5. Defining Classes and Methods

[ITP20003] Java Programming

Agenda

- **Class and Method Definitions**
- Information Hiding and Encapsulation
- Objects and References

Class and Method Definitions

- Java program consists of objects
  - Objects of class types
  - Objects that interact with one another
- Program objects can represent
  - Objects in real world
  - Abstractions

Class and Method Definitions

Ex) A class as a blueprint

Class Name: Automobile

Data:

amount of fuel_____

speed _____

license plate _____

Methods (actions):

accelerate:

How: Press on gas pedal.

decelerate:

How: Press on brake pedal.

Class and Method Definitions

Class Name: Automobile
Data:
amount of fuel_____
speed _____
license plate _____
Methods (actions):
accelerate: How: Press on gas pedal.
decelerate: How: Press on brake pedal.

amount of fuel: 10 gallons
speed: 55 miles per hour
license plate: "135 XJK"

Object name: patsCar

amount of fuel: 14 gallons
speed: 0 miles per hour
license plate: "SUES CAR"

Object name: suesCar

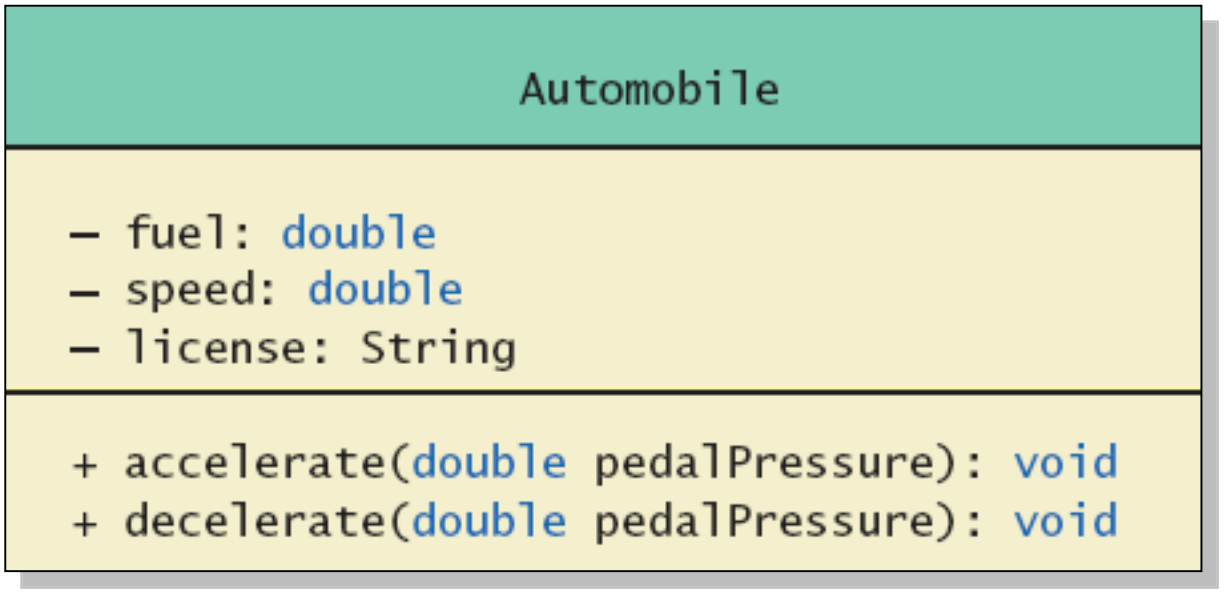
amount of fuel: 2 gallons
speed: 75 miles per hour
license plate: "351 WLF"

Object name: ronsCar

Class and Method Definitions

- A class outline as a UML class diagram

  cf. UML: Unified Modeling Language



Class Files and Separate Compilation

- Each Java class definition usually in a file
  - The filename should be *ClassName*.java
- Class can be compiled separately
  - Helpful to keep all class files used by a program in the same directory

Dog class and Instance Variables

```
public class Dog
{
    public String name;
    public String breed;
    public int age;
    public void writeOutput()
    {
        System.out.println("Name: " + name);
        System.out.println("Breed: " + breed);
        System.out.println("Age in calendar years: " + age);
        System.out.println("Age in human years: " + getAgeInHumanYears());
        System.out.println();
    }
    public int getAgeInHumanYears()
    {
        int humanAge = 0;
        if (age <= 2)
                humanAge = age * 11;
        else
                humanAge = 22 + ((age-2) * 5);

        return humanAge;
    }
}
```

Dog class and Instance Variables

- View sample program, listing 5.1
- The Dog class has
  - Three pieces of data (instance variables)
  - Two behaviors (methods)
- Each instance of this type has its own copies of the data items.
- Use of public
  - No restrictions on how variables used
  - Can be replaced with private

Java Access Modifiers

	public	protected	default	private
same class	O	O	O	O
same package	O	O	O	
derived classes	O	O		
other	O			

DogDemo

```
public class DogDemo
{
    public static void main(String[] args)
    {
        Dog balto = new Dog();
        balto.name = "Balto";
        balto.age = 8;
        balto.breed = "Siberian Husky";
        balto.writeOutput();

        Dog scooby = new Dog();
        scooby.name = "Scooby";
        scooby.age = 42;
        scooby.breed = "Great Dane";
        System.out.println(scooby.name + " is a " + scooby.breed + ".");
        System.out.print("He is " + scooby.age + " years old, or ");
        int humanYears = scooby.getAgeInHumanYears();
        System.out.println(humanYears + " in human years.");
    }
}
```

Use . to access variables and methods in the class.

Main method

Use new

Output

```
Name: Balto
Breed: Siberian Husky
Age in calendar years: 8
Age in human years: 52

Scooby is a Great Dane.
He is 42 years old, or 222 in human years.
```

Instance variables

- Data defined in the class are called *instance variables*.

Public
no restrictions on how these instance variables are used.

```
public class Dog
{
    public String name;
    public String breed;
    public int age;
    public void writeOutput()
    {
        …
    }
}
```

Data type: int, double, String

Methods

- When you use a method you "invoke" or "call" it
- Two kinds of Java methods
  - Return a single item
    - Use anywhere a value can be used
  - Perform some other action – a void method
    - Resulting statement performs the action defined by the method
- The method *main*

  public static void main(String[] args)
  - A void method
  - Invoked by the system

Methods

- Basic Syntax

Access_type Return_type Method_Name (input_type input_name)

- Example

```
public int getAgeInHumanYears()
public void writeOutput()
public int addition(int a, int b)
theNextInteger = keyboard.nextInt();
```

Defining Methods

- Method definitions appear inside class definition
  - Can be used only with objects of that class

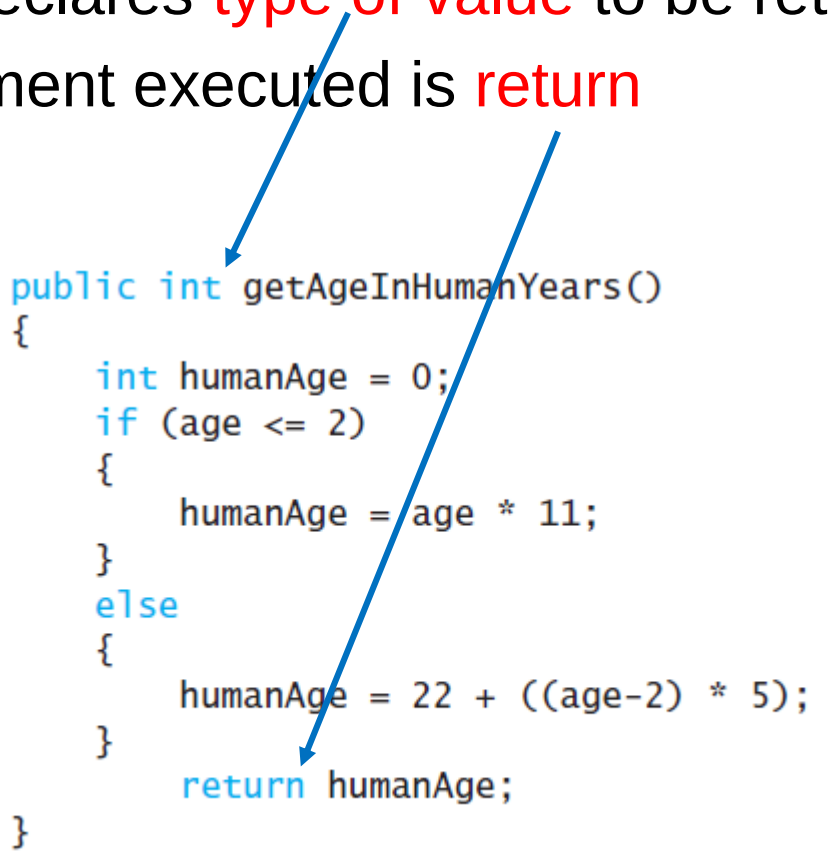
```
public void writeOutput()
{
    System.out.println("Name: " + name);
    System.out.println("Breed: " + breed);
    System.out.println("Age in calendar years: " +
                       age);
    System.out.println("Age in human years: " +
                       getAgeInHumanYears());
    System.out.println();
}
```

Defining Methods

- Most method definitions we will see as public
  - Void method does not return a value
- Head
  - Method name + parameters
- Body
  - Enclosed in braces { }
  - Think of method as defining an action to be taken

Methods That Return a Value

- Heading declares type of value to be returned
- Last statement executed is return



```
public int getAgeInHumanYears()
{
    int humanAge = 0;
    if (age <= 2)
    {
        humanAge = age * 11;
    }
    else
    {
        humanAge = 22 + ((age-2) * 5);
    }
        return humanAge;
}
```

Methods That Return a Value

- Use One return statement.

```
if (age <= 2)
        return age * 11;
else
        return 22 + ((age-2) * 5);
```

```
int output
if (age <= 2)
        output = age * 11;
else
        output = 22 + ((age-2) * 5);

return output;
```

The Keyword *this*

- Referring to instance variables outside the class

  Syntax) *ObjectName*.*VariableName*

- Referring to instance variables inside the class
  - Use *VariableName* alone
    - The object (unnamed) is understood to be there.

- Inside the class the unnamed object can be referred to with the name *this*

  Ex) this.name = keyboard.nextLine();
  - The keyword *this* stands for the receiving object

Local Variables

- Variables declared inside a method are called local variables
  - May be used only inside the method
  - All variables declared in method *main* are local to *main*
- Local variables having the same name and declared in different methods are different variables

Local Variables

- A variable declared within a method definition is called a local variable. One method's local variables have no meaning within another method. Moreover, if two methods each have a local variable with the same name, they are considered two different variables.

- Instance variables
  - Declared in a class.
  - Confined to the class.
  - Can be used in any methods in the class.

- Local variables
  - Declared in a method.
  - Confined to a method.
  - Can only be used inside the method.

BankAccount

```
public class BankAccount
{
    public double amount;
    public double rate;
    public void showNewBalance ()
    {
        double newAmount = amount + (rate / 100.0) * amount;
        System.out.println ("With interest added, the new amount is $“
                                                    + newAmount);
    }
}
```

newAmount is a local variable.

LocalVariablesDemoProgram

```
public class LocalVariablesDemoProgram
{
    public static void main (String [] args)
    {
        BankAccount myAccount = new BankAccount ();
        myAccount.amount = 100.00;
        myAccount.rate = 5;
        double newAmount = 800.00;
        myAccount.showNewBalance ();
        System.out.println ("I wish my new amount were $” + newAmount);
    }
}
```

```
With interest added, the new amount is $105.0
I wish my new amount were $800.0
```

Blocks

- Blocks or compound statements
  - Statements enclosed in braces { }
- When you declare a variable within a compound statement
  - The scope of the variable is from its declaration to the end of the block
- Variable declared outside the block usable both outside and inside the block

Parameters of Primitive Type

```
class MyCalculator {
      …
    public int addition (int a, int b)
       {
           int result = 0;
           result = a + b;

           return result;
       }
}
```

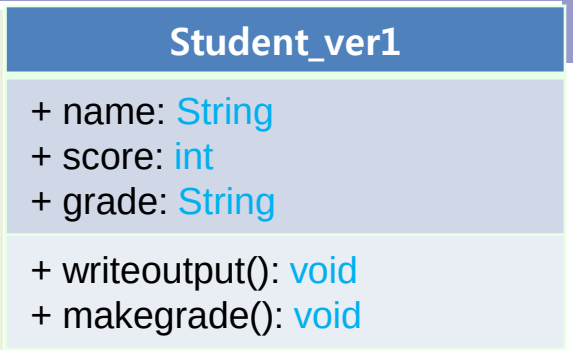
Parameters of Primitive Type

- Parameter names are local to the method
- When method invoked
  - Each parameter initialized to value in corresponding actual parameter
  - Primitive actual parameter cannot be altered by invocation of the method
- Automatic type conversion performed

```
byte -> short -> int ->
        long -> float -> double
```

Example: Student Class

```
public class Student_main_ver1 {
public static void main(String[] args) {
    Student_ver1 sman = new Student_ver1();
    int class_score = 80;
    sman.name = "SuperMan";
    sman.score = class_score;
    sman.makegrade();
    sman.writeoutput();
}
}
```



Output

```
SuperMan: 80: pass
```

```
public class Student_ver1 {
    public String name;
    public int score;
    public String grade;
    public void writeoutput() {
        System.out.println(name + ": " + score + ": " + grade);}
    public void makegrade() {
        int class_score;
        class_score = score;
        if (class_score > 50)    grade = "pass";
        else                     grade = "fail";
}}
```

Example: using the Keyword this

```
public void makegrade() {
int class_score;
class_score = score;
if (class_score > 50)
        grade = "pass";
else
        grade = "fail";
}
```

```
public void makegrade() {
int class_score;
class_score = this.score;
if (class_score > 50)
        this.grade = "pass";
else
        this.grade = "fail";
}
```

Example: Using public Variables outside the Class Definition

- Because ‘name’ and ‘score’ are declared as ‘public’, it can be accessed outside the class.

```
public class Student_main_ver1 {
    public static void main(String[] args) {
        Student_ver1 sman = new Student_ver1();
        int class_score = 80;
        sman.name = "SuperMan";
        sman.score = class_score;
        sman.makegrade();
        sman.writeoutput();
    }
}
```

- The methods (makegrade() and writeoutput()) are also ‘public’.

Example: Using Variables outside/Inside the Class Definition

- Instance variables (score, grade) are being used inside of the methods.

```
public void writeoutput() {
       System.out.println(name + ": " + score + ": " + grade);
}
public void makegrade() {
int class_score;
class_score = score;
if (class_score > 50)
        grade = "pass";
else
        grade = "fail";
}
```

Example: Local Variables

- They are different variables.
- They can be used only within the method where each one was made.

```
public class Student_main_ver1 {
    public static void main(String[] args) {
        Student_ver1 sman = new Student_ver1();
        int class_score = 80;
        sman.name = "SuperMan";
        sman.score = class_score;
        sman.makegrade();
        sman.writeoutput();
    }
}
```

```
public void makegrade() {
int class_score;
class_score = score;
if (class_score > 50)
        grade = "pass";
else
        grade = "fail";
}
```

Agenda

- Class and Method Definitions
- **Information Hiding and Encapsulation**
- Objects and References

Information Hiding

- Programmer using a class method need NOT know details of implementation
  - Only needs to know *what* the method does
- Information hiding
  - Designing a method so it can be used without knowing details
  - Also referred to as *abstraction*
- Method design should separate *what* from *how*

Pre- and Postcondition Comments

- Well defining what and how for your design of the method (or S/W) helps us to write well-organized/encapsulated code.

- If you defined what and how then write precondition and postcondition first, then write the body of the method.

- Example

```
/**
   Precondition: score should be set.
   Postcondition: grade is set based on score.
*/
public void makegrade() {
…
}
```

Pre- and Postcondition Comments

- Precondition comment
  - States conditions that must be true before method is invoked

```
/**
 Precondition: The instance variables of the calling
 object have values.
 Postcondition: The data stored in (the instance variables
 of) the receiving object have been written to the screen.
*/
public void writeOutput()
```

- Postcondition comment
  - Tells what will be true after method executed

```
/**
 Precondition: years is a nonnegative number.
 Postcondition: Returns the projected population of the
 receiving object after the specified number of years.
*/
public int predictPopulation(int years)
```

The *public* and *private* Modifiers

- Type specified as public
  - Any other class can directly access that object by name
  - Classes generally specified as *public*
- Instance variables usually not *public*
  - Instead specify as *private*
- View sample code, listing 5.8
  class SpeciesThirdTry

Programming Example

```
public class Rectangle
{
    private int width;
    private int height;
    private int area;
    public void setDimensions (int newWidth, int newHeight)
    {
        width = newWidth;
        height = newHeight;
        area = width * height;
    }
    public int getArea ()
    {
        return area;
    }
}
```

➔ Statement such as “box.width = 6;” is illegal.

Programming Example

```
public class Rectangle2
{
    private int width;
    private int height;

    public void setDimensions (int newWidth, int newHeight)
    {
        width = newWidth;
        height = newHeight;
    }

    public int getArea ()
    {
        return width * height;
    }
}
```

setDimensions() method is the only way the width and height may be altered outside the class.

Accessor and Mutator Methods

- When instance variables are private must provide methods to access values stored there
  - Typically named *getSomeValue()*
  - Referred to as an accessor method
- Must also provide methods to change the values of the private instance variable
  - Typically named *setSomeValue()*
  - Referred to as a mutator method

Accessor and Mutator Methods

- Consider an example class with accessor and mutator methods
  - View sample code, listing 5.11
  - Note the mutator method
    - setSpecies()
  - Note accessor methods
    - getName(), getPopulation(), getGrowthRate()

Programming Example

- View sample code, listing 5.13, class Purchase
  - Note use of private instance variables
  - Note also how mutator methods check for invalid values

```
public void setPrice (int count, double costForCount)
{
    if ((count <= 0) || (costForCount <= 0))
    {
        System.out.println ("Error: Bad parameter in setPrice.");
        System.exit (0);
    }  else {
        groupCount = count;
        groupPrice = costForCount;
    }
}
```

Methods Calling Methods

- A method body may call any other method
  - If the invoked method is within the same class, object name can be omitted.

- View sample code, listing 5.15
  class Oracle
  - chat() is *public*, but other methods are *private*
  - chat() calls answer();
  - answer() calls seekAdvice()

- View demo program, listing 5.16
  class OracleDemo
  - main() of OracleDemo class calls the chat() method of Oracle class

Encapsulation

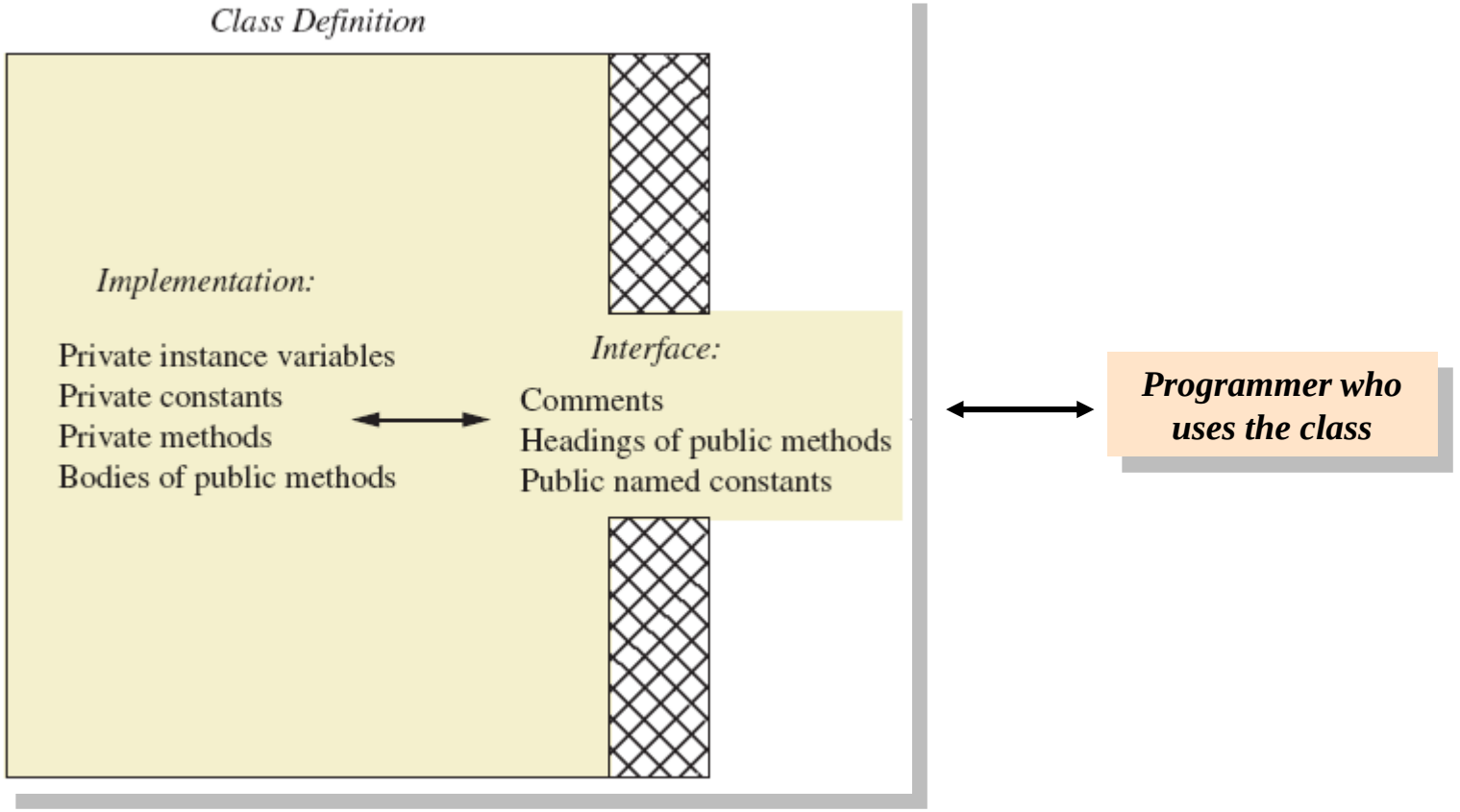
- Consider example of driving a car
  - We see and use break pedal, accelerator pedal, steering wheel – know what they do
  - We do not see mechanical details of how they do their jobs
- Encapsulation divides class definition into
  - Class interface
  - Class implementation

Encapsulation

- A *class interface*
  - Tells what the class does
  - Gives headings for public methods and comments about them
- A *class implementation*
  - Contains private variables
  - Includes definitions of public and private methods

Encapsulation

- A well encapsulated class definition



Encapsulation

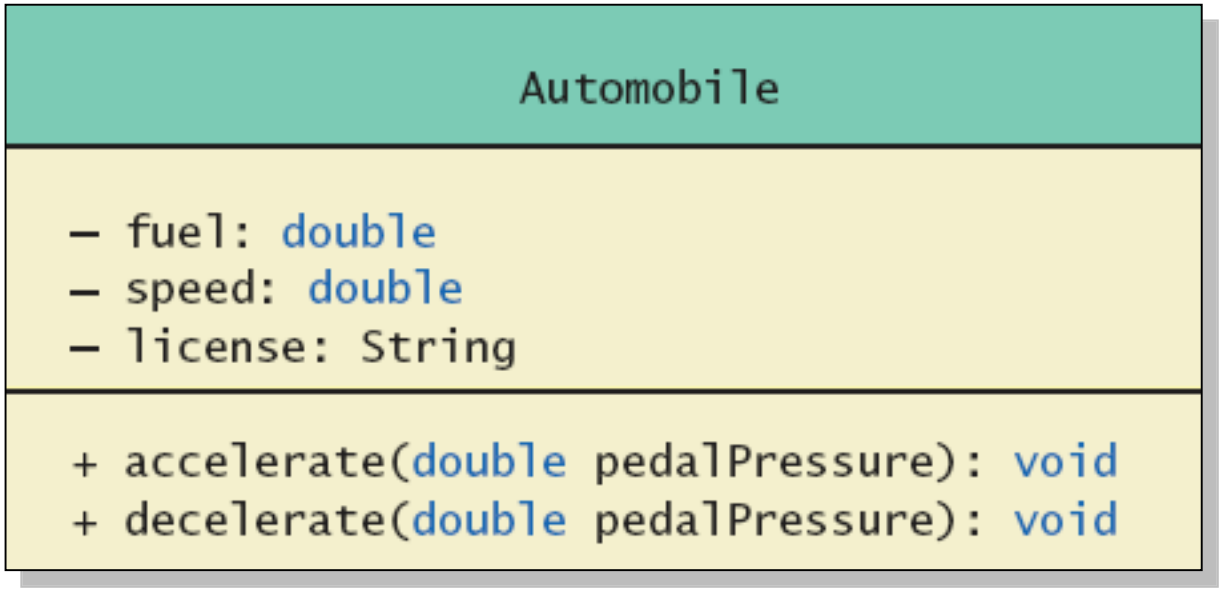
- Preface class definition with comment on how to use class
- Declare all instance variables in the class as private.
- Provide public accessor methods to retrieve data
- Provide public methods manipulating data
  - Such methods could include public mutator methods.
- Place a comment before each public method heading that fully specifies how to use method.
- Make any helping methods private.
- Write comments within class definition to describe implementation details.

Automatic Documentation *javadoc*

- Generates documentation for class interface
- Comments in source code must be enclosed in /** */
- Utility *javadoc* will include
  - These comments
  - Headings of public methods
- Output of *javadoc* is HTML format

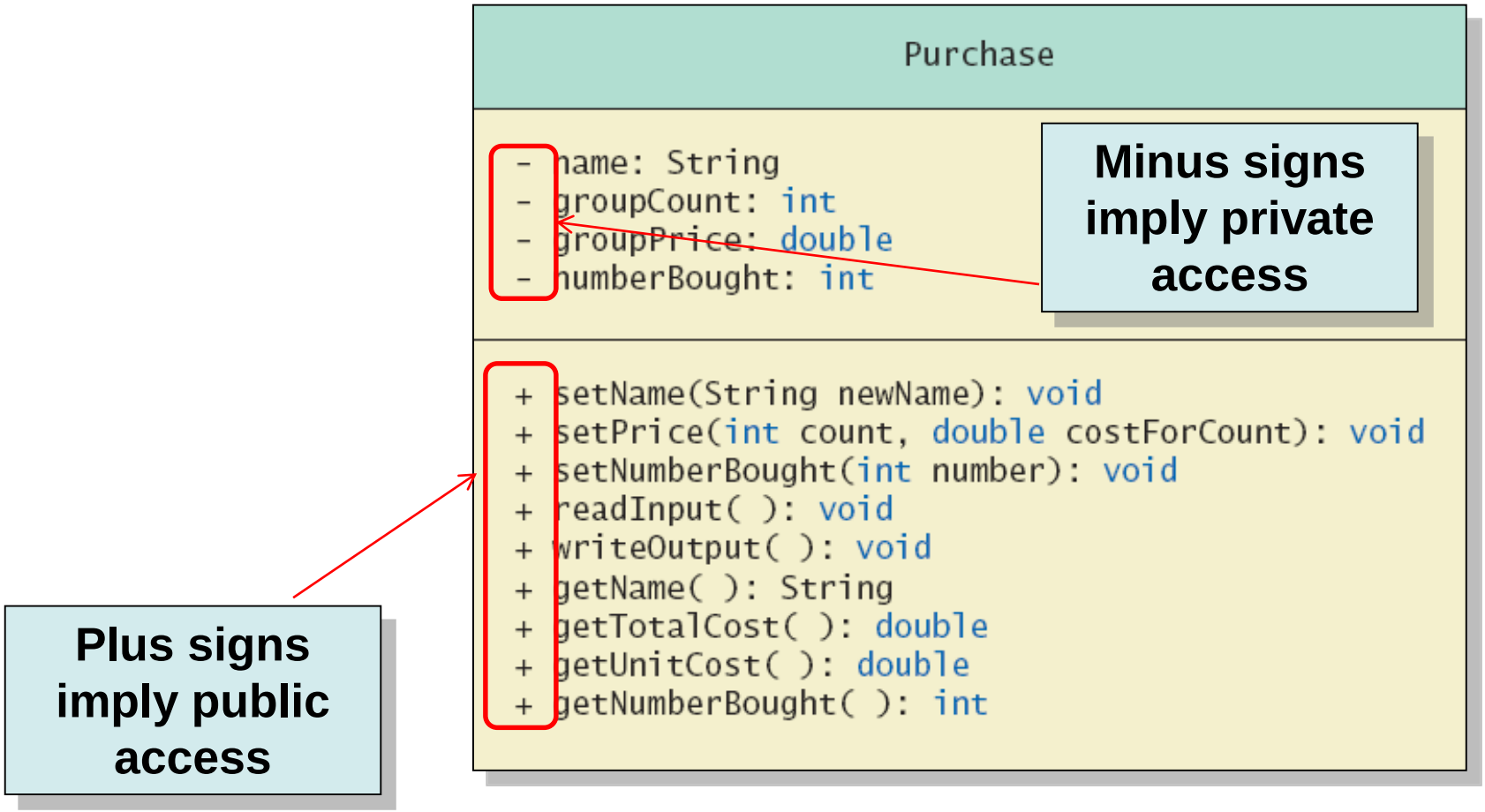
UML Class Diagrams

- A class outline as a UML class diagram



UML Class Diagrams

- The Purchase class



UML Class Diagrams

- Contains more than interface, less than full implementation
- Usually written before class is defined
- Used by the programmer defining the class
  - Contrast with the interface used by programmer who uses the class

Example: Student_ver1

```
public class Student_ver1 {
public String name;
public int score;
public String grade;
public void writeoutput() {
       System.out.println(name + ": " + score + ": " + grade);}

public void makegrade() {
    int class_score;
    class_score = score;
    if (class_score > 50)    grade = "pass";
    else                     grade = "fail";}
}
```

Example:
Accessor Methods and Mutator Methods

Student_ver1
+ name: String + score: int + grade: String
+ writeoutput(): void + makegrade(): void

Student_ver2
- name: String - score: int
+ writeoutput(): void + setdata(String s_name, int s_score): void + getName(): String + getScore(); int

+ public

- private

```
public class Student_ver2 {
private String name;
private int score;

public void writeoutput() {
        String grade;
        if (score > 50)
                grade = "pass";
        else
                grade = "fail";
        System.out.println(name + ": " + score + ": " + grade);
}

public int getScore()       {return score;}
public String getName()     {return name;}

public void setdata(String s_name, int s_score){
        name = s_name;
        score = s_score;
}
}
```

Please write Student_ver2.java and Student_main_ver2.java that makes same result.

```
SuperMan: 80: pass
```

Example:
Accessor Methods and Mutator Methods

```
public class Student_main_ver1 {
public static void main(String[] args) {
Student_ver1 sman = new Student_ver1();
int class_score = 80;
sman.name = "SuperMan";
sman.score = class_score;
sman.makegrade();
sman.writeoutput();
}
}
```

- **Same result.**

```
SuperMan: 80: pass
```

```
public class Student_main_ver2 {
public static void main(String[] args) {
Student_ver2 sman = new Student_ver2();
int class_score = 80;
sman.setdata("SuperMan", class_score);
sman.writeoutput();
}
}
```