More About Classes, Objects and Methods

Chapter 9

*JAVA: An Introduction to Problem Solving & Programming, 8*th Ed. By Walter Savitch
ISBN 0134462033

Objectives

- Define and use constructors
- Write and use static variables and methods
- Use predefined wrapper classes
- Write and use overloaded methods
- Define and use enumeration methods
- Define and use packages and `import` statements

Constructors: Outline

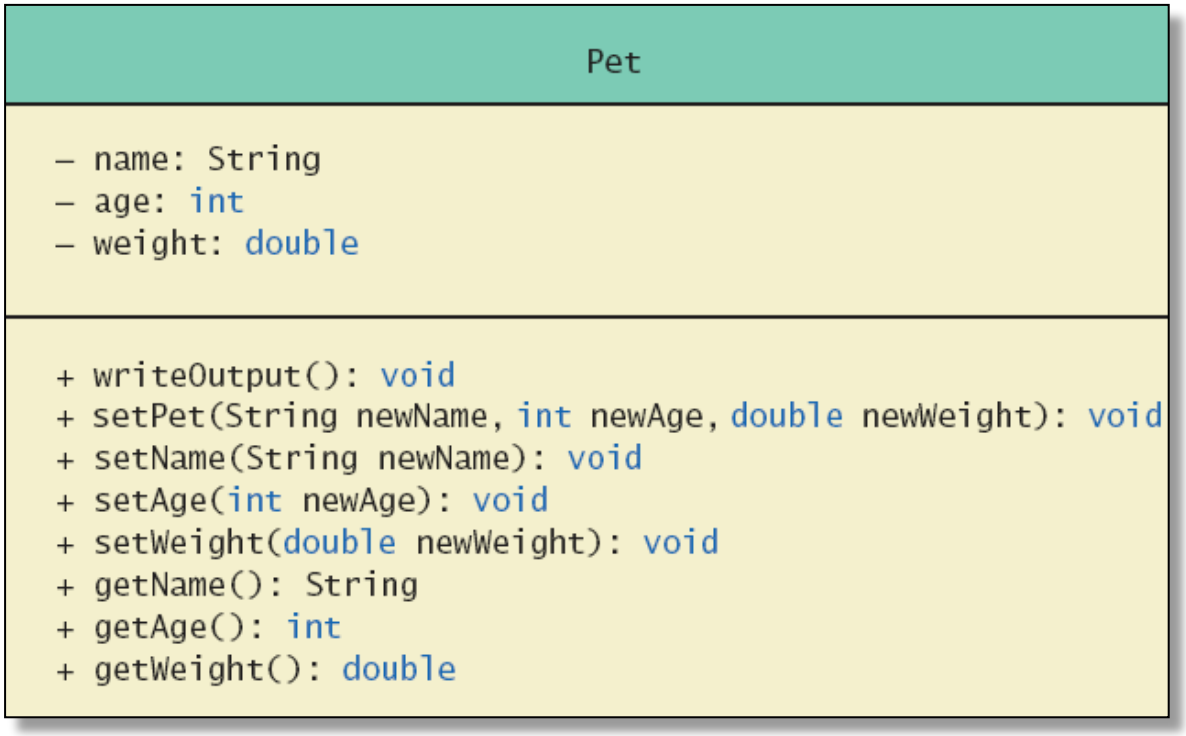
- Defining Constructors
- Calling Methods from Constructors
- Calling a Constructor from Other Constructors

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

Defining Constructors

- Example class to represent pets
- Figure 9.1 Class Diagram for Class `Pet`



Defining Constructors

- Note sample code, listing 9.1
  `class Pet`
- Note different constructors
  - Default
  - With 3 parameters
  - With String parameter
  - With double parameter
- Note sample program, listing 9.2
  `class PetDemo`

Defining Constructors

```
My records on your pet are inaccurate.
Here is what they currently say:
Name: Jane Doe
Age: 0
Weight: 0.0 pounds
Please enter the correct pet name:
Moon Child
Please enter the correct pet age:
5
Please enter the correct pet weight:
24.5
My updated records now say:
Name: Moon Child
Age: 5
Weight: 24.5 pounds
```

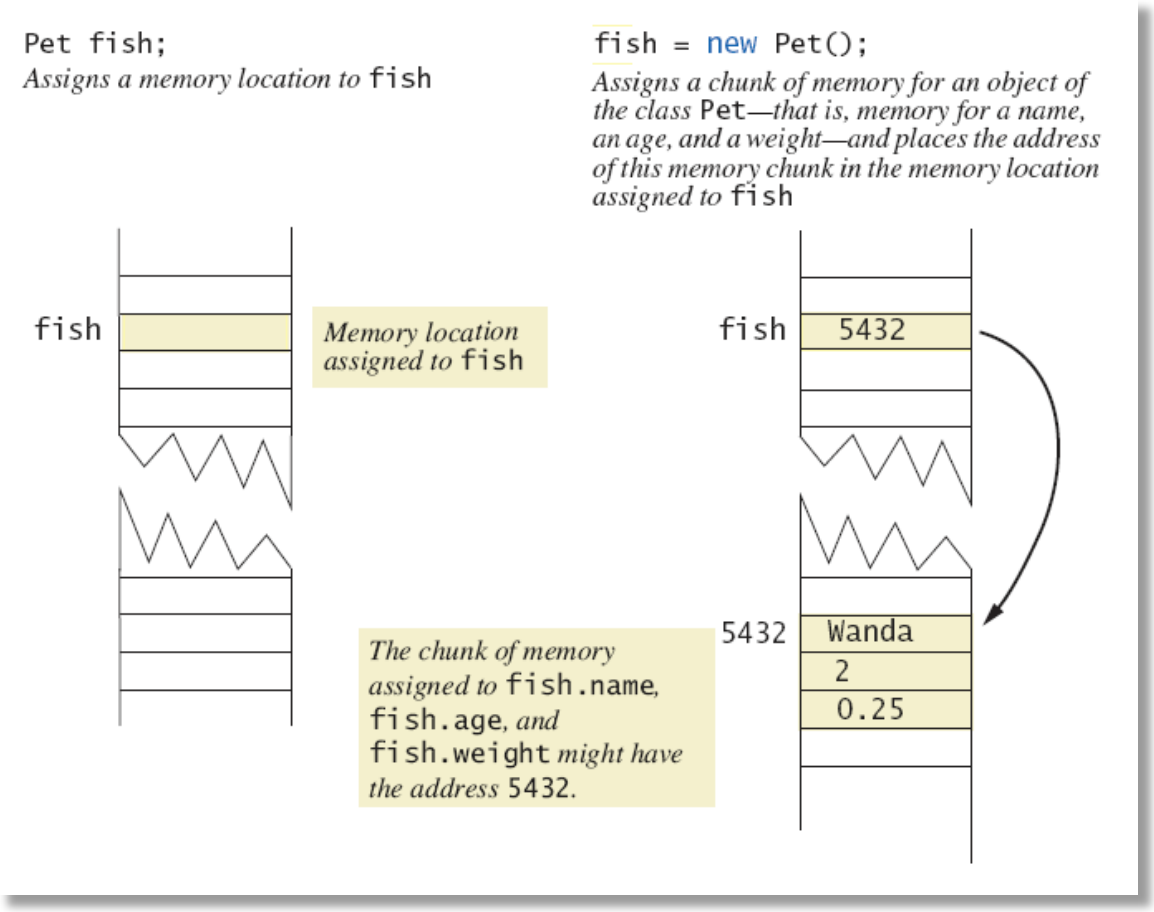
Sample screen output

Defining Constructors

- Constructor without parameters is the default constructor
  - Java will define this automatically if the class designer does not define any constructors
  - If you do define a constructor, Java will not automatically define a default constructor
- Usually default constructors not included in class diagram

Defining Constructors

- Figure 9.2 A constructor returning a reference



Calling Methods from Other Constructors

- Constructor can call other class methods

```
public Pet(String initialName, int initialAge,
           double initialWeight)
{
    setPet(initialName, initialAge, initialWeight);
}
```

- View sample code, listing 9.3
  `class Pet2`
  - Note method `set`
  - Keeps from repeating code

Calling Constructor from Other Constructors

- From listing 9.3 we have the initial constructor and method set
- In the other constructors use the this reference to call initial constructor
- View revised class, listing 9.4
  `class Pet3`
  - Note calls to initial constructor

Static Variables & Methods: Outline

- Static Variables
- Static Methods
- Dividing the Task of a `main` Method into Subtasks
- Adding a `main` Method to a class
- Wrapper Classes

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*
- Do not confuse class variables with variables of a class type
- 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

Static Methods

- View sample class, listing 9.5
  `class DimensionConverter`
- View demonstration program, listing 9.6
  `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

Mixing Static and Nonstatic Methods

- View sample class, listing 9.7
  `class SavingsAccount`
- View demo program, listing 9.8
  `class SavingsAccountDemo`

```
I deposited $10.75.
You deposited $75.
You deposited $55.
You withdrew $15.75.
You received interest.
Your savings is $115.3925
My savings is $10.75
We opened 2 savings accounts today.
```

Sample screen output

Tasks of `main` in Subtasks

- Program may have
  - Complicated logic
  - Repetitive code
- Create static methods to accomplish subtasks
- Consider example code, listing 9.9

  a `main` method with repetitive code
- Note alternative code, listing 9.10
  uses helping methods

Adding Method `main` to a Class

- Method main used so far in its own class within a separate file
- Often useful to include method main within class definition
  - To create objects in other classes
  - To be run as a program
- Note example code, listing 9.11
  a redefined `class Species`
  - When used as ordinary class, method `main` ignored

Wrapper Classes

- Recall that arguments of primitive type treated differently from those of a class type
  - May need to treat primitive value as an object
- Java provides *wrapper classes* for each primitive type
  - Methods provided to act on values

Wrapper Classes

- Allow programmer to have an object that corresponds to value of primitive type
- Contain useful predefined constants and methods
- Wrapper classes have no default constructor
  - Programmer must specify an initializing value when creating new object
- Wrapper classes have no `set` methods

Wrapper Classes

- Figure 9.3a Static methods in class **`Character`**

Name	Description	Argument Type	Return Type	Examples	Return Value
`toUpperCase`	Convert to uppercase	`char`	`char`	`Character.toUpperCase('a')` `Character.toUpperCase('A')`	`'A'` `'A'`
`toLowerCase`	Convert to lowercase	`char`	`char`	`Character.toLowerCase('a')` `Character.toLowerCase('A')`	`'a'` `'a'`
`isUpperCase`	Test for uppercase	`char`	`boolean`	`Character.isUpperCase('A')` `Character.isUpperCase('a')`	`true` `false`

Wrapper Classes

- Figure 9.3b Static methods in class **`Character`**

Name	Description	Argument Type	Return Type	Examples	Return Value
`isLowerCase`	Test for lowercase	`char`	`boolean`	`Character.isLowerCase('A')` `Character.isLowerCase('a')`	`false` `true`
`isLetter`	Test for a letter	`char`	`boolean`	`Character.isLetter('A')` `Character.isLetter('%')`	`true` `false`
`isDigit`	Test for a digit	`char`	`boolean`	`Character.isDigit('5')` `Character.isDigit('A')`	`true` `false`
`isWhitespace`	Test for whitespace	`char`	`boolean`	`Character.isWhitespace(' ')` `Character.isWhitespace('A')`	`true` `false`

Whitespace characters are those that print as white space, such as the blank, the tab character (`'\t'`), and the line-break character (`'\n'`).

Overloading: Outline

- Overloading Basics
- Overloading and Automatic Type Conversion
- Overloading and the Return Type
- Programming Example: A Class for Money

Overloading Basics

- When two or more methods have same name within the same class
- Java distinguishes the methods by number and types of parameters
  - If it cannot match a call with a definition, it attempts to do type conversions
- A method's name and number and type of parameters is called the *signature*

Overloading Basics

- View example program, listing 9.12
  `class Overload`
- Note overloaded method `getAverage`

```
average1 = 45.0
average2 = 2.0
average3 = b
```

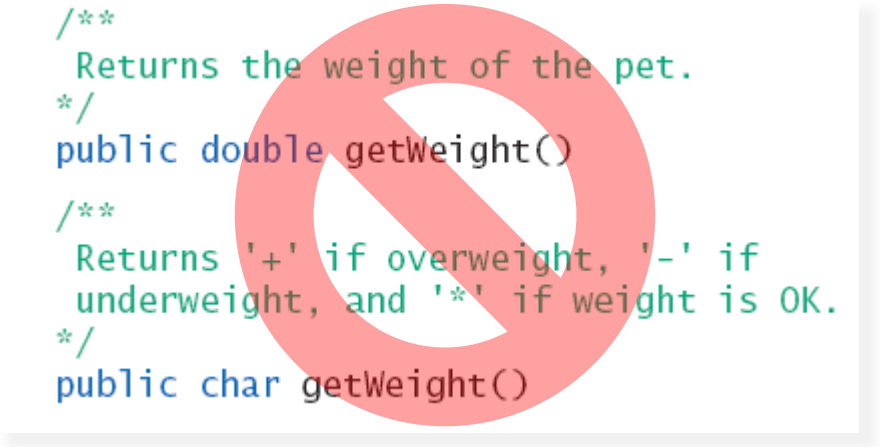
Sample screen output

Overloading and Type Conversion

- Overloading and automatic type conversion can conflict
- Recall definition of Pet class of listing 9.1
  - If we pass an integer to the constructor we get the constructor for age, even if we intended the constructor for weight
- Remember the compiler attempts to overload before it does type conversion
- Use descriptive method names, avoid overloading

Overloading and Return Type

- You must not overload a method where the only difference is the type of value returned



```
/**
 Returns the weight of the pet.
*/
public double getWeight()

/**
 Returns '+' if overweight, '-' if
 underweight, and '*' if weight is OK.
*/
public char getWeight()
```

Programming Example

- A class for money
- View sample class, listing 9.13
  `class Money`
- Note use of
  - Private instance variables
  - Methods to set values
  - Methods for doing arithmetic operations

Programming Example

- View demo program, listing 9.14
  `class MoneyDemo`

```
Enter your current savings:
Enter amount on a line by itself:
$500.99
If you double that, you will have $1001.98, or better yet:
If you triple that original amount, you will have
$1502.97
Remember: A penny saved
is a penny earned.
```

Sample screen output

Information Hiding Revisited

Privacy Leaks

- Instance variable of a class type contain address where that object is stored
- Assignment of class variables results in two variables pointing to same object
  - Use of method to change *either* variable, changes the actual object itself
- View insecure class, listing 9.15

```
class petPair
```

Information Hiding Revisited

- View sample program, listing 9.16

`class Hacker`

```
Our pair:
First pet in the pair:
Name: Faithful Guard Dog
Age: 5 years
Weight: 75.0 pounds
Second pet in the pair:
Name: Loyal Companion
Age: 4 years
Weight: 60.5 pounds
Our pair now:
First pet in the pair:
Name: Dominion Spy
Age: 1200 years
Weight: 500.0 pounds
Second pet in the pair:
Name: Loyal Companion
Age: 4 years
Weight: 60.5 pounds
The pet wasn't so private!
Looks like a security breach.
```

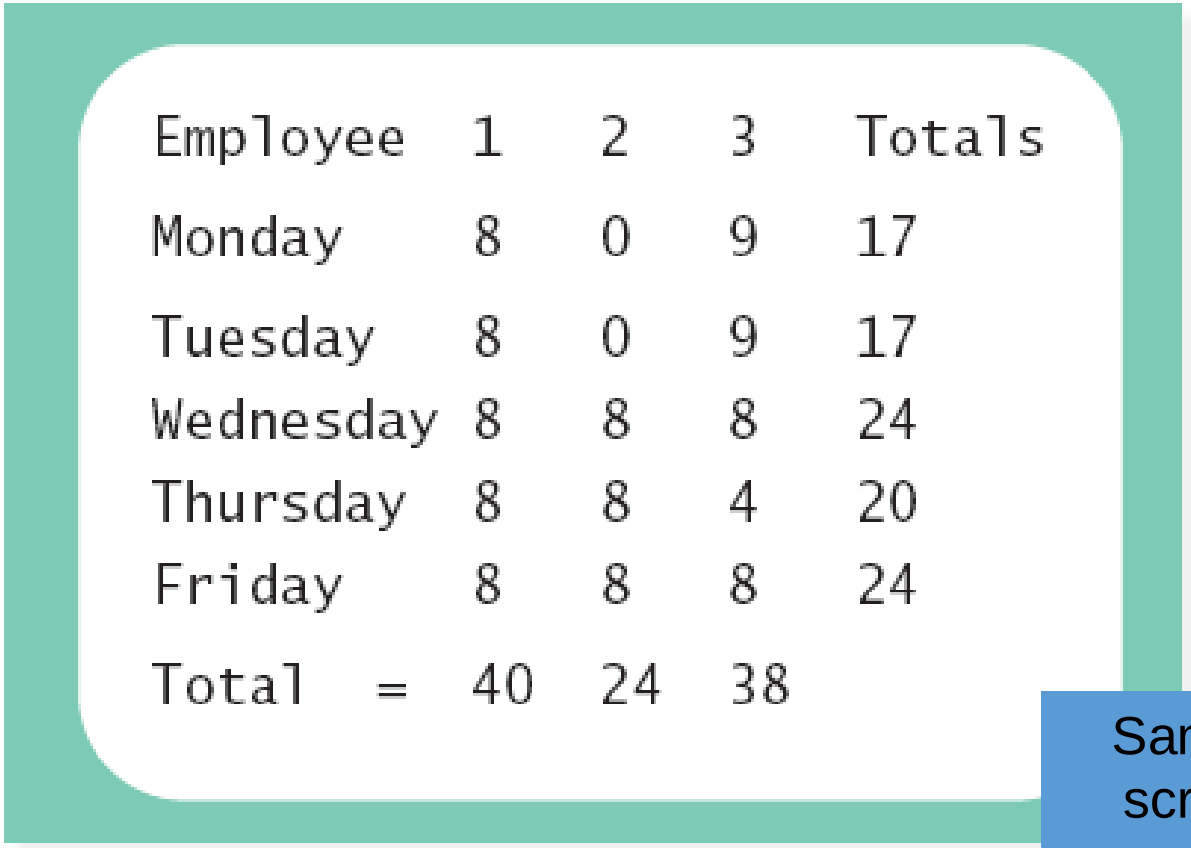
Sample screen output

This program has changed an object named by a private instance variable of the object `pair`.

Programming Example

- Employee Time Records
  - Two-dimensional array stores hours worked
    - For each employee
    - For each of 5 days of work week
  - Array is private instance variable of class
- View sample program, listing 9.17
  `class TimeBook`

Programming Example

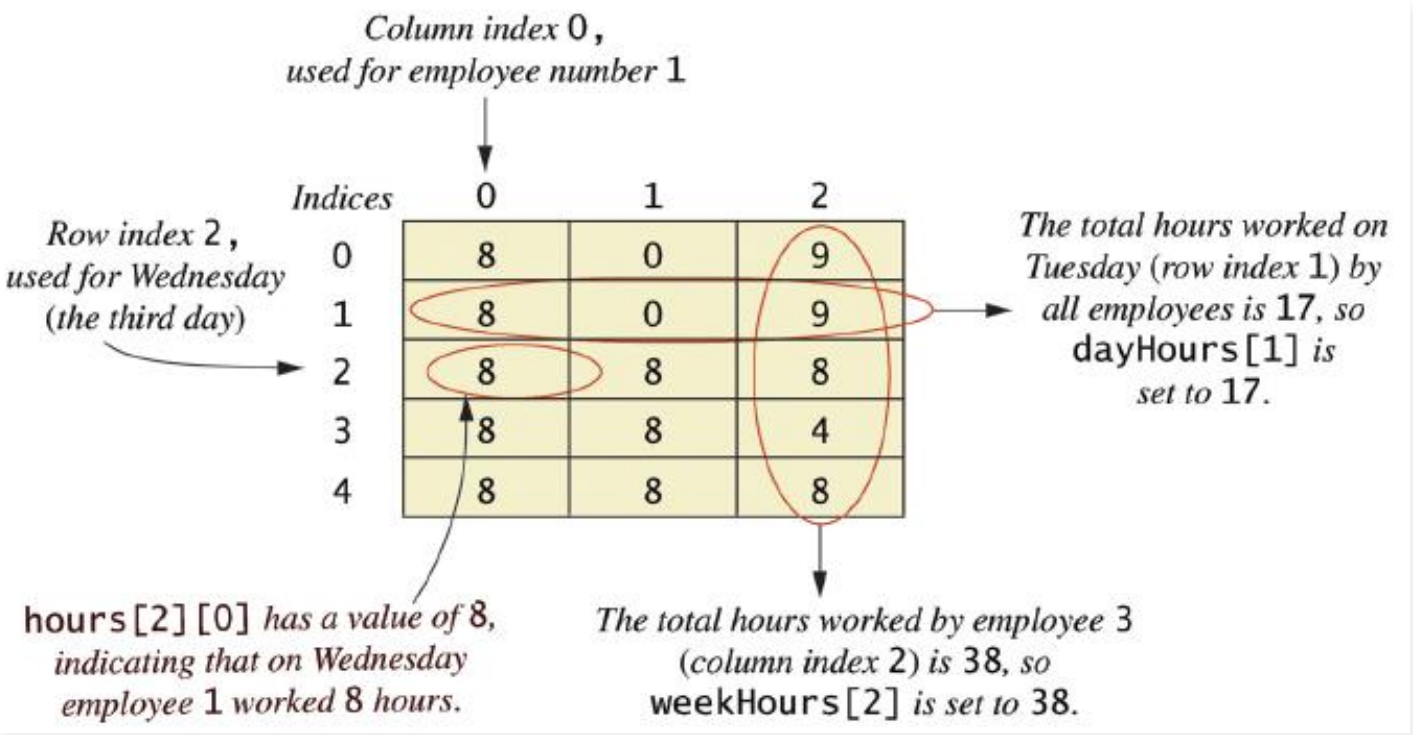


Employee	1	2	3	Totals
Monday	8	0	9	17
Tuesday	8	0	9	17
Wednesday	8	8	8	24
Thursday	8	8	4	20
Friday	8	8	8	24
Total =	40	24	38	

Sample screen output

Programming Example

- Figure 9.10 Arrays for the class **`TimeBook`**



Case Study: Sales Report

- Program to generate a sales report
- Class will contain
  - Name
  - Sales figure
- View class declaration, listing 9.18

```
class SalesAssociate
```

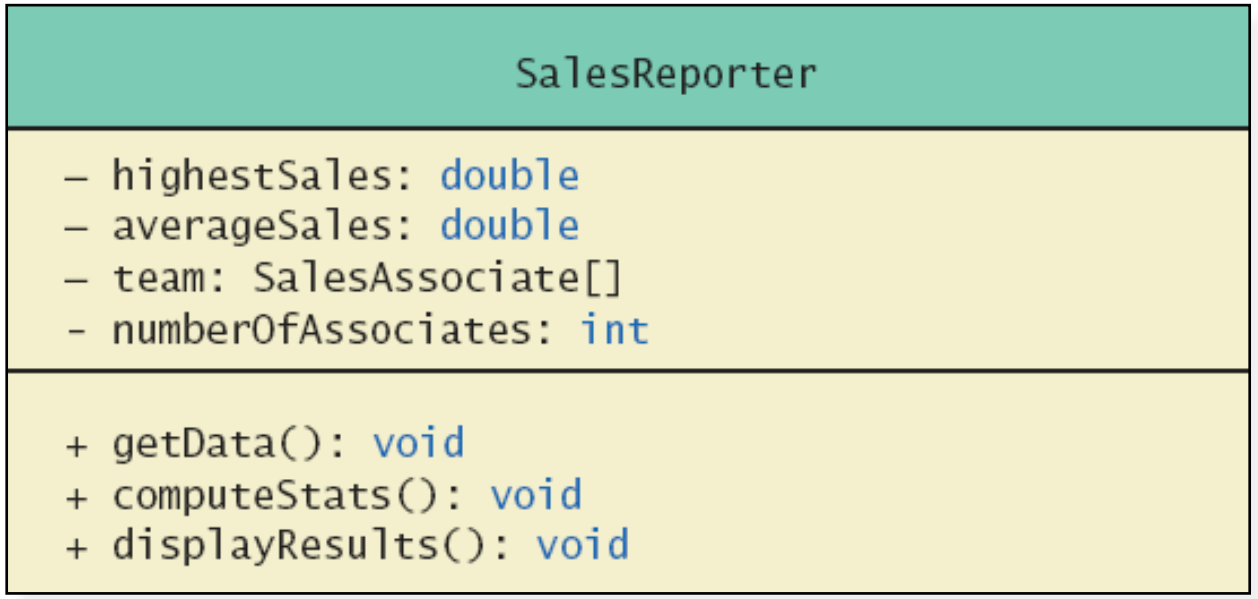
Case Study: Sales Report

Main subtasks for our program

1. Get ready
2. Obtain the data
3. Compute some statistics (update instance variables)
4. Display the results

Case Study: Sales Report

- Figure 9.11 Class diagram for class **`SalesReporter`**



Case Study: Sales Report

- View sales report program, listing 9.19

class SalesReporter

```
Average sales per associate is $32000.0
The highest sales figure is $50000.0

The following had the highest sales:
Name: Natalie Dressed
Sales: $50000.0
$18000.0 above the average.

The rest performed as follows:
Name: Dusty Rhodes
Sales: $36000.0
$4000.0 above the average.

Name: Sandy Hair
Sales: $10000.0
$22000.0 below the average.
```

Sample screen output

Gotcha – Creating an Array of Objects

- When you create an array of objects Java does not create instances of any of the objects! For example, consider the code:

```
SalesAssociate[] team = new SalesAssociate[10];
```

- We can't access team[0] yet; it is **null**. First we must create references to an object:

```
team[0] = new SalesAssociate("Jane Doe", 5000);
team[1] = new SalesAssociate("John Doe", 5000);
```

- we can now access team[0].getName() or team[1].getSalary()

Programming with Arrays and Classes: Outline

- Programming Example: A Specialized List Class
- Partially Filled Arrays

Programming Example

- A specialized List class
  - Objects can be used for keeping lists of items
- Methods include
  - Capability to add items to the list
  - Also delete entire list, start with blank list
  - But no method to modify or delete list item
- Maximum number of items can be specified

Programming Example

- View demo program, listing 9.20
  `class ListDemo`
- Note declaration of the list object
- Note method calls

Programming Example

```
Enter items for the list, when prompted.
Enter an item:
Buy milk
More items for the list? yes
Enter an item:
Walk dog
More items for the list? yes
Enter an item:
Buy milk
More items for the list? yes
Enter an item:
Write program
The list is now full.
The list contains:
Buy milk
Walk dog
Write program
```

Sample screen output

Programming Example

- Now view array wrapped in a class to represent a list, listing 9.21 **`class OneWayNoRepeatsList`**
- Notable code elements
  - Declaration of private array
  - Method to find n^{th} list item
  - Method to check if item is on the list or not

Enumeration as a Class

- Consider defining an enumeration for suits of cards

```
enum Suit {CLUBS, DIAMONDS, HEARTS, SPADES}
```

- Compiler creates a class with methods
  - `equals`
  - `compareTo`
  - `ordinal`
  - `toString`
  - `valueOf`

Enumeration as a Class

- View enhanced enumeration, listing 9.22
  `enum Suit`
- Note
  - Instance variables
  - Additional methods
  - Constructor

Packages: Outline

- Packages and Importing
- Package Names and Directories
- Name Clashes

Packages and Importing

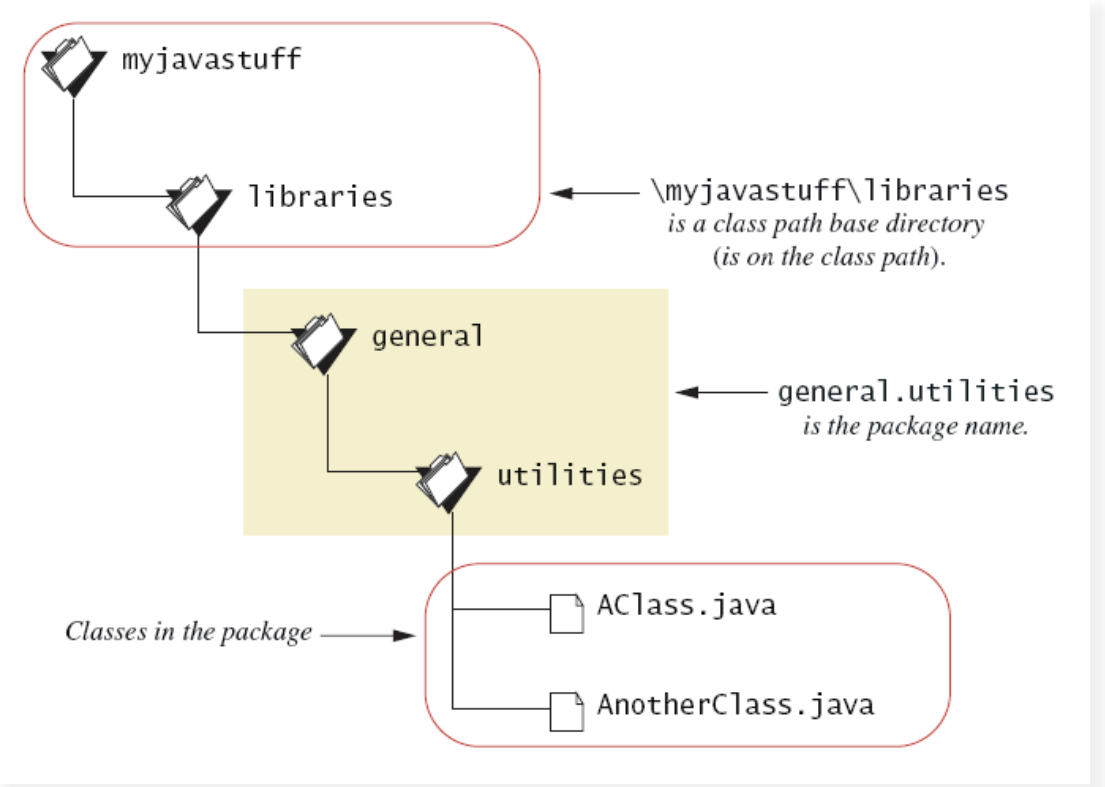
- A package is a collection of classes grouped together into a folder
- Name of folder is name of package
- Each class
  - Placed in a separate file
  - Has this line at the beginning of the file
    package `Package_Name;`
- Classes use packages by use of `import` statement

Package Names and Directories

- Package name tells compiler path name for directory containing classes of package
- Search for package begins in class path base directory
  - Package name uses dots in place of / or \
- Name of package uses relative path name starting from any directory in class path

Package Names and Directories

- Figure 9.12 A package name



Name Clashes

- Packages help in dealing with name clashes
  - When two classes have same name
- Different programmers may give same name to two classes
  - Ambiguity resolved by using the package name

Summary

- Constructor method creates, initializes object of a class
- Default constructor has no parameters
- Within a constructor use this as name for another constructor in same class
- A `static` variable shared by all objects of the class

Summary

- Primitive type has wrapper class to allow treatment as an object
- Java performs automatic type cast between primitive type and object of wrapper class as needed
- Divide method tasks into subtasks
- Test all methods individually

Summary

- Methods with same name, different signatures are overloaded methods
- Privacy leak caused by returning array corresponding to private instance variable
- An enumeration is a class – can have instance variables, constructors, methods
- A package of class definitions grouped together in same folder, contain a `package` statement at beginning of each class