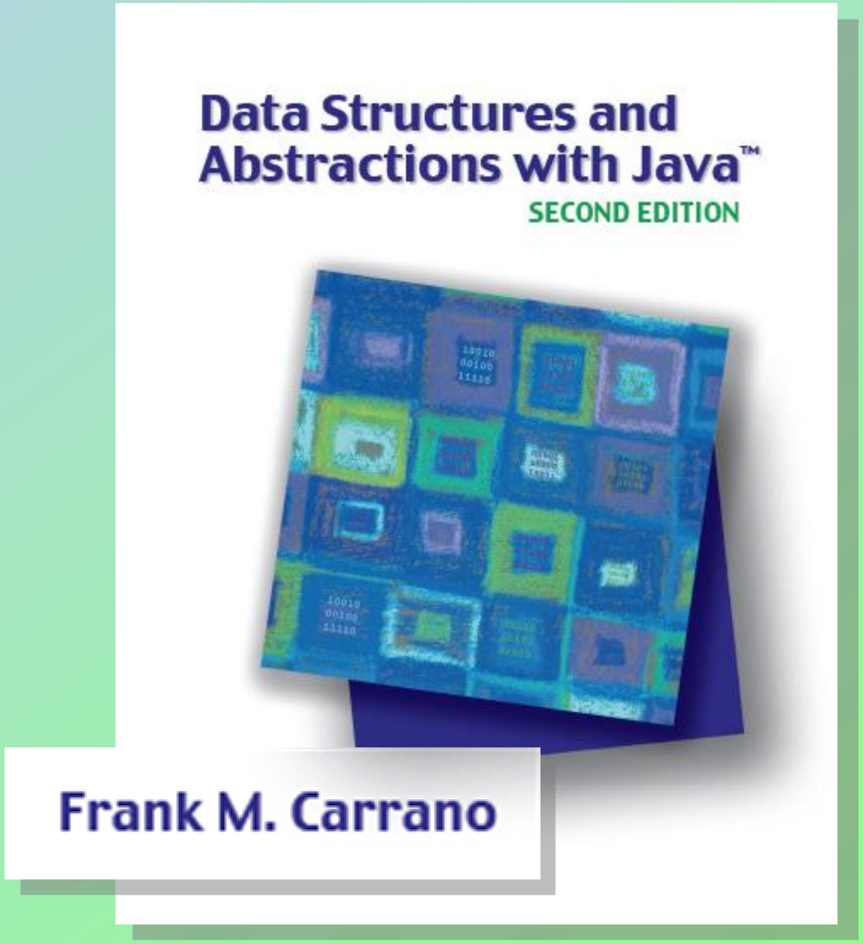
Java Classes

Appendix B

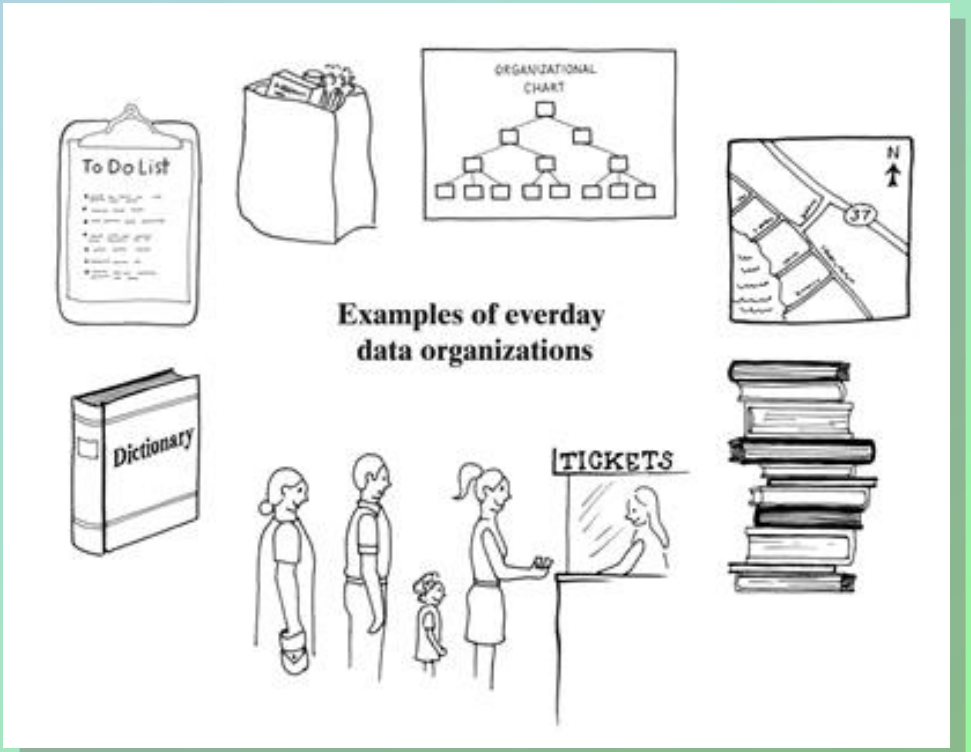


Slides by Steve Armstrong
LeTourneau University
Longview, TX

Chapter Contents

Chapter Contents

Organizing Our Lives



- For each of the above examples, consider how the objects are organized

Organizing Computer Data

- Computer stores/organizes items in similar manners as the examples
- Ways of organizing data are represented by Abstract Data Types (ADTs)
- An ADT specifies
  - data that is stored
  - operations that can be done on the data

ADT Terminology

- Data structure: implementation of an ADT within a programming language
- Collection: an ADT that contains a group of objects
- Container: a class that implements the collection
- These last two terms are sometimes used interchangeably

Types of ADTs

- Bag
  - Unordered collection, may contain duplicates
- List
  - A collection that numbers its items
- Stack
  - Orders items chronologically
  - Last In, First out
- Queue
  - Orders items chronologically
  - First in, First out

Match each of these to the pictures ?

Click here to return to pictures

Types of ADTs

- Dictionary
  - Pairs of items – one is a key
  - Can be sorted or not
- Tree
  - Arranged in a hierarchy
- Graph
  - Generalization of a tree

Match each of these to the pictures ?

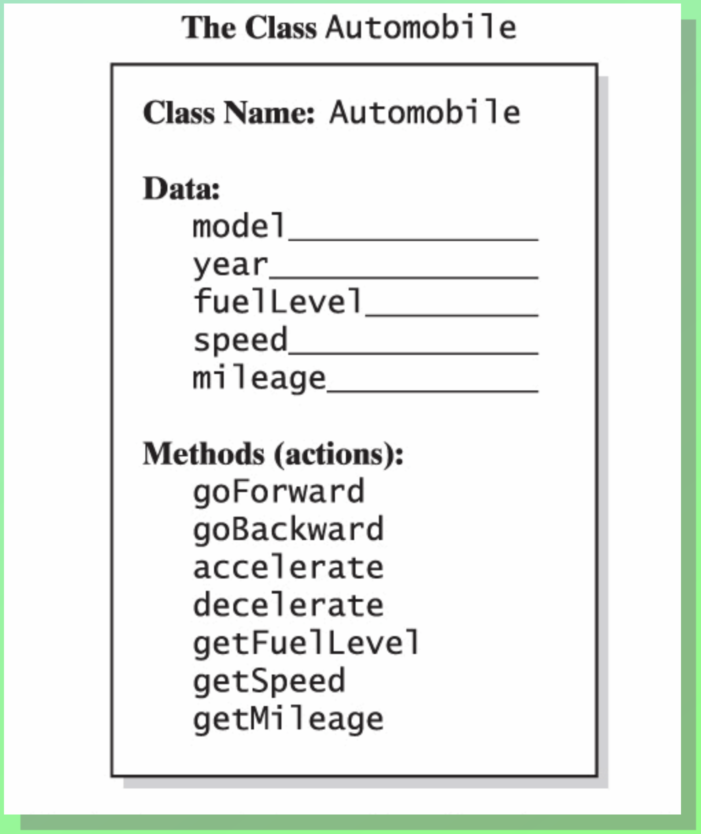
Click here to return to pictures

Objects and Classes 1

- An <u>object</u> is a program construct
  - Contains data
  - Performs actions
- Objects interact to solve problems
- Actions performed by objects are defined by <u>methods</u>

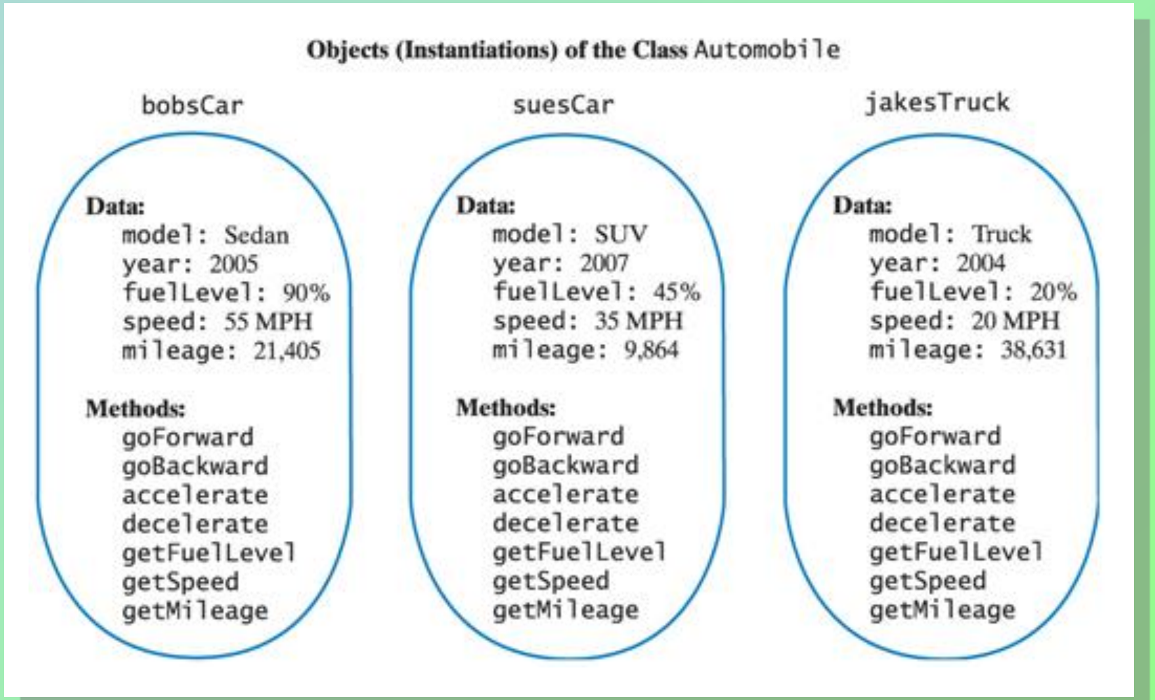
Objects and Classes

- A <u>class</u> is a kind of object
- A class definition is a general description of
  - what the object is
  - what it can do



Objects and Classes

- All objects in the same class have
  - the same kinds of data
  - the same methods



Using the Methods in a Java Class 2

- Given a class called **`Name`**
  - Declare a variable

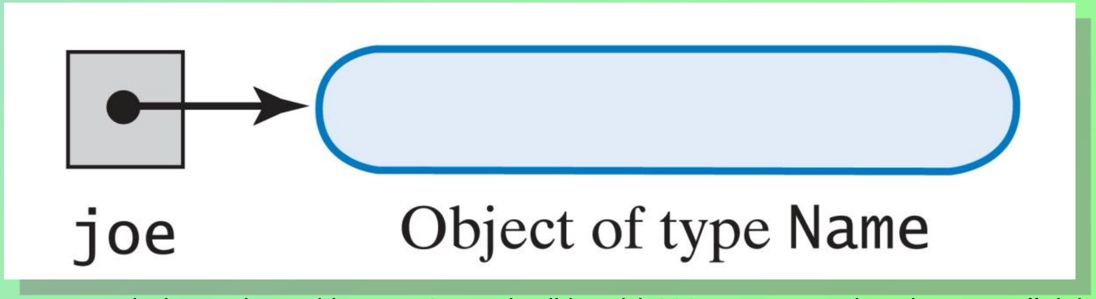
    ```
    Name joe;
    ```

  - Create an instance of **`Name`**

    ```
    joe = new Name();
    ```

  - Alternatively

    ```
    Name joe = new Name();
    ```



Using the Methods in a Java Class 3

- void methods are used to do a task such as set the first or last names

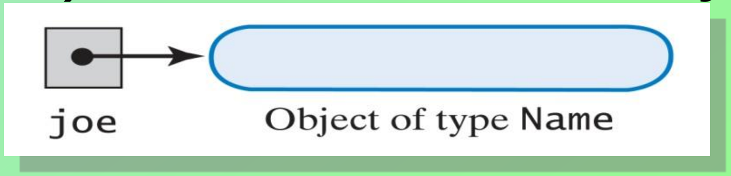
```
joe.setLast("Brown");
```

- valued methods return a single value

```
String hisLastName = joe.getLast();
```

References and Aliases 4

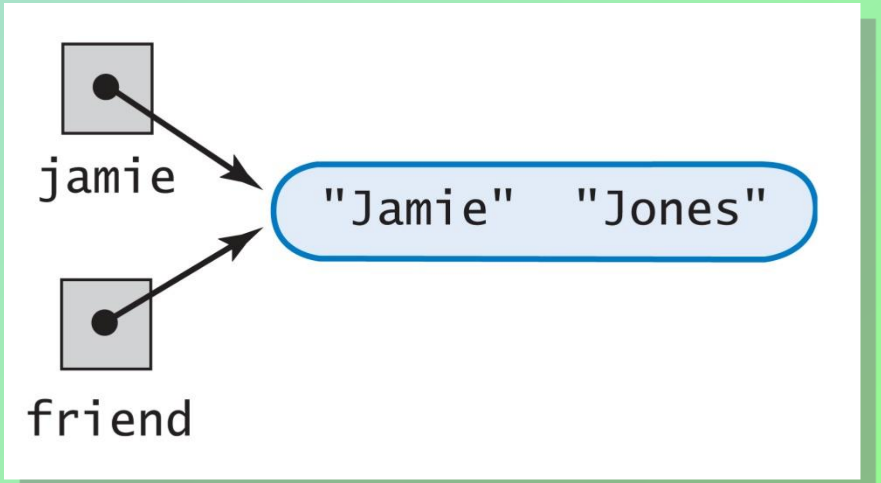
- Primitive data types
  - byte
  - short
  - int
  - long
  - float
  - double
  - char
  - boolean
- All other data types are <u>reference</u> or class types
- A reference variable contains <u>address</u> of (reference to) location in memory of an object



References and Aliases

- Consider the results of the code below:

```
Name jamie = new Name();
jamie.setFirst("Jamie");
jamie.setLast("Jones");
Name friend = jamie;
```



Defining a Java Class 5

Access or visibility modifiers

Specifies where a class, data field, or method can be used.

Data members

```
public class Name
{
private String first; // first name
private String last; // last name

< Definitions of methods are here >
. . .
} // end Name
```

Defining a Java Class

Methods that classes often use:

- Accessor (query) methods – return value of a data field
- Mutator methods – change the value of a data field

```
public class Name
{
private String first; // first name
private String last; // last name

< Definitions of methods are here >
. . .
} // end Name
```

Method Definitions 7

- General form of method definition

```
access-modifier use-modifier return-type
                    method-name(parameter-list)
{
    method-body
}
```

Examples

```
public String getFirst()
{
    return first;
} // end getFirst
```

```
public void
    setFirst(String firstName)
{
    first = firstName;
} // end setFirst
```

Method Definitions 10

- Note incorrect, ambiguous use of identifier **`first`**
- Solvable by use of **`this`**
  - **`this.first`** refers to data member
  - Note: possible but not typical
  - use different names

```
public void
    setFirst(String first)
{
    first = first;
} // end setFirst
```

data object
`first`

parameter
`first`

```
public void
    setFirst(String first)
{
    this.first = first;
} // end setFirst
```

Arguments and Parameters 12

- Consider statements:

```
Name joe = new Name();
joe.setFirst("Joseph");
joe.setLast("Brown");
```

- Arguments/parameters in call match in number and type to formal parameters in definition

```
public void setFirst(String firstName)
{
     first = firstName;
} // end setFirst
```

Passing Arguments 13

- When formal parameter is primitive type
  - parameter in method initialized by value
  - can be constant or variable

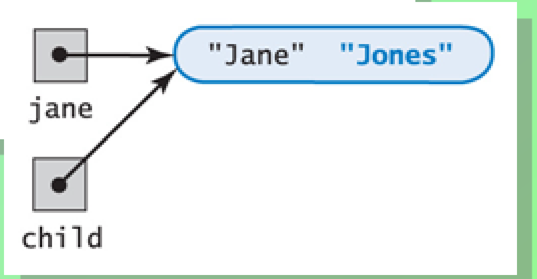
```
public void setMiddleInitial(char middleInitial)
{
    initial = middleInitial;
} // end setMiddleInitial
```

```
joe.setMiddleInitial('Q');
```

Passing Arguments 14

- When a formal parameter has a class type

```
public void giveLastNameTo(Name child)
{
child.setLast(last);
} // end giveLastNameTo
```



  - Formal parameter initialized with memory address of object passed to it.

```
jamie.giveLastNameTo(jane);
```

Passing Arguments

- However, a method <u>cannot replace</u> an object passed to it as an argument

```
public void giveLastNameTo2(Name child)
{
    String firstName = child.getFirst();
    child = new Name();
    child.setFirst(firstName);
    child.setLast(last);
} // end giveLastNameTo2
```

`child` is considered <u>local</u>.

It will disappear when the method finishes, argument remains unchanged

A Definition of the Class `Name` 16

- View definition of full class
- Note
  - Constructors
  - **`set`** methods – mutators
  - **`get`** methods – accessors
  - **`toString`** method
- Note demonstration program

Work on Item class

- Use the Name class as a model
- Write the default and explicit constructors
- Write the accessor and mutator methods
- Run the driver main method to verify your code

Constructors

- Tasks of a constructor
  - Allocate memory for object
  - Initialize data fields
- Properties
  - Same name as class
  - No return type (not even **`void`**)
  - Can have any number of parameters (including <u>no</u> parameters)
  - Note <u>constructors of **`Name`**</u>

The Method `toString` 21

- Note the `toString` method of class **`Name`**
  - Returns a string with value of person's name
- For any class, **`toString`** method invoked automatically for command

```
System.out.println (someObject);
```

Add toString to Item class

- Notice how you can explicity call toString
  - System.out.println(one.toString()) same as
  - System.out.println(one);

  - Anytime Java needs to combine your object with a String, invokes toString automatically

Methods That Call Other Methods 22

- Note **`setName`** method in class Name
  - Invokes **`setFirst`** and **`setLast`**
  - **`setName`** invokes them without preceding the method name with object variable and dot
- Consider the **`getName`** method
  - Calls **`toString`**
  - Thus both methods always give same result

Methods That Return an Instance of Their Class 26

- Consider a different version of **`setName`**

```
public Name setName(String firstName,
                    String lastName)
{
    setFirst(firstName);
    setLast(lastName);
    return this;
} // end setName
```

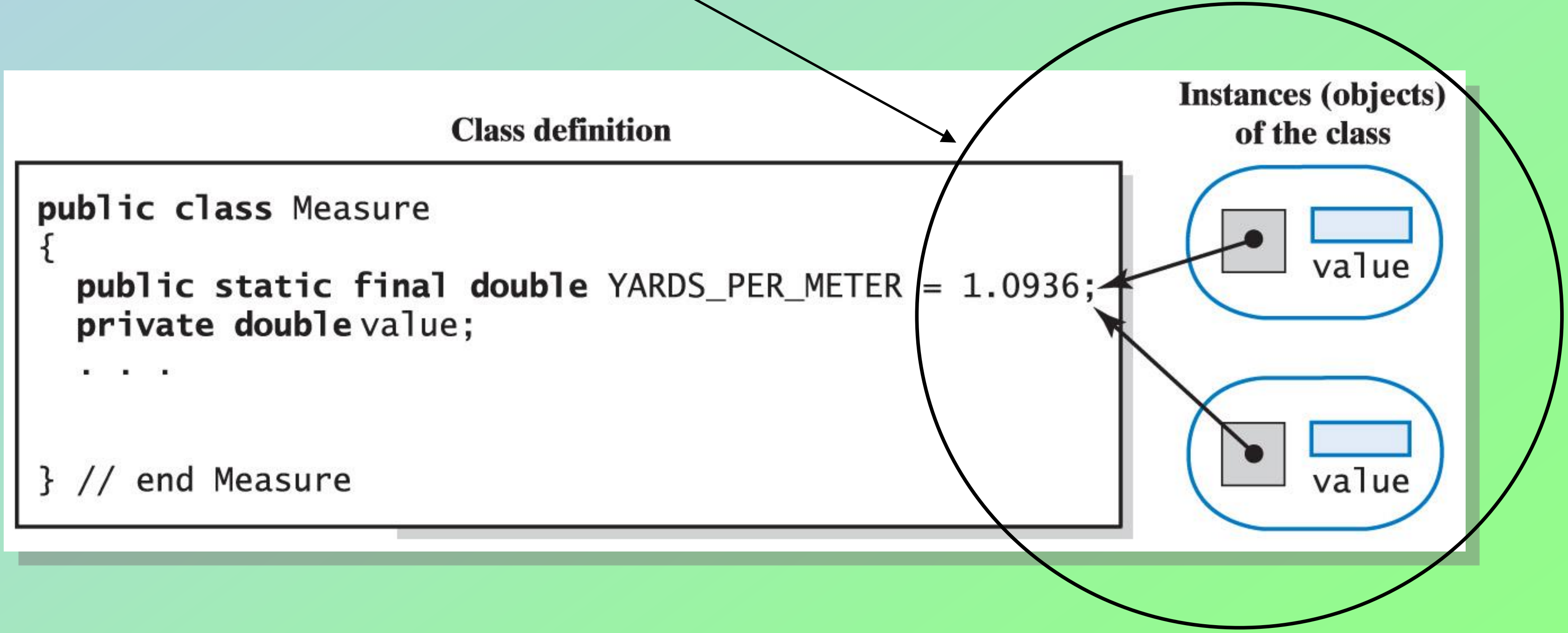
- The **`return this;`** returns a reference to the invoking object.

Static Fields and Methods 27

- A static data field does not belong to any one object
  - Also called a <u>class</u> variable
  - Only one instance of the variable exists for all instances of the class
- Note that a static data field is not a constant (final)

Static Fields and Methods

- All instances of the class reference that one variable



```
public class Measure
{
   public static final double YARDS_PER_METER = 1.0936;
   private double value;
   . . .

} // end Measure
```

Static Fields and Methods

- Consider the need of a method that does not belong to an object of any type
- Examples
  - A method to find the max or min of two or more numbers
  - A square root method

Static Fields and Methods

- When specified **`static`**, a method is still a member of the class
  - However, does not need an object as a prefix to the call
- Call with the name of the class

```
int maximum = Math.max(2, 3);
double root = Math.sqrt(4.2);
```

Additional Experiments (Optional)

- What happens if you try to create an object of the Math class?
- Add a public “count” static var to Item class
  - And a serialNumber instance variable
  - Mod the constructor to assign a unique serial number (the current count) to each object created
  - Mod the toString method to print the serial num
  - Print the number of Items creates using Item.count
  - Show one.count, two.count always the same

Overloading Methods 29

- Multiple methods within the same class can have the same name
- Java distinguishes them by noting the parameters
  - Different numbers of parameters
  - Different types of parameters
- This is called the <u>signature</u> of the method

Packages 34

- Packages enable grouping together multiple related classes
- Specify a class to be part of a package with first line

```
package myStuff;
```

- Place all classes in same directory which is named with the name of the package
- In your program which uses the package

```
import myStuff.*;
```

The Java Class Library 35

- The Java language has many classes defined
  - Recall the **`Math`** class with **`max`** and **`sqrt`**
- Collection known as

  Java Class Library or

  Java Application Programming Interface