Session 03
Classes and Objects

(http://docs.oracle.com/javase/tutorial/java/javaOO/index.html)

Objectives

1-Programming Paradigms

2-OOP basic concepts

3-How to identify classes

4-Hints for class design

5-How to declare/use a class

6-Common modifiers (a way to hide some members in a class)

7-Memory Management in Java

8-Garbage Collection

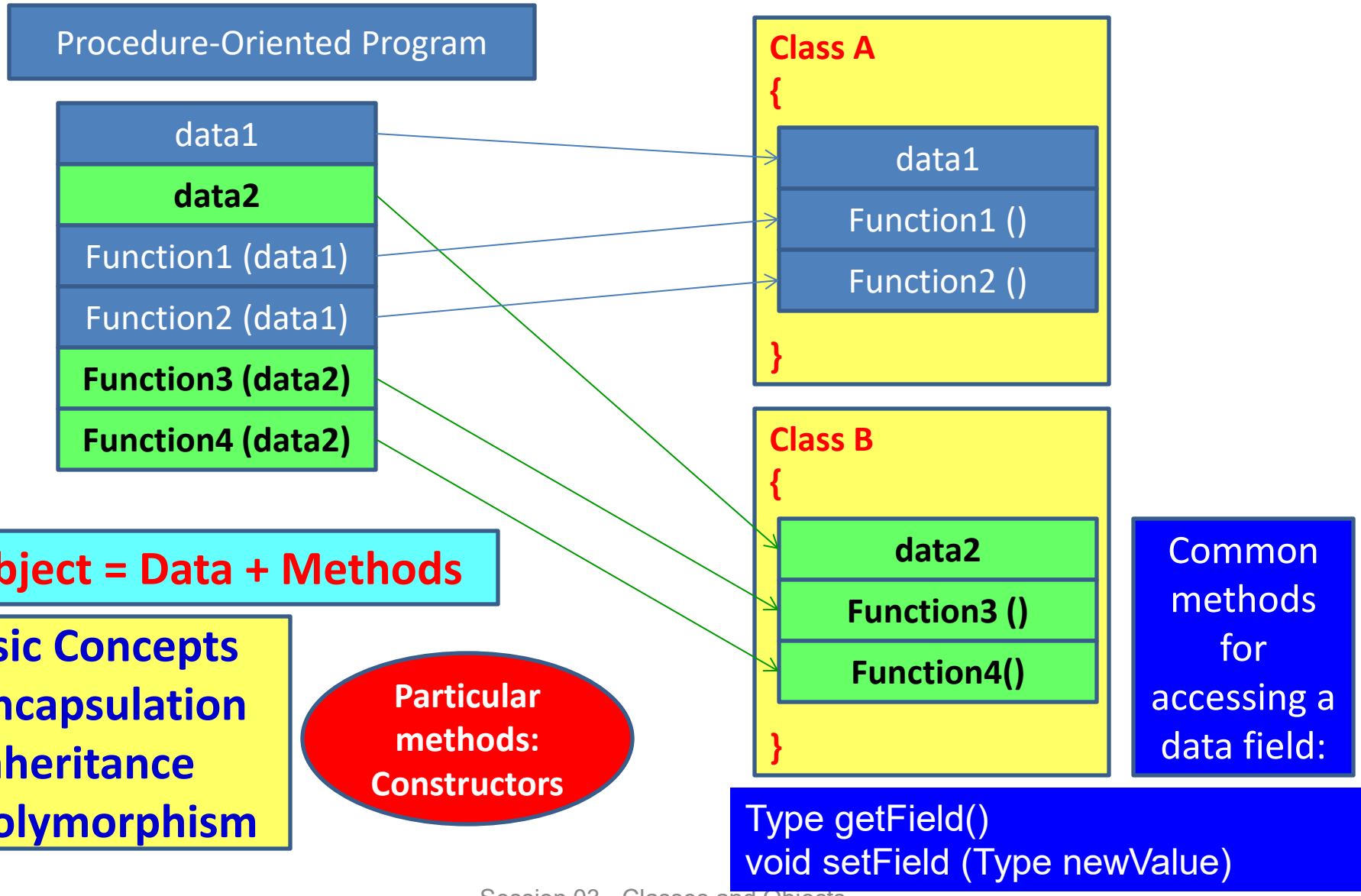
9-Case study: Java program for managing a list of persons

1- Programming Paradigms

- High-level programming languages (from 3rd generation languages) are divided into (Wikipedia):

Paradigm	Description
Procedural-oriented (imperative) paradigm-POP (3rd generation language)	Program= data + algorithms. Each algorithm is implemented as a function (group of statements) and data are it's parameters (C-language)
Object-oriented paradigm (OOP) (3rd generation language)	Programs = actions of some objects. Object = data + behaviors. Each behavior is implemented as a method (C++, Java, C#,…)
Functional paradigm (4th generation language)	Domain-specific languages. Basic functions were implemented. Programs = a set of functions (SQL)
Declarative/Logic paradigm (5th generation language)	Program = declarations + inference rules (Prolog, CLISP, …)

Programming Paradigms: POP vs. OOP



2-OOP Concepts

- Encapsulation
- Inheritance
- Polymorphism

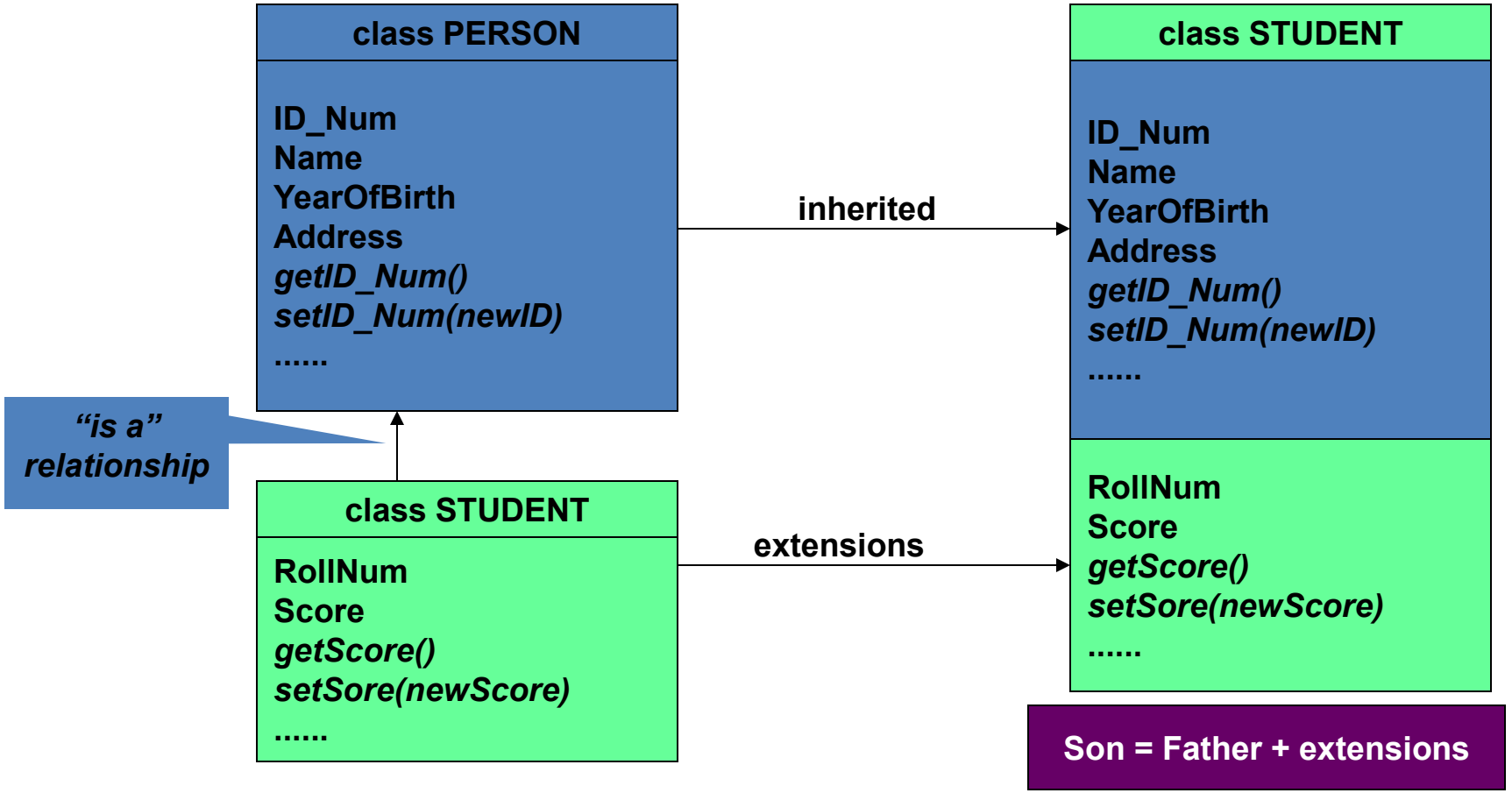
OOP Concepts: Encapsulation

Aggregation of data and behavior.

- Class = Data (fields/properties) + Methods
- Data of a class should be hidden from the outside.
- All behaviors should be accessed only via methods.
- A method should have a *boundary condition:* Parameters must be checked (use if statement) in order to assure that data of an object are always valid.
- **Constructor**: A special method it's code will execute when an object of this class is initialized.

OOP Concepts: Inheritance

Ability allows a class having members of an existed class ➔ Re-used code, save time

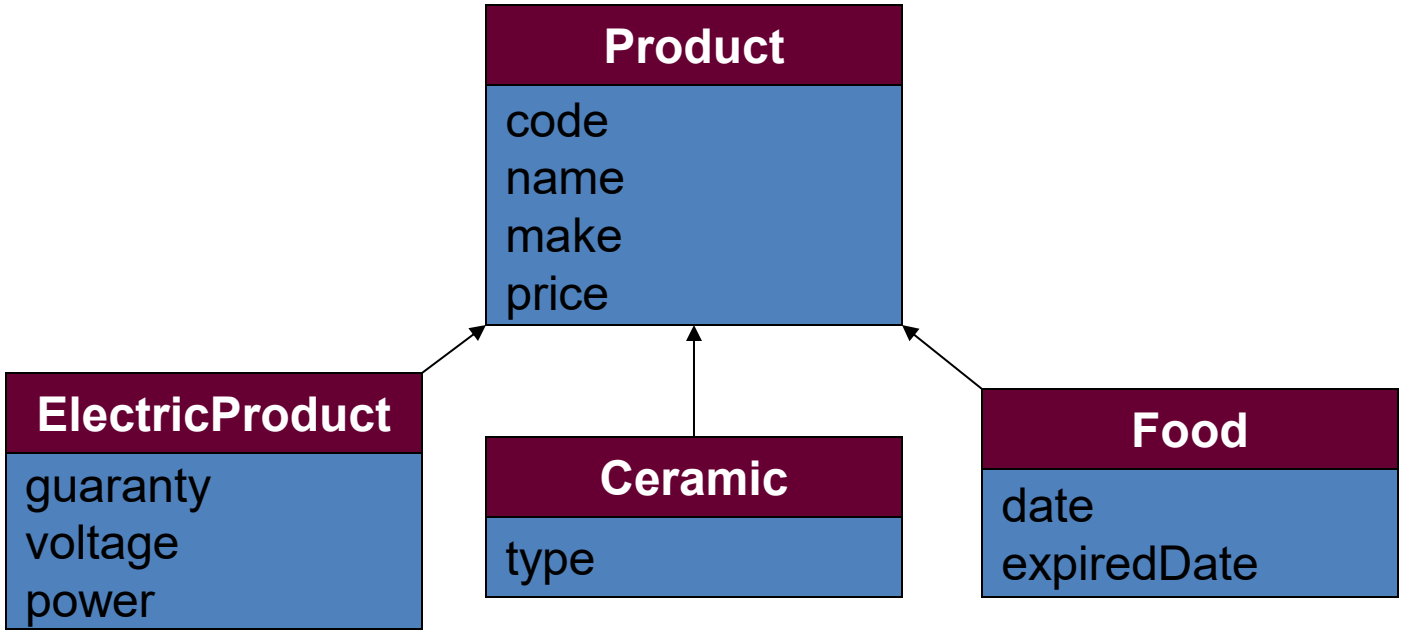


OOP Concepts: Inheritance

How to detect father class?

Finding the intersection of concerned classes.

- Electric Products< code, name, make, price, guaranty, voltage, power>
- Ceramic Products < code, name, make, price, type >
- Food Products < code, name, make, price , date, expiredDate >



OOP Concepts: Polymorphism

Ability allows many versions of a method based on overloading and overriding methods techniques.

Overloading: A class can have some methods which have the same name but their parameter types are different.

Overriding: A method in father class can be overridden in it's derived classes (body of a method can be replaced in derived classes).

3- How to Identity a Class

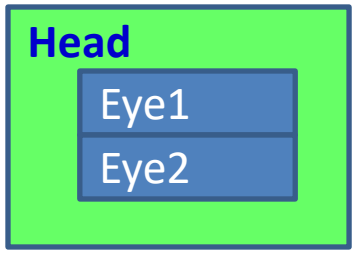
- Main noun: Class
- Nouns as modifiers of main noun: Fields
- Verbs related to main noun: Methods

Employee details of a **student** include **code, name, year of birth, address.**
Write a Java program that will allow **input** a student, **output** his/her.

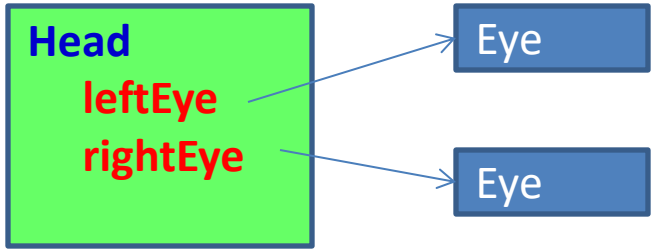
```
class Student {
    String code;
    String name;
    int bYear;
    String address;
    void input() {
        <code>
    }
    void output() {
        <code>
    }
}
```

4-Hints for class design

- ***Identifying classes***: Coupling
  - Is an object's reliance on knowledge of the internals of another entity's implementation.
  - When object A is tightly coupled to object B, a programmer who wants to use or modify A is required to have an inappropriately extensive expertise in how to use B.



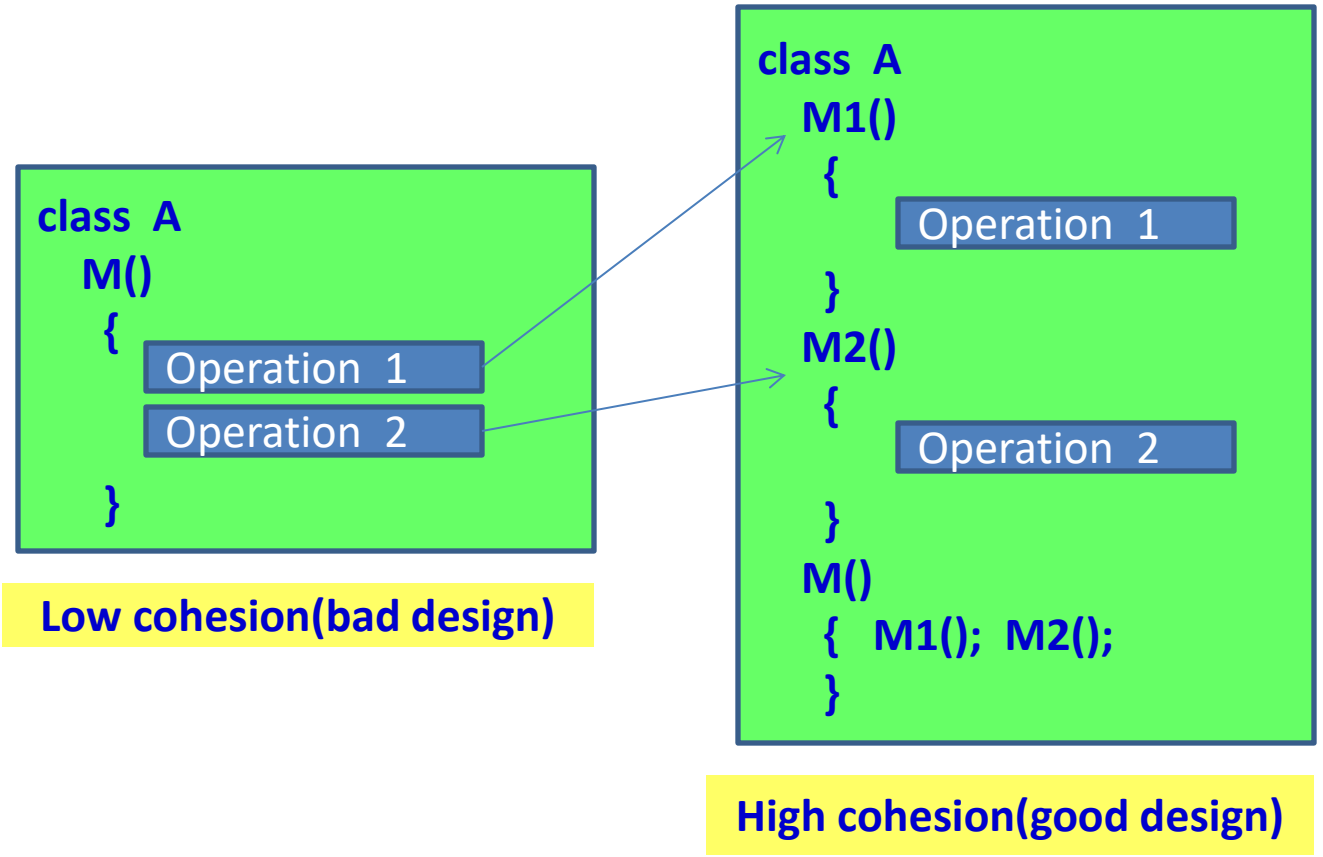
High coupling
(Bad design)



Low coupling
(Good design)

Hints for class design

- ***Implementing methods***: Cohesion is the degree to which a class or method resists being broken down into smaller pieces.



5- Declaring/Using a Java Class

```
[public] class ClassName [extends FatherClass] {
    [modifier] Type field1 [= value];
    [modifier] Type field2 [= value];
    // constructor
    [modifier] ClassName (Type var1,…) {
        <code>
    }
    [modifier] methodName (Type var1,…) {
        <code>
    }
    …….
}
```

Modifiers will be introduced later.

How many constructors should be implemented? ➔ Number of needed ways to initialize an object.

What should we will write in constructor's body? ➔ They usually are codes for initializing values to descriptive variables

Defining Constructors

- Constructors that are invoked to create objects from the class blueprint.
- Constructor declarations look like method declarations—except that they use the name of the class and have no return type.
- The compiler automatically provides a no-argument, default constructor for any class without constructors.

Defining Methods

- Typical method declaration:

```
[modifier] ReturnType methodName (params) {
  <code>
}
```

- Signature: data help identifying something
- Method Signature:

  name + order of parameter types

Passing Arguments a Constructor/Method

- Java uses the mechanism passing by value. Arguments can be:
  - Primitive Data Type Arguments
  - Reference Data Type Arguments (objects)

Creating Objects

- Class provides the blueprint for objects; you create an object from a class.
  - `Point p = new Point(23, 94);`
- Statement has three parts:
  - **Declaration**: are all variable declarations that associate a variable name with an object type.
  - **Instantiation**: The new keyword is a Java operator that creates the object (memory is allocated).
  - **Initialization**: The new operator is followed by a call to a constructor, which initializes the new object (values are assigned to fields).

Type of Constructors
Create/Use an object of a class

- ***Default constructor***: Constructor with no parameter.
- ***Parametric constructor***: Constructor with at least one parameter.

- **Create an object**

  ClassName obj1=new ClassName();

  ClassName obj2=new ClassName(params);
- **Accessing a field of the object**

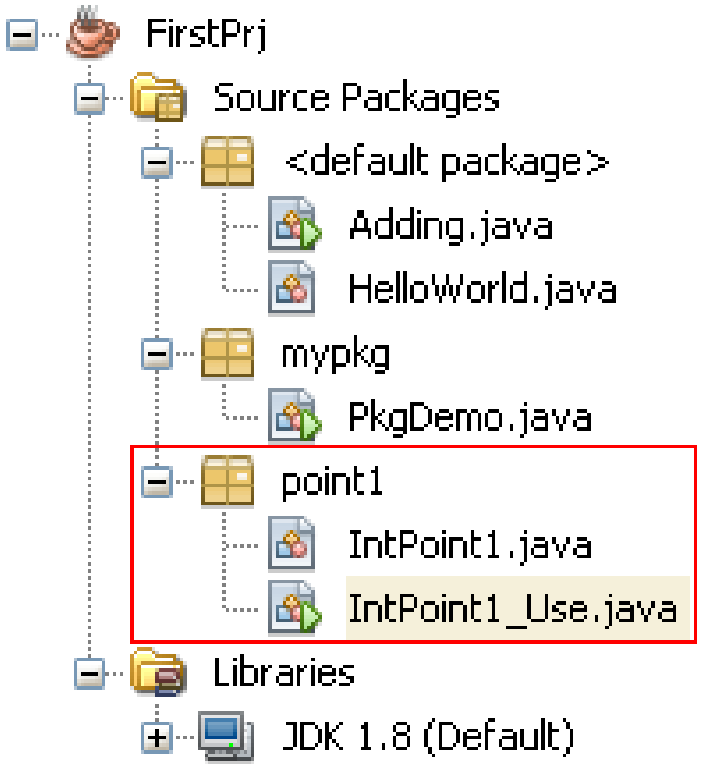
  object.field
- **Calling a method of an object**

  object.method(params)

Demo: If we do not implement any constructor, compiler will insert to the class a system default constructor

In this demonstration (package **point1**):

- The class **IntPoint1** represents a point in an integral two dimensional coordinate.
- The class **IntPoint1_Use** having the main method in which the class IntPoint1 is used.



Demo: If we do not implement any constructor, compiler will insert to the class a default constructor

```
package point1;
public class IntPoint1 {
    int x;
    int y;
    // If no constructor is implemented, the compier will insert
    // automatically a default constructor to the class
    public void output(){
        System.out.println ("[" + x + "," + y + "]");
    }
}
```

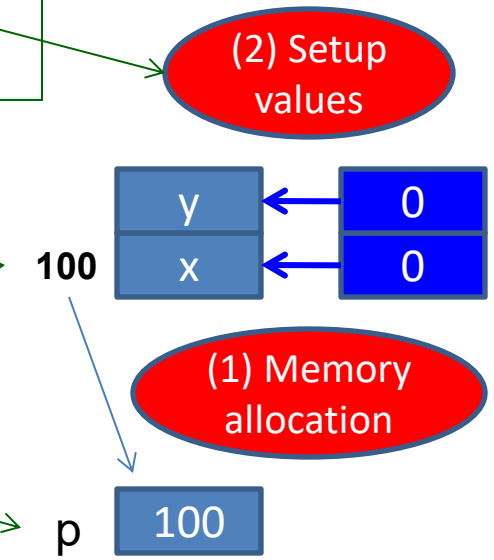
System constructor will clear all bits in allocated memory

Order for initializing an object

```
1  package point1;
2  public class IntPoint1_Use {
3      public static void main (String[] args){
4          // Create a point using default constructor
5          IntPoint1 p = new IntPoint1();
6          p.output();
7      }
8  }
```

```
Output - FirstPrj (run)
run:
[0,0]
BUILD SUCCESSFUL (total time: 0 seconds)
```

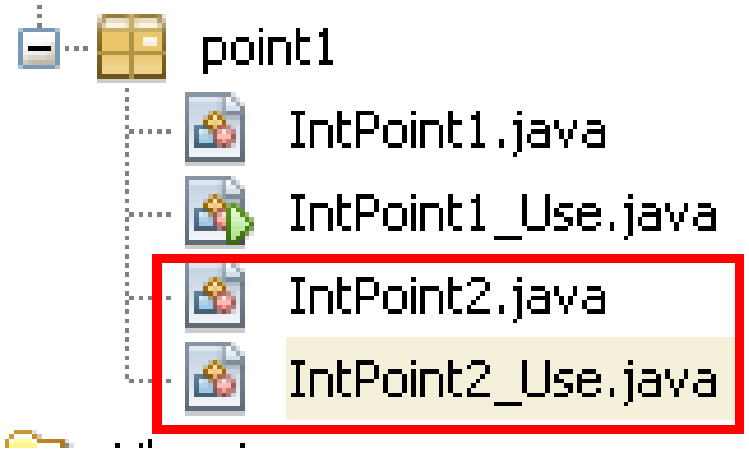
An object variable is a reference



Demo: If we implement a constructor, compiler does not insert default constructor

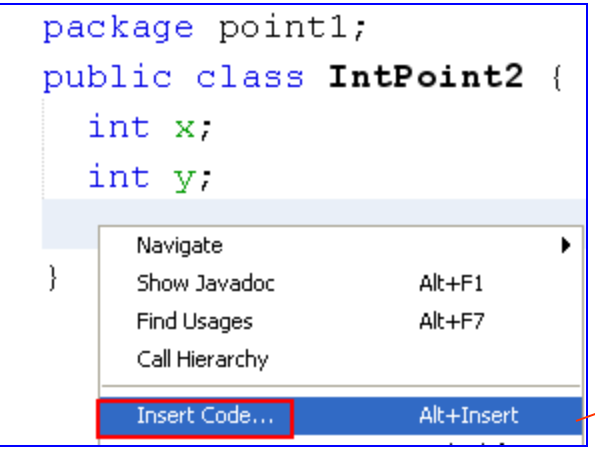
This demonstration will depict:

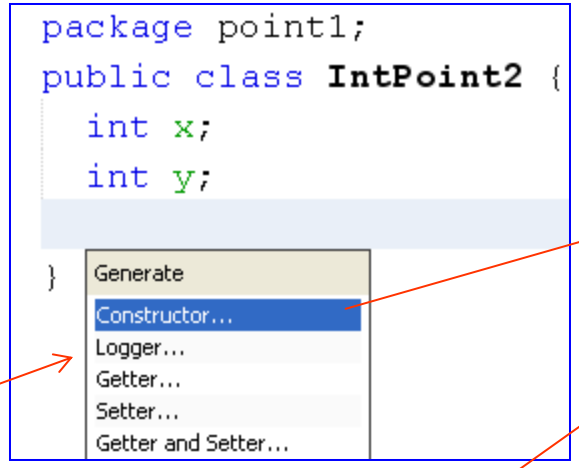
- The way to insert some methods automatically in NetBeans
- If user-defined constructors are implemented, compiler does not insert the system default constructor

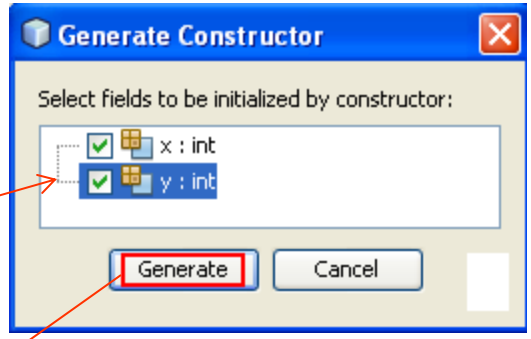


Demo: If we implement a constructor, compiler does not insert default constructor

Insert constructor








```
package point1;
public class IntPoint2 {
    int x;
    int y;

    public IntPoint2(int x, int y) {
        this.x = x;
        this.y = y;
    }

}
```

Parameter names are the same as those in declared data filed. So, the keyword **this** will help distinguish field name and parameter name.

this.x means that x of this object

Demo: If we implement a constructor, compiler does not insert default constructor

Accessing each data field is usually supported by :
A getter for reading value of this field
A setter for modifying this field

Insert getter/setter

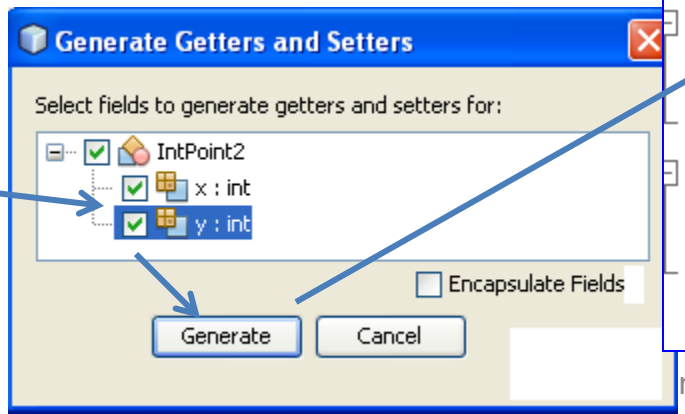
```
package point1;
public class IntPoint2 {
  int x;
  int y;

    public IntPoint2(int x, int y) {...4 lin

}
```

Navigate
Show Javadoc Alt+F1
Find Usages Alt+F7
Call Hierarchy
Insert Code... Alt+Insert

Generate
Constructor...
Logger...
Getter...
Setter...
Getter and Setter...
equals() and hashCode()...
toString()...
Override Method...
Add Property...




```
package point1;
public class IntPoint2 {
  int x;
  int y;

    public IntPoint2(int x, int y) {
    public int getX() {
        return x;
    }
    public void setX(int x) {
        this.x = x;
    }
    public int getY() {
        return y;
    }
    public void setY(int y) {
        this.y = y;
    }
}
```

Demo: If we implement a constructor, compiler does not insert system constructor

```
package point1;
public class IntPoint2 {
  int x;
  int y;

    public IntPoint2(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public int getX() {...3 lines }
    public void setX(int x) {...3 li
    public int getY() {...3 lines }
    public void setY(int y) {...3 li
}
```

```
1   package point1;
2   public class IntPoint2_Use {
3   public static void main (String[] args){
4         // Create a point using default constructor
5         // Error:Constructor InPoint2 in class IntPoint2 can
6         // not be appied to given type;required: int, int
          IntPoint2 p = new IntPoint2();
8     }
9   }
```

Explain the result of the following program

```
package point1;
public class IntPoint2 {
    int x=7;
    int y=3;
    public IntPoint2(){
        output();
        x=100;
        y=1000;
        output();
    }
    public IntPoint2(int x, int y) {
        output();
        this.x = x;
        this.y = y;
        output();
    }
    public void output(){
        String S= "[" + x + "," + y + "]";
        System.out.println(S);
    }
}
```

```
package point1;
public class IntPoint2_Use {
  public static void main (String[] args){
        System.out.println("Use default constructor:");
        IntPoint2 p1= new IntPoint2();
        System.out.println("Use parametric constructor:");
        IntPoint2 p2 = new IntPoint2(-7,90);
    }
}
```

```
Output - FirstPrj (run)
run:
Use default constructor:
[7,3]
[100,1000]
Use parametric constructor:
[7,3]
[-7,90]
BUILD SUCCESSFUL (total time: 0 seconds)
```

6- Common Modifiers

- Modifier (linguistics)is a word which can bring out the meaning of other word (adjective →noun, adverb → verb)
- Modifiers (OOP) are keywords that give the compiler information about the nature of code (methods), data, classes.
- Java supports some modifiers in which some of them are common and they are called as **access modifiers** (public, protected, default, private).
- Common modifiers will impose level of accessing on
  - class (where it can be used?)
  - methods (whether they can be called or not)
  - fields (whether they may be read/written or not)

Outside of a Class

```
package point1;
public class IntPoint2 {
    int x=7;
    int y=3;
    public IntPoint2(){
        output();
        x=100;
        y=1000;
        output();
    }
    public IntPoint2(int x, int y) {
        output();
        this.x = x;
        this.y = y;
        output();
    }
    public void output(){
        String S= "[" + x + "," + y + "]";
        System.out.println(S);
    }
}
```

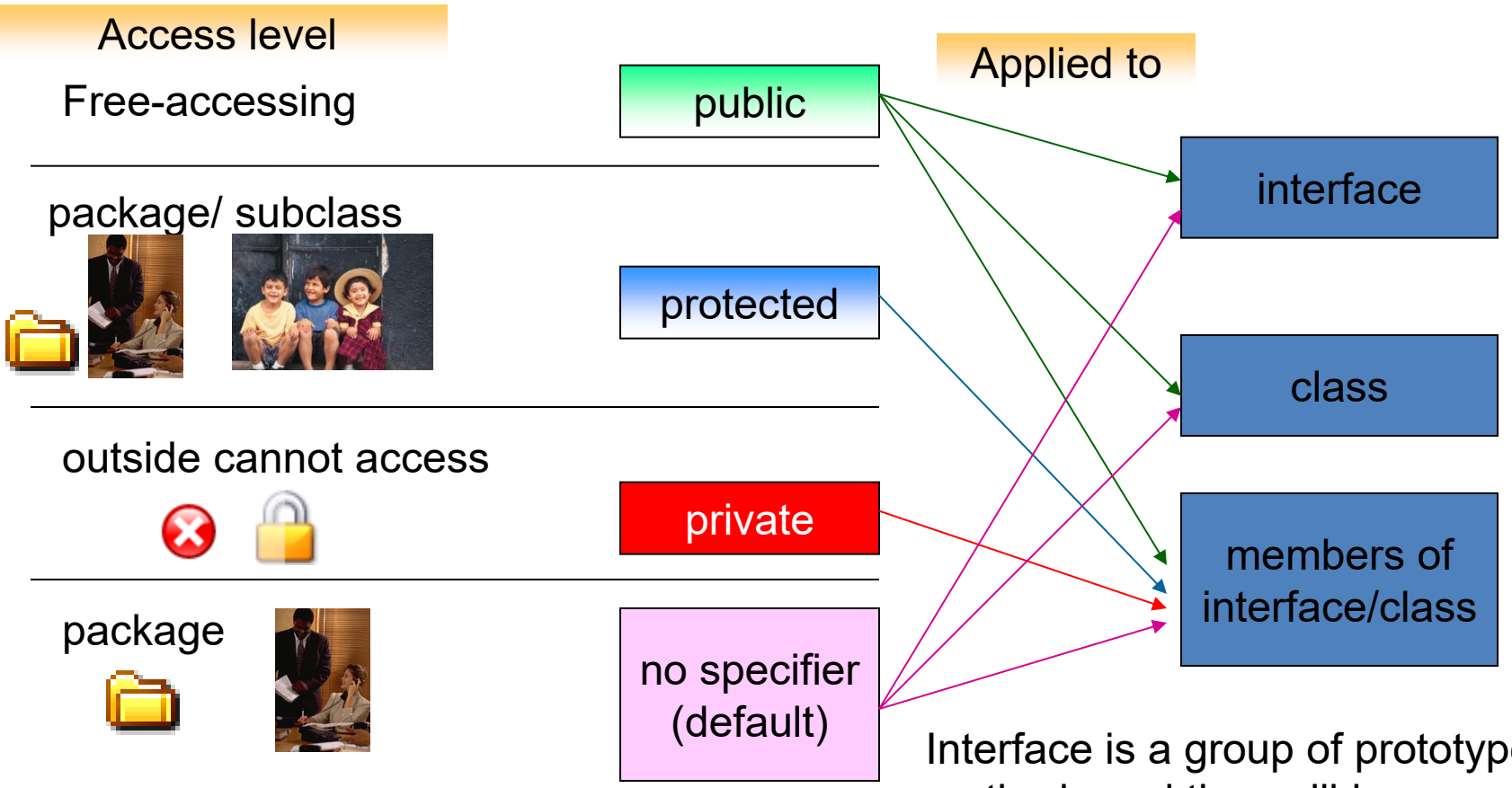
Inside of the class InPoint2

```
package point1;
public class IntPoint2_Use {
    public static void main (String[] args){
        System.out.println("Use default constructor:");
        IntPoint2 p1= new IntPoint2();
        System.out.println("Use parametric constructor:");
        IntPoint2 p2 = new IntPoint2(-7,90);
    }
}
```

Inside of the class InPoint2_Use and it is outside of the class IntPoint2

Outside of the class A is another class where the class A is accessed (used)

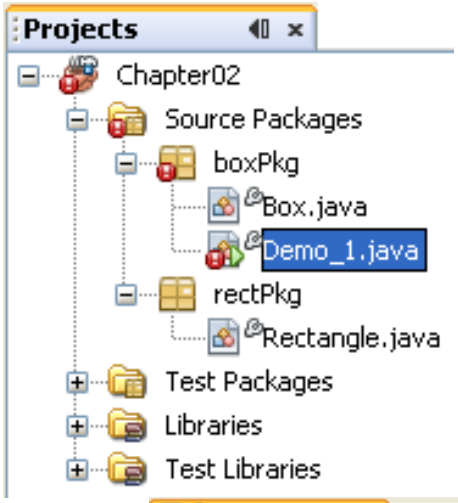
Common Modifiers



Order:
public > protected > default > private

Interface is a group of prototyped methods and they will be implemented in a class afterward. It will be introduced later.

Common Modifiers



Rectangle.java

```
1  package rectPkg;
2  public class Rectangle {
3    protected int length;
4    public int width;
5    public void setSize (int l, int w)
6    {  length = l>0? l: 0;
7       width = w>0? w: 0;
8    }
9  }
```

Box.java

```
1  package boxPkg;
2  import rectPkg.Rectangle;
3  public class Box extends Rectangle {
4     int height;
5     protected int price;
6     private int weight;
7     void setSize(int l, int w, int h)
8     {  super.setSize(l,w);
9        height = h>0? h : 0;
10    }
11    int volume ()
12    {  return length*width*height;
13    }
14 }
```

Demo_1.java

```
1  package boxPkg;
2  import rectPkg.Rectangle;
3  public class Demo_1 {
4    public static void main (String[] args)
5    { Box b = new Box();
6      b.setSize(1,2,3);
7      b.height=10;
8      b.price= 7;
       b.weight=9;
10     System.out.println("Volumn of the box:" + b.volume());
11     Rectangle r= new Rectangle();
12     r.setSize(3,5);
13     r.width=3;
       r.length=6;
15   }
16 }
```

Demo: Overloading Method

```
/* Overloading methods Demo. */
public class Box {
   int length=0;
   int width=0;
   int depth=0;
   // Overloading constructors
   public Box(){
   }
   public Box(int l){
       length = l>0? l: 0; // safe state
   }
   public Box(int l, int w){
       length = l>0? l: 0; // safe state
       width = w>0? w: 0;
   }
   public Box(int l, int w, int d){
       length = l>0? l: 0; // safe state
       width = w>0? w: 0;
       depth = d>0? d: 0;
   }
   // Overloading methods
   public void setEdge (int l,int w){
       length = l>0? l: 0; // safe state
       width = w>0? w: 0;
   }
   public void setEdge (int l,int w,int d){
       length = l>0? l: 0; // safe state
       width = w>0? w: 0;
       depth = d>0? d: 0;
   }
   public void output(){
      String S= "[" + length + "," + width
                + "," + depth + "]";
      System.out.println(S);
   }
}
```

```
/* Use the class Box */
public class BoxUse {
    public static void main(String[] args){
        Box b= new Box();
        b.output();
        b.setEdge(7,3);
        b.output();
        b.setEdge(90,100,75);
        b.output();
    }
}
```

Output - FirstPrj (run)

```
run:
[0,0,0]
[7,3,0]
[90,100,75]
```

Demo: Methods with Arbitrary Number of Arguments

A group is treated as an array
group.length →number of elements
group[i]: The element at the position i

```
1   public class ArbitraryDemo {
2       public double sum(double... group){
3           double S=0;
4           for (double x: group) S+=x;
5           return S;
6       }
7       public String concate(String... group){
8           String S="";
9           for (String x: group) S+=x + " ";
10          return S;
11      }
12      public static void main(String[] args){
13          ArbitraryDemo obj= new ArbitraryDemo();
14          double total= obj.sum(5.4, 3.2, 9.08, 4);
15          System.out.println(total);
16          String line = obj.concate("I", "love", "you", "!");
17          System.out.println(line);
18      }
19  }
```

```
Output - FirstPrj (run)
run:
21.68
I love you !
```

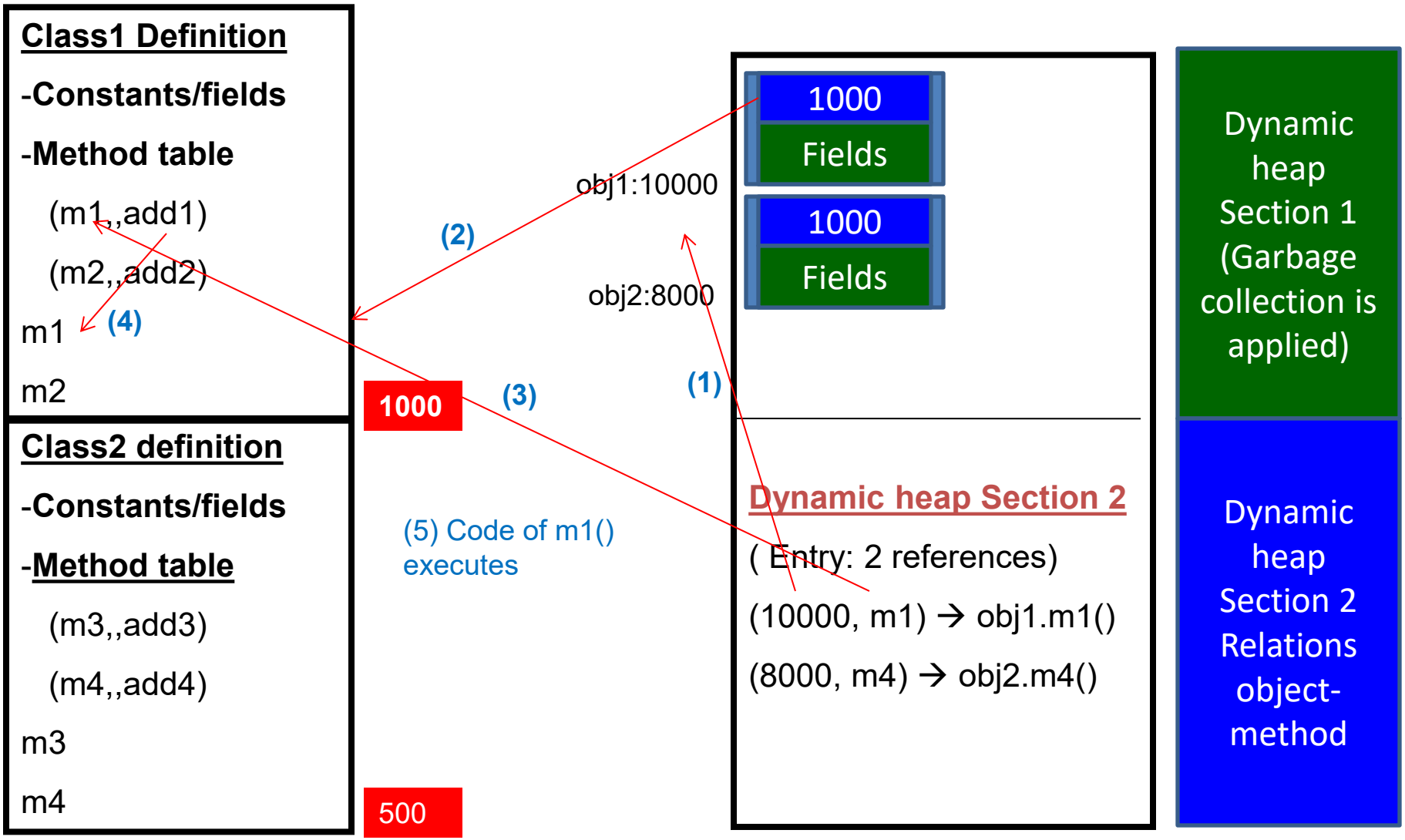
7- Memory Management in Java

- **Review:** In C, 4 basic regions: Data segment (for global data), code segment (for statements), stack (for local data of functions when they are called), heap (for dynamic data). C/C++ programmers must explicitly manage the heap of a program.
- **How Java heap is managed? (Refer to: http://docs.oracle.com/javase/specs/)**
  - JVM support the **garbage collector** in order to free Java programmers from explicitly managing heap
  - Java heap is managed by 2 lists: Free block list, Allocated block list
  - Initial, free block list is all the heap
  - After very much times for allocating and de-allocating memory, fragmented and free blocks are not contiguous

Memory Management in Java

- How are data allocated in heap?
  - Way: First fit
  - If there is no blank block is fit, Java memory manager must compact memory in order to create more larger free block
- Heap structure in Java
- Static heap contains class declarations → Invariable, garbage collection is not needed
- Dynamic heap is divided into two sections: The first contains objects and the second contains relations between object and appropriate method in static heap. When an object is not used (garbage), it's memory can be de-allocated.
- When an object is created, a field for reference to the class declaration is automatically added
- The next slide will depict it..

Memory Management in Java



8- Garbage Collection

- Most modern languages permit you to allocate data storage during a program run. In Java, this is done directly when you create an object with the new operation and indirectly when you call a method that has local variables or arguments.
- Local data of a method include: return data, parameters, variables are declared in the body of the method.
- Method locals are allocated space on the stack and are discarded when the method exits, but objects are allocated space on the heap and have a longer lifetime.

Garbage Collection…

- In Java, you never explicitly free memory that you have allocated; instead, Java provides automatic garbage collection.
- The runtime system keeps track of the memory that is allocated and is able to determine whether that memory is still useable.
- Garbage collector has the lowest priority. It runs only when the system heap becomes exhausted.
- A data is treated as garbage when it is out of it's scope or an object is assigned to null.

Garbage Collection …

```
Object obj1 = new Object();
int x= 5;
if (x<10) {
    Object obj2= new Object();
    int y=3;

    ……
}
int t=7;
obj1 = null;
t*=8;
……
```

Scope of a variable begins at the line where it is declared and ends at the closing bracket of the block containing it

obj2, y are out of scope (they are no longer used)

obj1= null → Memory allocated to obj1 is no longer used

Garbage Collection…

When does garbage collector execute?

- Garbage collector has the lowest priority. So, it runs only when program's memory is exhausted.
- It is called by JVM only. We can not activate it.

9- Case study and Sample Report

- Reports must be written in your notbook
- A report includes 5 parts:
  1- Problem Description
  2- Analysis
  3- Design
  4- Implementation
  5- Testing
- Hereafter, a sample report is introduced.

Case Study 1 Report

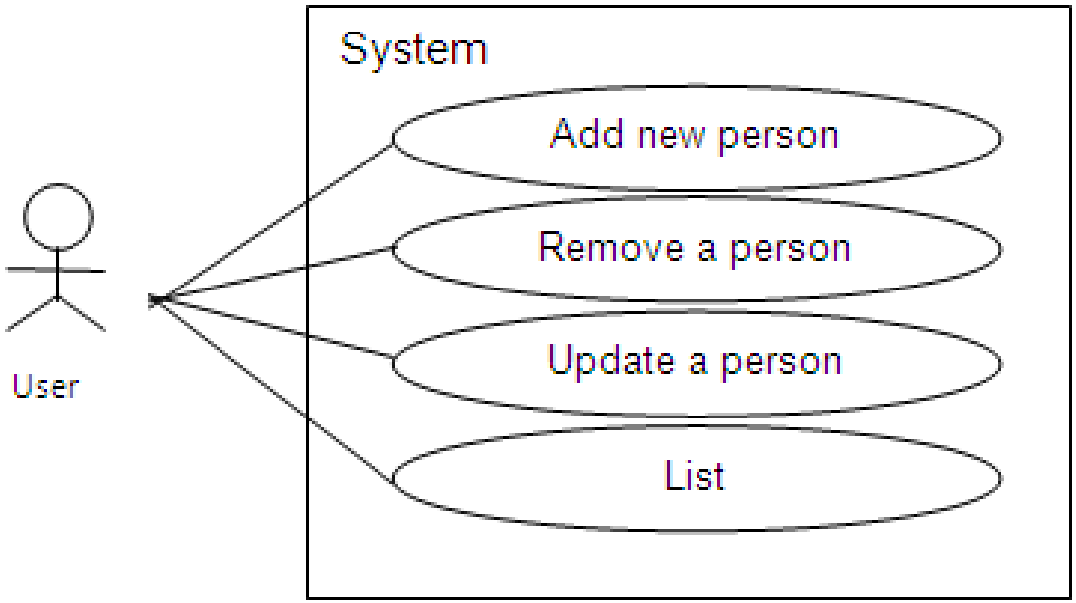
1- Problem Description

- Each person details include code, name, and age.
- Write a Java program that allows users adding a new person to the list, removing a person having a known code from the list, updating details of a known-code person, listing all managed persons in descending order of ages using a simple menu.

Report…

2- Analysis

From the problem description, following use-cases are identified:



-System/program is expressed as a bounded rectangle.
- Each function is expressed by a verb in an ellipse
-User runs a function is expressed as a line

Report…

3- Design

3.1- Class Design

From the problem description, concepts in the problem domain are expressed by following classes:

Class Person
Description for a person
Data: String code; String name; int age
Methods: Constructors Getters, setters void input() for collecting data String toString() to get data in string format

Report…

Class PersonList
Description for a list of persons

Data:
Person[] list; // current list
int count // current number of persons

Methods:
Constructors
Getters, setters
void add(); // add a new person. Data are collected from keyboard
int find (String aCode); // Find the index of the person whose code is known
void remove()// remove a person. His/ her code is accepted from keyboard
void sort(); // descending sort the list based on their ages
void update(); // update a person, data are accepted from keyboard
void print(); // print the list

Report…

Class Menu
Description for a menu

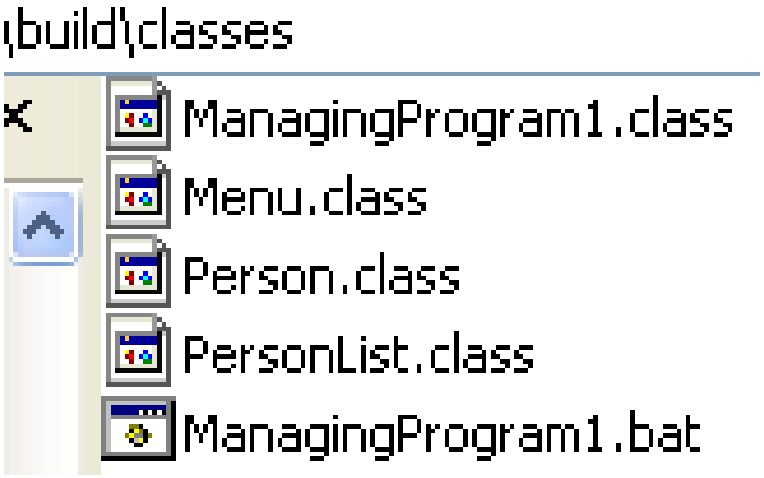
Data
String[] hints; // list of hints
int n; // current number of hints

Methods:
Menu(int n): constructor for initializing a menu containing n options
void add (String aHint); // add an option
int getChoice(); // get an option

Class ManagingProgram1
Description for the program
Data: none
Methods: main(…): main method of the program

Report…

3.2- Program structure



Algorithms

Please see comments in codes.

Report…

3.3- User interface

Menu of the program will be seen as:

1-Add new person
2-Remove a person
3-Update a person
4-List
5-Quit

Report…

4- Implementation

Initial data of the program (if any, file ……)

Please explore the software structure

Software

Please run the program

5- Testing

No.	Case	State
1	Add new person Code: not duplicate Name: …. Age: ….	Passed Passed **not passed** ……
2	Remove a person	Passed
3	Update aperson	Passed
4	List	……..
…	……	…..

Recommendations

Code Conventions:

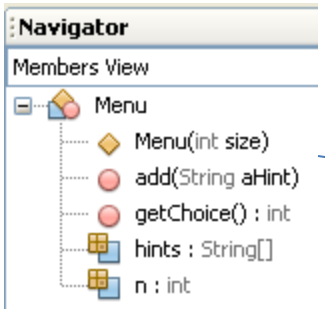
- **Indentation: 4 blanks at the beginning of each code line**
- **Comments in the code must be carried out.**
- **Names:**
  - **One-word name: lowercase**
  - **Multi-word name: The first word: lowercase, remaining words: The first character is uppercase, others are lowercase.**

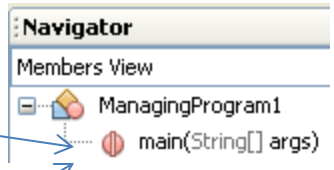
Recommendations

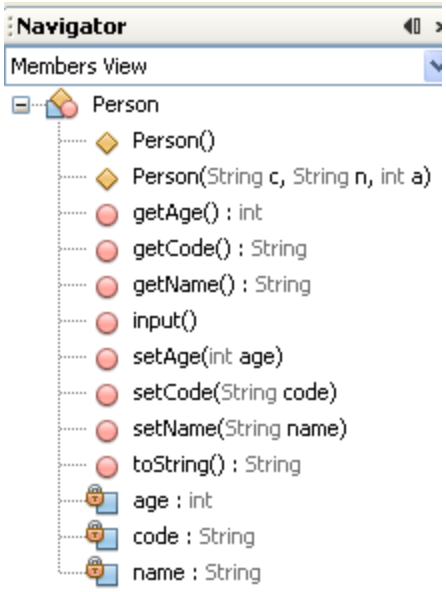
A sample :

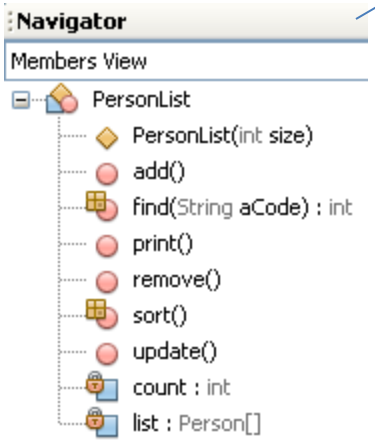
```
/*
Author:  ……
  Date:   …….
  This class represents ……..
*/
class  ClassName …….   {
   int data; //  Which does data represent?
   ….
   /*  What is the goal of the method
       Which does the return data represent?
   */
   Method implementation ….. {
   …
   }
}
```

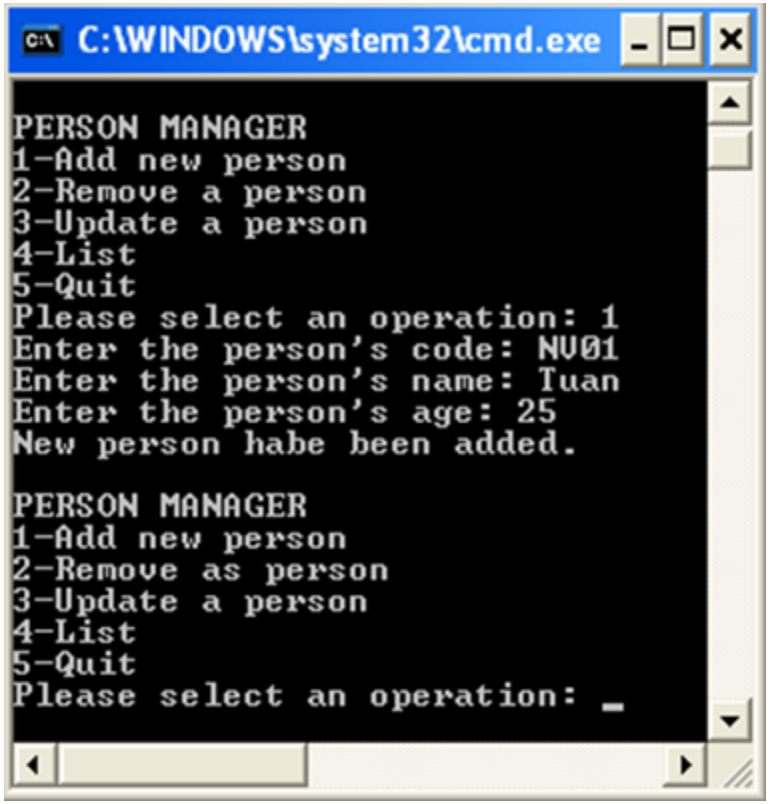
Case study: Design Guide











```java
import java.util.Scanner;
public class Menu {
  String[] hints;
  int n = 0; // current number of hints
  // create a menu with size elements
  public Menu (int size)
  {  if (size<1) size=10;
     hints = new String[size];
  }
  // add a hint
  public void add (String aHint)
  {  if (n<hints.length)
     {  hints[n++]=aHint;
     }
  }
  // get user choice
  public int getChoice()
  { int result=0;
    if (n>0)
    { // print out hints
      for (int i=0; i<n; i++)
        System.out.println( (i+1) + "-" + hints[i]);
      System.out.print("Please select an operation: ");
      Scanner sc= new Scanner(System.in);
      result= Integer.parseInt(sc.nextLine());// get user choice
    }
    return result;
  }
}
```

Case study: Code Supported

this: reference of the current object

Case study: Implementation

```
1   import java.util.Scanner;
2   public class Person {
3     private String code="", name=""; private int age=0;
4     // constructors
5     public Person()    {}
6     public Person (String c, String n, int a)
7     {  code=c; name=n; age=a>0? a: 0; }
8     // Getters and Setters
9     public String getCode()   { return code; }
10    public void setCode(String code)  { this.code = code;}
11    public String getName()   { return name;}
12    public void setName(String name)  { this.name = name;}
13    public int getAge()    { return age; }
14    public void setAge(int age)   { this.age = age;}
15    // Input details of the person
16    public void input()
17    { Scanner sc = new Scanner(System.in);
18      System.out.print("Enter the person's code: ");
19      code = sc.nextLine();
20      System.out.print("Enter the person's name: ");
21      name = sc.nextLine();
22      System.out.print("Enter the person's age: ");
23      age = Integer.parseInt(sc.nextLine());
24    }
25    // Method for output
26    public String toString()
27    { return code + ", " + name + ", " + age ;
28    }
29  }
```

Case study: Implementation

```
1  import java.util.Scanner;
2  public class PersonList {
3    private Person[] list= null;
4    private int count=0 ; // current number of persons
5    public PersonList( int size) // create a list with size persons
6    {  if (size<10) size=10;
7       list= new Person[size];
8    }
9    int find (String aCode) // find position of a known-code person
10   { for (int i=0; i<count; i++)
11       if (aCode.equals(list[i].getCode())) return i;
12     return -1;
13   }
```

Case study: Implementation

```
public class PersonList
```

```
14      public void add()
15      {  if (count == list.length) System.out.println("List is full!");
16         else
17         {  String newCode, newName; int newAge;
18            // Entering new person details
19            Scanner sc= new Scanner(System.in);
20            int pos; // variable for existing checking for new code
21            do
22            { System.out.print("Enter the person's code: ");
23              newCode = sc.nextLine().toUpperCase();
24              pos= find(newCode);
25              if (pos>=0) System.out.println("\tThis code existed!");
26            }
27            while (pos>=0);
28            System.out.print("Enter the person's name: ");
29            newName = sc.nextLine().toUpperCase();
30            System.out.print("Enter the person's age: ");
31            newAge = Integer.parseInt(sc.nextLine());
32            list[count++] = new Person(newCode, newName, newAge);
33            System.out.println("New person habe been added.");
34         }
35      }
```

Case study: Implementation

public class **PersonList**

```
36      public void remove()
37      {   if (count==0)
38          { System.out.println("Empty list.");
39            return;
40          }
41          String removedCode;
42          // Entering new person details
43          Scanner sc= new Scanner(System.in);
44          System.out.print("Enter the code of removed person: ");
45          removedCode = sc.nextLine().toUpperCase();
46          int pos = find (removedCode);
47          if (pos<0) System.out.println("This person does not exist.");
48          else
49          {   // Shift up the remainder of the list
50              for (int i=pos; i<count-1; i++) list[i]= list[i+1];
51              count--;
52              System.out.println("The person " + removedCode + " was removed");
53          }
54      }
```

Case study: Implementation

public class PersonList

```
55      public void update() // updating name and age only
56      {   if (count==0)
57          { System.out.println("Empty list.");
58            return;
59          }
60          String code;
61          // Entering the person's code
62          Scanner sc= new Scanner(System.in);
63          System.out.print("Enter the code of updated person: ");
64          code = sc.nextLine().toUpperCase();
65          int pos = find (code);
66          if (pos<0) System.out.println("This person does not exist.");
67          else
68          {   // Update name and age
69              String newName; int newAge;
70              System.out.print("Enter the person's name: ");
71              newName = sc.nextLine().toUpperCase();
72              System.out.print("Enter the person's age: ");
73              newAge = Integer.parseInt(sc.nextLine());
74              list[pos].setName(newName);
75              list[pos].setAge(newAge);
76              System.out.println("The person " + code + " was updated");
77          }
78      }
```

Case study: Implementation

```
public class PersonList

79      public void print()
80      {   if (count==0)
81          { System.out.println("Empty list.");
82            return;
83          }
84          System.out.println("LIST OF PERSONS:");
85          for (int i=0; i<count; i++)
86              System.out.println(list[i].toString());
87      }
88      void sort()
89      { if (count==0) return;
90        // Bubble Sort based on person's age
91        for (int i=0; i<count-1;i++)
92        for (int j=count-1; j>i; j--)
93            if (list[j].getAge()>list[j-1].getAge())
94            {   Person p = list[j];
95                list[j]=list[j-1];
96                list[j-1]=p;
97            }
98      }
99  }
```

```
public class ManagingProgram1 {
   public static void main(String[] args)
   {  Menu menu= new Menu(5);
      menu.add("Add new person");
      menu.add("Remove a person");
      menu.add("Update a person");
      menu.add("List");
      menu.add("Quit");
      int choice;
      PersonList list= new PersonList(50);
      do
      {  System.out.println("\nPERSON MANAGER");
         choice=menu.getChoice();
         switch(choice)
         { case 1: list.add(); break;
           case 2: list.remove(); break;
           case 3: list.update(); break;
           case 4: list.sort(); list.print(); break;
         }
      }
      while (choice>=1 && choice <5);
   }
}
```

Summary

- The anatomy of a class, and how to declare fields, methods, and constructors.
- Hints for class design:
  - Main noun → Class
  - Descriptive nouns → Fields
  - Methods: Constructors, Getters, Setters, Normal methods
- Creating and using objects.
- To instantiate an object: Using appropiate constructior
- Use the dot operator to access the object's instance variables and methods.