  - Typically it means public methods accessing private attributes via accessor and mutator methods.
  - Controlled access to attributes:
    - Can prevent invalid states
    - Reduce runtime errors



Summary

- Structured programming
- Object-Oriented programming
- Class
- Object
- Class attributes
- Class methods
- Object state
- Instantiation
- Constructor
  - The Default constructor
- Encapsulation/information hiding

Assignment 1

I. Create a rectangle class with the following

- The member data should be the length and height of the rectangle.
- Methods to:
  - Allow the user to enter class attributes
  - A method to display the values of the rectangle attributes.
  - A method for calculating the perimeter.
  - A method for calculating the area.
  - A default constrictor to initialize the attributes to zero values.

Thank you