CHAPTER 2

Classes and Objects

Objectives

- Programming Paradigms
- OOP basic concepts
- How to declare/use a class
  - Constructors
  - Current object – this
  - Member functions
  - Passing Arguments a Constructor/Method
  - Creating Objects
- Common modifiers (a way to hide some members in a class)
- Cotrolling access to members of a class using modifiers
- Overriding methods in sub-classes

- Nested classes (Inner class, Outer class)
- java.lang.Math, java.util.Random,
- Wrapper classes
- Case study

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
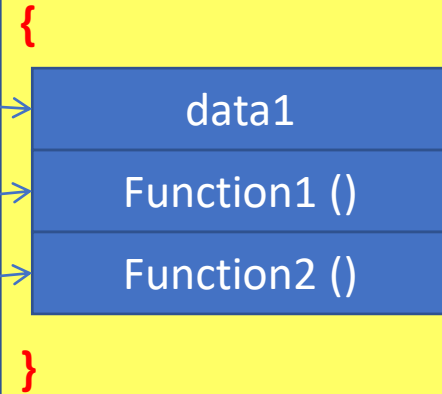
Procedure-Oriented Program

data1
data2
Function1 (data1)
Function2 (data1)
Function3 (data2)
Function4 (data2)
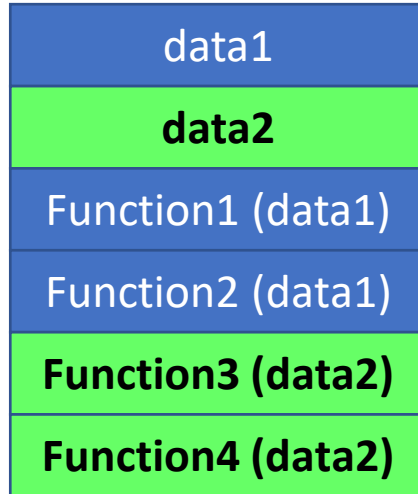
Object = Data + Methods

Basic Concepts
- **Encapsulation**
- **Inheritance**
- **Polymorphism**

Particular methods: Constructors

Class A
{

data1
Function1 ()
Function2 ()

}

Class B
{

data2
Function3 ()
Function4()

}
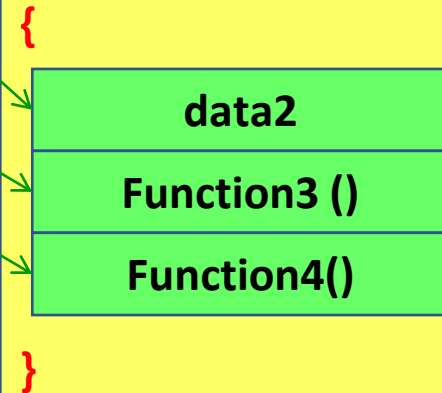
Common methods for accessing a data field:

Type getField()
void setField (Type newValue)

OOP Concepts

- Enc**a**psulation
- Inh**e**ritance
- Polym**o**rphism

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.

Inheritance (1)

Ability allows a class having members of an existed class ➔ Re-used code, save time
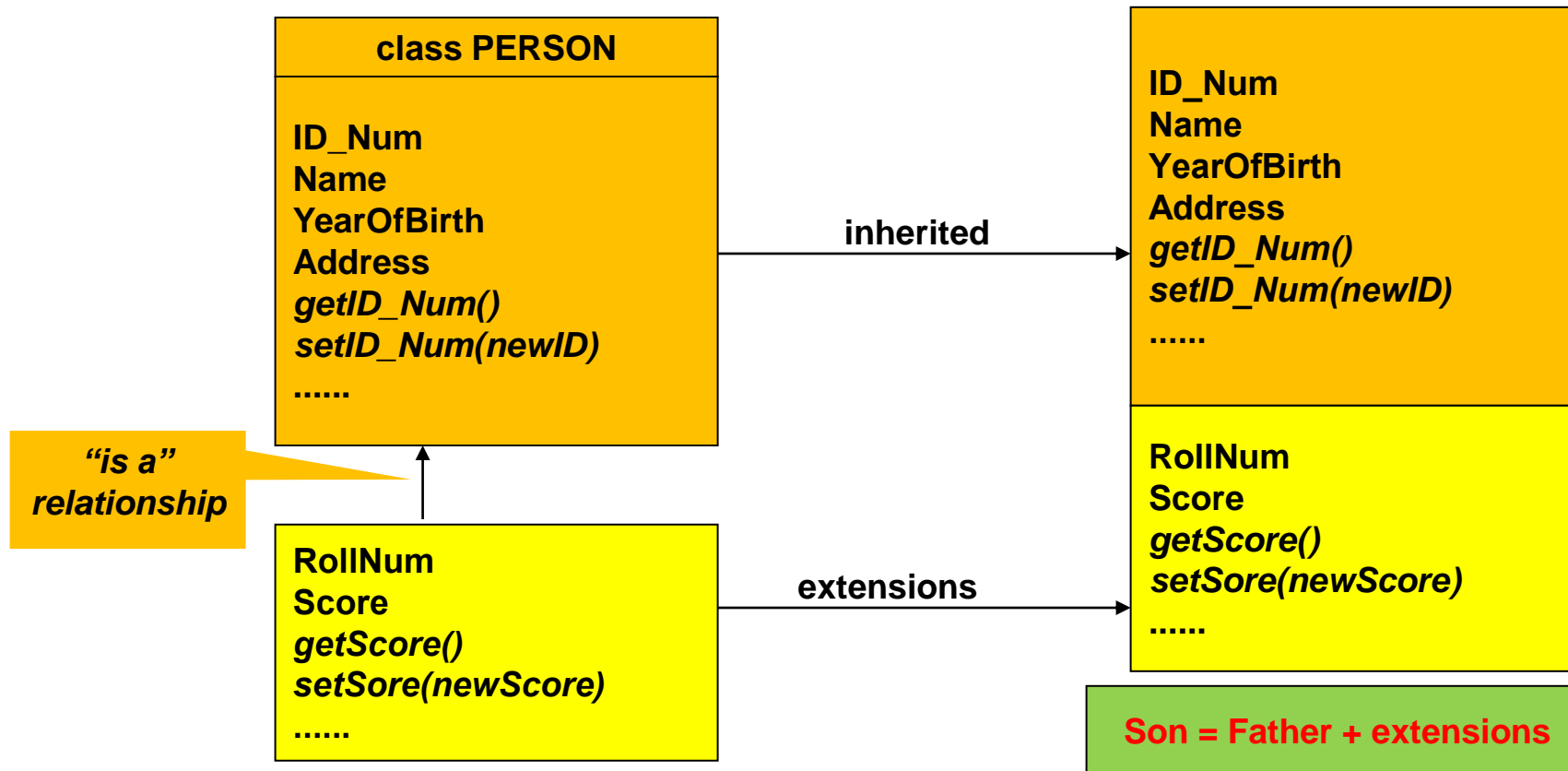
Inheritance (2)
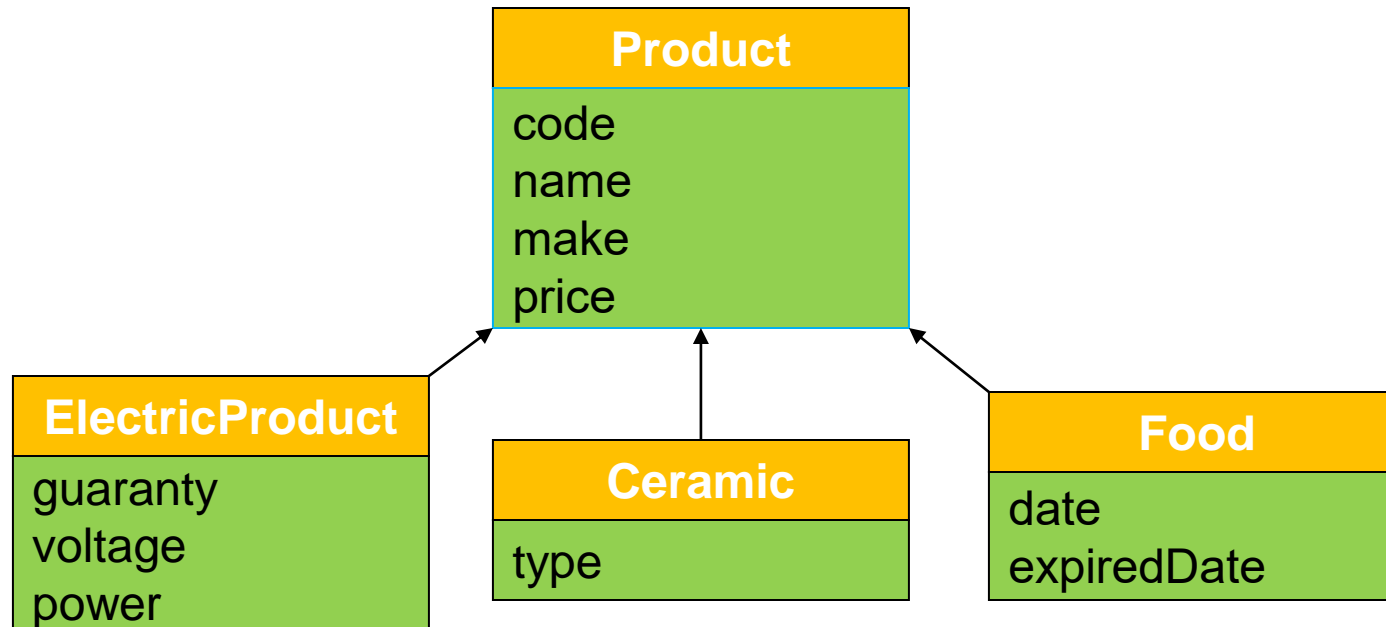
How to detect father class?

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 >

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).

How to Identity a Class

- Main noun: Class

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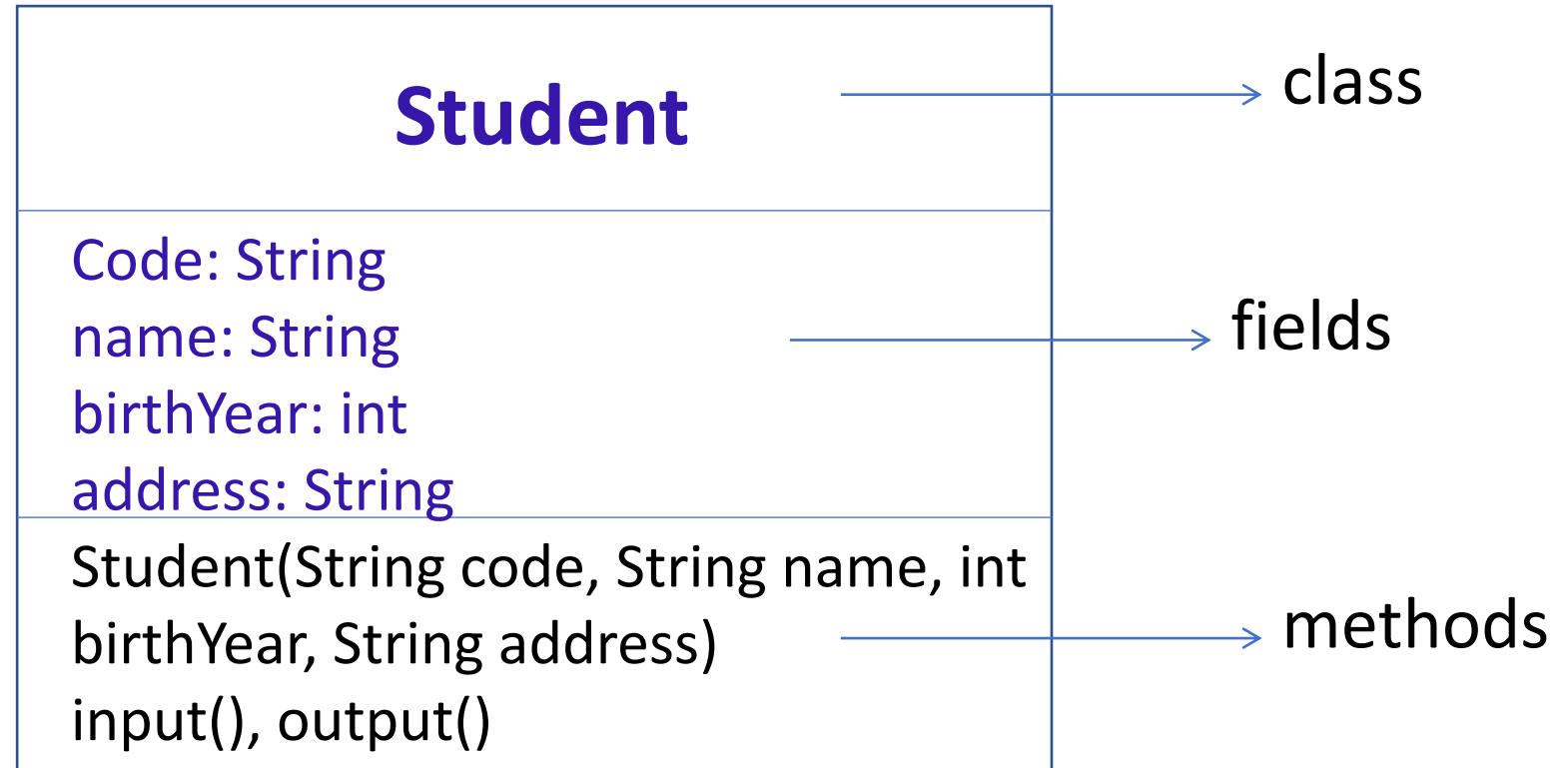
- Verbs related to main noun: Methods

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

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

Hints for class design

A UML class diagram is used to represent the Student class

class

Student

Code: String
name: String
birthYear: int
address: String

fields

Student(String code, String name, int birthYear, String address)
input(), output()

methods

Note: We can add more some functions

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

Constructors (1)

- 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.

- If we implement **a** constructor, compiler does **not** insert default constructor.

Constructors (2)

```
//default constructor 1 or 2
public Student(){}    //1

public Student(){     //2
    code="B19DCCN123";
    name="To An An";
    birthYear= 2000;
    address="1 Ba Trieu , HN".
}

//constructor with parameters
public Student(String code, String name, int
birthYear, String address){
    this.code=code;
    this.name=name;
    this.birthYear= 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

- Member functions are the functions, which have their declaration inside the class definition and work on the data members of the class.

- Typical method declaration:

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

- Signature: data help identifying something

- Method Signature: name + order of parameter types

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

- ```
public String getName(){
      return name;
}
```

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

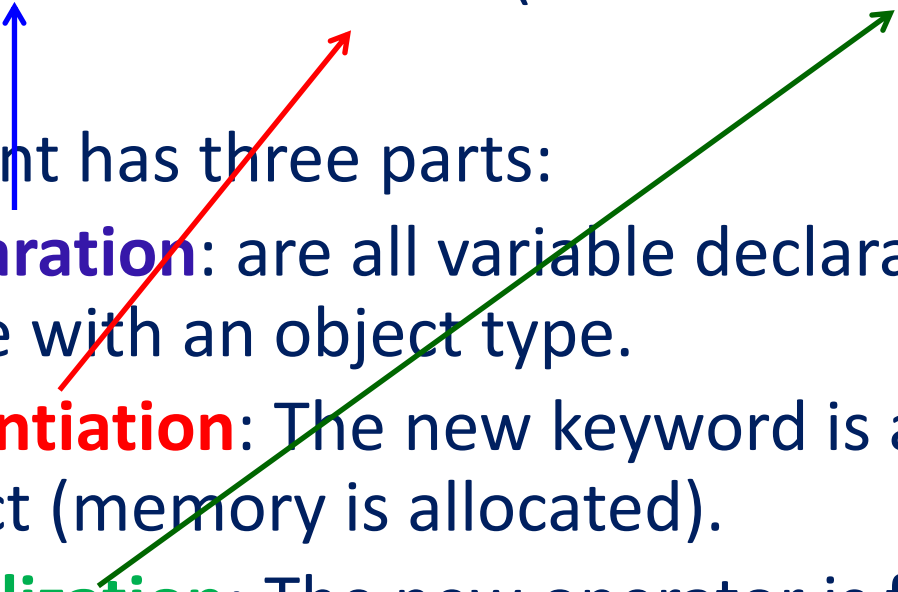# Member functions: other methods

- 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 st = new Student("`B19DCCN123`","To An An",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)

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

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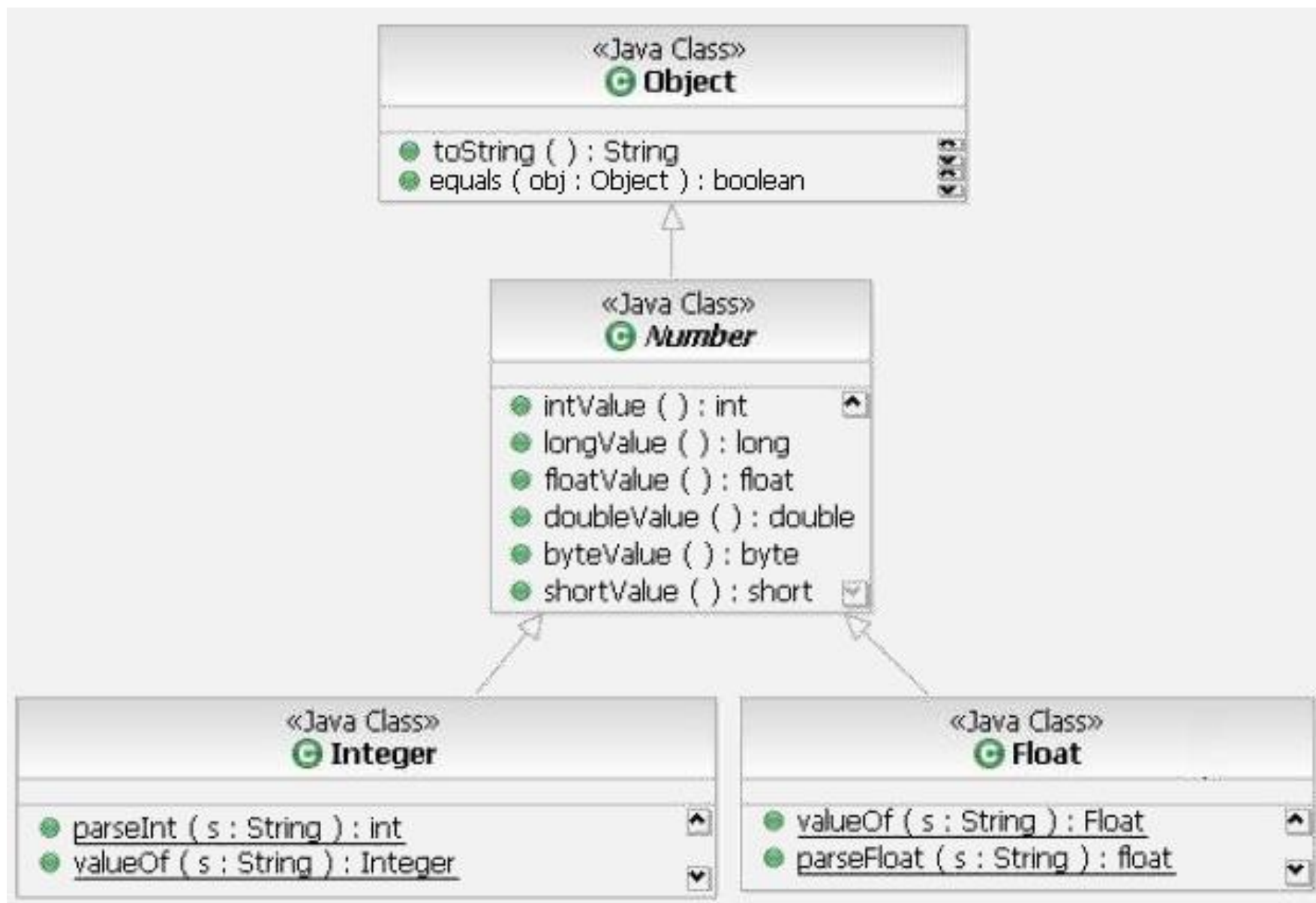
```java
package point1;
public class IntPoint2_Use {
  public static void main (String[] args){
        System.out.println("Use default constructor:");
        IntPoint2 p1= new IntPoint2();
        System.out.println("Use parametric constructor:");
        IntPoint2 p2 = new IntPoint2(-7,90);
    }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

Output - FirstPrj (run) ☒

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

Object class

- The java.lang.Object class is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

- Class constructors: Object() This is the Single Constructor.

- String toString() This method returns a string representation of the object.

- boolean equals(Object obj) This method indicates whether some other object is "equal to" this one.

- int hashCode() This method returns a hash code value for the object.

- Class getClass() This method returns the runtime class of this Object.
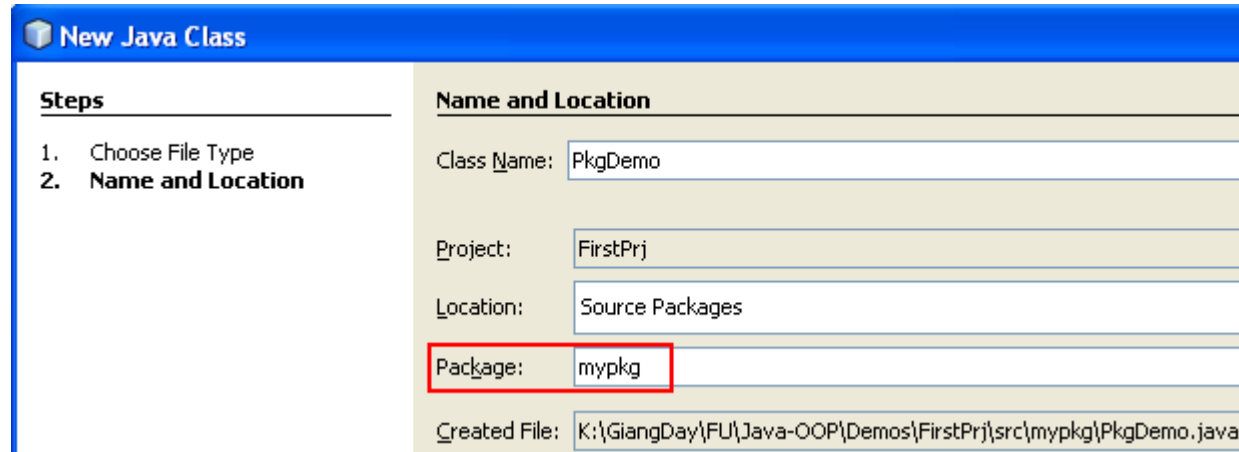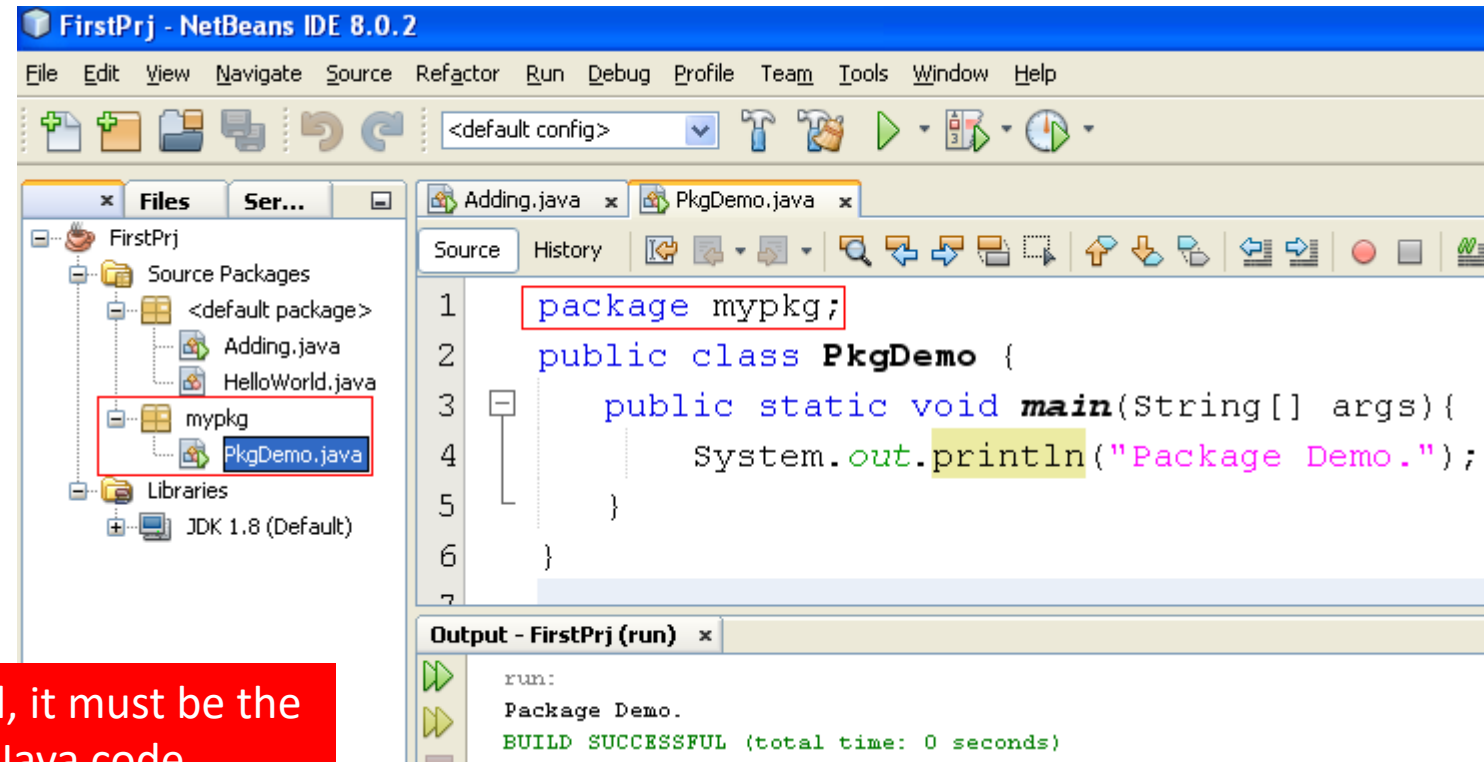
What Is a Package?
User-Defined Package

- A package is a namespace that organizes a set of related classes and interfaces.

- The Java platform provides an enormous class library (a set of packages) suitable for use in your own applications called API.

- For example, a String object contains state and behavior for character strings.

If package is used, it must be the first line in Java code

Access modifiers

- Modifier (linguistics)is a word which can bring out the meaning of other word (adjective for noun, adverb for 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, private).

- Access 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

```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);
    }
}
```

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)

Access Modifiers

Access level

Applied to

Free-accessing

public

interface

package/ subclass

protected

class

outside cannot access

private

members of interface/class

package

no specifier (default)

Interface is a group of prototyped methods and they will be implemented in a class afterward. **It will be introduced later.**
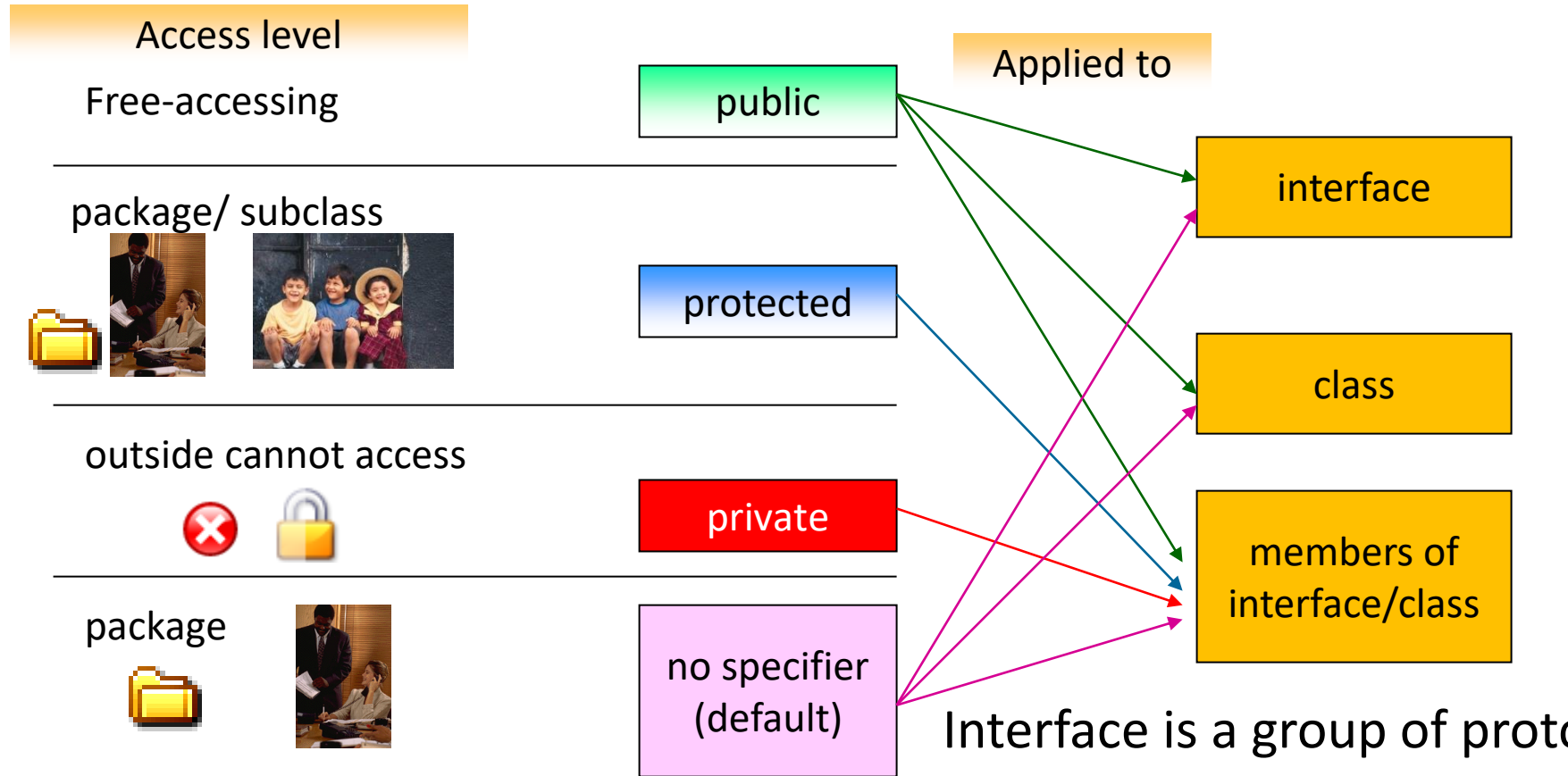
Note: If you don't use any modifier, it is treated as default by default.

Access Level

Modifier	Class	Same Package	Subclass-Outside package	World
private	Y	N	N	N
No (default)	Y	Y	N	N
protected	Y	Y	Y	N
public	Y	Y	Y	Y

Quiz?

Projects — Chapter02
- Source Packages
  - boxPkg
    - Box.java
    - Demo_1.java
  - rectPkg
    - Rectangle.java
- Test Packages
- Libraries
- Test Libraries

Rectangle.java
```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
```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
```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;
9       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;
14      r.length=6;
15    }
16  }
```

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.

Overloading Method

```java
/* 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;
    }
```

```java
    // 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);
    }
}
```

```java
/* 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) x

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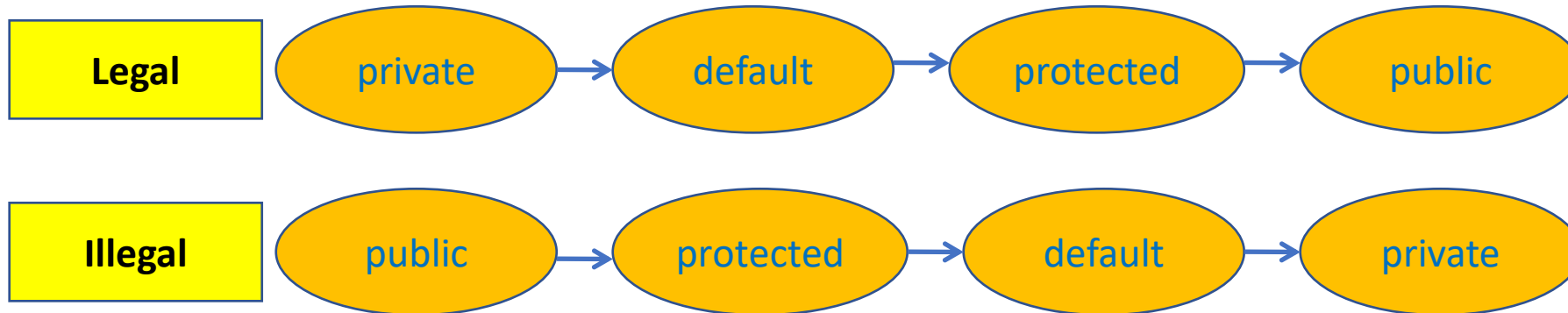
Access Modifier Overridden

The sub-class must be more opened than it's father

Modifier *final*

- final class: Class can not have sub-class

- final data is a constant.

- final method: a method can not be overridden.

Modifier *static* Class variable/ Object variable

- Object variable: Variable of each object
- Class Variable: A variable is shared in all objects of class. It is stored separately. It is declared with the modifier static
- Class variable is stored separately. So, it can be accessed as:

    object.staticVar

    ClassName.staticVar

Modifier *static*: Class variable/ Object variable

```java
public class StaticVarDemo {
    static int N=10; // class variable
    int x, y; // object variable
    public StaticVarDemo(int xx, int yy){
        x= xx; y=yy;
    }
    public void setN( int nn){
        N= nn;
    }
    public void output(){
      System.out.println(N + "," + x + "," + y);
    }
}
```

```java
public class StaticVarDemoUse {
    public static void main(String args[]){
        StaticVarDemo obj1= new StaticVarDemo(5,7);
        StaticVarDemo obj2= new StaticVarDemo(4,6);
        obj1.output();
        obj2.output();
        obj1.setN(9999);
        obj1.output();
        obj2.output();
        System.out.println(StaticVarDemo.N);
    }
}
```

y-=7

1000

x=4

800

1000

obj2   800

N   10 → 9999
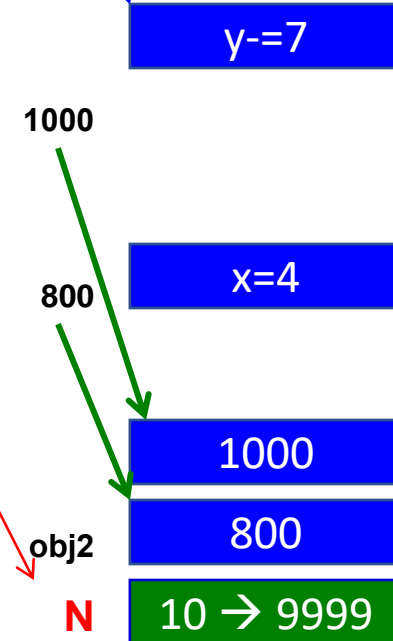
Output - FirstPrj (run) x

run:
10,5,7
10,4,6
9999,5,7
9999,4,6
9999

Static code – Free Floating Block

```java
public class StaticCodeDemo {
    public static int N=10;
    int x=5, y=7;
    static {
        System.out.println("Static code:" + N);
    }

    int sum(){
        return x+y;
    }

    static {
        System.out.println("Static code: Hello");
    }

}
```

All static code run only one time when the first time the class containing them is accesses

```java
public class StaticCodeDemoUse {
    public static void main(String args[]){
        System.out.println(StaticCodeDemo.N);
        StaticCodeDemo obj= new StaticCodeDemo();
        System.out.println(obj.sum());
    }
}
```

The second access

```
Output - FirstPrj (run)   x

run:
Static code:10
Static code: Hello
10
12
```

Static method

- It is called as class method/global method and it is called although no object of this class is created.

- Entry point of Java program is a static method

- Syntax for calling it: ClassName.staticMethod(args)

- Static methods:
  - can access class variables and class methods directly only.
  - cannot access instance variables or instance methods directly—they must use an object reference.
  - cannot use the this keyword as there is no instance for this to refer t

- Constants:
  - The static modifier, in combination with the final modifier, is also used to define constants. The final modifier indicates that the value of this field cannot change.

  static final double PI = 3.141592653589793;

Nested classes :Inner Class

- **Inner classes** are classes defined within other classes
    - The class that includes the inner class is called the **outer class**
    - There is no particular location where the definition of the inner class (or classes) must be place within the outer class
    - Placing it first or last, however, will guarantee that it is easy to find
- An inner class definition is a member of the outer class in the same way that the instance variables and methods of the outer class are members
    - An inner class is local to the outer class definition
    - The name of an inner class may be reused for something else outside the outer class definition
    - If the inner class is private, then the inner class cannot be accessed by name outside the definition of the outer class

Inner/Outer Classes (1)

```java
public class OuterClass {
    private int x;
    public int getX() {
        return x;
    }
    public void setX(int x) {
        this.x = x;
    }
    public class InnerClass{
        private int y;

        public int getY() {
            return y;
        }
        public void setY(int y) {
            this.y = y;
        }
    }
}
```

```java
public class OuterDemo {
    private class InnerDemo{
        public void print() {
            System.out.println("This is an
inner class");
        }
    }
    void display_Inner() {
        InnerDemo inner = new
InnerDemo();
        inner.print();
    }
}
```

Inner/Outer Classes (2)

```java
public class Main {
    public static void main(String[] args) {
        OuterClass o=new OuterClass();
        o.setX(10);
        OuterClass.InnerClass in=o.new InnerClass();
        in.setY(4);
        System.out.println("Out:"+o.getX()+"\n"+in.getY());
        OuterDemo outer = new OuterDemo();
        outer.display_Inner();
    }
}
```

java.lang.Math

- java.lang
- Math.PI
- Math.abs(-20);
- double c = Math.ceil(7.342); // 8.0
- double f = Math.floor(7.843); // 7.0
- double p = Math.pow(2, 3); // 8.0
- double s = Math.sin(Math.PI/2);//1
- double a = Math.sqrt(9);//3
- ……

java.util.Random

- import java.util.Random;
- Random rd=new Random();
- int a=rd.nextInt(n);//0-(n-1)
- int t = rd.nextInt(max-min+1) +a;//(from min to max)
- Float f=rd.nextFloat();// 0-1
- boolean b=r.nextBoolean();

Wrapper classes

- The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.

Primitive Type	Size	Minimum Value	Maximum Value	Wrapper Type
char	16-bit	Unicode 0	Unicode 2^{16}-1	Character
byte	8-bit	-128	+127	Byte
short	16-bit	-2^{15} (-32,768)	$+2^{15}$-1 (32,767)	Short
int	32-bit	-2^{31} (-2,147,483,648)	$+2^{31}$-1 (2,147,483,647)	Integer
long	64-bit	-2^{63} (-9,223,372,036,854,775,808)	$+2^{63}$-1 (9,223,372,036,854,775,807)	Long
float	32-bit	32-bit IEEE 754 floating-point numbers		Float
double	64-bit	64-bit IEEE 754 floating-point numbers		Double
boolean	1-bit	true or false		Boolean
void	-----	-----	-----	Void

Why we need Wrapper Class

- Wrapper Class will convert primitive data types into objects. The objects are necessary if we wish to modify the arguments passed into the method (because primitive types are passed by value).

- The classes in java.util package handles only objects and hence wrapper classes help in this case also.

- Data structures in the Collection framework such as ArrayList and Vector store only the objects (reference types) and not the primitive types.

- The object is needed to support synchronization in multithreading.

Autoboxing and Unboxing

- Since J2SE 5.0, autoboxing and unboxing feature convert primitives into objects and objects into primitives automatically.

- Autoboxing is used to convert primitive data types into corresponding objects.

- Unboxing is used to convert the Wrapper class object into corresponding primitive data types.

Primitive type
boolean
byte
char
float
int
long
short
double

Boxing →

Unboxing ←

Wrapper class
Boolean
Byte
Character
Float
Integer
Long
Short
Double
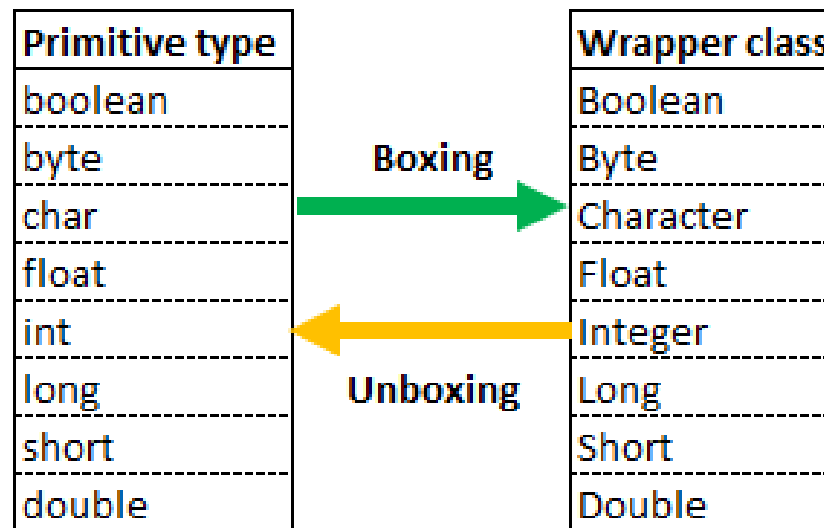
Example

- `Integer y = 567; // //boxing or auto boxing`
- `Double area(double radius) {`

  `    return Math.PI * radius * radius; //boxing`

  `    }`
- `Integer wi = 234;`

  `    int times9 = wi * 9; //unboxing`

Return primitive types

- To return primitive types: using method **type**Value()

```
// make a new wrapper object
Integer i2 = new Integer(42);
// byte
byte b = i2.byteValue();
// short
short s = i2.shortValue();
// double
double d = i2.doubleValue();
```

to convert from String to a primitive type

- Using methods (static) of wrapper classes

  ```
  static <type> parseType(String s)
  ```

- Example

  ```
  String s = "123";
  // int
  int i = Integer.parseInt(s);
  // short
  short j = Short.parseShort(s);
  String txt="13.5";
  Double x =Double.parseDouble(txt);
  ```

Java.lang.Integer Class

1. static int MAX_VALUE: 231-1.

2. static int MIN_VALUE : -231.

3. Integer(int value), Integer(String s)

4. java.lang.Integer.compare()

5. Integer obj1 = new Integer("25"); Integer obj2 = new Integer("10");

6. int retval = obj1.compareTo(obj2);

7. int retval = Integer.compare(obj1,obj2);

8. byte byteValue(), double doubleValue(), float floatValue()...

9. static int parseInt(String s), String toString(), static String toString(int i)

10. boolean equals(Object obj)

Java.lang.Double Class

- Double(double value), Double(String s)
- byte byteValue(), double doubleValue(), float floatValue(), int intValue(), ..
- static int compare(double d1, double d2), int compareTo(Double anotherDouble)
- boolean isNaN(),
- static double parseDouble(String s),
- String toString(), static String toString(double d)

java.lang.Character Class

- static char toUpperCase(char ch)

- static char toLowerCase(char ch)

- static String toString(char c)

- String toString()

- static boolean isWhitespace(char ch)

- static boolean isUpperCase(char ch)

- static boolean isLowerCase(char ch)

- static boolean isLetter(char ch)

Problem with .nextLine() in Java

- ```java
  import java.util.Scanner;
  ```
  ```java
  Scanner input = new Scanner( System.in );
  ```
- Solution:

```java
int option = input.nextInt();
input.nextLine(); // Consume newline left-over
String str1 = input.nextLine();
int option = Integer.parseInt(input.nextLine());
```

Case study (1)

- Create a new project named "StudentManager". It contains the file Student.java and Main.java.

- In the file Student.java, you implement the Student class base on the class diagram as below:

Student
- **String code**
- **String fullname**
- **boolean gender**
- **double mark**
- **Student()**
- **Student(String code)**
- **Student (String code,String fullname,boolean gender,double mark)**
- **String getCode(),void setCode()**
- **....**
- **String toString()**
- **void input()**

Main.java

- The program output might look something like:

1. TC = 1 – input a student

2. TC = 2 – Edit a student

3. TC = 3 – toUpper fullname

4. TC = 3 – set mark

5. TC = 4 - toString()

0. TC = 0 - Exit

Enter TC:

Case study (2)

- https://code.ptit.edu.vn/student/question (from 65 to 73, from 150 to 155, from 157 to 160)

- https://codelearn.io/learning/object-oriented-programming-in-java/766304