Encapsulation

Objectives

- Class and Object
- How to identify classes
- Hints for class design
- How to declare/use a class
- Member functions
- Common modifiers (a way to hide some members in a class)
- Case study

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.
- Getters/Setters: implementing **getter** and **setter** is one of the ways to enforce encapsulation in the program’s code.

How to Identity a Class

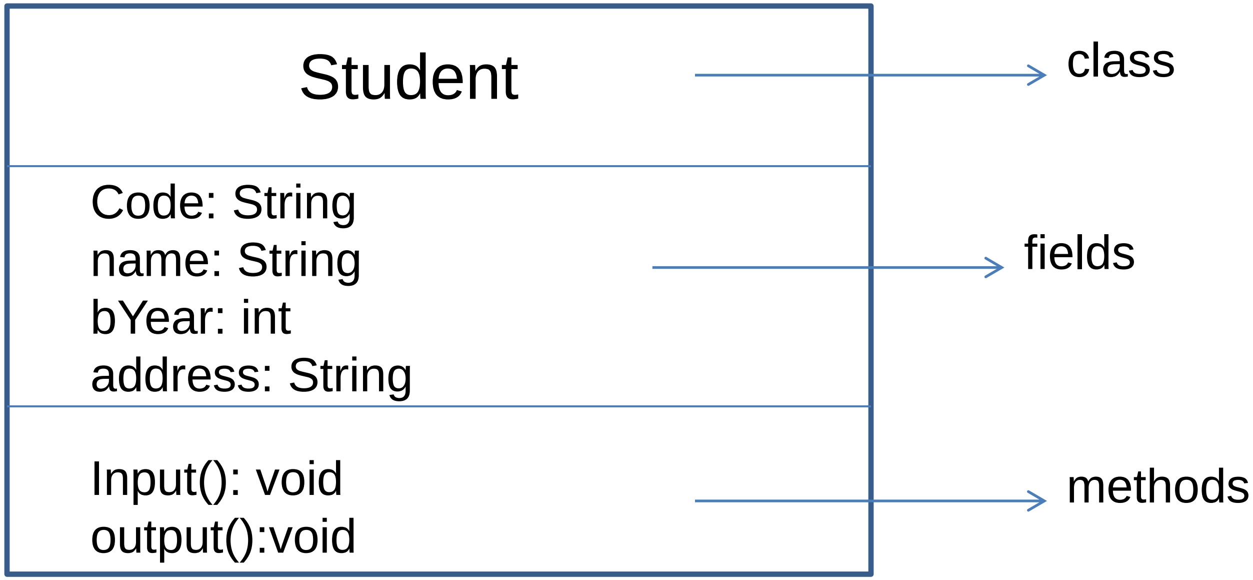
- Main noun: Class
- Nouns as modifiers of main noun: Fields
- Verbs related to main noun: Methods

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

Main noun: Student
Auxiliary nouns:code , name, bYear, address;
verbs: input() , output()

Hints for class design

A UML class diagram is used to represent the Student class



Student

Code: String
name: String
bYear: int
address: String

Input(): void
output():void

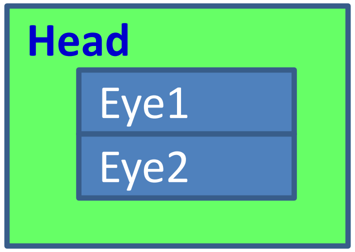
class

fields

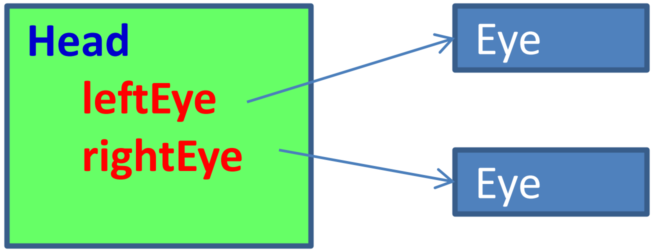
methods

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.



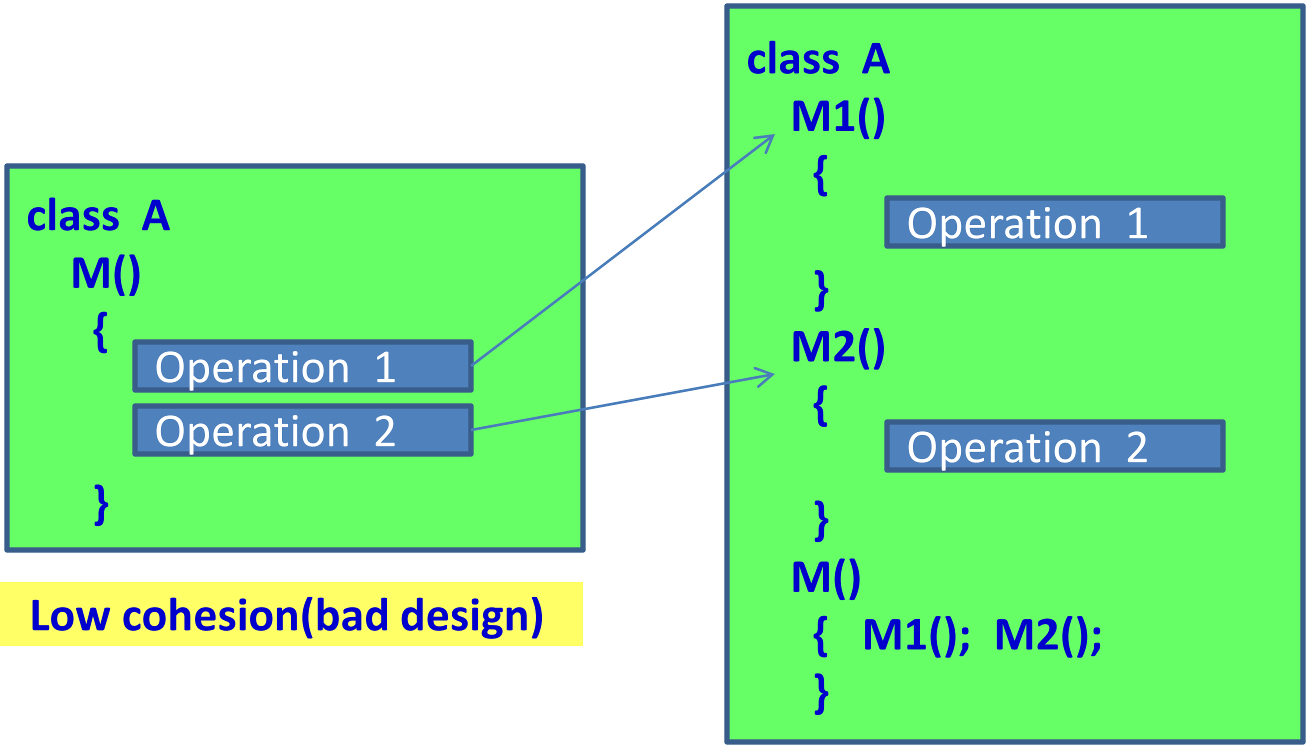
High coupling
(Bad design)



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.



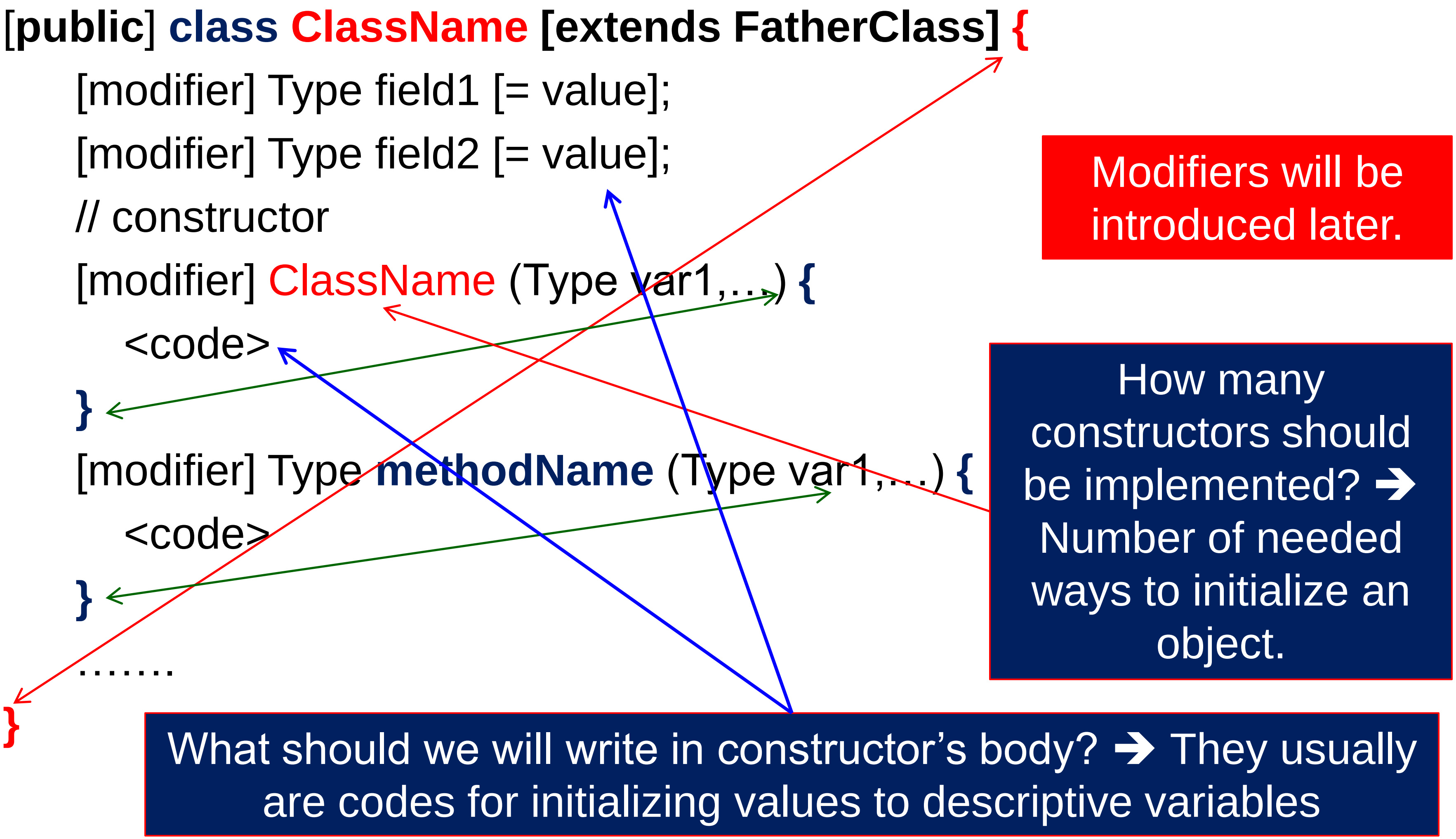
```
class A
   M()
    {
        Operation 1
        Operation 2
    }
```

```
class A
   M1()
    {
        Operation 1
    }
   M2()
    {
        Operation 2
    }
   M()
    {  M1();  M2();
    }
```

Low cohesion(bad design)

High cohesion(good design)

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] Type 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

Member functions: 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.

Member functions: Constructors

```
//default constructor
public Student(){
   code=“SE123”;
   name=“Hieu”;
   bYear= 2000;
   address=“1 Ba Trieu , HN”.
}
//constructor with parameters
public Student(String code, String name, int  bYear, String address){
   this.code=code;
   this.name=name;
   this.bYear= year;
   this.address=address.
}
```

The current object: this

- The keyword **this** returns the address of the current object.
- This holds the address of the region of memory that contains all of the data stored in the instance variables of current object.
- **Scope of this: this** is created and used just when the member method is called. After the member method terminates **this** will be discarded

Member functions: Getter/Setter

- A getter is a method that gets the value of a property.
- A setter is a method that sets the value of a property.
- Uses:
  - ➢for completeness of encapsulation
  - ➢to maintain a consistent interface in case internal details change

Member functions: Getter/Setter

- For example:

```
public String getName(){
        return name;
}
public void setName(String name){
        if(! name.isEmpty())
                this.name=name;
}
```

Member functions: other methods

- Typical method declaration:

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

- Signature: data help identifying something
- Method Signature:

  name + order of parameter types

Member functions: other Methods

- For example:

```
public void input(){
     //code here
}
public void output(){
    //code here
}
```

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.

```
Student stu = new
Student(“SE123”,”Minh”,2000,”1 Ba Trieu”);
```

- 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**

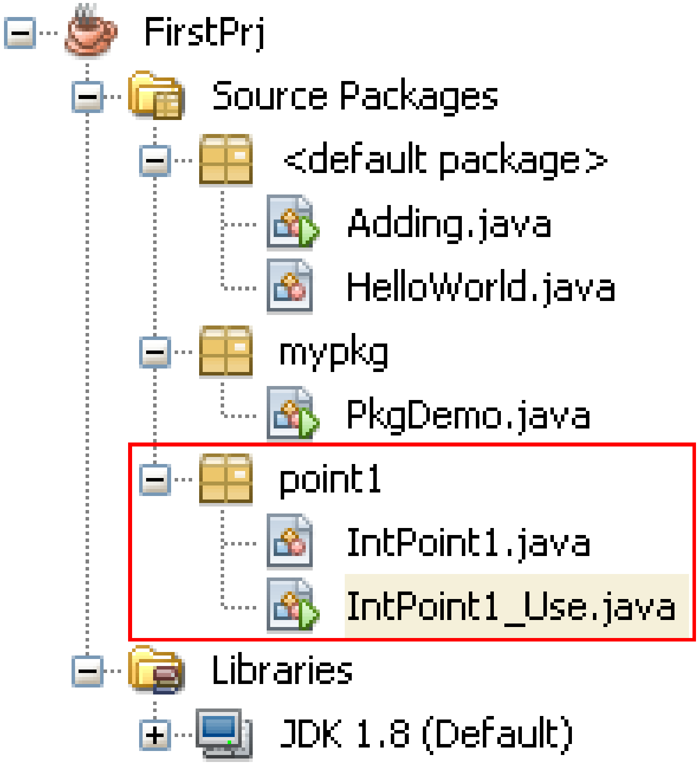
  object.field
- **Calling a method of an object**

  object.method(params)

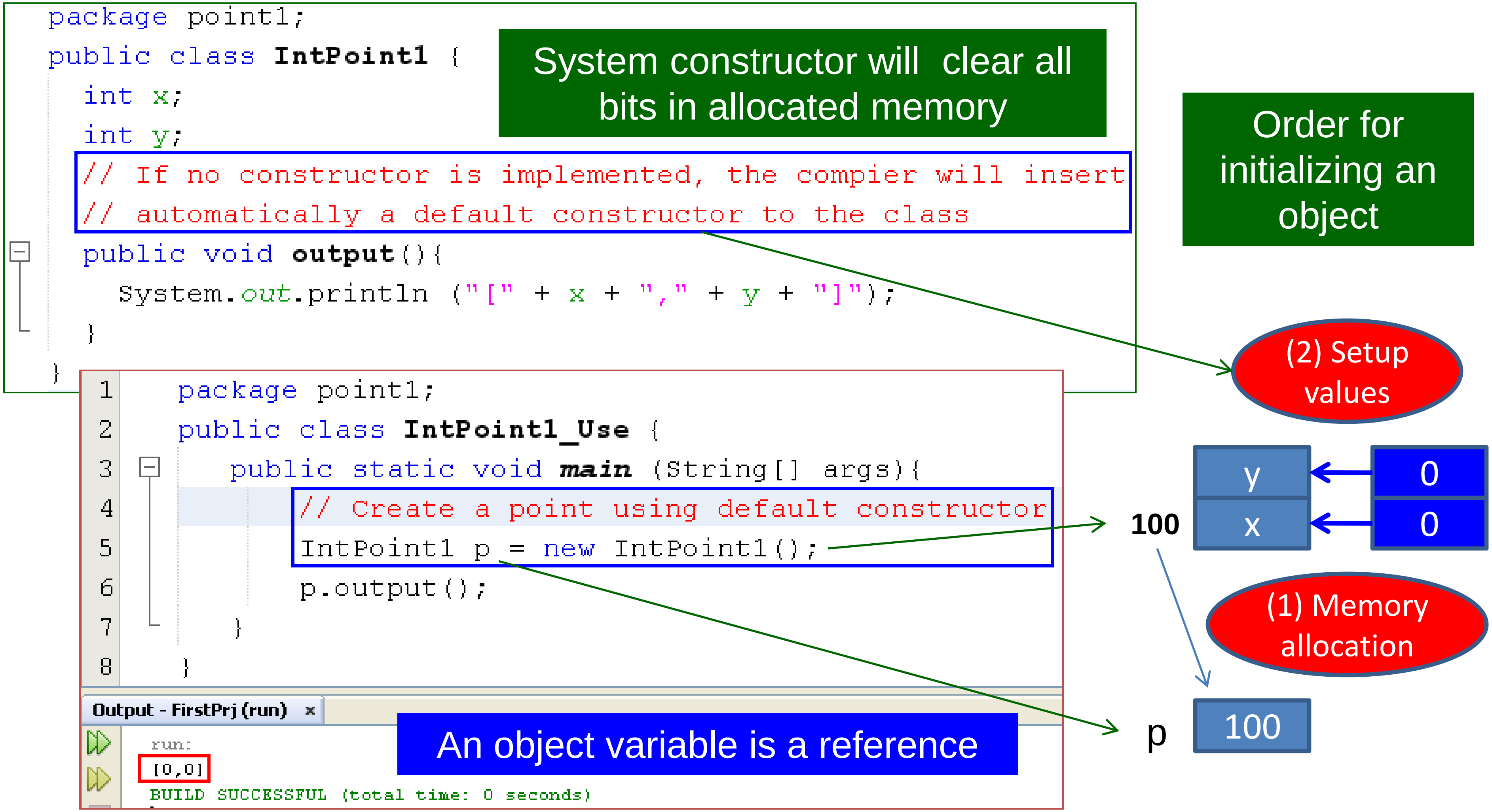
Demo: If we do not implement any constructor, compiler will insert to the class a system default constructor

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

(2) Setup values

```
package point1;
public class IntPoint1_Use {
   public static void main (String[] args){
      // Create a point using default constructor
      IntPoint1 p = new IntPoint1();
      p.output();
   }
}
```

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

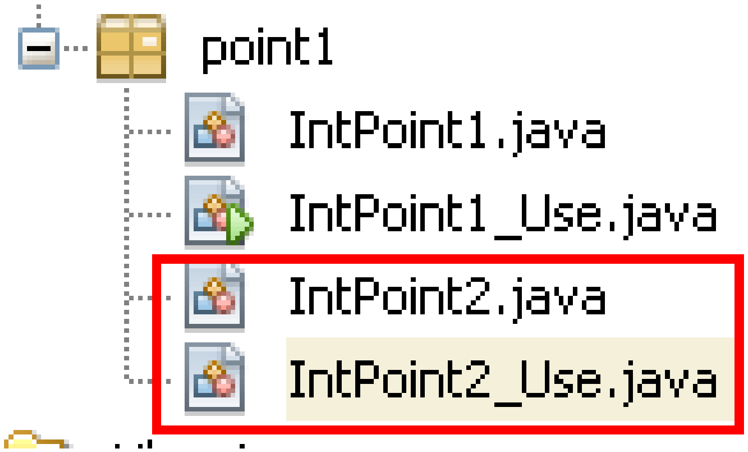
An object variable is a reference

(1) Memory allocation

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

This demonstration will depict:

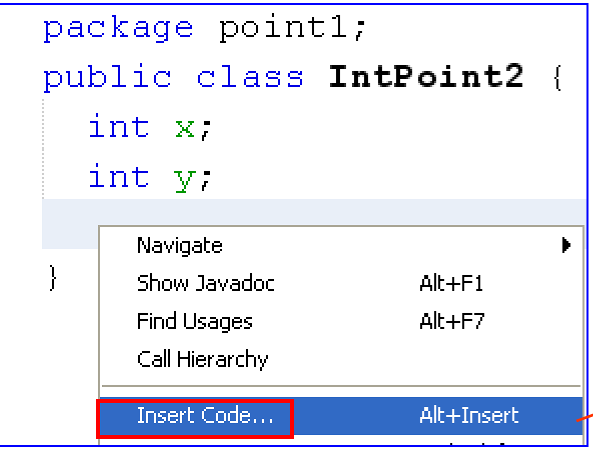
- The way to insert some methods automatically in NetBeans
- If user-defined constructors are implemented, compiler does not insert the system default constructor



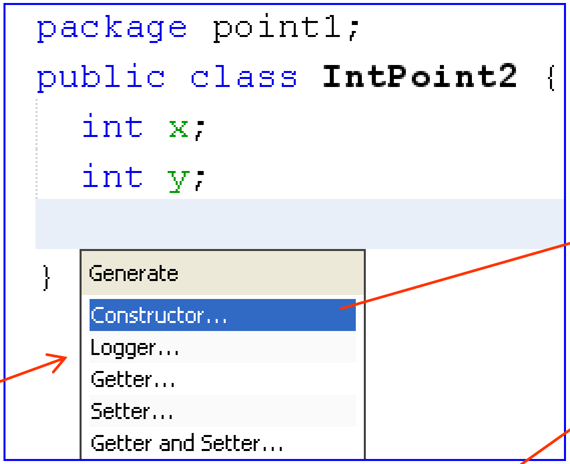
point1
IntPoint1.java
IntPoint1_Use.java
IntPoint2.java
IntPoint2_Use.java

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

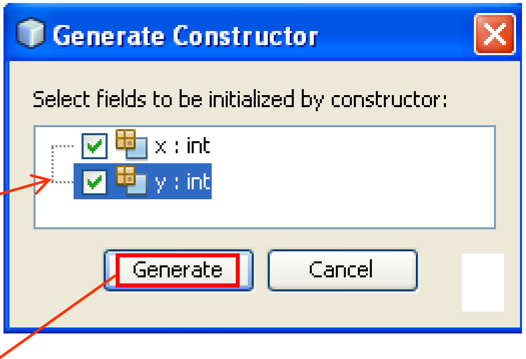
Insert constructor

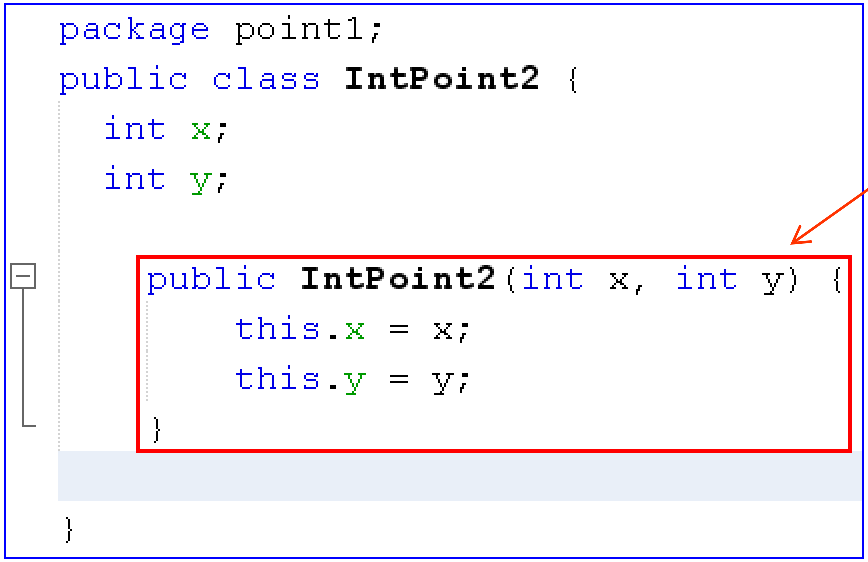


```
package point1;
public class IntPoint2 {
   int x;
   int y;

}
```



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

}
```





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

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

}
```

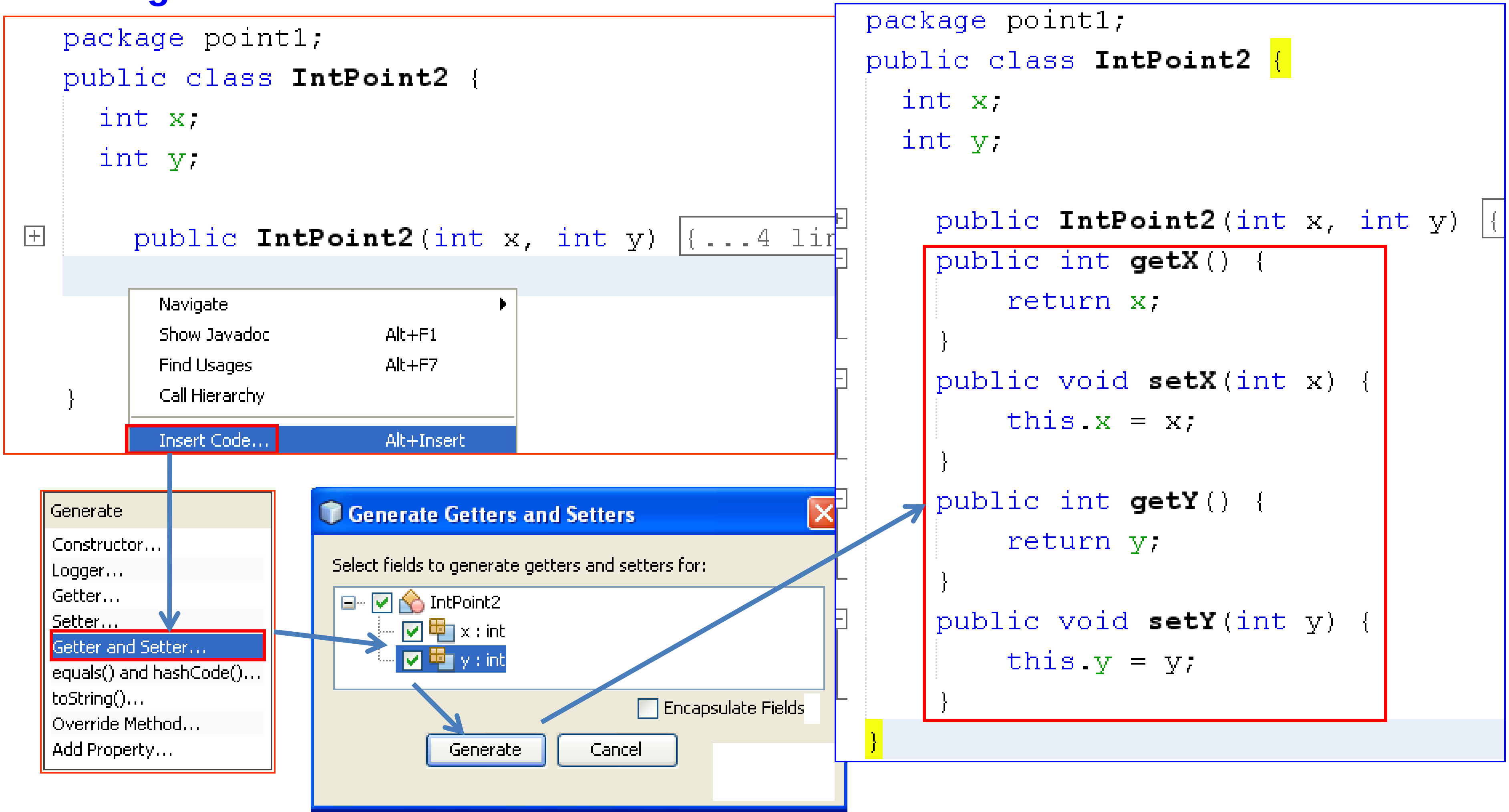
Parameter names are the same as those in declared data filed. So, the keyword **this** will help distinguish field name and parameter name.

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

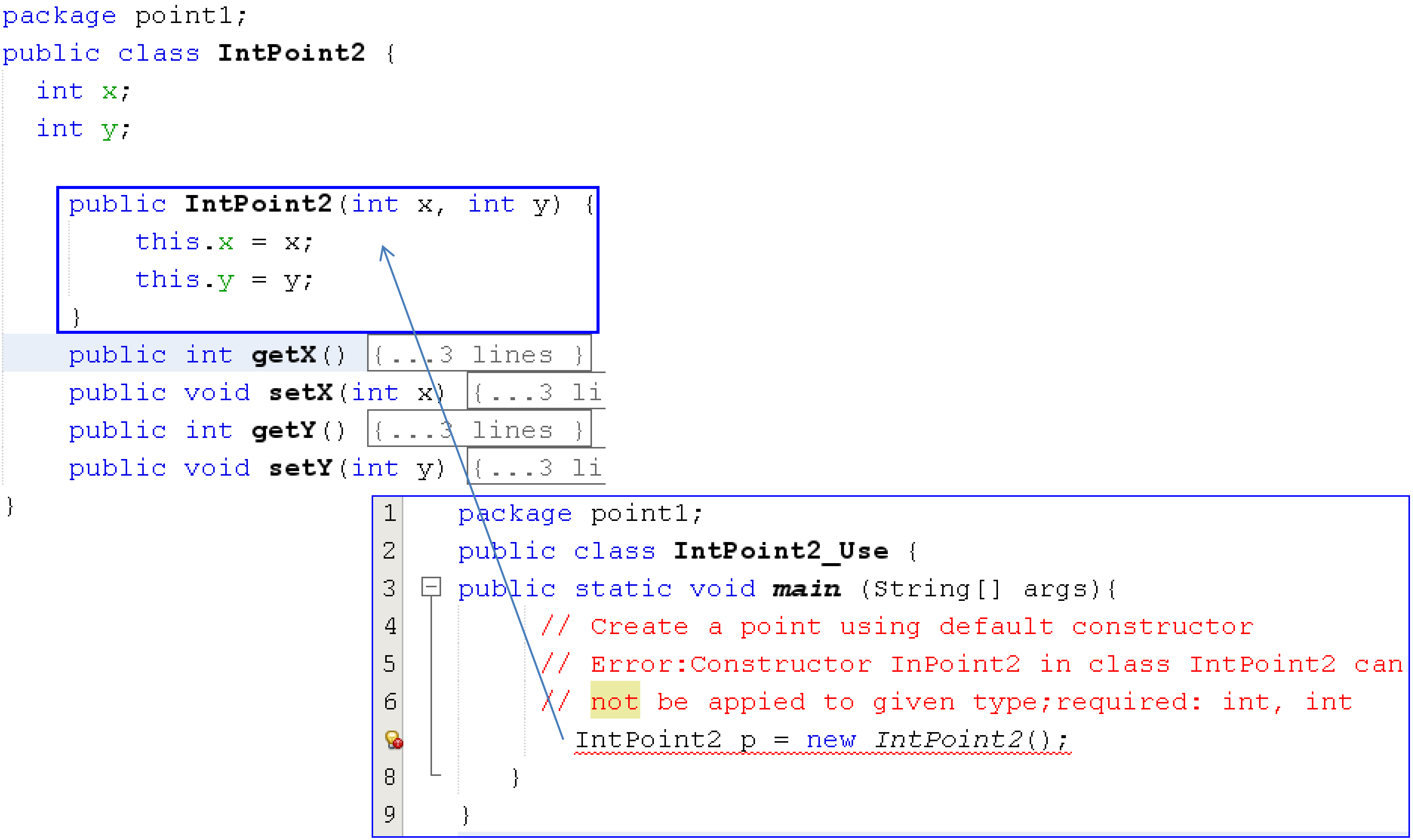
Insert getter/setter



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

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

}
```

```
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;
    }
}
```

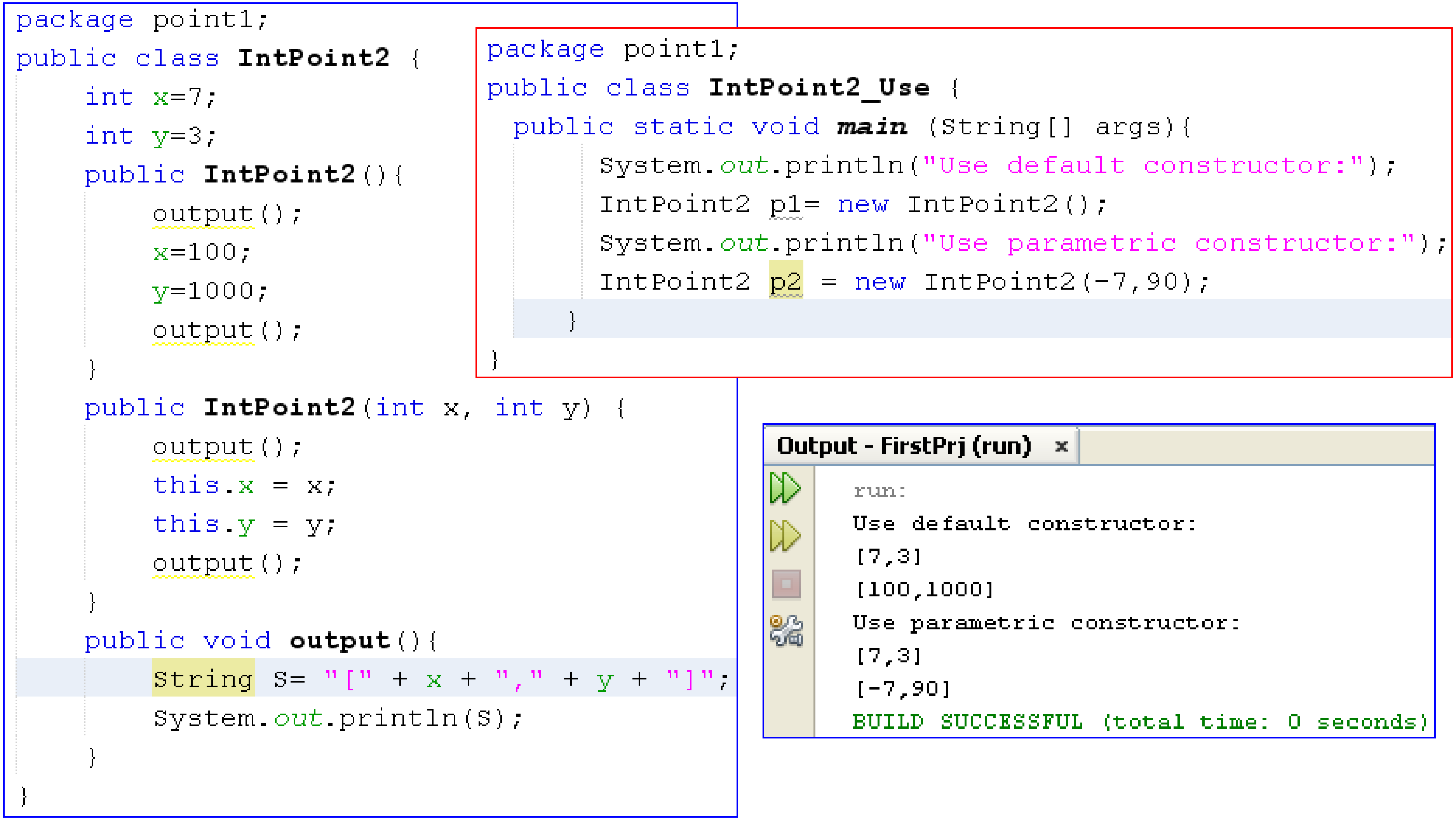
Demo: If we implement a constructor, compiler does not insert system constructor



```
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
}
```

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

Explain the result of the following program



```java
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);
    }
}
```

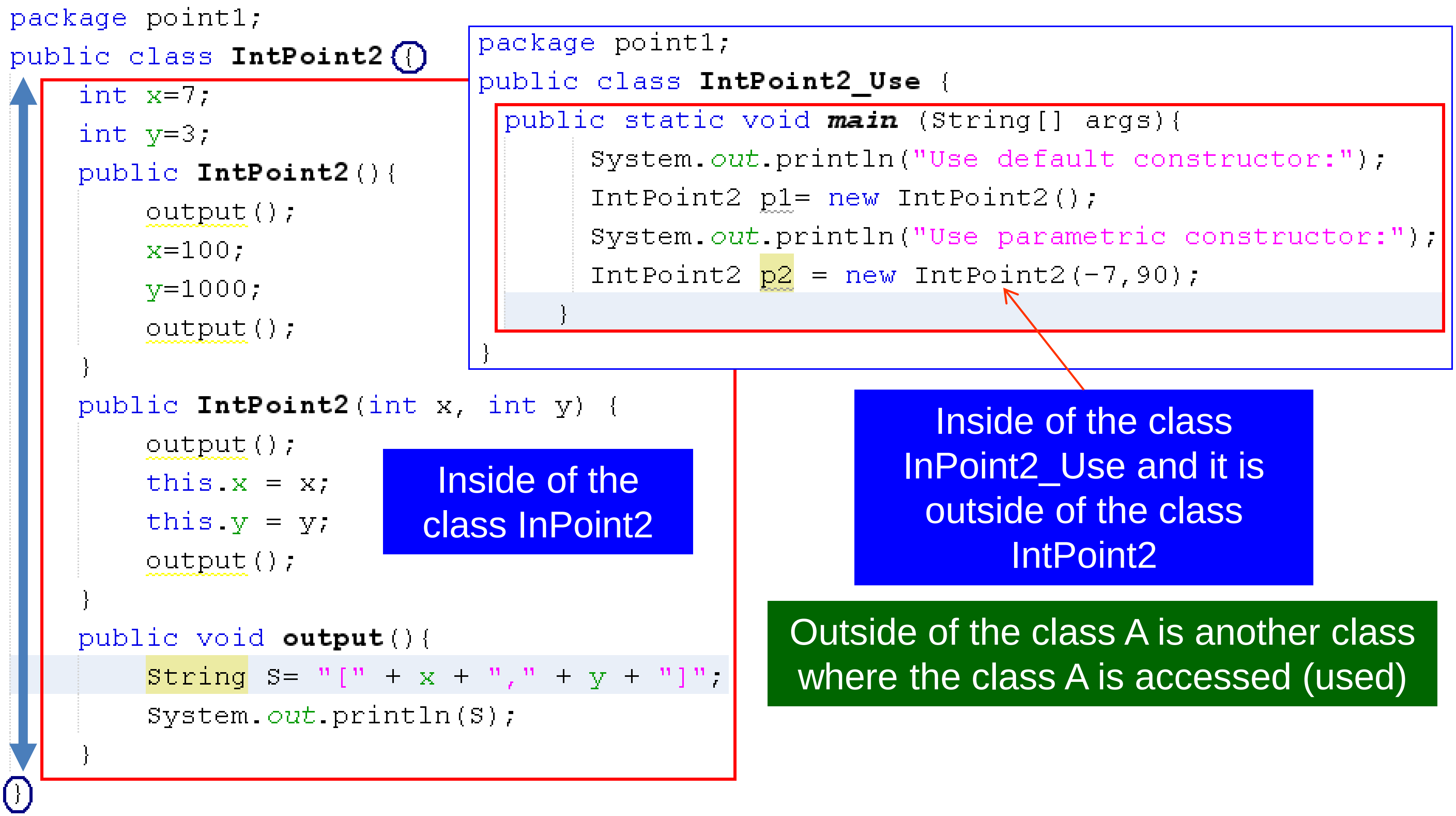
```java
package point1;
public class IntPoint2_Use {
  public static void main (String[] args){
        System.out.println("Use default constructor:");
        IntPoint2 p1= new IntPoint2();
        System.out.println("Use parametric constructor:");
        IntPoint2 p2 = new IntPoint2(-7,90);
    }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

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

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);
    }
}
```

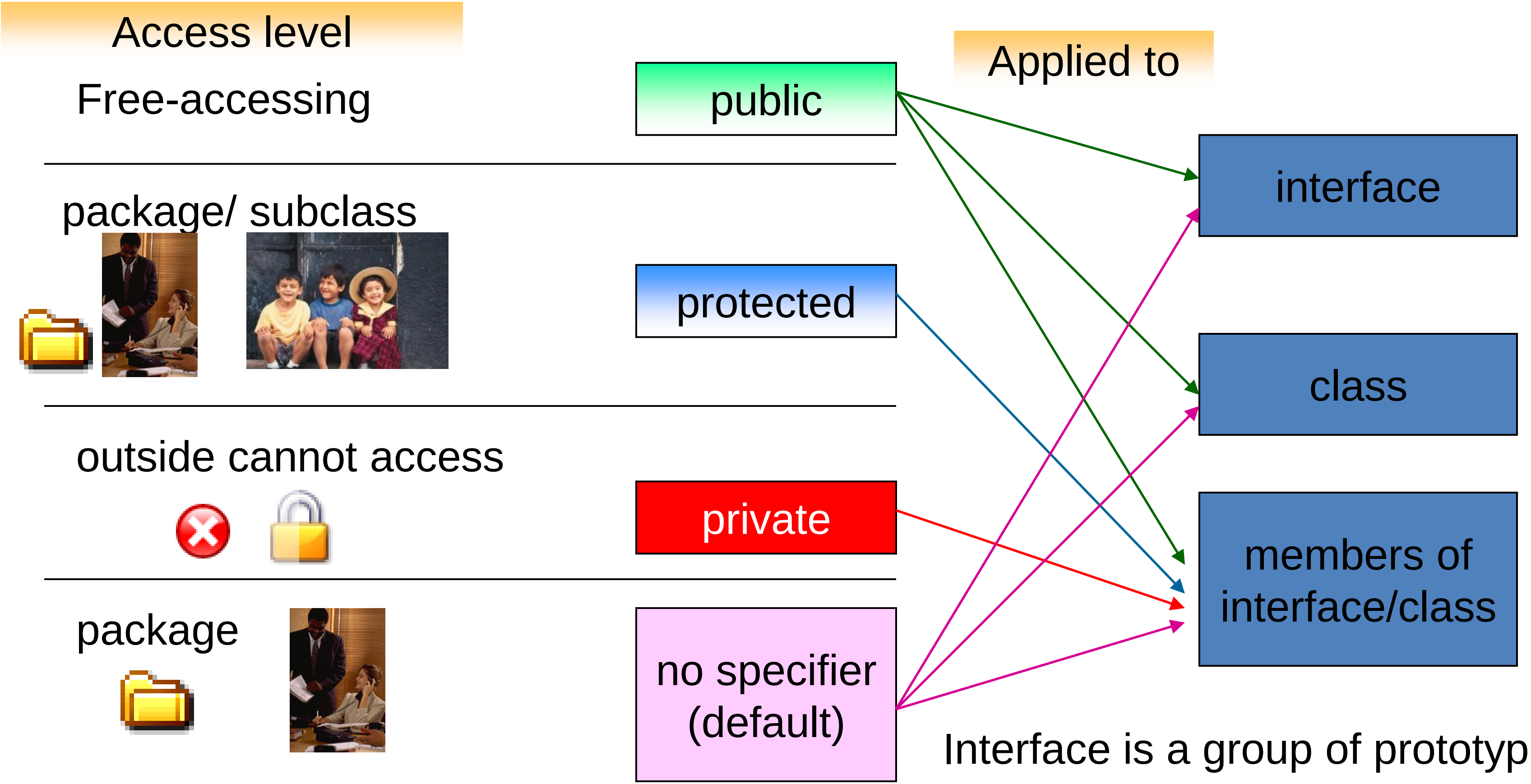
```
package point1;
public class IntPoint2_Use {
  public static void main (String[] args){
        System.out.println("Use default constructor:");
        IntPoint2 p1= new IntPoint2();
        System.out.println("Use parametric constructor:");
        IntPoint2 p2 = new IntPoint2(-7,90);
  }
}
```

Inside of the class InPoint2

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



Access level

Free-accessing

public

package/ subclass

protected

outside cannot access

private

package

no specifier (default)

Applied to

interface

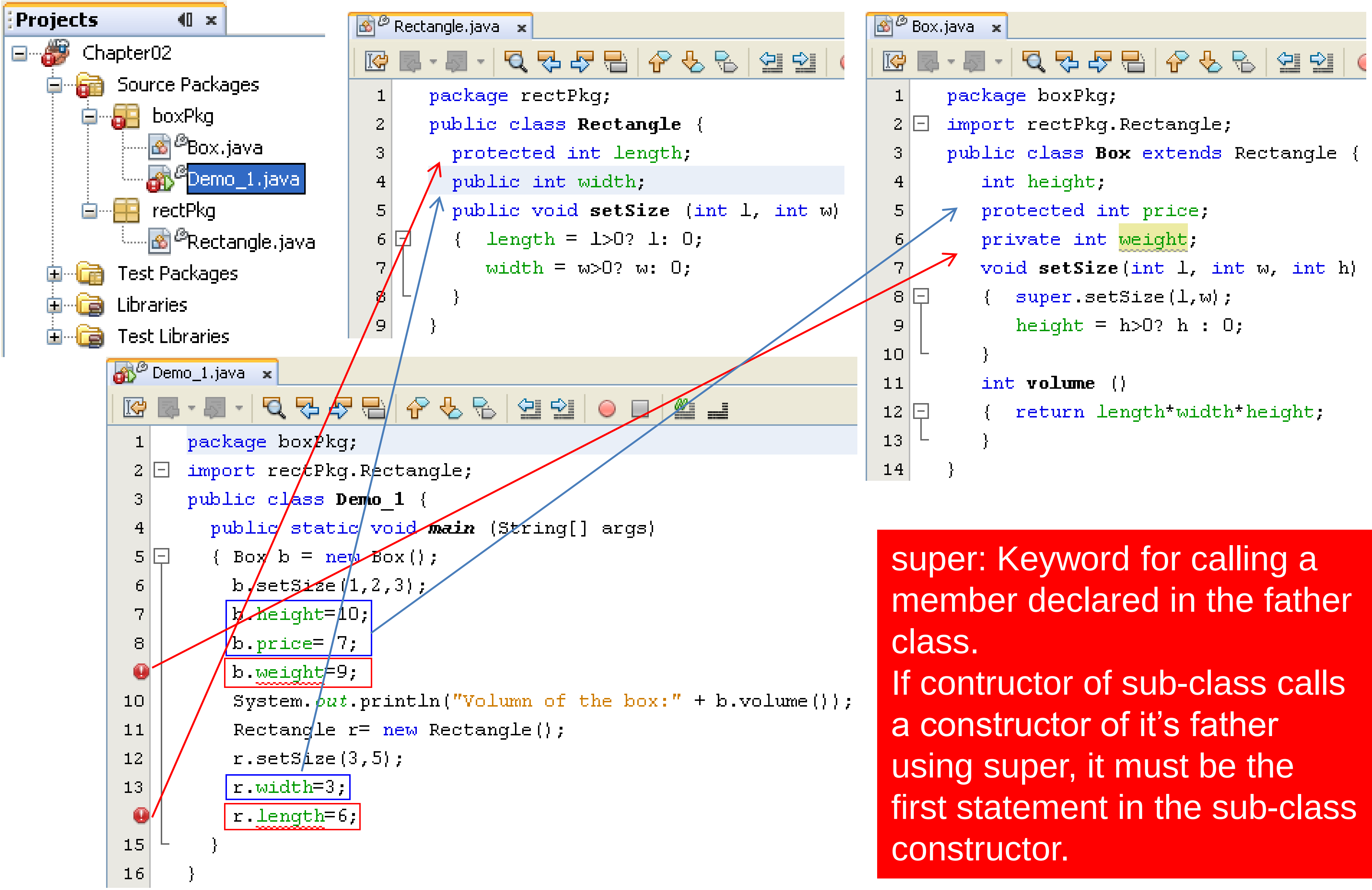
class

members of interface/class

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

```java
package rectPkg;
public class Rectangle {
  protected int length;
  public int width;
  public void setSize (int l, int w)
  {   length = l>0? l: 0;
      width = w>0? w: 0;
  }
}
```

Box.java

```java
package boxPkg;
import rectPkg.Rectangle;
public class Box extends Rectangle {
   int height;
   protected int price;
   private int weight;
   void setSize(int l, int w, int h)
   {  super.setSize(l,w);
      height = h>0? h : 0;
   }
   int volume ()
   {  return length*width*height;
   }
}
```

Demo_1.java

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

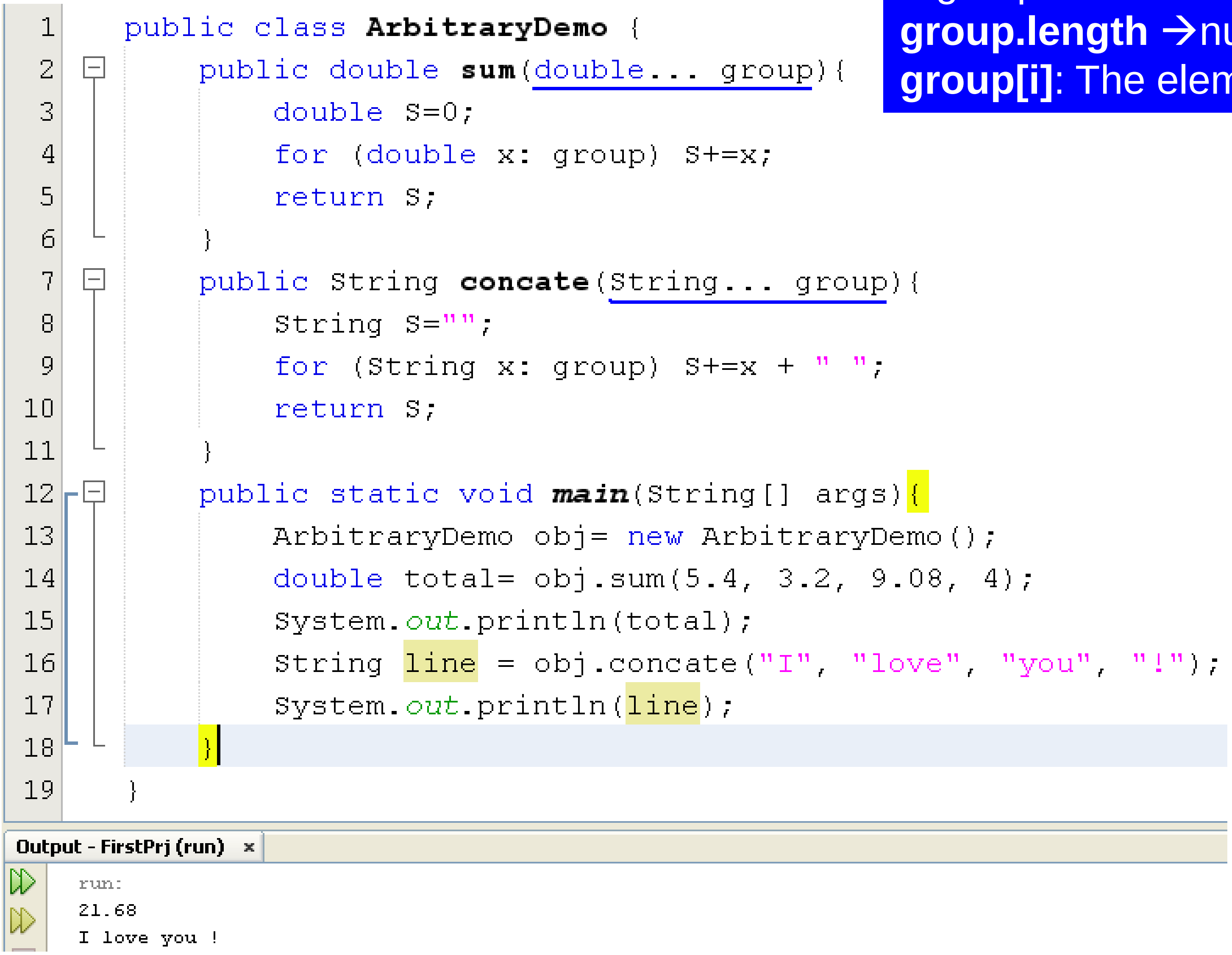
super: Keyword for calling a member declared in the father class.
If contructor of sub-class calls a constructor of it's father using super, it must be the first statement in the sub-class constructor.

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



```
public class ArbitraryDemo {
    public double sum(double... group){
        double S=0;
        for (double x: group) S+=x;
        return S;
    }
    public String concate(String... group){
        String S="";
        for (String x: group) S+=x + " ";
        return S;
    }
    public static void main(String[] args){
        ArbitraryDemo obj= new ArbitraryDemo();
        double total= obj.sum(5.4, 3.2, 9.08, 4);
        System.out.println(total);
        String line = obj.concate("I", "love", "you", "!");
        System.out.println(line);
    }
}
```

```
run:
21.68
I love you !
```

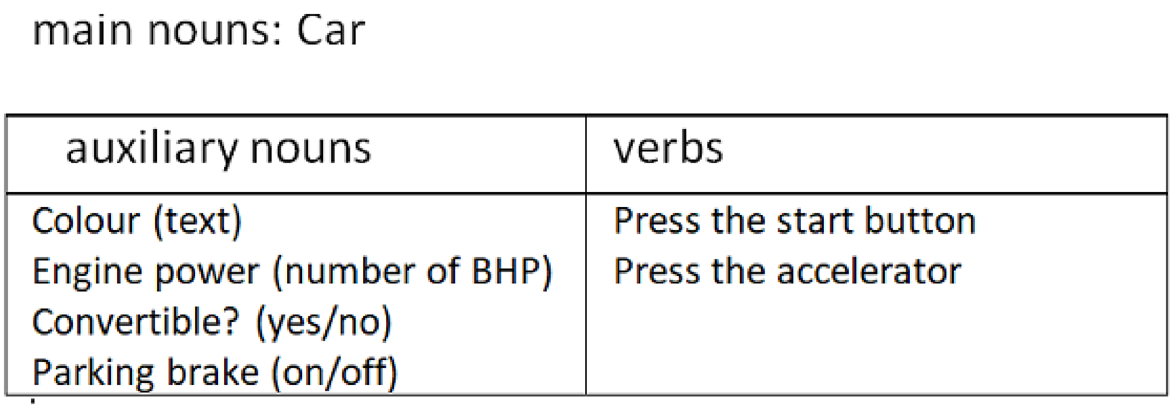
Case study

- **Problem**:
  A sports car can be one of a variety of colours, with an engine power between 100 HP and 200 HP. It can be a convertible or a regular model. The car has a button that starts the engine and a parking brake. When the parking brake is released and you press the accelerator, it drives in the direction determined by the transmission setting.

Report…

Class Design

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

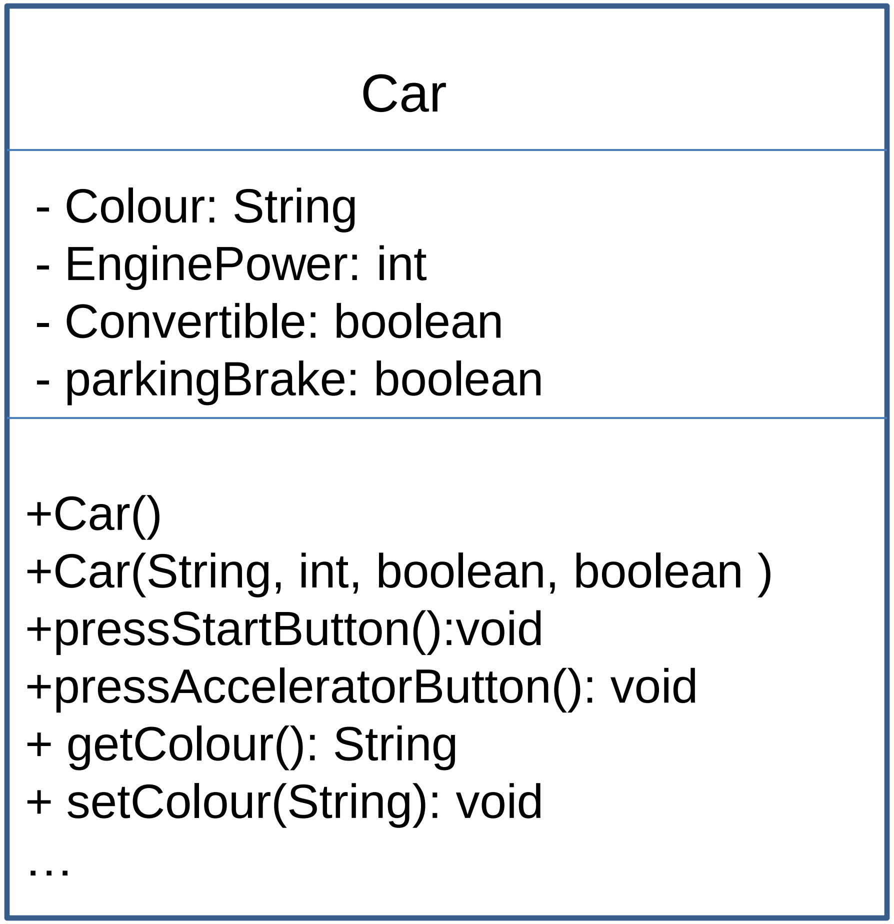


main nouns: Car

auxiliary nouns	verbs
Colour (text) Engine power (number of BHP) Convertible? (yes/no) Parking brake (on/off)	Press the start button Press the accelerator

Report…

- A UML class diagram is used to represent the Car class

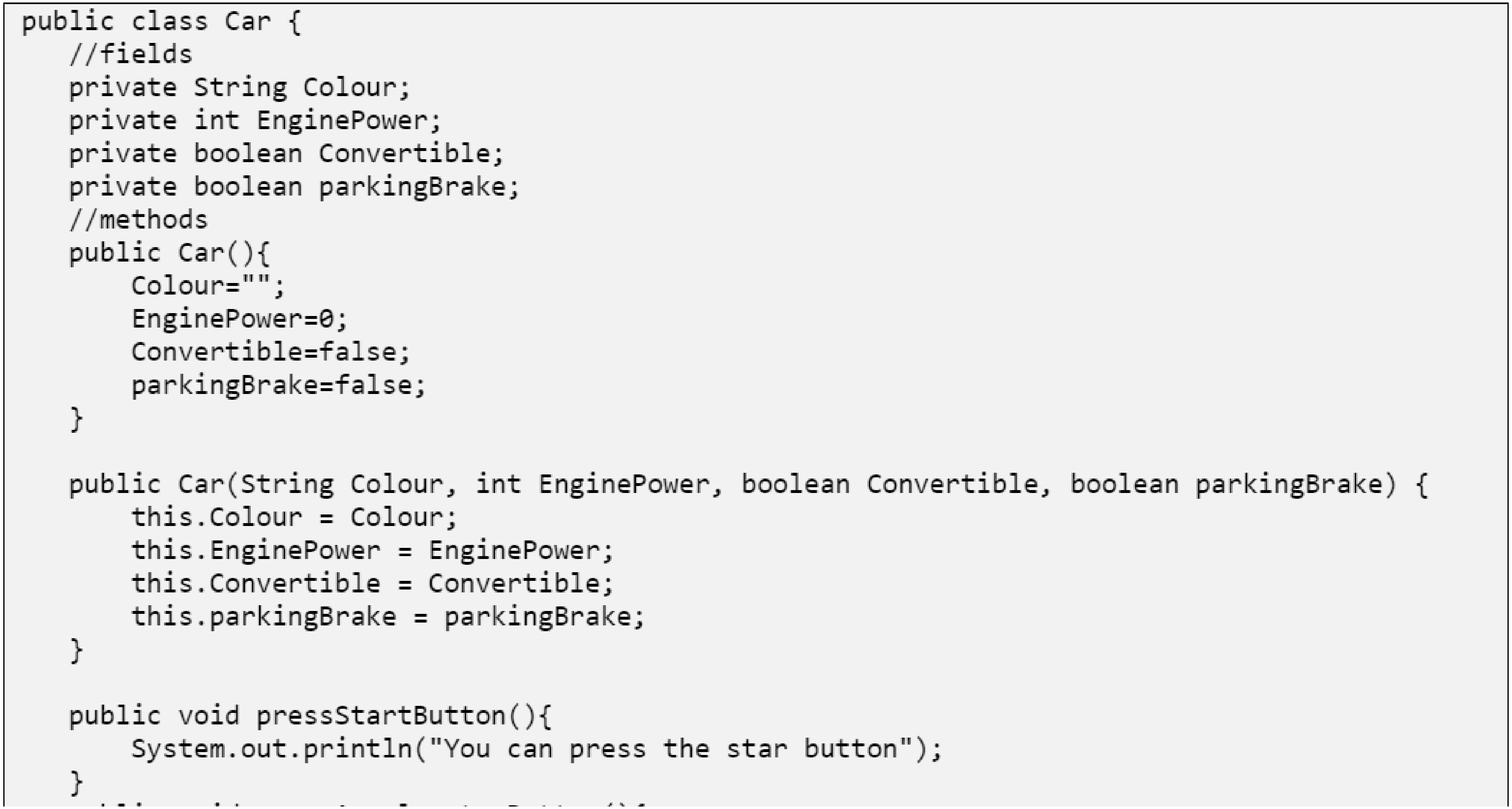


Car

- Colour: String
- EnginePower: int
- Convertible: boolean
- parkingBrake: boolean

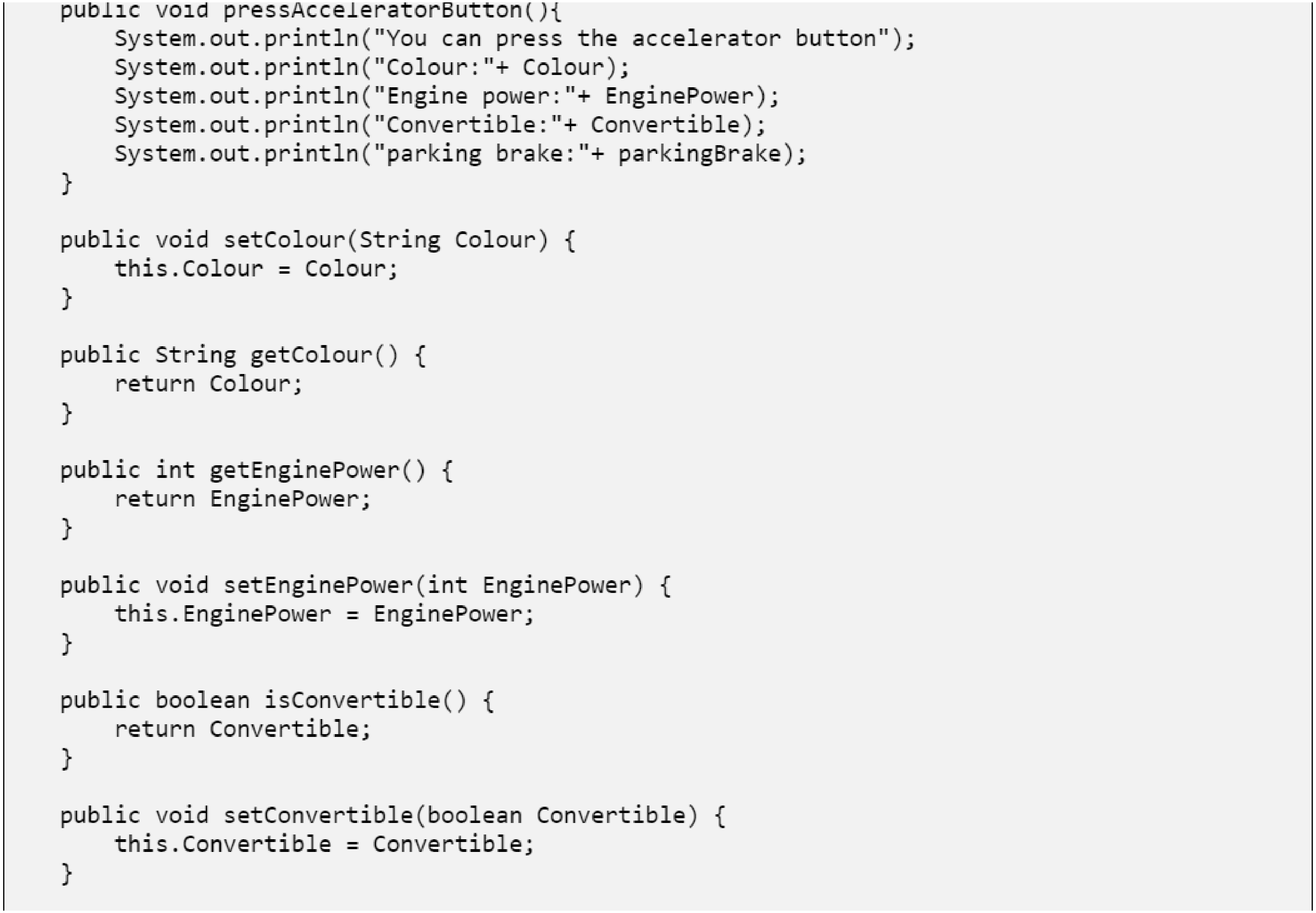
+Car()
+Car(String, int, boolean, boolean)
+pressStartButton():void
+pressAcceleratorButton(): void
+ getColour(): String
+ setColour(String): void
…

Implement



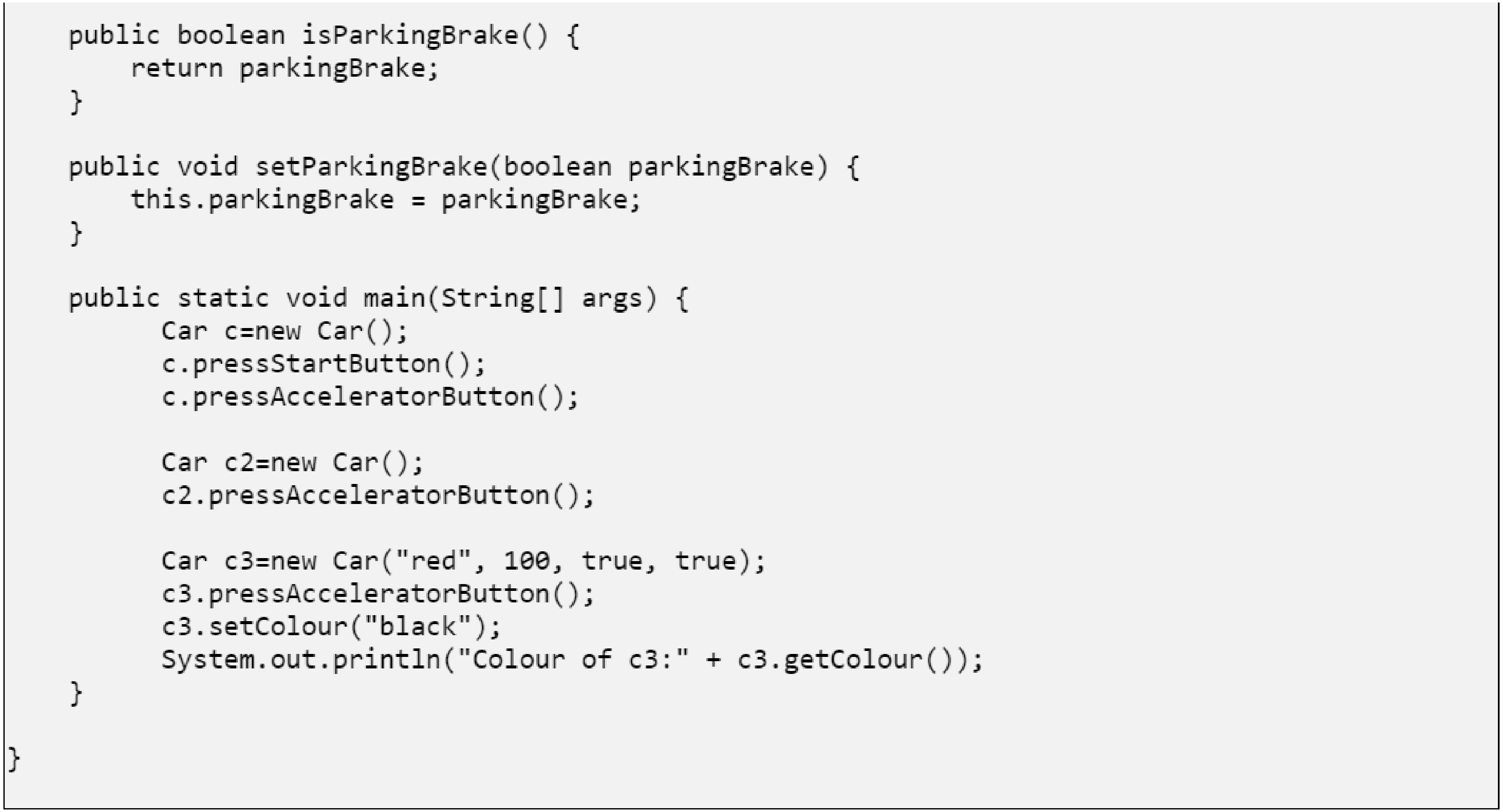
```java
public class Car {
   //fields
   private String Colour;
   private int EnginePower;
   private boolean Convertible;
   private boolean parkingBrake;
   //methods
   public Car(){
       Colour="";
       EnginePower=0;
       Convertible=false;
       parkingBrake=false;
   }

   public Car(String Colour, int EnginePower, boolean Convertible, boolean parkingBrake) {
       this.Colour = Colour;
       this.EnginePower = EnginePower;
       this.Convertible = Convertible;
       this.parkingBrake = parkingBrake;
   }

   public void pressStartButton(){
       System.out.println("You can press the star button");
   }
```

Implement



```
    public void pressAcceleratorButton(){
        System.out.println("You can press the accelerator button");
        System.out.println("Colour:"+ Colour);
        System.out.println("Engine power:"+ EnginePower);
        System.out.println("Convertible:"+ Convertible);
        System.out.println("parking brake:"+ parkingBrake);
    }

    public void setColour(String Colour) {
        this.Colour = Colour;
    }

    public String getColour() {
        return Colour;
    }

    public int getEnginePower() {
        return EnginePower;
    }

    public void setEnginePower(int EnginePower) {
        this.EnginePower = EnginePower;
    }

    public boolean isConvertible() {
        return Convertible;
    }

    public void setConvertible(boolean Convertible) {
        this.Convertible = Convertible;
    }
```

Implement



```
    public boolean isParkingBrake() {
        return parkingBrake;
    }

    public void setParkingBrake(boolean parkingBrake) {
        this.parkingBrake = parkingBrake;
    }

    public static void main(String[] args) {
          Car c=new Car();
          c.pressStartButton();
          c.pressAcceleratorButton();

          Car c2=new Car();
          c2.pressAcceleratorButton();

          Car c3=new Car("red", 100, true, true);
          c3.pressAcceleratorButton();
          c3.setColour("black");
          System.out.println("Colour of c3:" + c3.getColour());
    }

}
```

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.