Object Oriented Programming (OOP)

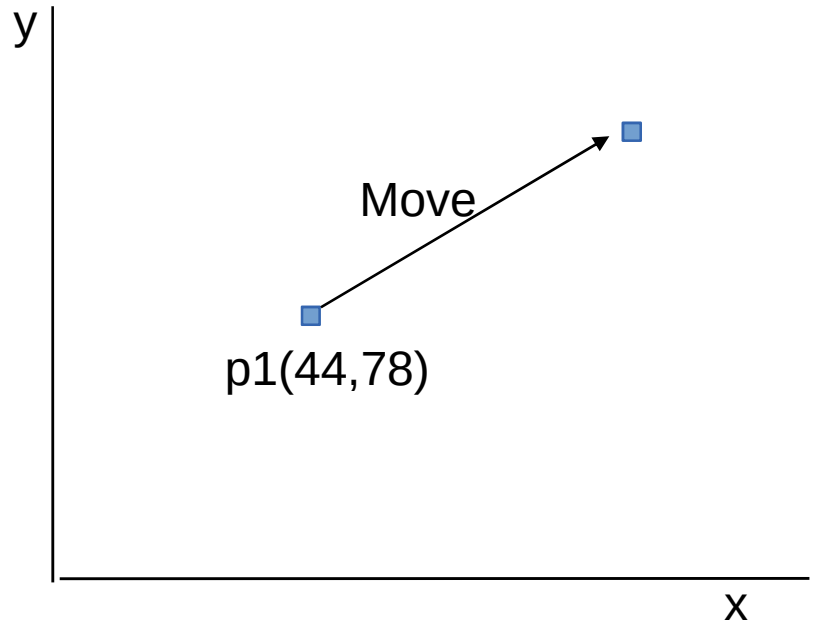
Mohamed Ezz

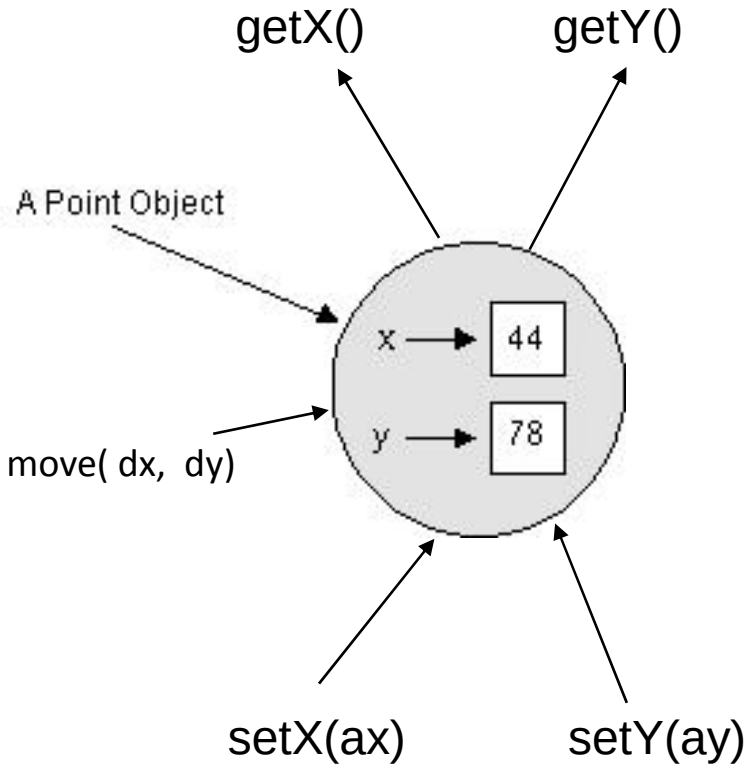
Lecture 3

Review

1. How to create Object?
2. What is the Object reference?
3. How to access member variables/methods from outside class?
4. How to access member variables/methods from within class?
5. Why using setter & getter?

Review: **Point Class Example**





Which concept we learned ?

Review: Test Point Class

TestPoint.java

```
public class TestPoint
{

        public static void main(String arg[]){
            System.out.println("Test Point Class");
            Point p1 =  new Point();
            Point p2 = new Point();
            p1.setX(1);
            p1.setY(3);
            p2.setX(4);
            p2.setY(5);
            System.out.println(p1);
            p1.move(2,2);
            System.out.println(p1);

       }// end of main
}// end of class
```

Lecture Objectives

•Understand how to represent Class, Object, method to communicate design with developer

•Understand how to constructor objects

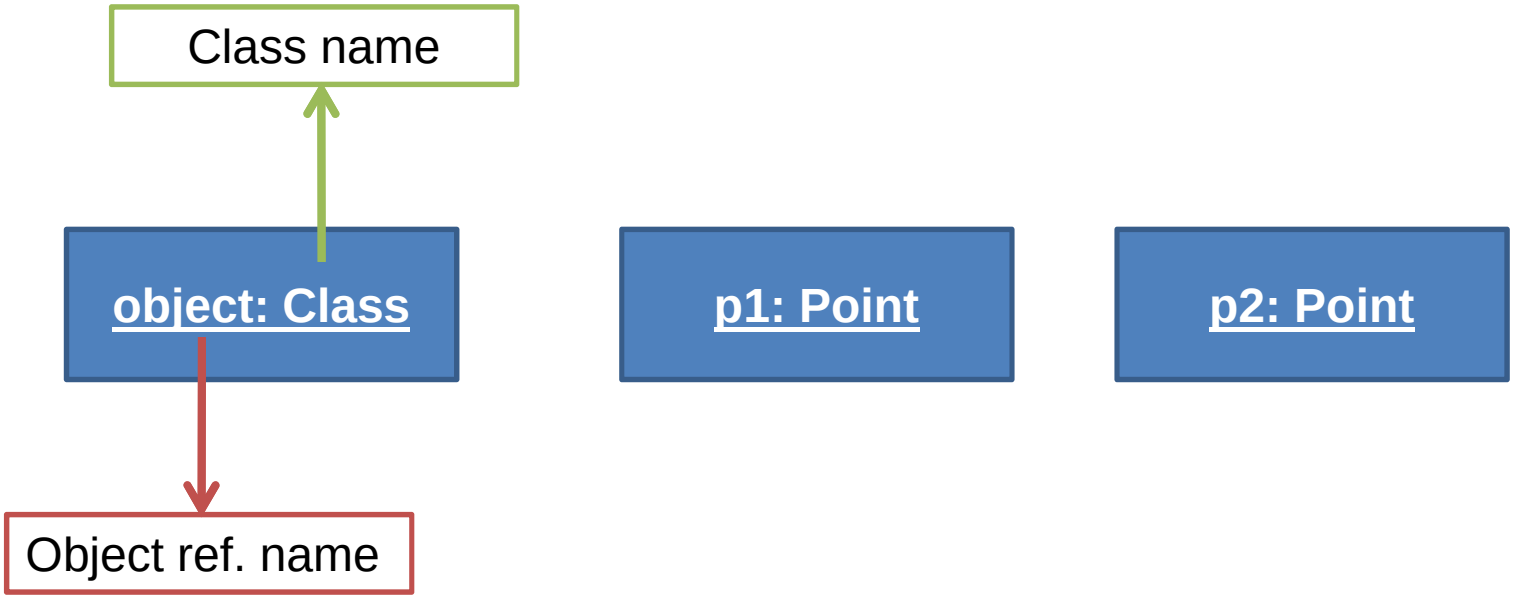
• Understand difference between Primitive and reference types

• Understand how to Packages and group classes.

• Practice UML design

How to Represent Object in the design

We will use **UML** notating as following:



Object in UML represented as

- box
- name underline
- qualified with class name

How to Represent Class in the design

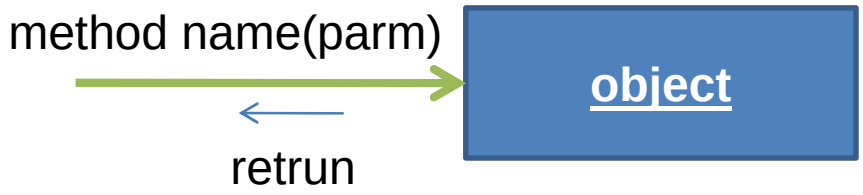
We will use **UML** notating as following:

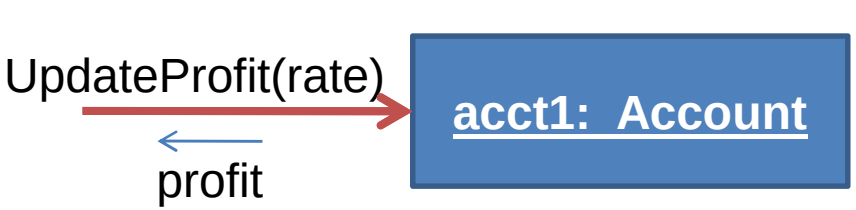
Class in UML represented as

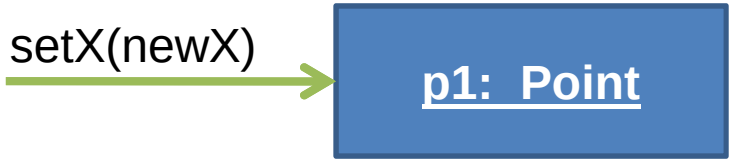
- box
- Class name in the top of the box without underline
- May has attributes & methods as will be explained later

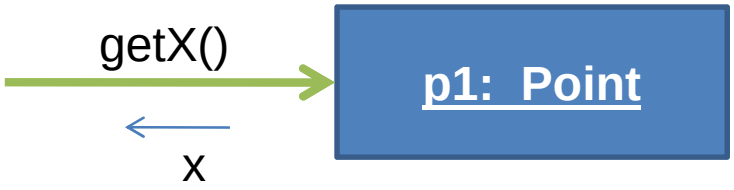
How to Represent message in the design

We will use **UML** notating as following:





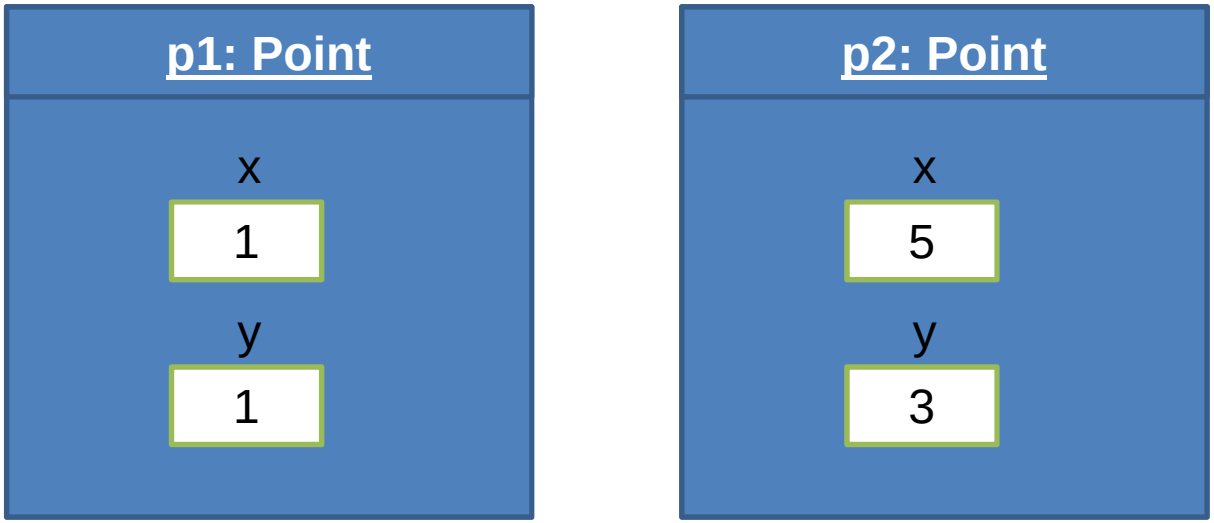




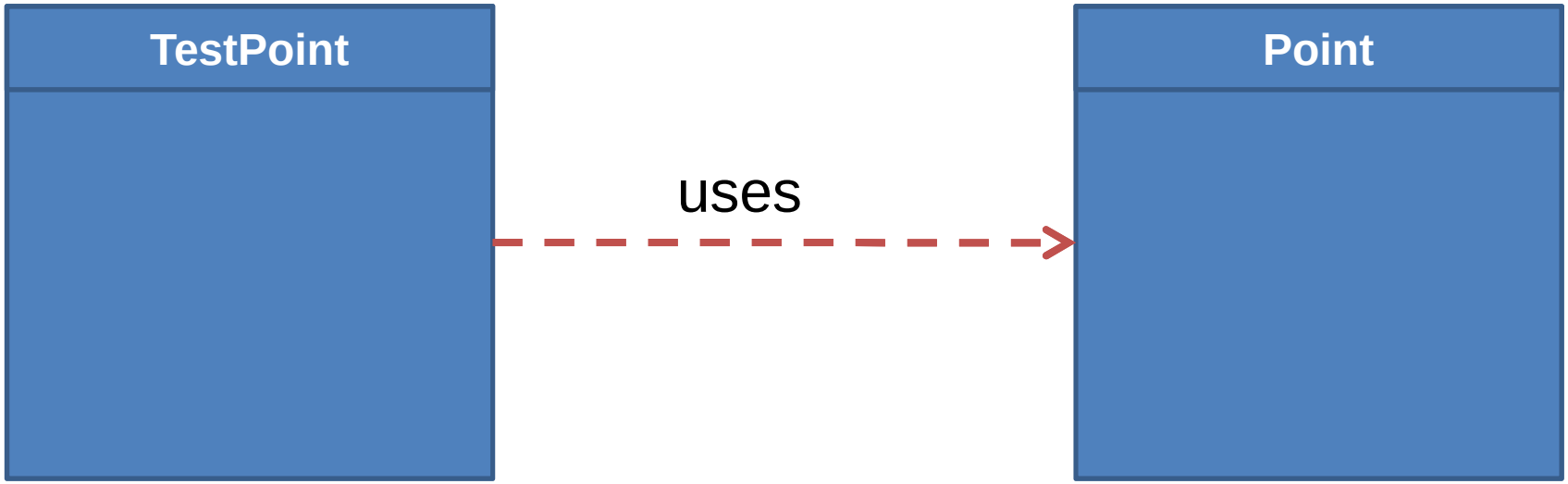
Synchronies method

Asynchronies method

How to Represent Object with data values



Classes Relationship (Dependency)



First relationship, next lectures we will explain more relationship between classes

Constructors

- Code that gets executed when "**new**" is called
- Constructor is a special method used for object creation to initialize object variables
-Allow enforce/ensure that all instances have certain properties (valid state)

Example

Point p1 = new ***Point()***;

-Default constructor (zero-argument constructor)

- A constructor without any parameter
- If a programmer doesn't define any constructor for a class, JRE will implicitly create a default constructor as in Point class

User Defined constructor

- Constructor explicitly defined in class
- "Method" that exactly named as the class name and has no return type (**not even void**).
- May have parameters (Zero argument or not)

```
public class MyClass {
        public MyClass(...) //no return :"void" or any other data type
        {
        …//init object
        …
        }
}
```

Default Constructors

Person.java

```
public class Person {
        public String firstName, lastName;
}
```

PersonTest.java

```
public class Person1Test {
        public static void main(String[] args) {
                Person p = new Person();
                p.firstName = “Hamza";
                p.lastName = “Ezz";

/*It took three lines of code to make a properly constructed person. It would be possible for a programmer to build a person and forget to assign a first or last name.*/
        }
}
```

User-Defined Constructor

Person.java

```
public class Person {
        public String firstName, lastName;
        public Person(String initialFirstName, String initialLastName) {
                firstName = initialFirstName;
                lastName = initialLastName;
        }
}
```

PersonTest.java

```
public class PersonTest {
        public static void main(String[] args) {
        Person p = new Person(“Hamza", “Ezz");
        /*It took one line of code to make a properly constructed person. It would not be possible for a programmer to build a person and forget to assign a first or last name*/
        }
}
// Are developer can use default constructor?
```

User-Defined Constructor Example

Date.java

```
public Class Date {
        private int day, month, year;
        public Date (int d, int m, int y){
                day = d;
                month= m;
                year= y;
        }
}
```

DateTest.java

```
public Class TestDate {
    public static void main (String arg[]){
        Date meeting= new Date(5,11,2015);
        Date today = new Date(20,10,2015);
  }
}
```

User-Defined Constructor (zero-Arg.)

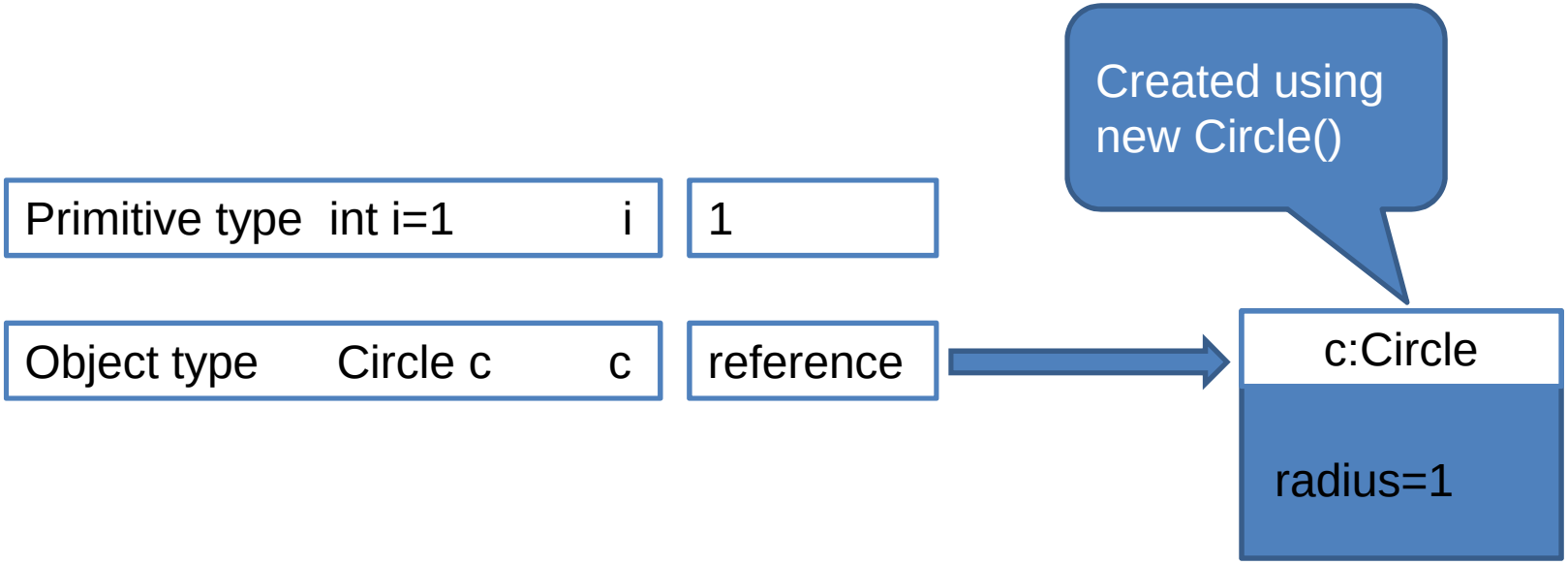
Date.java

```
public Class Date {
        public int day, month, year;
        public Date (){
                day = 1;
                month= 1;
                year= 2013;
        }
}
```

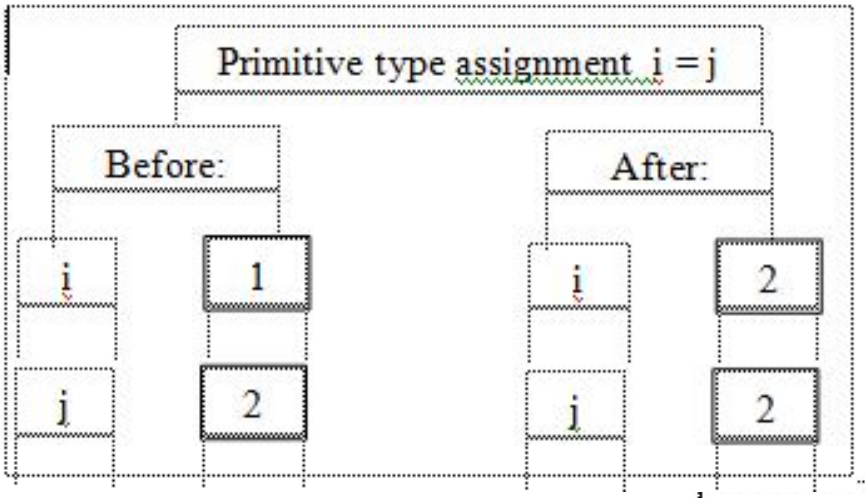
DateTest.java

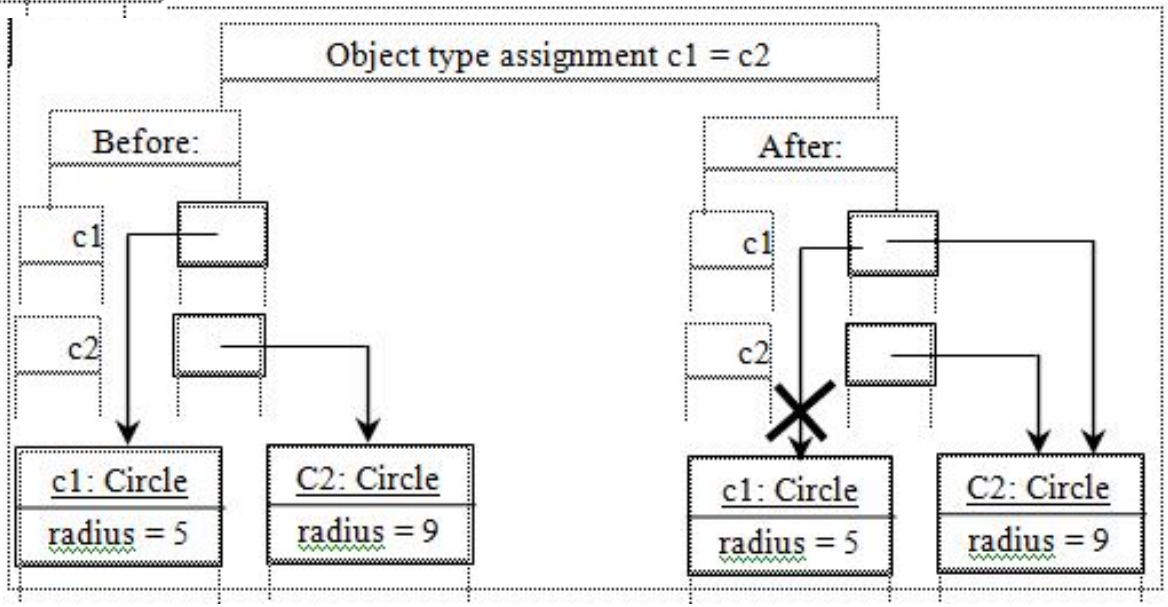
```
public Class TestDate {
  public static void main (String arg[]){
        Date meeting= new Date();
        Date today = new Date();
  }
}
```

Differences between Variables of Primitive Data Types and Object Types



Copying Variables of Primitive Data Types and Object Types





Garbage Collection

As shown in the previous figure, after the assignment statement c1 = c2,

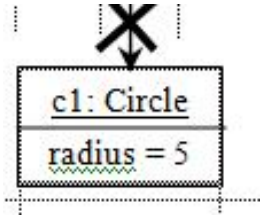
c1 points to the same object referenced by c2.

The object previously referenced by c1 is no longer referenced.

This object is known as garbage.

Garbage is automatically collected by JVM.

- The Garbage Collector
- Automatically frees an object, if not referenced.
  prevents memory leaks



Packages group related classes

Java hierarchically organizes classes into packages*

- java.lang
- java.text
- java.util
- …

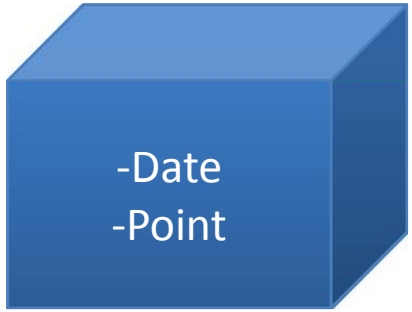
Classes need to be referred using its complete name (package + class name): for example, java.util.Calendar

Packages can be “imported” to avoid writing package names every time you use a class (except java.lang)

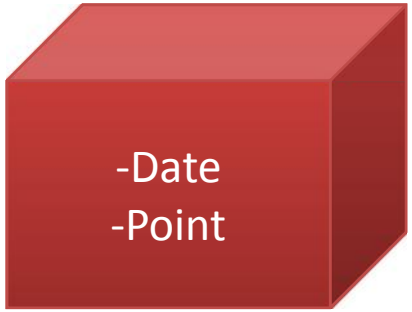
```
import java.util.*;
```

Creating Package

second.section1



second.section2



second.section3

```
package second.section1;
public Class Date {
   private int day, month, year;
   public Date (){
                day = 1;
                month= 1;
                year= 206;
   }
}
```

```
package second.section2;
public Class Date {
   private int day, month, year;
   public Date (){
                day = 1;
                month= 1;
                year= 206;
   }
}
```

```
package second.section3;
public Class Date {
   private int day, month, year;
   public Date (){
                day = 1;
                month= 1;
                year= 206;
   }
}
```

```
package second.section1;
public Class Point{
```

```
package second.section2;
public Class Point{
```

```
package second.section3;
public Class Point{
```

Using Packages

```
import second.section1.*; // first approach
import second.section1.Point; // second approach
import second.section2.Date; // second approach
Class TestPoint{
  //Third approach without any import in classes belong to same
   //package
            public static void main (String arg[]){
            Point p1= new Point(); // which class
            Date d1= new date(); //which class
            second.section1.Date d2 = new second.section1.Date();
            // Fourth in-line approach
            }
}
```

Using Package

- Using Package
  - Organizing your classes into packages
    - A class can only be in one package
    - No duplicate class definition in the same package
    - Put the package statement at the beginning
    - Packages correspond to directories in local file system

  - Examples:

    ```
    package section1;
    package section2.assignment;
    package section3.lecture.web;
    ```

- Default Package
  - A class without any package defined is in a “default package”
  - The default package is NOT the root package!
    - Classes in the default package cannot be referenced outside the default package

Object-Oriented Design

- Step one: Create a UML class diagram of your classes & objects
- Step two: Create a detailed description of the services to be performed

UML: Unified Modeling Language

http://www.UML.org

Structure of a Class Definition

```
class name {
        declarations

        constructor definition(s)

        method definitions
}
```

attributes and symbolic constants

how to create and initialize objects

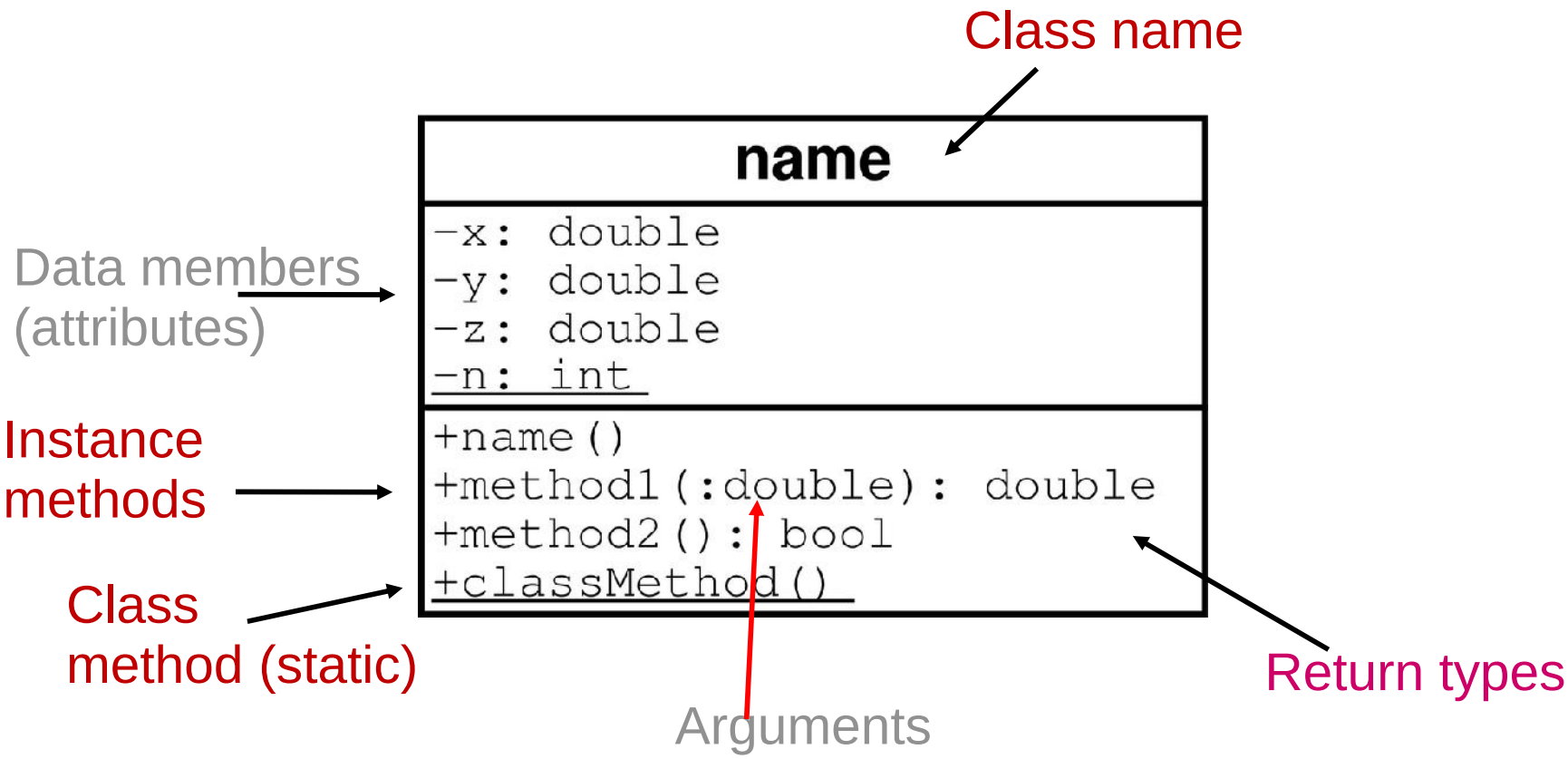
how to manipulate the state of objects

These parts of a class can actually be in any order

UML Class

Visibility shown as

+	public
-	private
#	protected



Data members, arguments and methods are specified by visibility name : type

Class Attributes

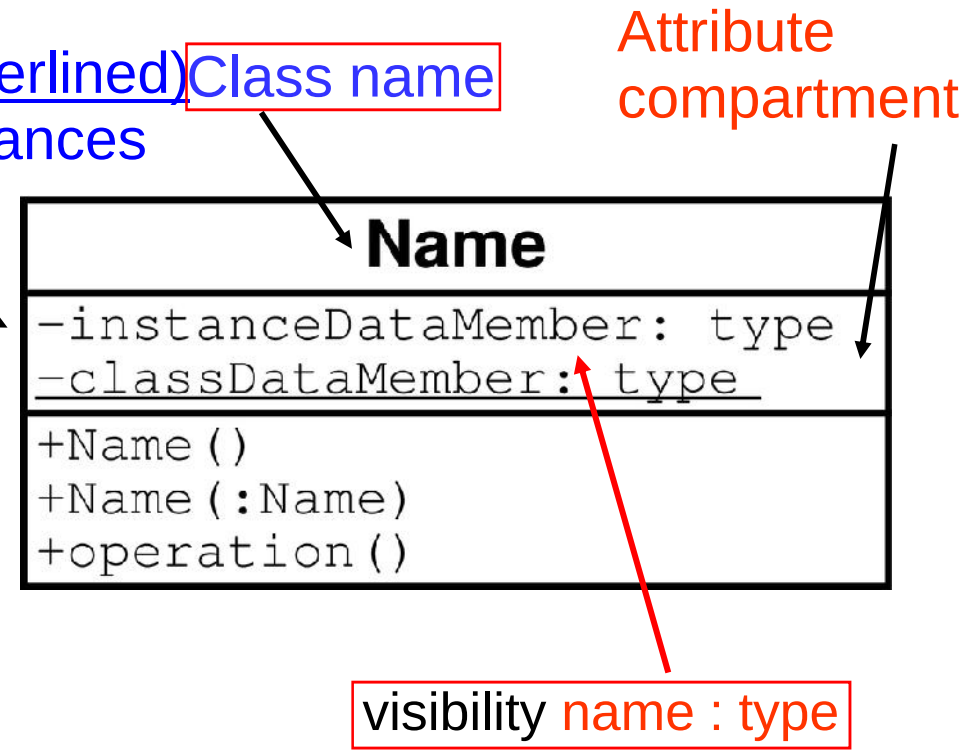
Attributes are the instance data members and class data members

Class data members (underlined) are shared between all instances (objects) of a given class

Data types shown after ":"

Visibility shown as

+	public
-	private
#	protected

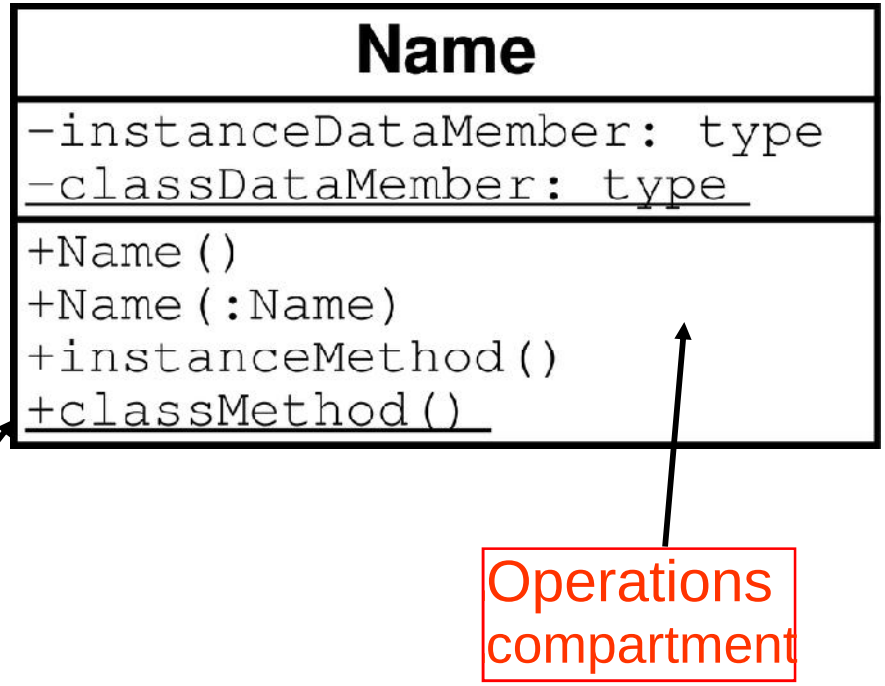


Class Operations

Operations are the class methods with their argument and return types

Public (+) operations define the class interface

Class methods (underlined) can only access to class data members, no need for a class instance (object)



Home Work

- Create Five classes: A, B, C, D, E each will contains a string member variable **s**
- Make A, B under package x
- Make C, D under package y
- Make E under package x.z
- Compile the classes and check the generated classes structure (folders).
- Create main method inside E class, and try to create objects from classes A,B,C,D using approaches 1,2,3 to access these classes