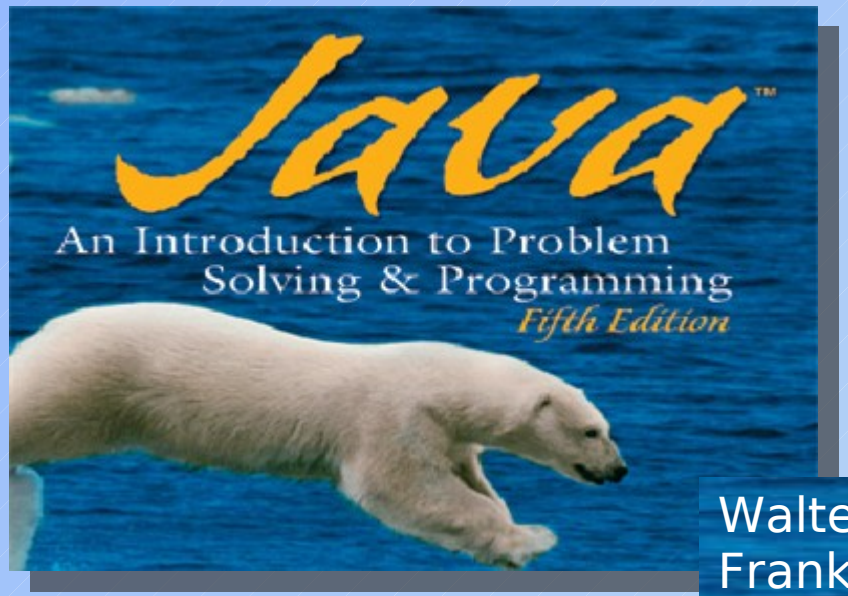


Walter Savitch
Frank M. Carrano

Defining Classes and Methods

Chapter 5

*JAVA: An Introduction to Problem Solving & Programming, 5*th Ed. By Walter Savitch and Frank Carrano.
ISBN 0136130887

Class and Method Definitions: Outline

- Class Files and Separate Compilation
- Instance Variables
- Methods
- The Keyword this
- Local Variables
- Blocks
- Parameters of a Primitive Type

Class and Method Definitions

- A Java program consists of objects which interact with each other
  - Objects of class types (String, Scanner)
  - Objects have both data and methods
- Program objects can represent
  - Objects in real world
  - Abstractions

Class and Method Definitions

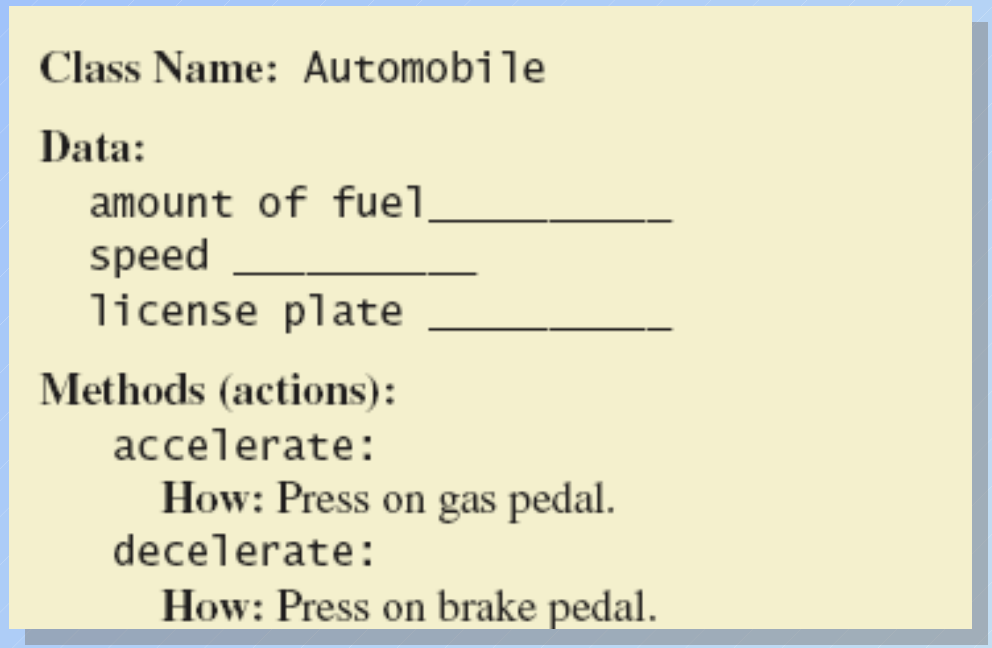
- A **class definition** is a **template** or **blueprint** for creating objects
- A class definition is like a cookie-cutter
- A cookie cutter is not a cookie, but it can be used to create cookies
- Each cookie created by a particular cookie-cutter will have **the same attributes** (thickness, decoration), but **different values for those attributes** (3mm, “#1 Luke”)

Class and Method Definitions

- An **instance** of a class is an object of that class type

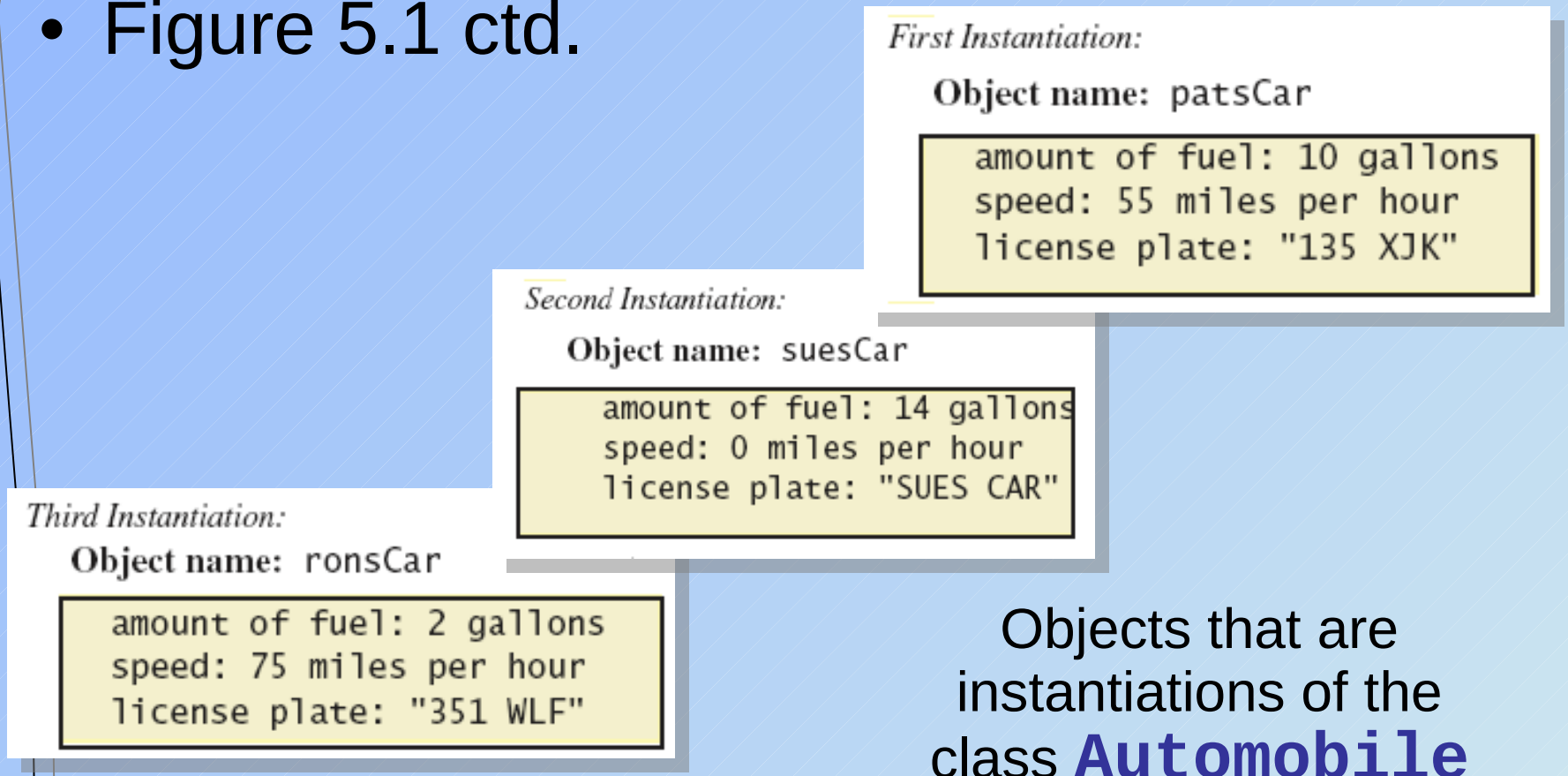
Class and Method Definitions

- Figure 5.1 A class as a blueprint



Class and Method Definitions

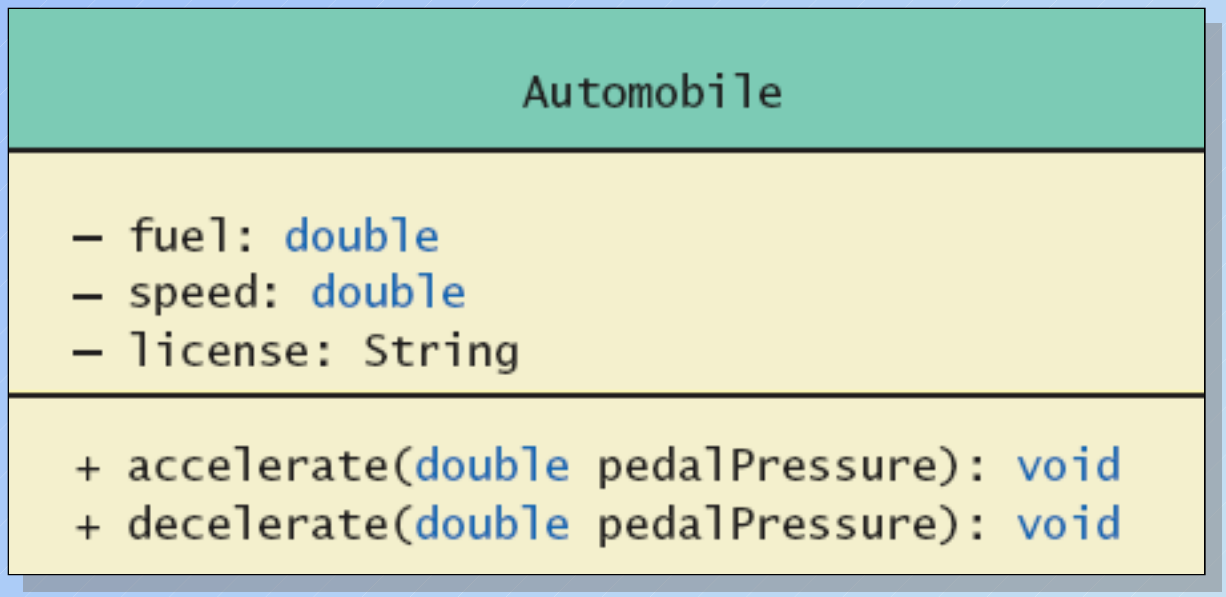
- Figure 5.1 ctd.



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

Class and Method Definitions

- Figure 5.2 A class outline as a UML class diagram (UML = Universal Modeling Language)



Class Files and Separate Compilation

- Each **`Java`** class definition usually in a file by itself
  - File name begins with name of the class
  - Ends with .**`java`**
- Class can be compiled separately
- Useful to keep all class files used by a program in the same directory

Instance Variable

- Download **`SpeciesFirstTry` and `SpeciesFirstTryDemo`**
- Note **`SpeciesFirstTry`** has
  - three pieces of data (instance variables)
  - three behaviors (methods)
- Each instance of this type has its own copies of the data items
- Meaning of **`public`**
  - No restrictions on how variables used

Methods

- When you use a method you "invoke" or "call" it
- Two kinds of Java methods
  - Return a single item
  - Perform some other action – a **`void`** method
- The method **`main`** is a **`void`** method
  - Invoked by the system
  - Not by the application program

Methods

- Calling a method that returns a quantity
  - Use anywhere a value can be used
    - `if (keyboard.nextInt() > 0) ...`
- Calling a void method
  - Write the invocation followed by a semicolon
  - Resulting statement performs the action defined by the method
    - `System.out.println("hello");`

Defining `void` Methods

- Consider method **`writeOutput`** from **`SpeciesFirstTry`**

```
public void writeOutput()
{
     System.out.println("Name = " + name);
     System.out.println("Population = " + population);
     System.out.println("Growth rate = " + growthRate + "%");
}
```

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

Defining `void` Methods

- Most method definitions we will see as **`public`**
- Method does not return a value
  - Specified as a **`void`** method
- Heading includes parameters
- Body enclosed in braces **`{ }`**
- Think of a method as defining an action to be taken

Methods That Return a Value

- Consider method **`getPopulationIn10( )`**

```
public int getPopulationIn10()
{
    int result = 0;
    double populationAmount = population;
    int count = 10;
    while ((count > 0)
           ...
    if (populationAmount > 0)
        result = (int)populationAmount;
    return result;
}
```

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

Naming Methods

- Use a verb to name a void method
  - writeOutput
- Use a noun to name a method that returns a value
  - nextInt
- All method names should start with a lowercase letter

Referring to Instance Variables

- 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
  - can usually be omitted
- We will see some situations later that require the **`this`**

Local Variables

```
public class SpeciesFirstTry
{
    public String name;
    public int population;
    public double growthRate;
```

- Note beginning of class in listing 5.1
- Variables declared inside the class are considered *local* variables
  - May be used only inside this class
- Variable with same name inside a different class is considered a different variable
- All variables declared in method **`main`** are local to **`main`**

Local Variables

- **Download `BankAccount`** and **`LocalVariablesDemoProgram`**
- Note two different variables **`newAmount`**
  - Note different values output

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

Sample screen output

Blocks

- Recall compound statements
  - Enclosed in braces { }
- When you declare a variable within a compound statement
  - The compound statement is called a *block*
  - The scope of the variable is from its declaration to the end of the block
- A Variable declared outside the block is usable both outside and inside the block

Parameters of Primitive Type

```
public int getPopulationIn10()
{
    int result = 0;
    double populationAmount = population;
    int count = 10;
```

- Recall method declaration in **SpeciesFirstTry**
  - Note it only works for 10 years
  - We can make it more versatile by giving the method a parameter to specify how many years
- Download **SpeciesSecondTry** and **SpeciesSecondTryDemo**

Parameters of Primitive Type

- Note the declaration
  ```
  public int predictPopulation(int years)
  ```
  - The *formal* parameter is `years`
- Calling the method
  ```
  int futurePopulation =
   speciesOfTheMonth.predictPopulation(10);
  ```
  - The *actual* parameter is the integer 10

Parameters of Primitive Type

- Parameter names are local to the method
- When a method is 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
```