Praktische Softwaretechnologie

Lecture 3.

Károly Bósa
(Karoly.Bosa@jku.at)

Research Institute for Symbolic Computation
(RISC)

Class Definition


```
class Point {
    //Fields
    …

    //Methods
    …
}
```

Class Definition


```
public abstract class Point extends SuperClass {
    //Fields
    …

    //Methods
    …
}
```

Class Definition


```
class Point {
    /** the x coordinate */
    double x;

    /** the y coordinate */
    double y;

    //Methods
    …
}
```

Class Definition


```
class Point {
    /** the x coordinate */
    public double x;

    /** the y coordinate */
    public double y;

    //Methods
    …
}
```

Class Definition


```
class Point {
    /** the x coordinate */
    private double x;

    /** the y coordinate */
    private double y;

    //Methods
    …
}
```

Class Definition


```
class Point {
    /** the x coordinate */
    private double x;

    /** the y coordinate */
    private double y;

    /** Return the point's x coordinate */
    public double getX() {
            return x;
    }
}
```

Class Definition


```
class Point {
    /** the x coordinate */
    private double x;

    /** the y coordinate */
    private double y;

    /** Return the point's x coordinate */
    public double getX() {
            return x;
    }

    /** Set the point's x coordinate */
    public void setX(double newX) {
            x = newX;
    }
}
```

Class Definition


```
class Point {
    /** the x coordinate */
    public double x;

    /** the y coordinate */
    public double y;

    /** Test whether another point is equal to this */
    public boolean eqPoint(Point other) {
            return (x == other.x) && (y == other.y);
    }
}
```

Class Definition


```
class Point {
    /** the x coordinate */
    public double x;

    /** the y coordinate */
    public double y;

    /** Test whether another point is equal to this */
    public boolean eqPoint(Point other) {
            return (this.x == other.x) && (this.y == other.y);
    }
}
```

Class Definition


```
class Point {
    /** the x coordinate */
    private double x;

    /** the y coordinate */
    private double y;

    /** Test whether another point is equal to this */
    public boolean eqPoint(Point other) {
            return (this.x == other.getX()) &&
                    (this.y == other.getY());
    }
}
```

Class Definition


```
class Point {
    /** the x coordinate */
    private double x;

    /** the y coordinate */
    private double y;

    /** Test whether another point is equal to this */
    public boolean eqPoint(Point other) {
            return (this.getX() == other.getX()) &&
                    (this.getY() == other.getY());
    }
}
```

Class Definition


```
class Point {
    /** the x coordinate */
    private double x;

    /** the y coordinate */
    private double y;

    /** Return a String representation of this point */
    public String toString() {
            return "(" + x + "," + y + ")";
    }
}
```

Class Definition


```
class Point {
    /** the x coordinate */
    private double x;

    /** the y coordinate */
    private double y;

    /** Return a String representation of this point */
    public String toString() {
            return "(" + x + "," + y + ")";
    }
}
...
Point p = …;
String s = "The point: " + p;
System.out.println(s);
…
```

Class Definition


```
class Point {
    /** the x coordinate */
    private double x;

    /** the y coordinate */
    private double y;

    /** Create a new point object with given coordinates */
    public Point(double x, double y) {
            this.x = x;
            this.y = y;
    }
}
```

Class Definition


```
class Point {
    /** the x coordinate */
    private double x;

    /** the y coordinate */
    private double y;

    /** Create a new point object with given coordinates */
    public Point(double x, double y) {
            this.x = x;
            this.y = y;
    }
}

…
Point p = new Point(10.0, 20.0);
…
```

Class Definition


```
class Point {
    /** the x coordinate */
    private double x;

    /** the y coordinate */
    private double y;

    /** Create a new point object with default values */
    public Point() {
            this(0.0, 0.0)          //Located at the origin
    }
}

…
Point p = new Point();
…
```

Usage

Karoly.Bosa@jku.at

```
class UsePoint {
   public static void comparePoints(Point p1, Point p2) {
      System.out.print("Points " + p1 + " and " + p2 + " are ");
      if (p1.eqPoint(p2)) {
         System.out.println("equal.");
      } else {
         System.out.println("unequal.");
      }
   }

   public static void main(String[] args) {
      Point p1 = new Point();
      Point p2 = new Point(1.0,2.0);
      comparePoints(p1,p2);
      p1.setX(1.0);
      p1.setY(2.0);
      comparePoints(p1,p2);
   }
}
```

Sub Classes


```
class WeightedPoint extends Point {

   private double weight;

}
```

Sub Classes


```
class WeightedPoint extends Point {

   private double weight;

   public WeightedPoint(double x, double y, double w) {
      super(x,y);
      weight = w;
   }
}
```

Sub Classes


```
class WeightedPoint extends Point {

   private double weight;

   public WeightedPoint(double x, double y, double w) {
      super(x,y);
      weight = w;
   }

   public WeightedPoint(double x, double y) {
      this(x, y, 1.0);
   }
}
```

Sub Classes


```
class WeightedPoint extends Point {

   private double weight;

   public WeightedPoint(double x, double y, double w) {
      super(x,y);
      weight = w;
   }

   public WeightedPoint(double x, double y) {
      this(x, y, 1.0);
   }

   public WeightedPoint() {
      weight = 1.0;
   }
}
```

Sub Classes


```
class WeightedPoint extends Point {

   private double weight;

   public double getWeight() {
      return weight;
   }

   public void setWeight(double weight) {
      this.weight = weight;
   }
}
```

Sub Classes

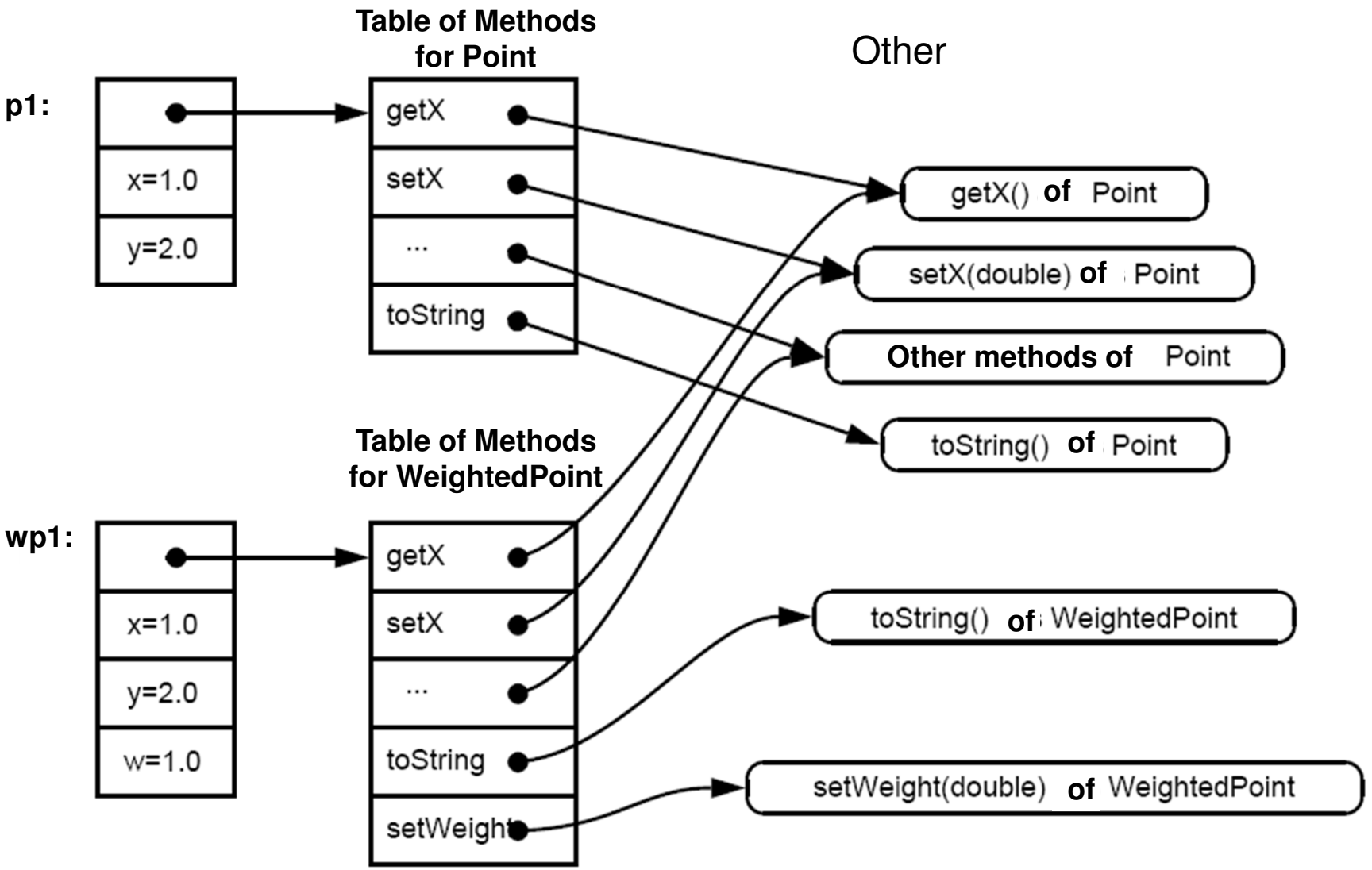

```
class WeightedPoint extends Point {

   private double weight;

   public boolean eqWeightedPoint(WeightedPoint other) {
      return eqPoint(other) && (this.weight == other.weight);
   }
}
```

Sub Classes


```
class WeightedPoint extends Point {

   private double weight;

   public boolean eqWeightedPoint(WeightedPoint other) {
      return eqPoint(other) && (this.weight == other.weight);
   }

   public String toString() {
      return super.toString() + "*" + weight;
   }
}
```

➔ Overriding the inherited method toString()

Implementation



Implementation 2

Call of Dynamic Methods (from the WeightedPoint):

- getX() = super.getX() = this.getX()
- getWeight() = this.getWeight()
- toString() = this.toString()
- super.toString()

Overloading of Methods

Overriding:

- A method of a sub class overrides the method of its super class
- The real calling sequence of methods will be determined only in run time
- It always depends on the dynamic type of the caller

Overloading:

- More methods (maybe belonging to the same class) have the same name
- The identity of the called method is determined by the Compiler
- It depend on the number and the types of the arguments

Overloading of Methods 2

For instance:

```
public class PrintStream {
    ...
    println(boolean b) {...}
    println(char c) {...}
    println(int i) {...}
    ...
    println(Object o) {...}
    ...
}
```

Overloading of Methods 2

For instance:

```
public class PrintStream {
    ...
    println(boolean b) {...}
    println(char c) {...}
    println(int i) {...}
    ...
    println(Object o) {...}
    ...
}

public class GermanPrintStream extends PrintStream {

    println(boolean b) {...}
```

Static Fields

Fields marked with *word* ***static*** is shared among all instances/objects of a class.

```
class A {
        public static int x;
}

A objectA = …;
```

Access:

```
        objectA.x = objectA.x + 1;
```

Or (better)

```
        A.x = A.x + 1;
```

Static Methods

Methods marked with word ***static***:

- They can access only static fields
- They can be called without creating any object/instance of a class
- Consequently ***this*** cannot be usage within such methods

```
class A {
   public static initClass();
}

A objectA = …;
```

Access:

```
objectA.initClass();
```

Or (better)

```
A.initClass();
```

Packages

Each java class is part of a “Package”

The file HelloWorld.java is in the actual directory:

```
class HelloWorld {
  …
}
```

“default package”

Usage:

```
new HelloWorld(…)
```

Packages 2

Each java class is part of a “Package”

The file PrintStream.java in the directiory java\io.

```
package java.io;

class PrintStream {
  …
}
```

Usage:

```
new java.io.PrintStream(…);
```

Packages 3

Using the short class name PrintStream is possible instead of java.io.PrintStream:

- In another class of package java.io.
- After import (it must be given at the beginning of the class file between the words “package” and “class”).

```
import java.io.PrintStream;
```

- After the import of the whole package.

```
import java.io.*;
```

- Automatic import for package java.lang.

Package Example


```
import java.io.*;

public class HelloWorld2 {

   public static void main (String[] args) throws IOException {
      PrintWriter out  = new PrintWriter(new FileWriter("hello.txt"));
      out.println("Hello World!");
      out.close();
   }
}
```

Package Names

Note: The hierarchy is not always important

After internet domain names, e.g.:
Bundesministerium für Finanzen: http://www.bmf.gv.at
Package Namen: at.gv.bmf….

But,
Standard Libraries (refer to each other): java.lang, java.io, java.util, …

Recommended to Read

Reading and completing the course material from the online Java Tutorial:

http://java.sun.com/docs/books/tutorial/java/index.html

- Classes and Objects
- especially the part about the “Nested Classes”
- “enum types”

Exercise 4

Deadline: 09.04.2014

Karoly.Bosa@jku.at

Implement a class Stack (LIFO) with the following methods:

- public Stack(int n)
  Create a stack with place for n objects

- public void push(String s)
  Inserts “s” into the stack

- public String pop()
  Retrieves and removes the top element of the stack

- public boolean isEmpty()
  Checks whether the stack is empty

- public String toString()
  Returns with all elements of the stack in a String

Exercise 4 (continuation) Deadline: 09.04.2014

Karoly.Bosa@jku.at

Write a main program in another class, which:

- creates 2 stacks
- pushes some strings into the first stack
- then prints out the content of the first stack on the screen
- in a loop takes out (pop) all the element of the first stack and inserts (push) them into the second stack
- Finally prints out the content of the second stack on the screen.