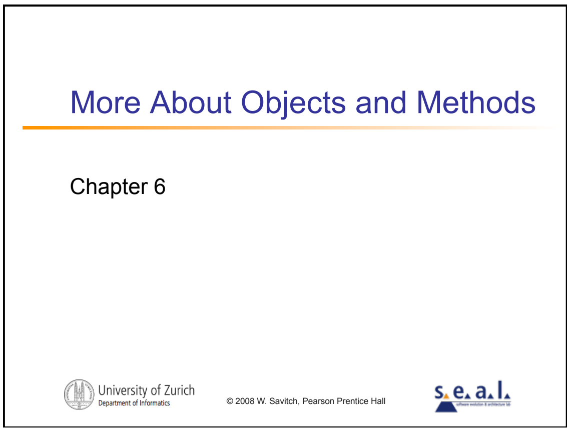
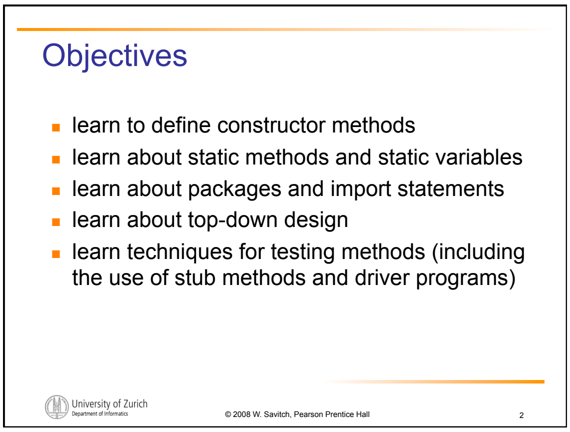


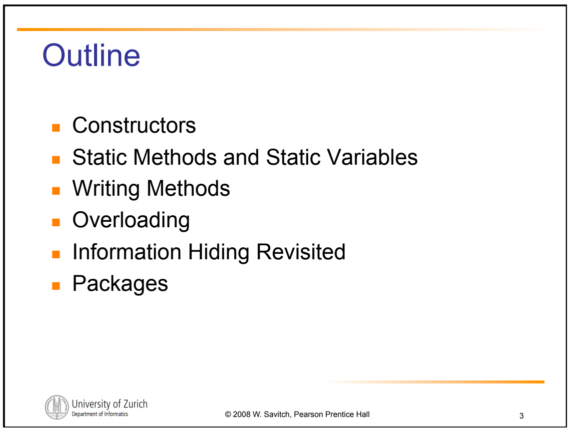
More About Objects and Methods

Chapter 6



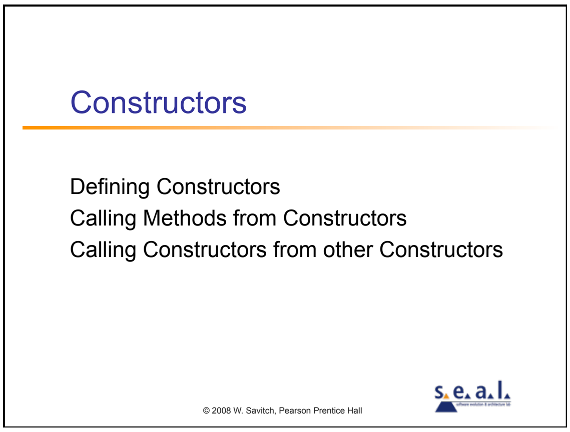
Objectives

- learn to define constructor methods
- learn about static methods and static variables
- learn about packages and import statements
- learn about top-down design
- learn techniques for testing methods (including the use of stub methods and driver programs)



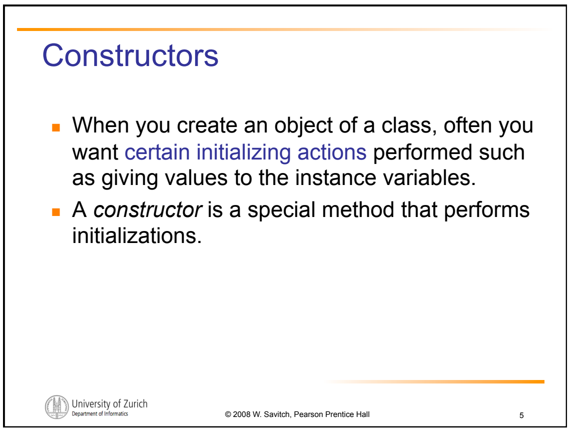
Outline

- Constructors
- Static Methods and Static Variables
- Writing Methods
- Overloading
- Information Hiding Revisited
- Packages



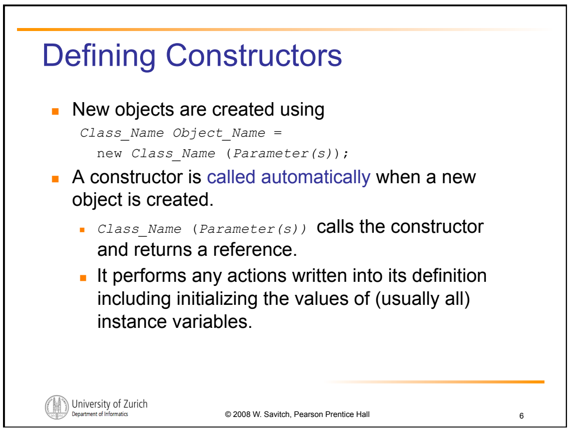
Constructors

Defining Constructors
Calling Methods from Constructors
Calling Constructors from other Constructors



Constructors

- 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));
```

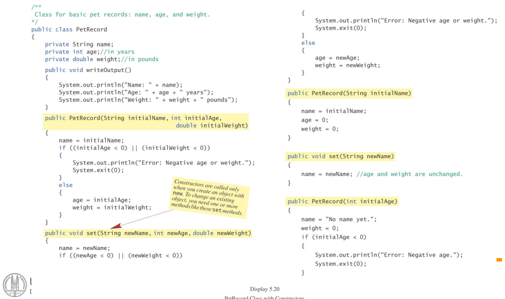
- A constructor is called automatically when a new object is created.
  - `Class_Name (Parameter(s))` calls the constructor and returns a reference.
  - It performs any actions written into its definition including initializing the values of (usually all) instance variables.

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.

- `class Pet`



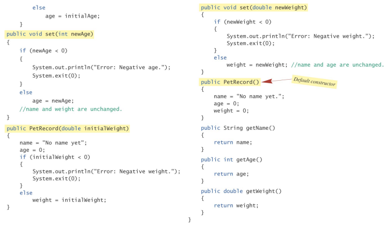
```
/**
 Class for basic pet records: name, age, and weight.
*/
public class PetRecord
{
    private String name;
    private int age;//in years
    private double weight;//in pounds

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

    public PetRecord(String initialName, int initialAge,
                                         double initialWeight)
    {
        name = initialName;
        if ((initialAge < 0) || (initialWeight < 0))
        {
            System.out.println("Error: Negative age or weight.");
            System.exit(0);
        }
        else
        {
            age = initialAge;
            weight = initialWeight;
        }
    }

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

    public PetRecord(String initialName)
    {
        name = initialName;
        age = 0;
        weight = 0;
    }

    public void set(String newName)
    {
        name = newName; //age and weight are unchanged.
    }

    public PetRecord(int initialAge)
    {
        name = "No name yet.";
        weight = 0;
        if (initialAge < 0)
        {
            System.out.println("Error: Negative age.");
            System.exit(0);
        }
```

Constructors are called only when you create an object with `new`*. To change an existing object, you need one or more methods like these* `set` *methods.*

Display 5.20
PetRecord Class with Constructors

Defining Constructors, cont.

- `class Pet, contd.`



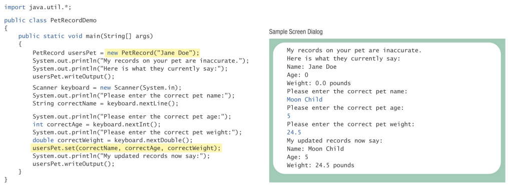
```
        else
            age = initialAge;
    }

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

    public PetRecord(double initialWeight)
    {
        name = "No name yet";
        age = 0;
        if (initialWeight < 0)
        {
            System.out.println("Error: Negative weight.");
            System.exit(0);
        }
        else
            weight = initialWeight;
    }

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

    public PetRecord()
    {
        name = "No name yet.";
        age = 0;
        weight = 0;
    }

    public String getName()
    {
        return name;
    }

    public int getAge()
    {
        return age;
    }

    public double getWeight()
    {
        return weight;
    }
}
```

Default constructor

Display 5.20
PetRecord Class with Constructors

Defining Constructors, cont.

- class PetDemo



```
import java.util.*;

public class PetRecordDemo
{
    public static void main(String[] args)
    {
        PetRecord usersPet = new PetRecord("Jane Doe");
        System.out.println("My records on your pet are inaccurate.");
        System.out.println("Here is what they currently say:");
        usersPet.writeOutput();

        Scanner keyboard = new Scanner(System.in);
        System.out.println("Please enter the correct pet name:");
        String correctName = keyboard.nextLine();

        System.out.println("Please enter the correct pet age:");
        int correctAge = keyboard.nextInt();
        System.out.println("Please enter the correct pet weight:");
        double correctWeight = keyboard.nextDouble();
        usersPet.set(correctName, correctAge, correctWeight);
        System.out.println("My updated records now say:");
        usersPet.writeOutput();
    }
}
```

Sample Screen Dialog

```
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
```

Display 5.21

Using Constructors and set Methods

Defining Constructors, cont.

- When a class definition does not have a constructor definition, Java creates a default constructor automatically.
- Once you define at least one constructor for the class, no additional constructor is created automatically.

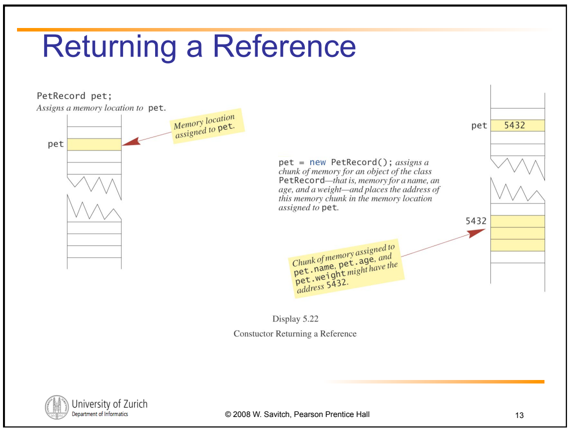
Using Constructors

- A constructor can be called only when you create a new object.

```
newborn.Pet("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

```
PetRecord pet;
```

Assigns a memory location to `pet`.

Memory location assigned to `pet`.

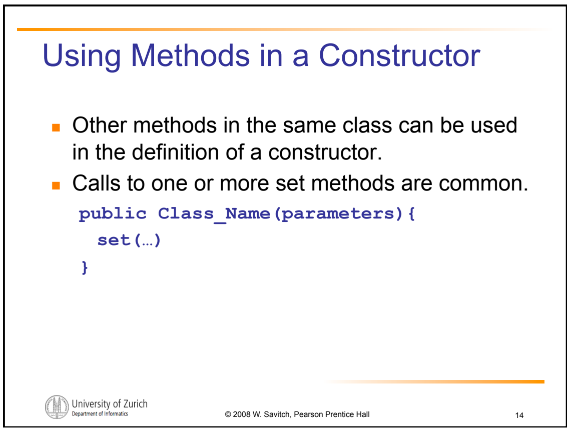
`pet = new PetRecord();` *assigns a chunk of memory for an object of the class* `PetRecord`*—that is, memory for a name, an age, and a weight—and places the address of this memory chunk in the memory location assigned to* `pet`.

Chunk of memory assigned to `pet.name`, `pet.age`, *and* `pet.weight` *might have the address* `5432`.

Display 5.22

Constuctor Returning a Reference

University of Zurich
Department of Informatics

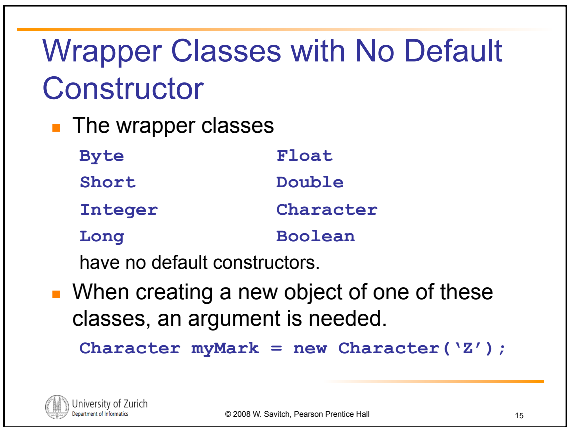


Using Methods in a Constructor

- 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(parameters){
  set(…)
}
```

University of Zurich
Department of Informatics



Wrapper Classes with No Default Constructor

- The wrapper classes

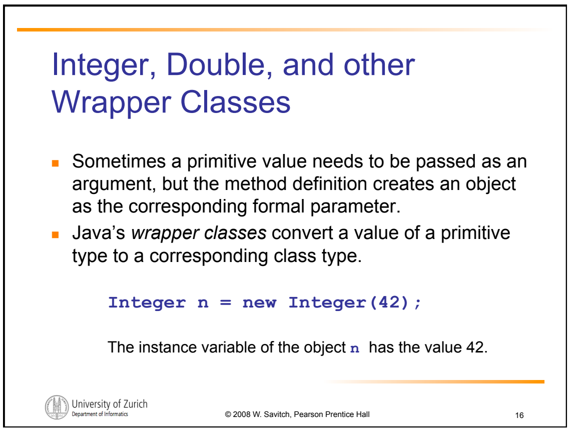
```
Byte          Float
Short         Double
Integer       Character
Long          Boolean
```

have no default constructors.

- When creating a new object of one of these classes, an argument is needed.

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

University of Zurich
Department of Informatics



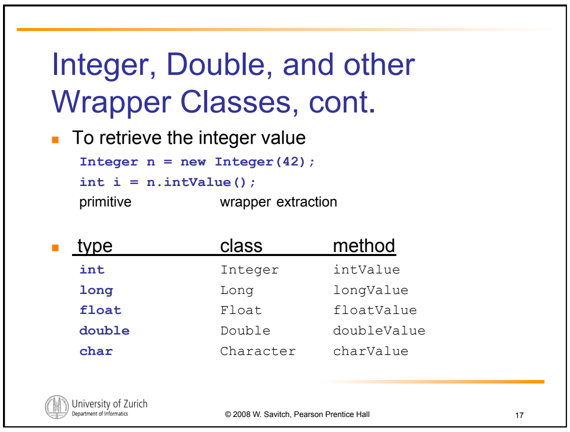
Integer, Double, and other Wrapper Classes

- 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.

University of Zurich
Department of Informatics



Integer, Double, and other Wrapper Classes, cont.

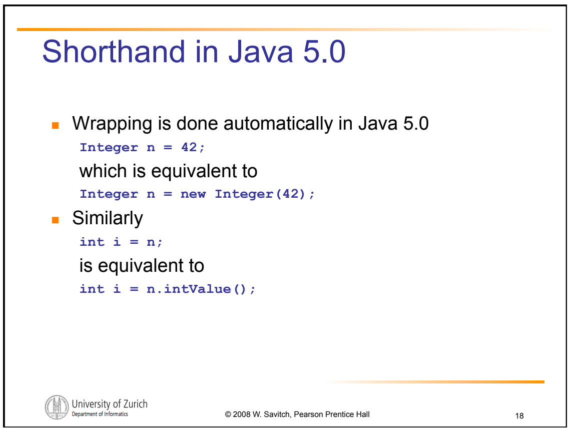
- To retrieve the integer value

```
Integer n = new Integer(42);
int i = n.intValue();
```

primitive wrapper extraction

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

University of Zurich
Department of Informatics



Shorthand in Java 5.0

- Wrapping is done automatically in Java 5.0

```
Integer n = 42;
```

which is equivalent to

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

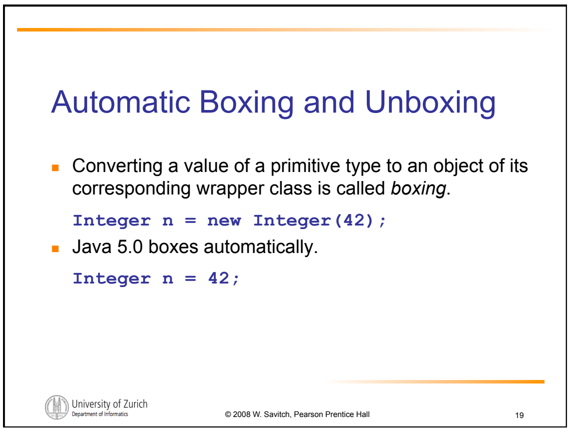
- Similarly

```
int i = n;
```

is equivalent to

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

University of Zurich
Department of Informatics



Automatic Boxing and Unboxing

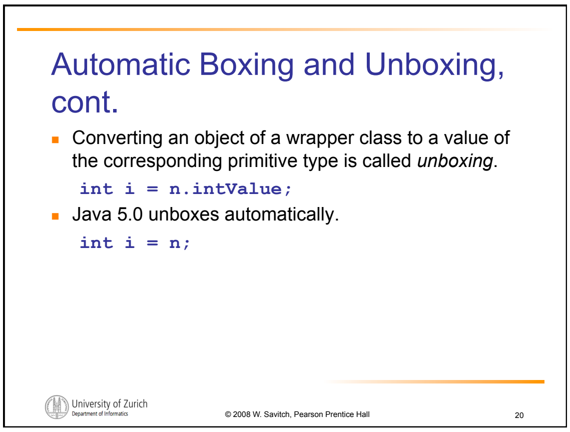
- Converting a value of a primitive type to an object of its corresponding wrapper class is called *boxing*.

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

- Java 5.0 boxes automatically.

```
Integer n = 42;
```

University of Zurich
Department of Informatics



Automatic Boxing and Unboxing, cont.

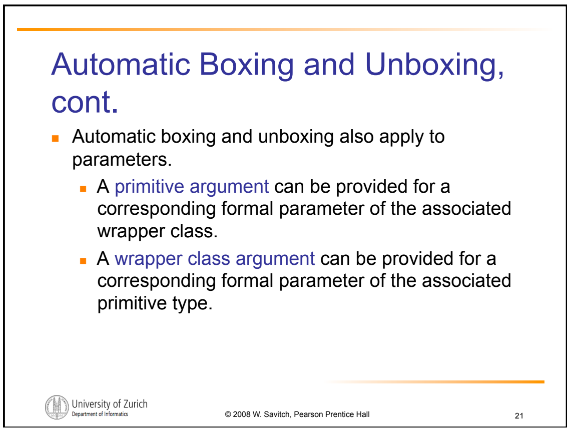
- Converting an object of a wrapper class to a value of the corresponding primitive type is called *unboxing*.

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

- Java 5.0 unboxes automatically.

```
int i = n;
```

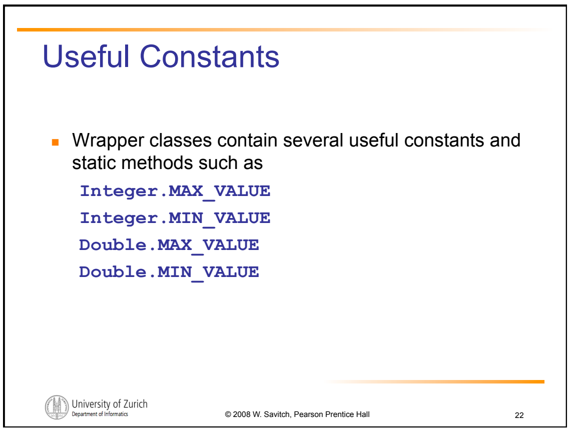
University of Zurich
Department of Informatics



Automatic Boxing and Unboxing, cont.

- 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.

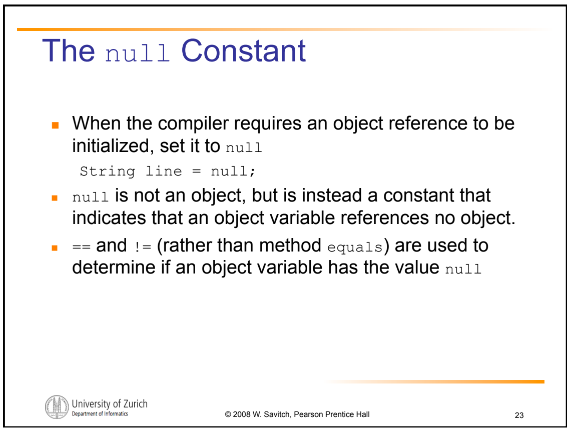
University of Zurich
Department of Informatics



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
```

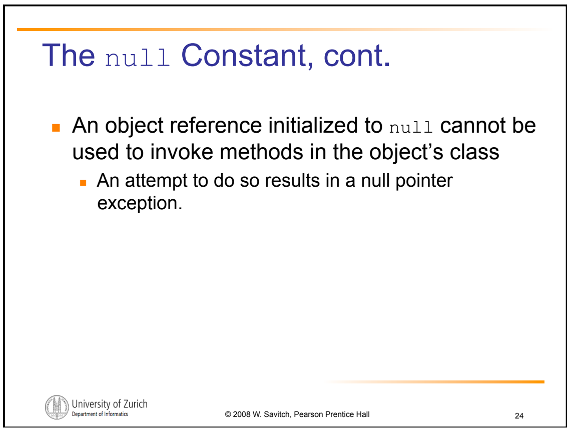


The `null` Constant

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

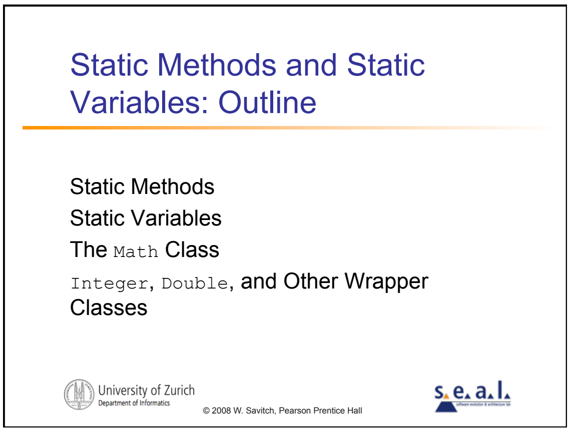
```
String line = null;
```

- `null` is not an object, but is instead 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`



The `null` Constant, cont.

- An object reference initialized to `null` cannot be used to invoke methods in the object's class
  - An attempt to do so results in a null pointer exception.

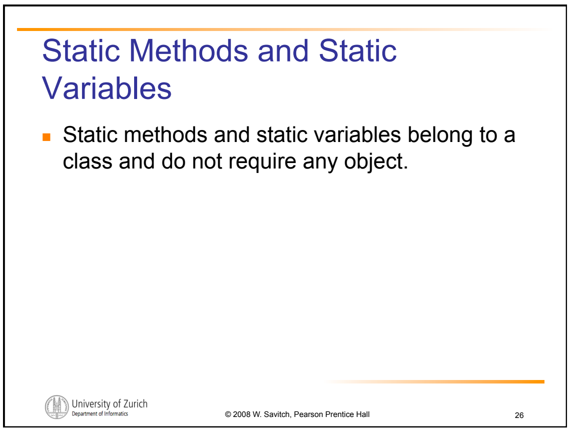


Static Methods and Static Variables: Outline

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

University of Zurich
Department of Informatics

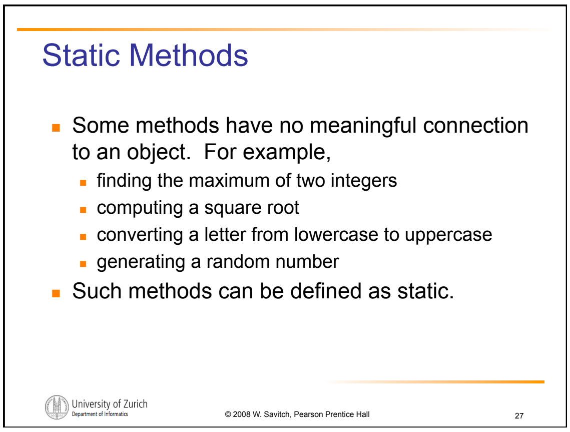
s.e.a.l.



Static Methods and Static Variables

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

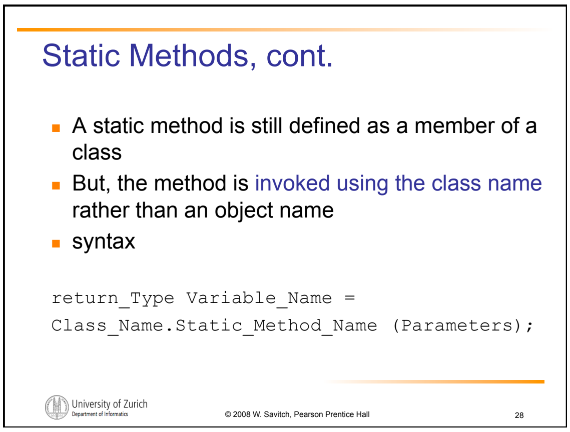
University of Zurich
Department of Informatics



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.

University of Zurich
Department of Informatics

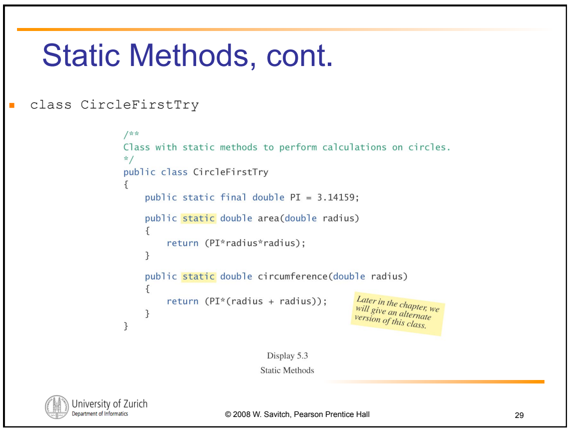


Static Methods, cont.

- 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
- syntax

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

University of Zurich
Department of Informatics



Static Methods, cont.

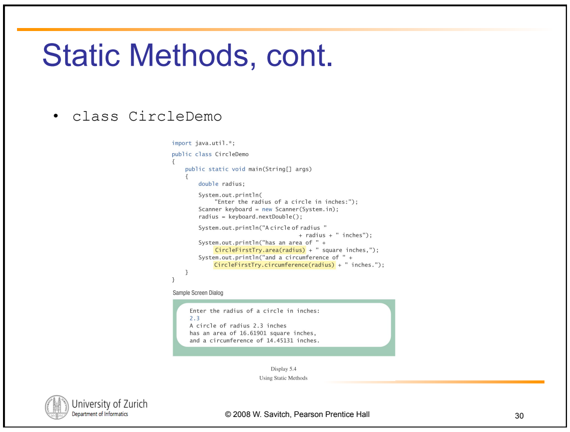
- class CircleFirstTry

```
/**
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

University of Zurich
Department of Informatics



Static Methods, cont.

- 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

University of Zurich
Department of Informatics

Defining a Static Method

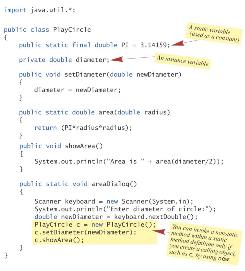
- A static method is defined in the same way as any other method, but includes the keyword `static` in the heading.

```
public static double area (double radius);
```

- Nothing in the definition can refer to a calling object; no instance variables can be accessed.

Mixing Static and Nonstatic Methods

- `class PlayCircle`



```
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 diameter 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

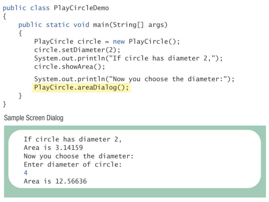
You can invoke a nonstatic method within a static method definition only if you create a calling object, such as c*, by using* new*.*

Display 5.5

Mixing Static and Nonstatic Methods

Mixing Static and Nonstatic Methods

- `class PlayCircleDemo`



```
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

Using an Object to Call a Static Method

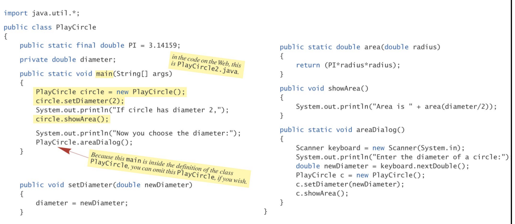
- An object of the class can be used to call a static method of the class even though it is more common to use the class name to call the static method.
- You cannot invoke a nonstatic method within a static method unless you create and use a calling object for the nonstatic method.

Putting `main` in Any 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

Putting `main` in Any Class

- `class PlayCircle`



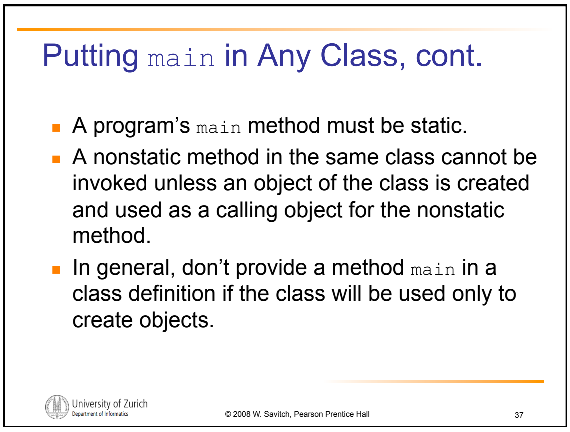
```
import java.util.*;

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 this `PlayCircle`, if you wish.

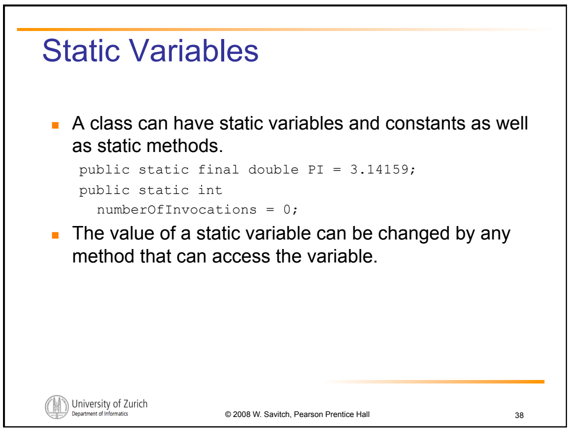
Display 5.7

Placing a main Method in a Class Definition



Putting `main` in Any Class, cont.

- A program's `main` method must be static.
- A nonstatic method in the same class cannot be invoked unless an object of the class is created and used as a calling object for the nonstatic method.
- In general, don't provide a method `main` in a class definition if the class will be used only to create objects.

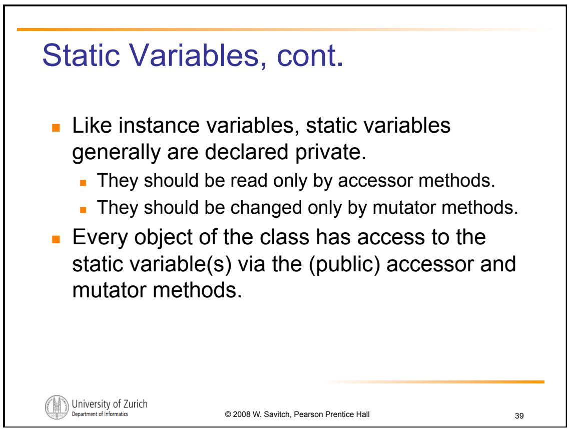


Static Variables

- A class can have static variables and constants as well as static methods.

```
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.

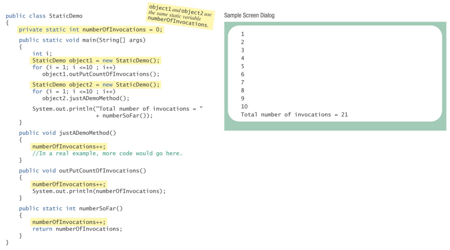


Static Variables, cont.

- Like instance variables, static variables generally are declared private.
  - They should be read only by accessor methods.
  - They should be changed only by mutator methods.
- Every object of the class has access to the static variable(s) via the (public) accessor and mutator methods.

Static Variables, cont.

- `class StaticDemo`



object1 *and* object2 *use the same static variable* numberOfInvocations.

```
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;
    }
}
```

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)

Static Variables, cont.

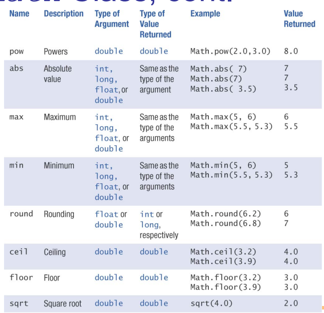
- Static variables are also called *class variables*
- The primary purpose of static variables (class variables) is to store information that relates to the class as a whole.

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

University of Zu
Department of Informatics

Display 5.9

Static Methods in the Class Math

The Math Class, cont.

- Method round returns a number as the nearest whole number.
  - If its argument is of type double, it returns a whole number of type long.
- Method floor (ceil) returns the largest (smallest) whole number that is less (greater) than or equal to its argument.

University of Zurich
Department of Informatics

The Math Class, cont.

- To store a value returned by either floor or ceil in a variable of type int, a cast must be used.

```
double start = 3.56;
int lowerBound = (int)Math.floor(start);
int upperBound =
   (int)Math.ceil(start);
```

University of Zurich
Department of Informatics

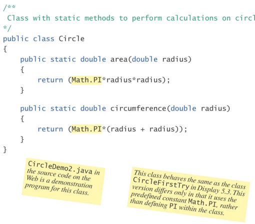
The `Math` Class, cont.

- The `Math` class has two predefined constants, `E` and `PI`.
  - example

```
area = Math.PI * radius * radius;
```

The `Math` Class, cont.

- `class Circle`



```
/**
 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

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.

Type Conversions, cont.

- Methods for converting strings to the corresponding numbers

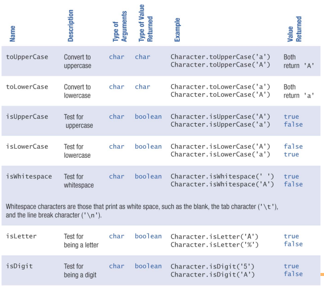
```
Integer.parseInt("42")
Long.parseLong("42")
Float.parseFloat("199.98")
Double.parseDouble("199.98")
```

Type Conversions, cont.

- Methods for converting strings to the corresponding numbers

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

Static Methods in Class `Character`



Name	Description	Type of Arguments	Type of Value Returned	Example	Value Returned

toUpperCase	Convert to uppercase	char	char	Character.toUpperCase('a')
				Character.toUpperCase('A')
					Both return 'A'

toLowerCase	Convert to lowercase	char	char	Character.toLowerCase('a')
				Character.toLowerCase('A')
					Both return 'a'

isUpperCase	Test for uppercase	char	boolean	Character.isUpperCase('A')
				Character.isUpperCase('a')
					true
					false

isLowerCase	Test for lowercase	char	boolean	Character.isLowerCase('A')
				Character.isLowerCase('a')
					false
					true

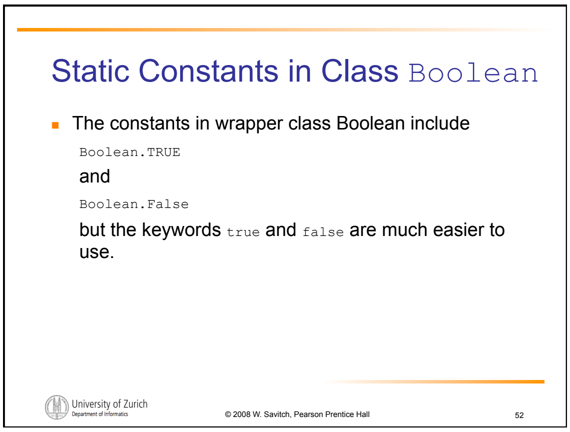
isWhitespace	Test for whitespace	char	boolean	Character.isWhitespace(' ')
				Character.isWhitespace('A')
					true
					false

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

isLetter	Test for being a letter	char	boolean	Character.isLetter('A')
				Character.isLetter('%')
					true
					false

isDigit	Test for being a digit	char	boolean	Character.isDigit('5')
				Character.isDigit('A')
					true
					false

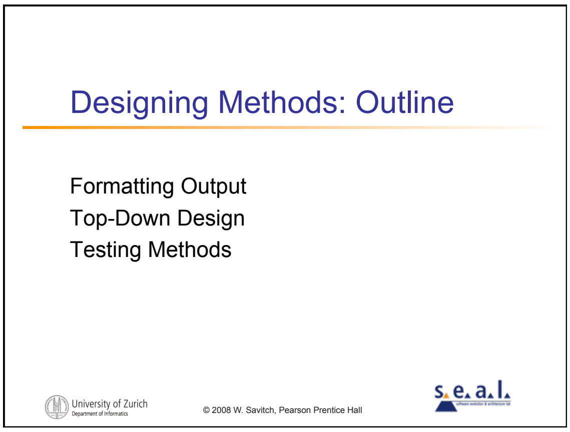
Display 5.11
Static Methods in the Class Character



Static Constants in Class `Boolean`

- The constants in wrapper class Boolean include `Boolean.TRUE` and `Boolean.False` but the keywords `true` and `false` are much easier to use.

University of Zurich
Department of Informatics

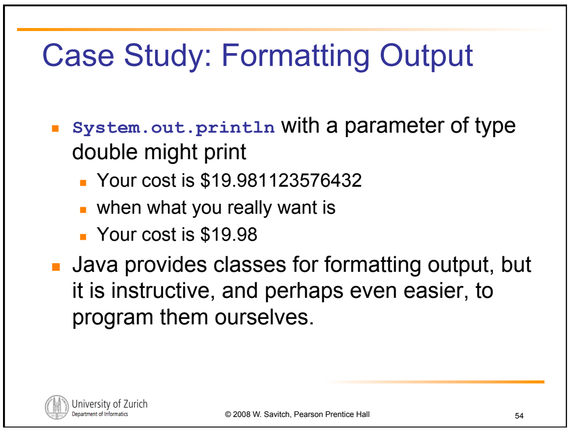


Designing Methods: Outline

Formatting Output
Top-Down Design
Testing Methods

University of Zurich
Department of Informatics

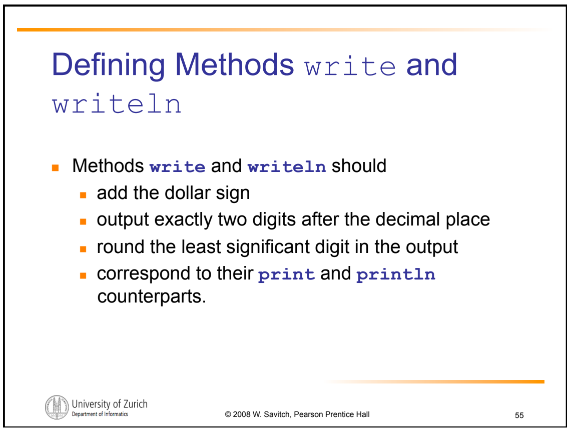
s.e.a.l.



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.

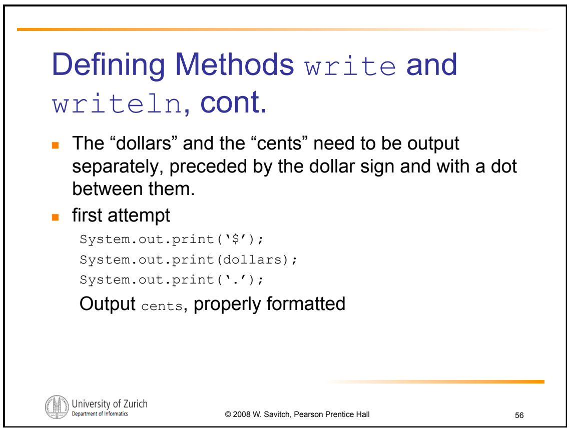
University of Zurich
Department of Informatics



Defining Methods `write` and `writeln`

- Methods **`write`** and **`writeln`** should
  - add the dollar sign
  - output exactly two digits after the decimal place
  - round the least significant digit in the output
  - correspond to their **`print`** and **`println`** counterparts.

University of Zurich
Department of Informatics



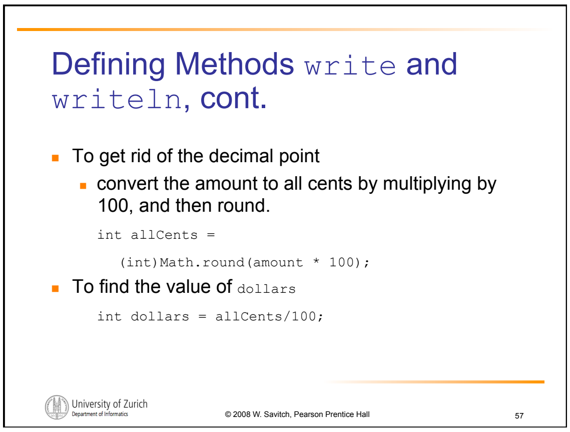
Defining Methods `write` and `writeln`, cont.

- The “dollars” and the “cents” need to be output separately, preceded by the dollar sign and with a dot between them.
- first attempt

```
System.out.print(‘$’);
System.out.print(dollars);
System.out.print(‘.’);
```

Output `cents`, properly formatted

University of Zurich
Department of Informatics



Defining Methods `write` and `writeln`, cont.

- To get rid of the decimal point
  - convert the amount to all cents by multiplying by 100, and then round.

```
int allCents =

   (int)Math.round(amount * 100);
```

- To find the value of `dollars`

```
int dollars = allCents/100;
```

University of Zurich
Department of Informatics

Defining Methods `write` and `writeln`, cont.

- To find the value of cents

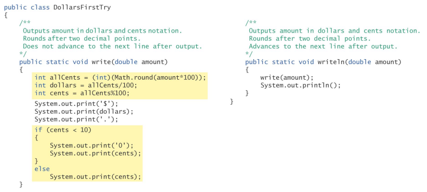
```
int cents = allCents%100;
```

- To provide a leading zero when `cents` has a value less than `10`

```
if (cents < 10)
    System.out.print(‘0’);
System.out.print(cents);
```

Defining Methods `write` and `writeln`, cont.

- `class DollarsFirstTry`

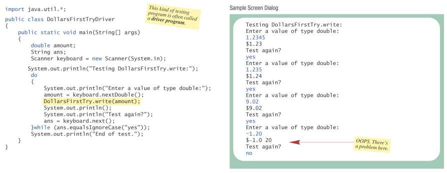


```
public class DollarsFirstTry
{
    /**
     Outputs amount in dollars and cents notation.
     Rounds after two decimal points.
     Does not advance to the next line after output.
    */
    public static void write(double amount)
    {
        int allCents = (int)(Math.round(amount*100));
        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

Defining Methods `write` and `writeln`, cont.

- `class 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
$-1.0 20
Test again?
no
```

OOPS. There's a problem here.

Display 5.13
Testing a Method

Defining Methods `write` and `writeln`, cont.

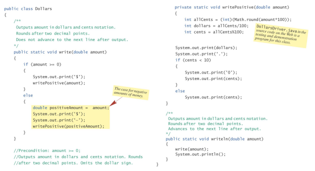
- A program used to test a method or class under development is called a *driver program*.

Defining Methods `write` and `writeln`, cont.

- Negative numbers are not handled properly by class `DollarsFirstTry:`
  `$-1.0-20` instead of `-$1.20`, for example
- To handle negative amounts, convert the amount to a positive number, output a minus sign, and output the properly formatted amount.

Defining Methods `write` and `writeln`, cont.

- `class Dollars`



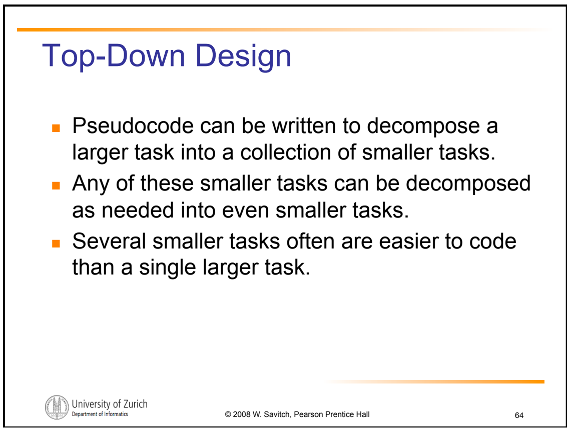
```
public class Dollars
{
    /**
     Outputs amount in dollars and cents notation.
     Rounds after two decimal points.
     Does not advance to the next line after output.
    */
    public static void write(double amount)
    {
        if (amount >= 0)
        {
            System.out.print('$');
            writePositive(amount);
        }
        else
        {
            double positiveAmount =  amount;
            System.out.print('$');
            System.out.print('-');
            writePositive(positiveAmount);
        }
    }

    //Precondition: amount >= 0;
    //Outputs amount in dollars and cents notation. Rounds
    //after two decimal points. Omits the dollar sign.
    private static void writePositive(double amount)
    {
        int allCents = (int)(Math.round(amount*100));
        int dollars = allCents/100;
        int cents = allCents%100;

        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();
    }
}
```

The case for negative amounts of money.

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

Display 5.14
The Corrected Class Dollars

- Retest after changing the definition.

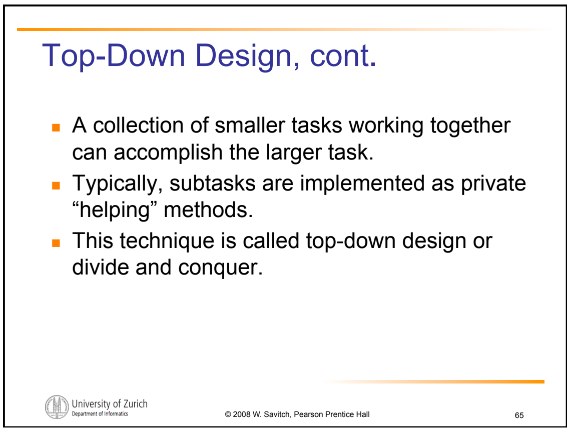


Top-Down Design

- Pseudocode can be written to decompose a larger task into a collection of smaller tasks.
- Any of these smaller tasks can be decomposed as needed into even smaller tasks.
- Several smaller tasks often are easier to code than a single larger task.

University of Zurich
Department of Informatics

64

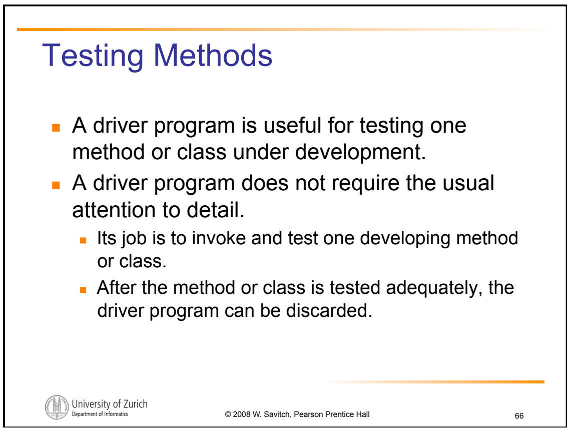


Top-Down Design, cont.

- A collection of smaller tasks working together can accomplish the larger task.
- Typically, subtasks are implemented as private “helping” methods.
- This technique is called top-down design or divide and conquer.

University of Zurich
Department of Informatics

65

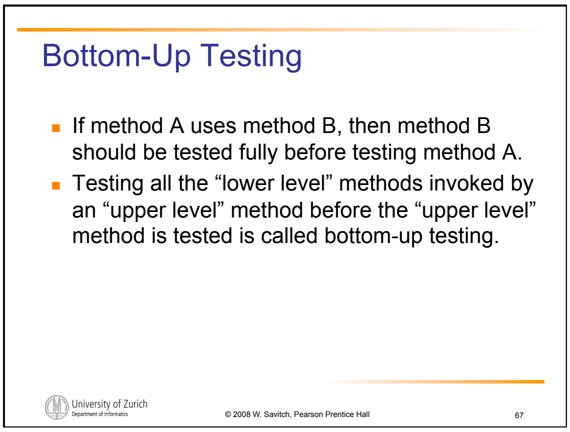


Testing Methods

- A driver program is useful for testing one method or class under development.
- A driver program does not require the usual attention to detail.
  - 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.

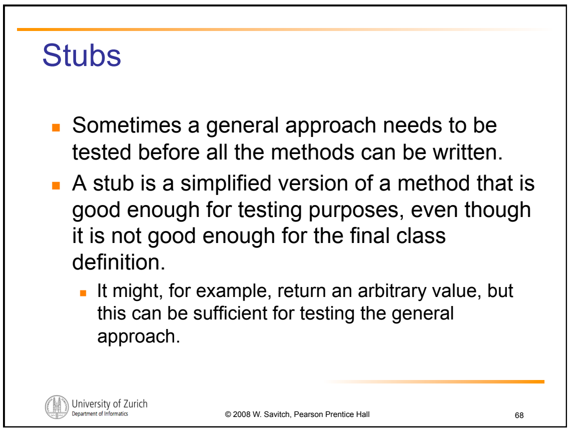
University of Zurich
Department of Informatics

66



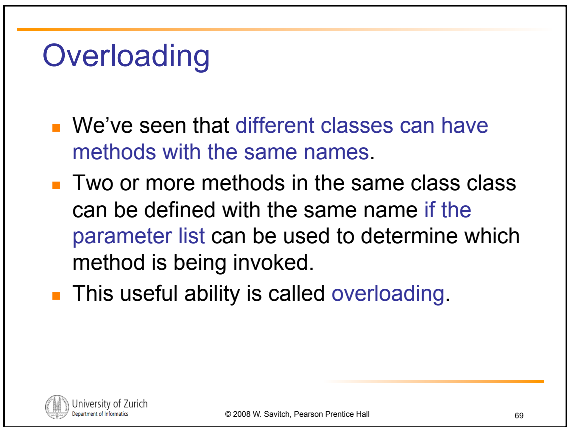
Bottom-Up Testing

- 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.



Stubs

- Sometimes a general approach needs to be tested before all the methods can be written.
- 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.
  - It might, for example, return an arbitrary value, but this can be sufficient for testing the general approach.

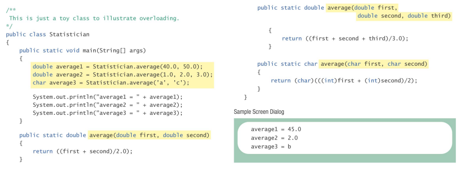


Overloading

- We've seen that 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.

- `class Statistician`



```
/**
 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 determines which method `average` is invoked.
  - If there is no match, Java attempts simple type conversions of the kinds discussed earlier.
  - If there is still no match, an error message is produced.

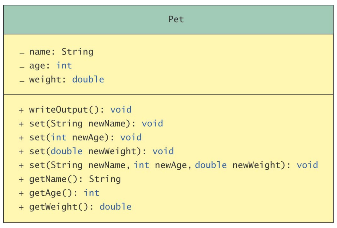
Overloading, cont.

- Overloading can be applied to all kinds of methods.
  - void methods
  - methods that return a value
  - static methods
  - nonstatic methods
  - or any combination

Overloading, cont.

- We've been using overloading, perhaps without recognizing it.
- examples
  - method `max` (from the `Math` class)
  - method `println`
  - the `/` operator

Programming Example

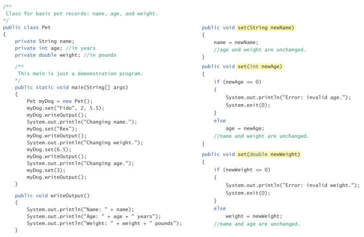


```
Pet
– name: String
– age: int
– weight: double
+ writeOutput(): void
+ set(String newName): void
+ set(int newAge): void
+ set(double newWeight): void
+ set(String newName, int newAge, double newWeight): void
+ getName(): String
+ getAge(): int
+ getWeight(): double
```

Display 5.16
Class Diagram for Pet Class

Programming Example, cont.

- `class Pet`



```
/**
 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 set(String 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.
```

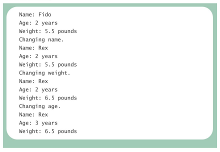
Display 5.17
Pet Class

Programming Example, cont.

- class Pet, contd.

```
    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
```

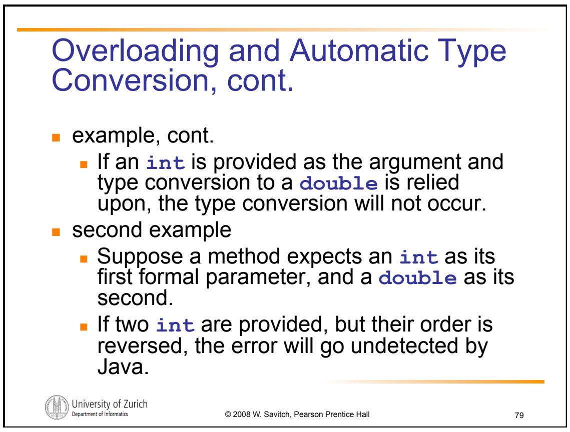
Display 5.17
Pet Class

Overloading and Automatic Type Conversion

- Overloading can be helpful.
- Automatic type conversion of arguments can be helpful.
- But, overloading and automatic type conversion can interfere with each other.

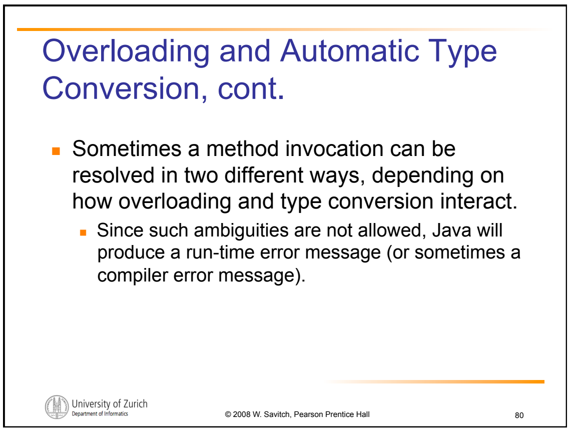
Overloading and Automatic Type Conversion, cont

- Example
  - Suppose method **set** is overloaded; one method has an **int** as its formal parameter and the other has a **double** as its formal parameter.



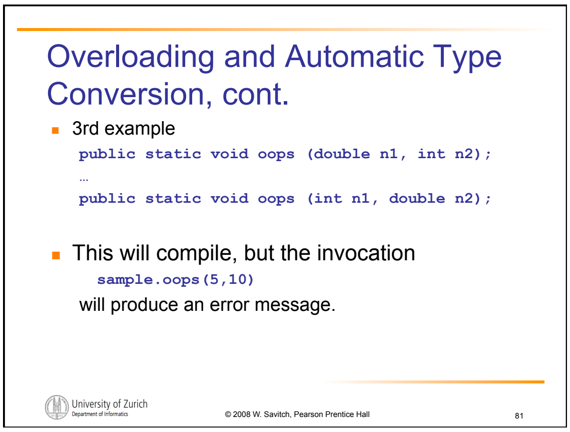
Overloading and Automatic Type Conversion, cont.

- example, cont.
  - If an `int` is provided as the argument and type conversion to a `double` is relied upon, the type conversion will not occur.
- second example
  - Suppose a method expects an `int` as its first formal parameter, and a `double` as its second.
  - If two `int` are provided, but their order is reversed, the error will go undetected by Java.



Overloading and Automatic Type Conversion, cont.

- Sometimes a method invocation can be resolved in two different ways, depending on how overloading and type conversion interact.
  - Since such ambiguities are not allowed, Java will produce a run-time error message (or sometimes a compiler error message).



Overloading and Automatic Type Conversion, cont.

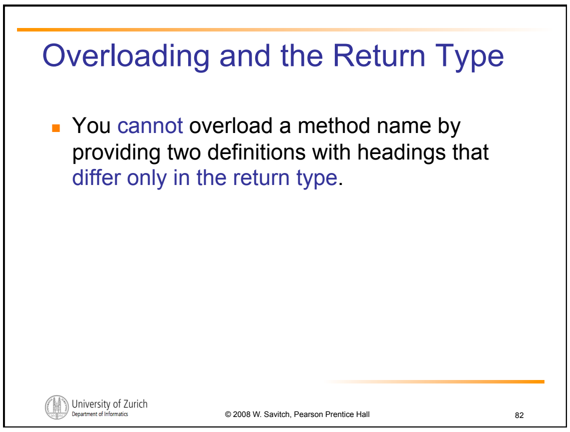
- 3rd example

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

- This will compile, but the invocation

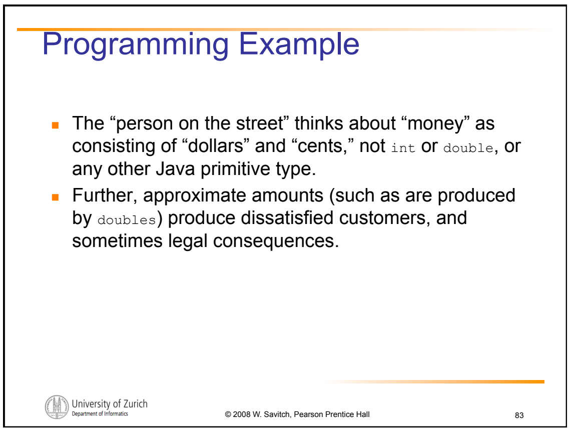
```
  sample.oops(5,10)
```

will produce an error message.



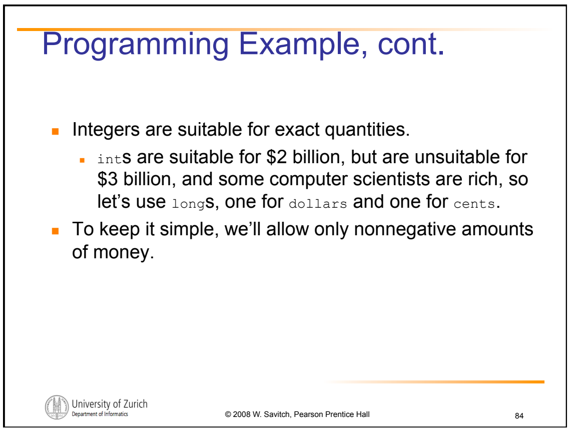
Overloading and the Return Type

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



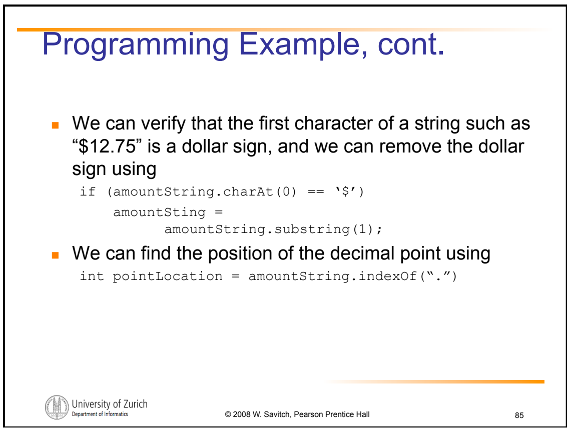
Programming Example

- The “person on the street” thinks about “money” as consisting of “dollars” and “cents,” not `int` or `double`, or any other Java primitive type.
- Further, approximate amounts (such as are produced by `doubles`) produce dissatisfied customers, and sometimes legal consequences.



Programming Example, cont.

- Integers are suitable for exact quantities.
  - `int`s are suitable for $2 billion, but are unsuitable for $3 billion, and some computer scientists are rich, so let’s use `long`s, one for `dollars` and one for `cents`.
- To keep it simple, we’ll allow only nonnegative amounts of money.



Programming Example, cont.

- We can verify that the first character of a string such as “$12.75” is a dollar sign, and we can remove the dollar sign using

```
if (amountString.charAt(0) == ‘$’)
     amountSting =
            amountString.substring(1);
```

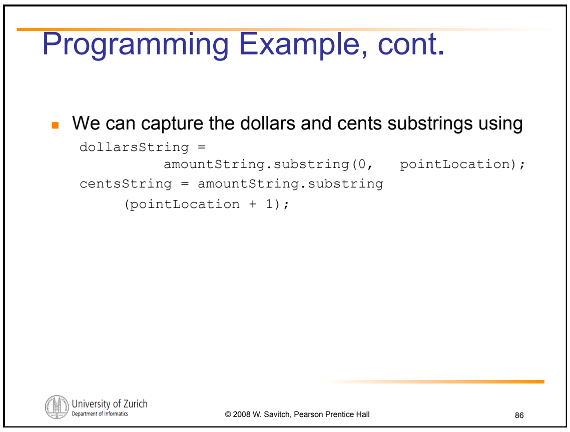
- We can find the position of the decimal point using

```
int pointLocation = amountString.indexOf(“.”)
```

University of Zurich
Department of Informatics

© 2008 W. Savitch, Pearson Prentice Hall

85



Programming Example, cont.

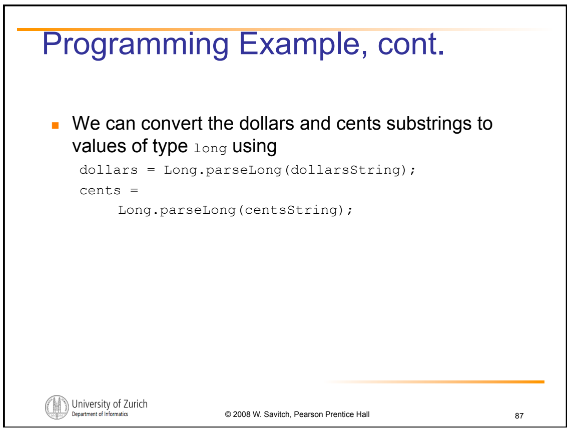
- We can capture the dollars and cents substrings using

```
dollarsString =
            amountString.substring(0,   pointLocation);
centsString = amountString.substring
      (pointLocation + 1);
```

University of Zurich
Department of Informatics

© 2008 W. Savitch, Pearson Prentice Hall

86



Programming Example, cont.

- We can convert the dollars and cents substrings to values of type `long` using

```
dollars = Long.parseLong(dollarsString);
cents =
     Long.parseLong(centsString);
```

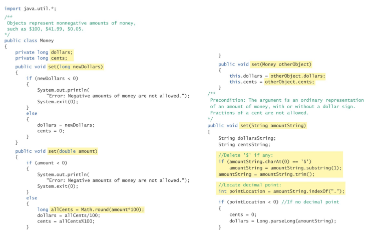
University of Zurich
Department of Informatics

© 2008 W. Savitch, Pearson Prentice Hall

87

Programming Example, cont.

- `class Money`

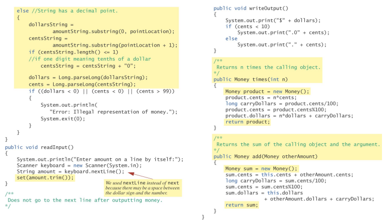


```
import java.util.*;
/**
 Objects represent nonnegative amounts of money,
 such as $100, $41.99, $0.05.
*/
public class Money
{
    private long dollars;
    private long cents;
    public void set(long newDollars)
    {
        if (newDollars < 0)
        {
            System.out.println(
      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

Programming Example, cont.

- `class Money, contd.`



```
    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

Information Hiding Revisited

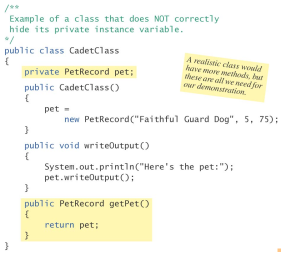
- A class can have instance variables of any type, including any class type.
- Variables of a class type contain the memory address of the associated object.
- Any change made using an instance variable of a class type indirectly, and sometimes unintentionally, affects all other references to the associated object.

Information Hiding Revisited, cont.

- If an alias can be created, the otherwise private methods of the class can be accessed.

Information Hiding Revisited, cont.

- class CadetClass



```
/**
 Example of a class that does NOT correctly
 hide its private instance variable.
*/
public class CadetClass
{
    private PetRecord pet;

    public CadetClass()
    {
        pet =
            new PetRecord("Faithful Guard Dog", 5, 75);
    }

    public void writeOutput()
    {
        System.out.println("Here's the pet:");
        pet.writeOutput();
    }

    public PetRecord getPet()
    {
        return pet;
    }
}
```

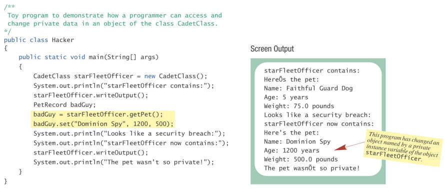
A realistic class would have more methods, but these are all we need for our demonstration.

Display 5.23

An Insecure Class

Information Hiding Revisited, cont.

- class Hacker



```
/**
 Toy program to demonstrate how a programmer can access and
 change private data in an object of the class CadetClass.
*/
public class Hacker
{
    public static void main(String[] args)
    {
        CadetClass starFleetOfficer = new CadetClass();
        System.out.println("starFleetOfficer contains:");
        starFleetOfficer.writeOutput();
        PetRecord badGuy;
        badGuy = starFleetOfficer.getPet();
        badGuy.set("Dominion Spy", 1200, 500);
        System.out.println("Looks like a security breach:");
        System.out.println("starFleetOfficer now contains:");
        starFleetOfficer.writeOutput();
        System.out.println("The pet wasn't so private!");
    }
}
```

Screen Output

```
starFleetOfficer contains:
HereÕs the pet:
Name: Faithful Guard Dog
Age: 5 years
Weight: 75.0 pounds
Looks like a security breach:
starFleetOfficer now contains:
Here's the pet:
Name: Dominion Spy
Age: 1200 years
Weight: 500.0 pounds
The pet wasnÕt so private!
```

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

Display 5.24

Changing Private Data in a Poorly Defined Class

Avoiding the Problem

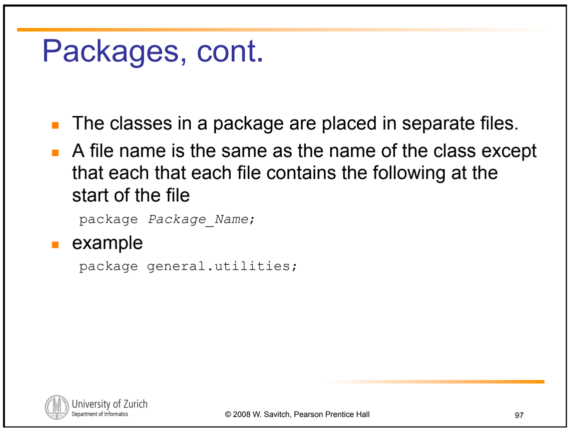
- An easy solution is to use only instance variables of a primitive type or of type String, which has no methods than can change its data.
- A harder (and better) solution produces an exact copy of the object called a clone.
  - A reference to the clone is returned instead of a reference to the object.
  - See Appendix 8 for details.

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.



Packages, cont.

- The classes in a package are placed in separate files.
- A file name is the same as the name of the class except that each that each file contains the following at the start of the file

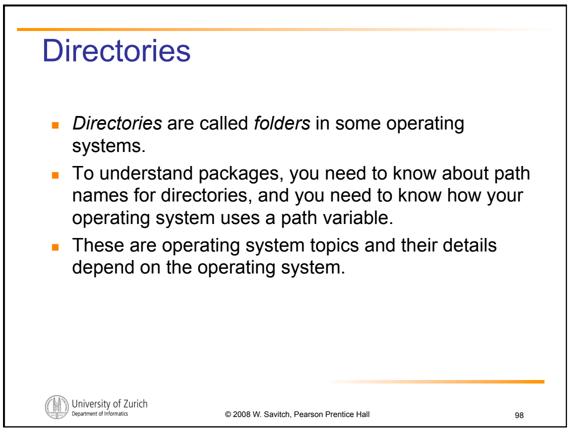
```
package Package_Name;
```

- example

```
package general.utilities;
```

University of Zurich
Department of Informatics

© 2008 W. Savitch, Pearson Prentice Hall 97

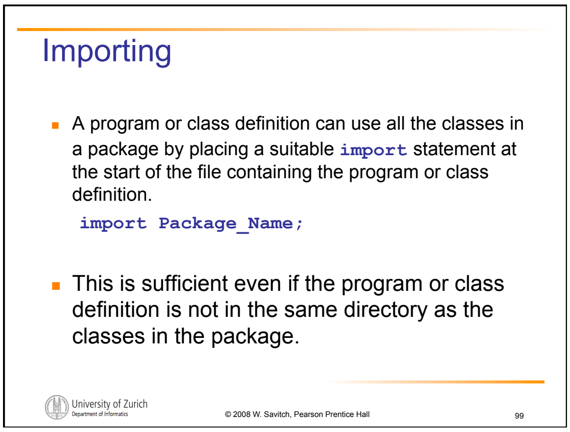


Directories

- *Directories* are called *folders* in some operating systems.
- To understand packages, you need to know about path names for directories, and you need to know how your operating system uses a path variable.
- These are operating system topics and their details depend on the operating system.

University of Zurich
Department of Informatics

© 2008 W. Savitch, Pearson Prentice Hall 98



Importing

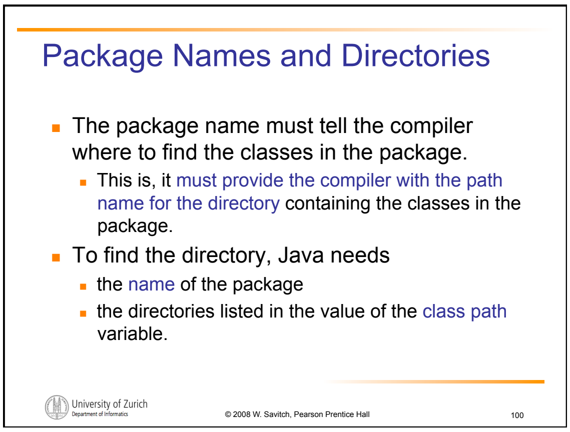
- A program or class definition can use all the classes in a package by placing a suitable `import` statement at the start of the file containing the program or class definition.

```
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.

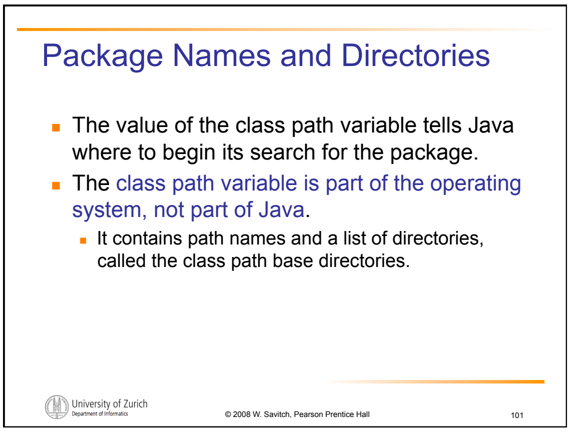
University of Zurich
Department of Informatics

© 2008 W. Savitch, Pearson Prentice Hall 99



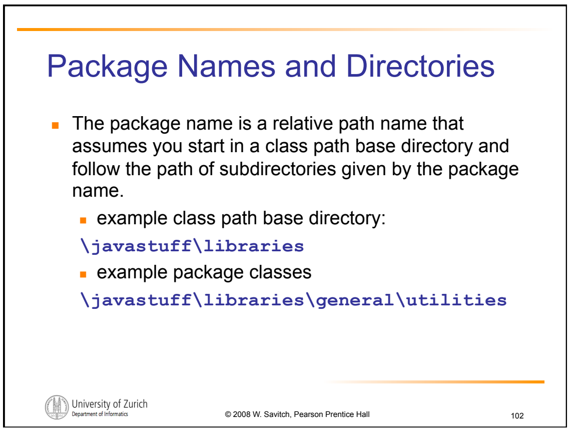
Package Names and Directories

- The package name must tell the compiler where to find the classes in the package.
  - This is, it must provide the compiler with the path name for the directory containing the classes in the package.
- To find the directory, Java needs
  - the name of the package
  - the directories listed in the value of the class path variable.



Package Names and Directories

- 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.



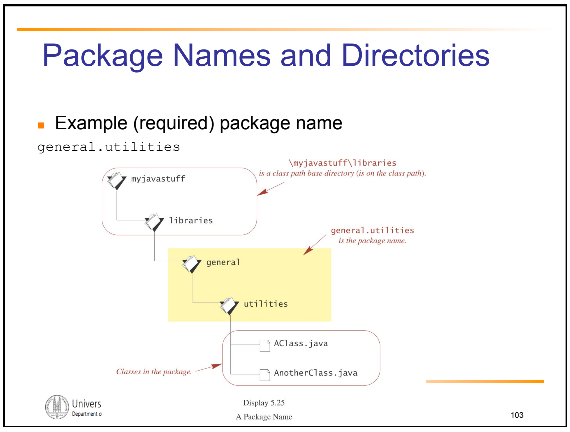
Package Names and Directories

- 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
```



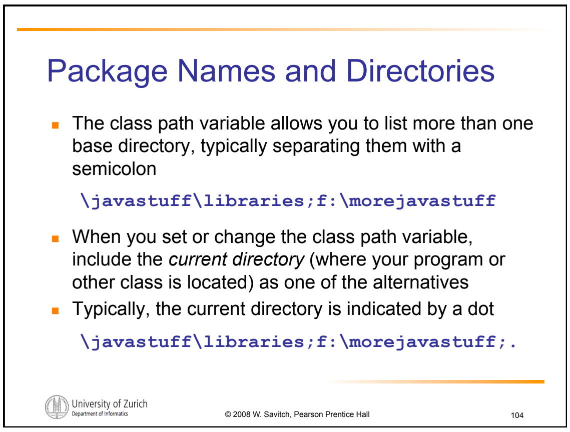
Package Names and Directories

- Example (required) package name

`general.utilities`

Univers
Department o

Display 5.25
A Package Name



Package Names and Directories

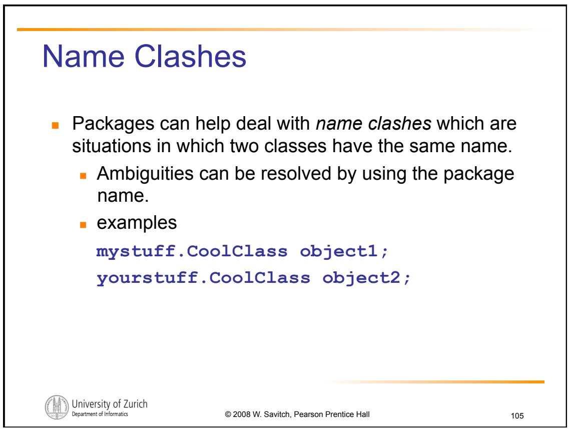
- The class path variable allows you to list more than one base directory, typically separating them with a semicolon

```
\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) as one of the alternatives
- Typically, the current directory is indicated by a dot

```
\javastuff\libraries;f:\morejavastuff;.
```

University of Zurich
Department of Informatics

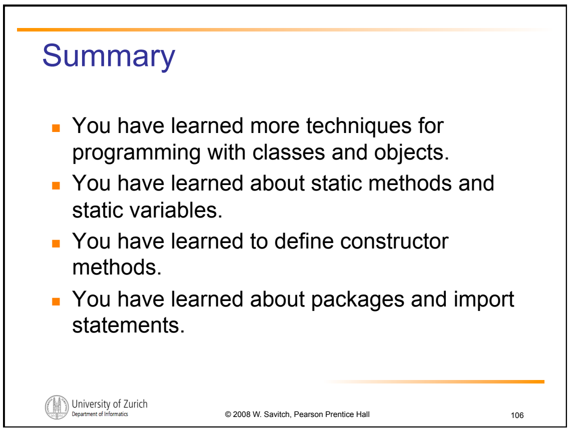


Name Clashes

- 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

```
mystuff.CoolClass object1;
yourstuff.CoolClass object2;
```

University of Zurich
Department of Informatics



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.

University of Zurich
Department of Informatics