Classes and Objects

Ms. Layal Kazma

Learning Outcomes

- How to declare a class and use it to create an object.
- How to implement a class's behaviors as methods.
- How to implement a class's attributes as instance variables and properties.
- How to call an object's methods to make them perform their tasks.
- What instance variables of a class and local variables of a method are.
- How to use a constructor to initialize an object's data.
- Encapsulation and data hiding (using private access specifier and data validation).
- To use keyword this.
- To use static variables and methods.
- To import static members of a class.

Class and Method Definitions

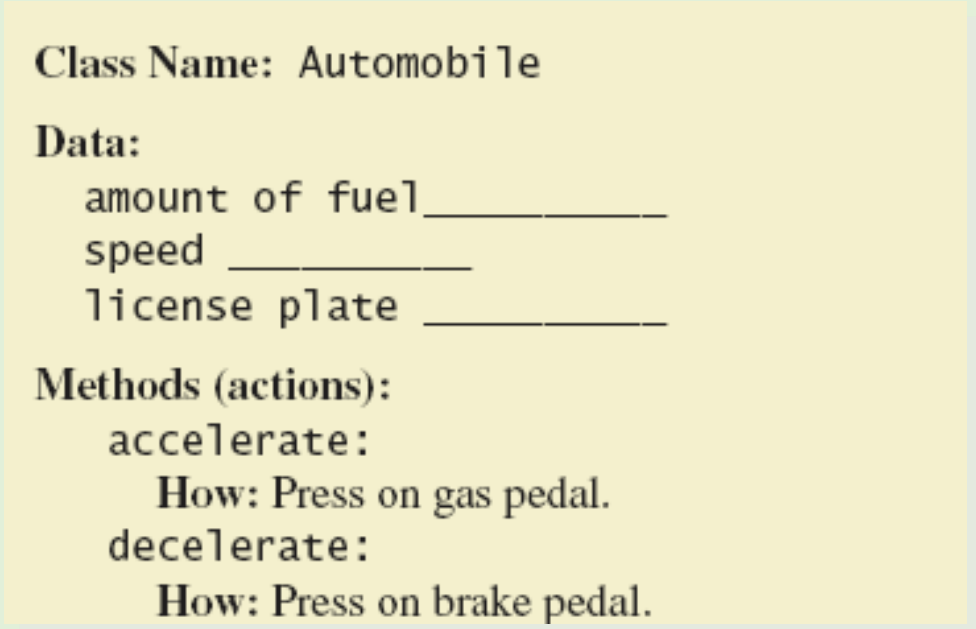
- Java program consists of objects
  - Objects of class types
  - Objects that interact with one another
- Program objects can represent
  - Objects in real world
  - Abstractions

Class

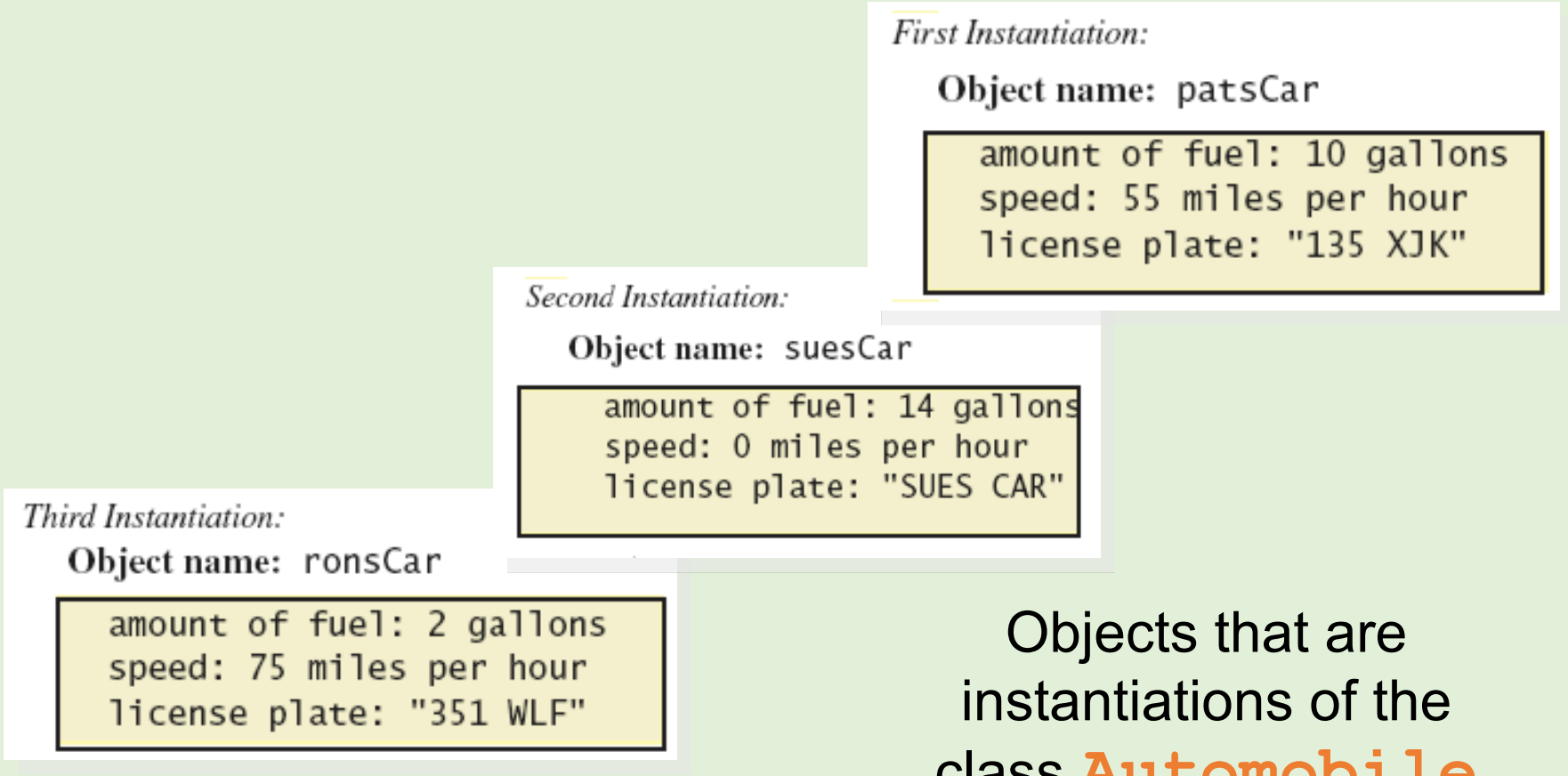
- A class is the definition of a kind of object.
- It is like a plan or a blueprint for constructing specific objects.

Class and Method Definitions

- Figure 5.1 A class as a blueprint



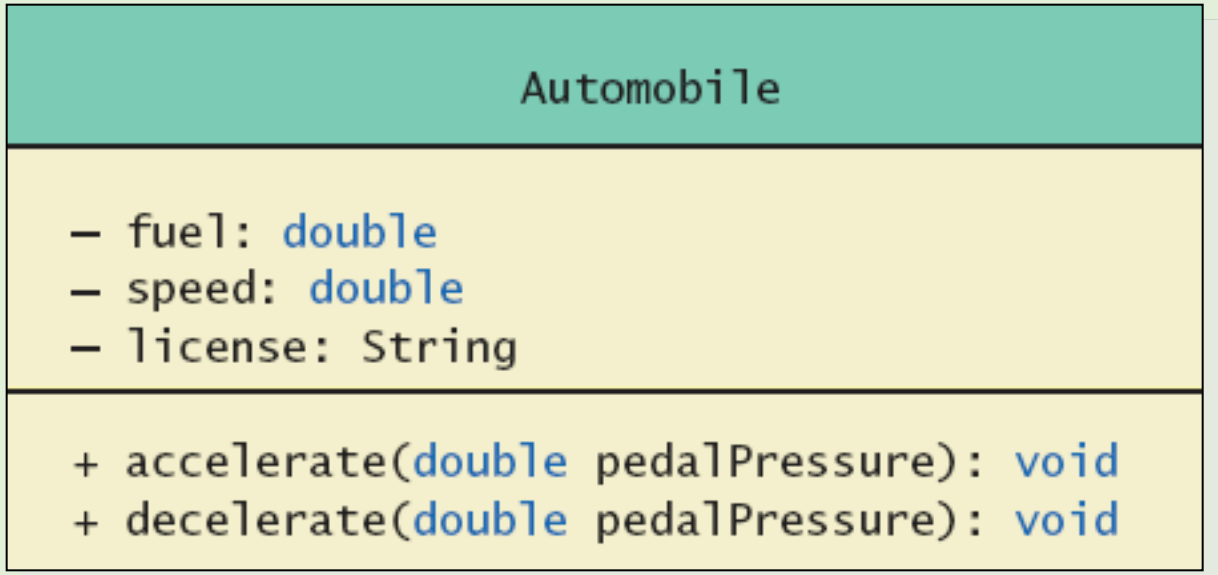
Class and Method Definitions



Objects that are instantiations of the class **`Automobile`**

Class and Method Definitions

- A class outline as a UML class diagram



Book Reading - Walter Savitch book ch 5 (5.1)

5.1 CLASS AND METHOD DEFINITIONS

The greatest invention of the nineteenth century was the invention of the method of invention.

ALFRED NORTH WHITEHEAD, SCIENCE AND THE MODERN WORLD

A Java program consists of objects of various class types, interacting with one another. Before we go into the details of how you define your own classes and objects in Java, let's review and elaborate on what we already know about classes and objects.

Objects in a program can represent real-world things or abstractions

Objects in a program can represent either objects in the real word—like automobiles, houses, and employee records—or abstractions like colors, shapes, and words. A class is the definition of a kind of object. It is like a plan or a blueprint for constructing specific objects. For example, Figure 5.1 describes a class called `Automobile`. The class is a general description of what an automobile is and what it can do.

An instance of a class is an object

Objects of this class are particular automobiles. The figure shows three `Automobile` objects. Each of these objects satisfies the class definition of an `Automobile` object and is an **instance** of the `Automobile` class. Thus, we can create, or **instantiate**, several objects of the same class. The objects here are individual automobiles, while the `Automobile` class is a generic description of what an automobile is and does. This is, of course, a very simplified version of an automobile, but it illustrates the basic idea of what a class is. Let's look at some details.

A class is like a blueprint for creating objects

A class specifies the attributes, or data, that objects of the class have. The `Automobile` class definition says that an `Automobile` object has three attributes or pieces of data: a number telling how many gallons of fuel are in the fuel tank, another number telling how fast the automobile is moving, and a string that shows what is written on the license plate. The class definition has no data—that is, no numbers and no string. The individual objects have the data, but the class specifies what kind of data they have.

A class specifies an object's attributes and defines its behaviors as methods

The class also specifies what actions the objects can take and how they accomplish those actions. The `Automobile` class specifies two actions: `accelerate` and `decelerate`. Thus, in a program that uses the class `Automobile`, the only actions an `Automobile` object can take are `accelerate` and `decelerate`. These actions are described within the class by methods. All objects of any one class have the same methods. In particular, all objects of the class `Automobile` have the same methods. As you can see in our sample `Automobile` class, the definitions of the methods are given in the class definition and describe how objects perform the actions.

Use a UML class diagram to help design a class

The notation in Figure 5.1 is a bit cumbersome, so programmers often use a simpler graphical notation to summarize some of the main properties of a class. This notation, illustrated in Figure 5.2, is called a **UML class diagram**, or simply a **class diagram**. **UML** is an abbreviation for **Universal Modeling Language**. The class described in Figure 5.2 is the same as the one described in Figure 5.1. Any annotations in Figure 5.2 that are new will be explained later in the chapter.

Class

- A class is an entity that contains **attributes** and **methods**. Attributes or **instance variables** represent a characteristic of the entity. The methods represent **operations** on these attributes.
  - The general declaration and creation of an object in Java is in the form

  ClassName ObjectName = new ClassName();

  Example:
  Student student1= new Student();

Class Student

- Attributes:
  - firstName;
  - lastName;
  - phone;
  - gpa;
  - address;
- Methods
  - PrintGPA
  - PrintInfo

Class and Instance Variables

- Note class has
  - pieces of data (instance variables)
  - behaviors
- Each instance of this type has its own copies of the data items

Constructor

- A **constructor**: it is a method that:
  - does not have any return type
  - has the same name of the class
  - must be public
- Java defines for you a **default constructor** that initialized all your attributes to null or default values.

When you define a constructor with parameters, the default constructor will no longer be available.
It is overridden. You must then define your own default constructor.

- Example of default constructor for Student class:

```
public Student(){

}
```

Defining Constructors

- A special method called when instance of an object created with new
  - Create objects
  - Initialize values of instance variables
- Can have parameters
  - To specify initial values if desired
- May have multiple definitions
  - Each with different numbers or types of parameters

Constructors

- **Constructor overloading**: means having several constructors inside the same class.

Information Hiding

- **Access specifier**: determines how to access an attribute or a method. Access specifiers are public by default or private
- A good programming practice is to make all attributes as private and restrict their access from external classes. This called **data hiding** or **encapsulation**, because data is hidden from external classes.

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*

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`

Data hiding and data Validation

- Why data hiding is a good programming practice?
  - this is needed to perform **data validation** , i.e. ,make sure that values assigned to attributes are valid
  - does not allow external program to put non valid values.
- How to access (read/write) the private attributes?
  - use get methods to read an attribute value
  - use the set methods to modify the value of the attribute

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

- 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.

Using object-oriented design (Student class)

code Student Version1 Class and Student Test

The Keyword `this`

- This is a keyword in Java that refers to the object of the same class where it is called.
- This keyword can be used to call the constructor of the class.
- Referring to instance variables outside the class – must use
  - Name of an object of the class
  - Followed by a dot
  - Name of instance variable
- Inside the class,
  - Use name of variable alone
  - The object (unnamed) is understood to be there

The Keyword `this`

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

```
this.name = keyboard.nextLine();
```

- The keyword `this` stands for the receiving object
- We will seem some situations later that require the `this`

Static keyword

- static keyword: creates a **shared attribute or method** for all created objects, i.e. class-wide attribute or method.
- The keyword final is used to create **constant** values, .i.e cannot be changed.

Static Variables

- Static variables are shared by all objects of a class
  - Variables declared `static final` are considered constants – value cannot be changed
- Variables declared `static` (without `final`) can be changed
  - Only one instance of the variable exists
  - It can be accessed by all instances of the class

Static Variables

- Static variables also called *class variables*
  - Contrast with *instance variables*
- Both static variables and instance variables are sometimes called *fields* or *data members*

Static Methods

- Some methods may have no relation to any type of object
- Example
  - Compute max of two integers
  - Convert character from upper- to lower case
- Static method declared in a class
  - Can be invoked without using an object
  - Instead use the class name

Programming Example- version 2

- Write another version of class Student that has a firstName, lastName, id, mobilePhone and a count. Declare the count as a static variable. The count is incremented every time an object of this class is created. Write the default constructor, a constructor that takes parameters: firstName, lastName, id and mobilePhone. Write a third constructor that takes only firstName and lastName. Write the set and get methods for your attributes. Make sure that the phone number consists of 10 numbers and that the first name and last name consist of more than 2 characters. Create a function that displays all the student information and another that displays only the name.
- Then write a demo program that tests the methods of your Student class.

Student Class Code

- Student Version 2

toString() Method

- toString() method is a special method that provides a String representation for any object in Java. By default, it returns a reference to the object

but, you can **override** (changing its definition) it to return another **string representation** of the object.

equals method

- public boolean equals(Object obj)
- the **public boolean equals(Object obj)** of the **Object** class compares object references (i.e. addresses) for equality: For any non-null reference values **x** and **y**, the call: **x.equals(y)** to the equals method of the **Object** class returns true if and only if, **x** and **y** refer to the same object, i.e., if **x == y** has the value true. It is recommended to override the **equals** method in our class to compare objects based on one or more instance variables.

equals method

```
// Two student objects are equal if theirIDs are equal

public boolean equals(Student other){
        return(this.id == other.getId());
  }
```

You can test in Main by creating two objects st1 and st2

```
if (st1.equals(st2))
   System.out.println("Equal IDs");

else
    System.out.println("Not Equal IDs");
```

Copy Constructors

- A copy constructor is a constructor with a single argument of the same type as the class. The copy constructor should create an object that is a separate object but with the instance variables set so that the object is an exact copy of the argument object.

Copy Constructors

```
public Student (Student s){
    firstName=s.firstName;
    lastName=s.lastName;
    id=s.id;
    mobilePhone=s.mobilePhone;
  }
```

Static Methods

- View sample class, listing 6.5
  `class DimensionConverter`
- View demonstration program, listing 6.6
  `class DimensionConverterDemo`

```
Enter a measurement in inches: 18
18.0 inches = 1.5 feet.
Enter a measurement in feet: 1.5
1.5 feet = 18.0 inches.
```

Sample screen output

LISTING 6.5 Static Methods

```
/**
 Class of static methods to perform dimension conversions.
*/
public class DimensionConverter
{
    public static final int INCHES_PER_FOOT = 12;

    public static double convertFeetToInches(double feet)
    {
        return feet * INCHES_PER_FOOT;
    }

    public static double convertInchesToFeet(double inches)
    {
        return inches / INCHES_PER_FOOT;
    }
}
```

A static constant; it could be private here.

LISTING 6.6 Using Static Methods

```
import java.util.Scanner;
/**
 Demonstration of using the class DimensionConverter.
*/
public class DimensionConverterDemo
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter a measurement in inches: ");
        double inches = keyboard.nextDouble();
        double feet =
               DimensionConverter.convertInchesToFeet(inches);
        System.out.println(inches + " inches = " +
                           feet + " feet.");

        System.out.print("Enter a measurement in feet: ");
        feet = keyboard.nextDouble();
        inches = DimensionConverter.convertFeetToInches(feet);
        System.out.println(feet + " feet = " +
                           inches + " inches.");
    }
}
```

Sample Screen Output

```
Enter a measurement in inches: 18
18.0 inches = 1.5 feet.
Enter a measurement in feet: 1.5
1.5 feet = 18.0 inches.
```

Composition

- **Composition** is the **has-a** relationship between two classes. We say that a class **is composed of** an object of another class or a class **has-a** object of another class. For example, the class Student has an object of the class Date.

Student Version 3

- Modify your student class to include a date object that represents the birth date of the students in addition to all the previous attributes.
- A Date object is characterized by its day, month and year.
- The date class should have a constructor, set and get methods for all its attributes. In addition the class should include a method to display Date.
- Note that you can write a simple version of class date without validation the attributes.
- Write also a demo program to test your classes.

- Code of Student Version 3