More About Objects and Methods

Harald Gall, Prof. Dr.
Institut für Informatik
Universität Zürich
http://seal.ifi.uzh.ch/info1

University of Zurich
Department of Informatics

s.e.a.l.

Objectives

- learn more techniques for programming with classes and objects
- learn about static methods and static variables
- learn to define constructor methods
- learn about packages and import statements

Outline

- Programming with methods
- Static methods and static variables
- Overloading
- Constructors
- Packages

Programming with Methods - Methods Calling Methods

- A method body may contain an invocation of another method.
  - Methods invoked from method **`main`** typically involve a calling object.
  - Invocation of a method in the same class typically does not involve a calling object.

Methods Calling Methods

```
import java.util.*;

public class Oracle
{
    private String oldAnswer = "The answer is in your heart.";
    private String newAnswer;
    private String question;

    public void dialog()
    {
        String ans;
        Scanner keyboard = new Scanner(System.in);
        do
        {
            answerOne();
            System.out.println("Do you wish to ask another question?");
            ans = keyboard.next();
        } while (ans.equalsIgnoreCase("yes"));

        System.out.println("The oracle will now rest.");
    }

    private void answerOne()
    {
        System.out.println("I am the oracle.");
        System.out.println("I will answer any one-line question.");
        System.out.println("What is your question?");
        Scanner keyboard = new Scanner(System.in);
        question = keyboard.nextLine();

        seekAdvice();
        System.out.println("You asked the question:");
        System.out.println(question);
        System.out.println("Now, here is my answer:");
        System.out.println(oldAnswer);
        update();
    }

    private void seekAdvice()
    {
        System.out.println("Hmm, I need some help on that.");
        System.out.println("Please give me one line of advice.");
        Scanner keyboard = new Scanner(System.in);
        newAnswer = keyboard.nextLine();
        System.out.println("Thank you. That helped a lot.");
    }

    private void update()
    {
        oldAnswer = newAnswer;
    }
}
```

Display 5.1
Methods Calling Other Methods

Methods Calling Methods

- Method **`main`** invokes method **`dialog`** in the class **`Oracle`** using object **`delphi`**
- Within the same class, the name of the calling object is omitted.
  - Method **`dialog`** invokes method **`answerOne`** in the same class.
  - Method **`answerOne`** invokes method **`seekAdvice`** and method **`update`** in the same class.

Methods Calling Methods, cont.

- Omission of the calling object and the dot applies only when the calling object can be expressed with the `this` parameter.

The `null` Constant

- When the compiler requires an object reference to be initialized, set it to `null`

```
String line = null;
```

- `null` is not an object, but a constant that indicates that an object variable references no object
- `==` and `!=` (rather than method `equals`) are used to determine if an object variable has the value `null`
- An object reference initialized to `null` cannot be used to invoke methods in the object's class
  - An attempt results in a null pointer exception!

Static Methods and Static Variables: Outline

Static Methods
Static Variables
The `Math` Class
`Integer`, `Double`, and Other Wrapper Classes

Static Methods and Static Variables

- Static methods and static variables belong to a class and do not require any object.

Static Methods

- Some methods have no meaningful connection to an object.
- For example,
  - finding the maximum of two integers
  - computing a square root
  - converting a letter from lowercase to uppercase
  - generating a random number
- Such methods can be defined as static.

Static Methods, cont.

- A static method is still defined as a member of a class
- But, the method is invoked using the class name rather than an object name
- Nothing can refer to a calling object; no instance variables can be accessed.
- Syntax

```
return_Type Variable_Name =
Class_Name.Static_Method_Name (Parameters);
```

Class `Circle` with static methods

```
/**
Class with static methods to perform calculations on circles.
*/
public class CircleFirstTry
{
    public static final double PI = 3.14159;

    public static double area(double radius)
    {
        return (PI*radius*radius);
    }

    public static double circumference(double radius)
    {
        return (PI*(radius + radius));
    }
}
```

Later in the chapter, we will give an alternate version of this class.

Display 5.3
Static Methods

Class `CircleDemo`

```
import java.util.*;
public class CircleDemo
{
    public static void main(String[] args)
    {
        double radius;
        System.out.println(
            "Enter the radius of a circle in inches:");
        Scanner keyboard = new Scanner(System.in);
        radius = keyboard.nextDouble();
        System.out.println("A circle of radius "
                                + radius + " inches");
        System.out.println("has an area of " +
            CircleFirstTry.area(radius) + " square inches,");
        System.out.println("and a circumference of " +
            CircleFirstTry.circumference(radius) + " inches.");
    }
}
```

Sample Screen Dialog

```
Enter the radius of a circle in inches:
2.3
A circle of radius 2.3 inches
has an area of 16.61901 square inches,
and a circumference of 14.45131 inches.
```

Display 5.4
Using Static Methods

Mixing Static and Nonstatic Methods

```
import java.util.*;

public class PlayCircle
{
    public static final double PI = 3.14159;

    private double diameter;

    public void setDiameter(double newDiameter)
    {
        diameter = newDiameter;
    }

    public static double area(double radius)
    {
        return (PI*radius*radius);
    }
    public void showArea()
    {
        System.out.println("Area is " + area(diameter/2));
    }

    public static void areaDialog()
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter dianeter of circle:");
        double newDiameter = keyboard.nextDouble();
        PlayCircle c = new PlayCircle();
        c.setDiameter(newDiameter);
        c.showArea();
    }
}
```

A static variable (used as a constant)

An instance variable

You can invoke a nonstatic method within a static method definition only if you create a calling object, such as c, by using `new`.

Display 5.5
and Nonstatic Methods

Mixing Static and Nonstatic Methods

```
public class PlayCircleDemo
{
    public static void main(String[] args)
    {
        PlayCircle circle = new PlayCircle();
        circle.setDiameter(2);
        System.out.println("If circle has diameter 2,");
        circle.showArea();

        System.out.println("Now you choose the diameter:");
        PlayCircle.areaDialog();
    }
}
```

Sample Screen Dialog

```
If circle has diameter 2,
Area is 3.14159
Now you choose the diameter:
Enter diameter of circle:
4
Area is 12.56636
```

Display 5.6
Using Static and Nonstatic Methods

Putting `main` in a class

- A class, which contains a method `main` serves two purposes:
  - It can be run as a program
  - It can be used to create objects for other classes
- A program's `main` method must be static.
- In general, don't provide a method `main` in a class definition if the class will be used only to create objects.

Putting `main` in Any Class

```
import java.util.*;
public class PlayCircle
{
    public static final double PI = 3.14159;
    private double diameter;
    public static void main(String[] args)
    {
        PlayCircle circle = new PlayCircle();
        circle.setDiameter(2);
        System.out.println("If circle has diameter 2,");
        circle.showArea();

        System.out.println("Now you choose the diameter:");
        PlayCircle.areaDialog();
    }

    public void setDiameter(double newDiameter)
    {
        diameter = newDiameter;
    }

    public static double area(double radius)
    {
        return (PI*radius*radius);
    }

    public void showArea()
    {
        System.out.println("Area is " + area(diameter/2));
    }

    public static void areaDialog()
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter the diameter of a circle:");
        double newDiameter = keyboard.nextDouble();
        PlayCircle c = new PlayCircle();
        c.setDiameter(newDiameter);
        c.showArea();
    }
}
```

In the code on the Web, this is PlayCircle2.java.

Because this main is inside the definition of the class PlayCircle, you can omit the PlayCircle. if you wish.

Display 5.7
Placing a main Method in a Class Definition

Static variables

- A class can have static variables and constants

```
public static final double PI = 3.14159;
public static int numberOfInvocations = 0;
```

- The value of a static variable can be changed by any method that can access the variable.
- Static variables generally are declared private.
  - They should be read/changed only by accessor/mutator methods.
- Every object of the class has access to the static variables via the (public) accessor and mutator methods.

Static variables, cont.

- Static variables are also called *class variables*
- The primary purpose of static variables (class variables) is to store information that relates to the class as a whole.
- Example:
  - `Car, numberOfRegisteredCars`
  - `Invoice, numberOfInvoices`

Class `StaticDemo`

```
public class StaticDemo
{
    private static int numberOfInvocations = 0;
    public static void main(String[] args)
    {
        int i;
        StaticDemo object1 = new StaticDemo();
        for (i = 1; i <=10 ; i++)
            object1.outPutCountOfInvocations();
        StaticDemo object2 = new StaticDemo();
        for (i = 1; i <=10 ; i++)
            object2.justADemoMethod();
        System.out.println("Total number of invocations = "
                           + numberSoFar());
    }
    public void justADemoMethod()
    {
        numberOfInvocations++;
        //In a real example, more code would go here.
    }
    public void outPutCountOfInvocations()
    {
        numberOfInvocations++;
        System.out.println(numberOfInvocations);
    }
    public static int numberSoFar()
    {
        numberOfInvocations++;
        return numberOfInvocations;
    }
}
```

object1 and object2 use the same static variable numberOfInvocations.

Sample Screen Dialog

```
1
2
3
4
5
6
7
8
9
10
Total number of invocations = 21
```

Display 5.8
A Static Variable (Optional)

The `Math` Class

- The predefined class `Math` provides several standard mathematical methods.
  - All of these methods are `static` methods.
  - You do not need to create an object to call the methods of the `Math` class.
  - These methods are called by using the class name (`Math`) followed by a dot and a method name.

```
Return_Value =
 Math.Method_Name(Parameters);
```


The `Math` Class, cont.

Name	Description	Type of Argument	Type of Value Returned	Example	Value Returned
pow	Powers	double	double	Math.pow(2.0,3.0)	8.0
abs	Absolute value	int, long, float, or double	Same as the type of the argument	Math.abs(7) Math.abs(7) Math.abs(3.5)	7 7 3.5
max	Maximum	int, long, float, or double	Same as the type of the arguments	Math.max(5, 6) Math.max(5.5, 5.3)	6 5.5
min	Minimum	int, long, float, or double	Same as the type of the arguments	Math.min(5, 6) Math.min(5.5, 5.3)	5 5.3
round	Rounding	float or double	int or long, respectively	Math.round(6.2) Math.round(6.8)	6 7
ceil	Ceiling	double	double	Math.ceil(3.2) Math.ceil(3.9)	4.0 4.0
floor	Floor	double	double	Math.floor(3.2) Math.floor(3.9)	3.0 3.0
sqrt	Square root	double	double	sqrt(4.0)	2.0

Display 5.9
Static Methods in the Class Math

`Circle` using `Math`

```
/**
 Class with static methods to perform calculations on circles.
*/
public class Circle
{
    public static double area(double radius)
    {
        return (Math.PI*radius*radius);
    }

    public static double circumference(double radius)
    {
        return (Math.PI*(radius + radius));
    }
}
```

`CircleDemo2.java` in the source code on the Web is a demonstration program for this class.

This class behaves the same as the class `CircleFirstTry` in Display 5.3. This version differs only in that it uses the predefined constant `Math.PI`, rather than defining `PI` within the class.

Display 5.10
Predefined Constants

Integer, Double, and Other Wrapper Classes

- Sometimes a primitive value needs to be passed as an argument, but the method definition creates an object as the corresponding formal parameter.
- Java's *wrapper classes* convert a value of a primitive type to a corresponding class type.

```
Integer n = new Integer(42);
```

  - The instance variable of the object `n` has the value 42.

Integer, Double, and Other Wrapper Classes, cont.

- To retrieve the integer value

```
int i = n.intValue();
```

primitive type	wrapper extraction class	method
`int`	`Integer`	`intValue`
`long`	`Long`	`longValue`
`float`	`Float`	`floatValue`
`double`	`Double`	`doubleValue`
`char`	`Character`	`charValue`

Shorthand in Java

- Wrapping is done automatically in Java

```
Integer n = 42;
```

  which is equivalent to

```
Integer n = new Integer(42);
```

- Similarly

```
int i = n;
```

  is equivalent to

```
int i = n.intValue;
```

Automatic Boxing and Unboxing

- Converting a value of a primitive type to an object of its corresponding wrapper class is called *boxing*.

```
Integer n = new Integer(42);
```

- Java boxes automatically.

```
Integer n = 42;
```


Automatic Boxing and Unboxing..

- Converting an object of a wrapper class to a value of the corresponding primitive type is called *unboxing*.

```
int i = n.intValue;
```

- Java unboxes automatically.

```
int i = n;
```


Automatic Boxing and Unboxing..

- Automatic boxing and unboxing also apply to parameters.
  - A primitive argument can be provided for a corresponding formal parameter of the associated wrapper class.
  - A wrapper class argument can be provided for a corresponding formal parameter of the associated primitive type.

Useful Constants

- Wrapper classes contain several useful constants and static methods such as

```
Integer.MAX_VALUE
Integer.MIN_VALUE
Double.MAX_VALUE
Double.MIN_VALUE
```


Type Conversions

- Static methods in the wrapper classes can be used to convert a string to the corresponding number of type int, long, float, or double.

```
String theString = "199.98";
double doubleSample =
       Double.parseDouble(theString);
```

or

```
Double.parseDouble(theString.trim());
```

if the string has leading or trailing whitespace.

Converting Strings to Numbers

- Methods for converting strings to the corresponding numbers

```
Integer.parseInt("42")
Long.parseLong("42")
Float.parseFloat("199.98")
Double.parseDouble("199.98")
```

Converting Numbers to Strings

- Methods for converting strings to the corresponding numbers

```
Integer.toString(42)
Long.toString(42)
Float.toString(199.98)
Double.toString(199.98)
```


Designing Methods: Outline

- Formatting Output
- Top-Down Design
- Testing Methods

Case Study: Formatting Output

- `System.out.println` with a parameter of type double might print

```
Your cost is $19.981123576432
```

  when what you really want is

```
Your cost is $19.98
```

- Java provides classes for formatting output, but it is instructive, and perhaps even easier, to program them ourselves.

DollarsFirstTry

```
public class DollarsFirstTry
{
    /**
     Outputs amount in dollars and cents notation.
     Rounds after two decimal points.
     Does not advance to the next line after output.
    */
    public static void write(double amount)
    {
        int allCents = (int)(Math.round(amount*100));
        int dollars = allCents/100;
        int cents = allCents%100;
        System.out.print('$');
        System.out.print(dollars);
        System.out.print('.');
        if (cents < 10)
        {
            System.out.print('0');
            System.out.print(cents);
        }
        else
            System.out.print(cents);
    }
```

```
    /**
     Outputs amount in dollars and cents notation.
     Rounds after two decimal points.
     Advances to the next line after output.
    */
    public static void writeln(double amount)
    {
        write(amount);
        System.out.println();
    }
}
```

Display 5.12
The DollarsFirstTry

DollarsFirstTryDriver

```
import java.util.*;
public class DollarsFirstTryDriver
{
    public static void main(String[] args)
    {
        double amount;
        String ans;
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Testing DollarsFirstTry.write:");
        do
        {
            System.out.println("Enter a value of type double:");
            amount = keyboard.nextDouble();
            DollarsFirstTry.write(amount);
            System.out.println();
            System.out.println("Test again?");
            ans = keyboard.next();
        }while (ans.equalsIgnoreCase("yes"));
        System.out.println("End of test.");
    }
}
```

This kind of testing program is often called a driver program.

Sample Screen Dialog

```
Testing DollarsFirstTry.write:
Enter a value of type double:
1.2345
$1.23
Test again?
yes
Enter a value of type double:
1.235
$1.24
Test again?
yes
Enter a value of type double:
9.02
$9.02
Test again?
yes
Enter a value of type double:
-1.20
$-1.20
Test again?
no
```

OOPS. There's a problem here.

Display 5.13
Testing a Method

Testing Methods

- A driver program is useful for testing one method or class under development.
  - Its job is to invoke and test one developing method or class.
  - After the method or class is tested adequately, the driver program can be discarded.

Bottom-Up Testing & Stubs

- If method A uses method B, then method B should be tested fully before testing method A.
- Testing all the "lower level" methods invoked by an "upper level" method before the "upper level" method is tested is called *bottom-up testing*.
- A *stub* is a simplified version of a method that is good enough for testing purposes, even though it is not good enough for the final class definition.

Overloading

- Different classes can have methods with the same names.
- Two or more methods in the same class class can be defined with the same name
  - if the parameter list can be used to determine which method is being invoked.
  - This useful ability is called overloading.

Overloading, cont.

```
/**
 This is just a toy class to illustrate overloading.
*/
public class Statistician
{
    public static void main(String[] args)
    {
        double average1 = Statistician.average(40.0, 50.0);
        double average2 = Statistician.average(1.0, 2.0, 3.0);
        char average3 = Statistician.average('a', 'c');

        System.out.println("average1 = " + average1);
        System.out.println("average2 = " + average2);
        System.out.println("average3 = " + average3);
    }
    public static double average(double first, double second)
    {
        return ((first + second)/2.0);
    }
    public static double average(double first,
                                 double second, double third)
    {
        return ((first + second + third)/3.0);
    }
    public static char average(char first, char second)
    {
        return (char)((((int)first + (int)second)/2);
    }
}
```

Sample Screen Dialog

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

Display 5.15 Overloading

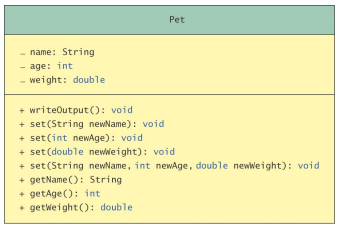
Overloading, cont.

- The number of arguments and the types of the arguments determine, which method `average` is invoked.
  - if there is no match, Java attempts simple type conversions
  - if there is still no match, an error message is produced.

Overloading, cont.

- Overloading can be applied to all kinds of methods:
  - `void` methods,
  - methods that return a value,
  - static methods
  - non-static methods, or any combination
- Examples
  - method `max` (from the `Math` class)
  - method `println`
  - the `/` operator

Programming Example



Display 5.16
Class Diagram for Pet Class

Programming Example, cont.

```
/**
 Class for basic pet records: name, age, and weight.
*/
public class Pet
{
    private String name;
    private int age; //in years
    private double weight; //in pounds

    /**
     This main is just a demonstration program.
    */
    public static void main(String[] args)
    {
        Pet myDog = new Pet();
        myDog.set("Fido", 2, 5.5);
        myDog.writeOutput();
        System.out.println("Changing name.");
        myDog.set("Rex");
        myDog.writeOutput();
        System.out.println("Changing weight.");
        myDog.set(6.5);
        myDog.writeOutput();
        System.out.println("Changing age.");
        myDog.set(3);
        myDog.writeOutput();
    }

    public void writeOutput()
    {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age + " years");
        System.out.println("Weight: " + weight + " pounds");
    }

    public void setCString newName)
    {
        name = newName;
        //age and weight are unchanged.
    }
    public void set(int newAge)
    {
        if (newAge <= 0)
        {
            System.out.println("Error: invalid age.");
            System.exit(0);
        }
        else
            age = newAge;
        //name and weight are unchanged.
    }
    public void set(double newWeight)
    {
        if (newWeight <= 0)
        {
            System.out.println("Error: invalid weight.");
            System.exit(0);
        }
        else
            weight = newWeight;
        //name and age are unchanged.
```

Display 5.17
Pet Class

Programming Example, cont.

```
public void set(String newName, int newAge, double newWeight)
{
    name = newName;
    if (newAge <= 0) || (newWeight <= 0))
    {
        System.out.println("Error: invalid age or weight.");
        System.exit(0);
    }
    else
    {
        age = newAge;
        weight = newWeight;
    }
}
public String getName()
{
    return name;
}
public int getAge()
{
    return age;
}
public double getWeight()
{
    return weight;
}
}
```

Sample Screen Dialog

```
Name: Fido
Age: 2 years
Weight: 5.5 pounds
Changing name.
Name: Rex
Age: 2 years
Weight: 5.5 pounds
Changing weight.
Name: Rex
Age: 2 years
Weight: 6.5 pounds
Changing age.
Name: Rex
Age: 3 years
Weight: 6.5 pounds
```

Display 5.17
Pet Class

Overloading and Automatic Type Conversion

- Example:

```
set(int i){ … }
set(double i) { … }

    set(10);
    set(10.0);
```

- `set(int i, double d) { … }`

```
    set(10,20);
    set(10.0,20);
```

Overloading and Automatic Type Conversion, cont.

- Example

```
public static void oops(double n1, int n2);
…
public static void oops(int n1, double n2);
```

- This will compile, but the invocation

```
sample.oops(5,10);
```

will produce an error message.

- You cannot overload a method by providing two definitions with headings that differ only in the return type.

Class Money

```
import java.util.*;
/**
 Objects represent nonnegative amounts of money,
 such as $100, $41.99, $0.05.
*/
public class Money
{
    private long dollars;
    private long cents;
    public void set(long newDollars)
    {
        if (newDollars < 0)
        {
            System.out.println(
              "Error: Negative amounts of money are not allowed.");
            System.exit(0);
        }
        else
        {
            dollars = newDollars;
            cents = 0;
        }
    }
    public void set(double amount)
    {
        if (amount < 0)
        {
            System.out.println(
              "Error: Negative amounts of money are not allowed.");
            System.exit(0);
        }
        else
        {
            long allCents = Math.round(amount*100);
            dollars = allCents/100;
            cents = allCents%100;
        }
    }
    public void set(Money otherObject)
    {
        this.dollars = otherObject.dollars;
        this.cents = otherObject.cents;
    }
    /**
     Precondition: The argument is an ordinary representation
     of an amount of money, with or without a dollar sign.
     Fractions of a cent are not allowed.
    */
    public void set(String amountString)
    {
        String dollarsString;
        String centsString;

        //Delete "$" if any:
        if (amountString.charAt(0) == '$')
            amountString = amountString.substring(1);
        amountString = amountString.trim();

        //Locate decimal point:
        int pointLocation = amountString.indexOf(".");

        if (pointLocation < 0) //If no decimal point
        {
            cents = 0;
            dollars = Long.parseLong(amountString);
        }
```

Display 5.18
Money Class

Class Money

```
        else //String has a decimal point.
        {
            dollarsString =
                amountString.substring(0, pointLocation);
            centsString =
                amountString.substring(pointLocation + 1);
            if (centsString.length() == 1)
            //If one digit meaning tenths of a dollar
                centsString = centsString + "0";

            dollars = Long.parseLong(dollarsString);
            cents = Long.parseLong(centsString);
            if ((dollars < 0) || (cents < 0) || (cents > 99))
            {
                System.out.println(
                  "Error: Illegal representation of money.");
                System.exit(0);
            }
        }
    }
    public void readInput()
    {
        System.out.println("Enter amount on a line by itself:");
        Scanner keyboard = new Scanner(System.in);
        String amount = keyboard.nextLine();
        set(amount.trim());
    }
    /**
     Does not go to the next line after outputting money.
    */
    public void writeOutput()
    {
        System.out.print("$" + dollars);
        if (cents < 10)
            System.out.print("0" + cents);
        else
            System.out.print("." + cents);
    }
    /**
     Returns n times the calling object.
    */
    public Money times(int n)
    {
        Money product = new Money();
        product.cents = n*cents;
        long carryDollars = product.cents/100;
        product.cents = product.cents%100;
        product.dollars = n*dollars + carryDollars;
        return product;
    }
    /**
     Returns the sum of the calling object and the argument.
    */
    public Money add(Money otherAmount)
    {
        Money sum = new Money();
        sum.cents = this.cents + otherAmount.cents;
        long carryDollars = sum.cents/100;
        sum.cents = sum.cents%100;
        sum.dollars = this.dollars
                    + otherAmount.dollars + carryDollars;
        return sum;
    }
}
```

We used nextLine instead of next because there may be a space between the dollar sign and the number.

Display 5.18
Money Class

Constructors

- Creating objects with parameters and/or initializations
- When you create an object of a class, often you want certain initializing actions performed such as giving values to the instance variables.
- A *constructor* is a special method that performs initializations.

Defining Constructors

- New objects are created using

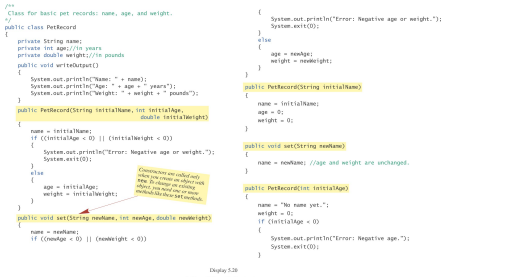
```
Class_Name Object_Name =
  new Class_Name (Parameter(s));
```

- A constructor is called automatically when a new object is created.
  - `Class_Name (Parameter(s)` calls the constructor and returns a reference.
  - It performs any actions written into its definition including initializing the values of (usually all) instance variables.

Defining Constructors, cont.

- Each constructor has the same name as its class.
- A constructor does not have a return type, not even `void`
- Constructors often are overloaded, each with a different number of parameters or different types of parameters.
- Typically, at least one constructor, the *default constructor*, has no parameters.

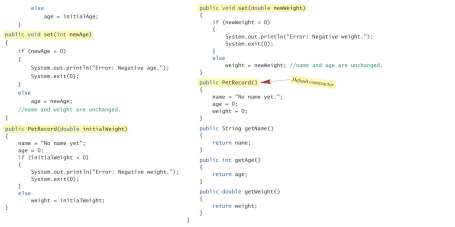
Defining Constructors, cont.



Display 5.20
PetRecord Class with Constructors

Defining Constructors, cont.

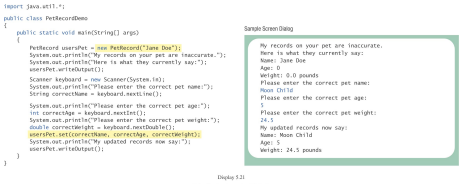
- `class PetRecord, contd.`



Display 5.20
PetRecord Class with Constructors

Defining Constructors, cont.

- `class PetRecordDemo`



Display 5.21
Using Constructors and set Methods

Defining Constructors, cont.

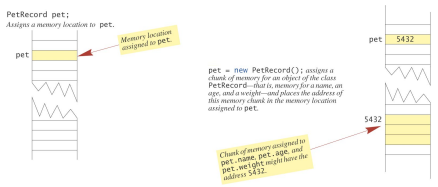
- When a class definition does not have a constructor, Java creates a default constructor automatically.
- Once you define at least one constructor for the class, no additional constructor is created automatically.
- A constructor can be called only when you create a new object.

```
newborn.PetRecord("Fang", 1, 150.0); // invalid
```

- After an object is created, a `set` method is needed to change the value(s) of one or more instance variables.

```
newBorn.set("Fang", 1, 150.0); // valid
```


Returning a Reference



Display 5.22
Constuctor Returning a Reference

Using Methods in a Constructor

- Other methods in the same class can be used in the definition of a constructor.
- Calls to one or more `set` methods are common.

```
public Class_Name(parameter(s));
{
  set(…)
}
```

Wrapper Classes with No Default Constructor

- The wrapper classes

```
Byte        Float
Short       Double
Integer     Character
Long        Boolean
```

  have no default constructors.
- When creating a new object of one of these classes, an argument is needed.

```
Character myMark = new Character('Z);
```


Packages: Outline

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

Packages

- A *package* groups and names a collection of related classes.
  - It can serve as a library of classes for any program.
  - The collection of classes need not reside in the same directory as a program that uses them.
- The classes are grouped together in a directory and are given a package name.
- Each file contains the following at the start of the file:

```
Package general.utilities;
```

Importing

- A program or class definition can use all the classes in a package by placing a suitable `import` statement at the start of the file containing the program or class definition.

```
import Package_Name;
```

  - This is sufficient even if the program or class definition is not in the same directory as the classes in the package.

Package Names and Directories

- The package name must tell the compiler the path name for the directory containing the classes and the name of the package
- The value of the *class path variable* tells Java where to begin its search for the package.
  - The class path variable is part of the operating system, not part of Java.
  - It contains path names and a list of directories, called the *class path base directories*

Package Names and Directories..

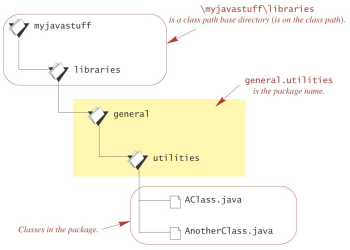
- The package name is a relative path name that assumes you start in a class path base directory and follow the path of subdirectories given by the package name.
  - example class path base directory:

```
\javastuff\libraries
```

  - example package classes

```
\javastuff\libraries\general\utilities
```

Package Names and Directories…



Display 5.25
A Package Name

Package Names and Directories…

- The class path variable allows you to list more than one base directory, separating them with a semicolon.
  - Example: `\javastuff\libraries;f:\morejavastuff`
- When you set or change the class path variable, include the *current directory* (where your program or other class is located) e
  - Example: `\javastuff\libraries;f:\morejavastuff;.`
  - Omitting the dot limits the locations you can use for packages and can interfere with programs that do not use packages.

Name Clashes

- Packages can help deal with *name clashes*, which are situations in which two classes have the same name.
  - Ambiguities can be resolved by using the package name.
  - Examples:

```
mypackage.CoolClass object1;
yourpackage.CoolClass object2;
```

Summary

- You have learned more techniques for programming with classes and objects.
- You have learned about static methods and static variables.
- You have learned to define constructor methods.
- You have learned about packages and import statements.