合肥學院
HEFEI UNIVERSITY

OOP with Java
15. Instance Methods

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

Outline

website

Introduction

- We have learned to make our own classes and provide them with instance variables

Introduction

- We have learned to make our own classes and provide them with instance variables
- But we can also provide them with instance methods

Introduction

- We have learned to make our own classes and provide them with instance variables
- But we can also provide them with instance methods
- These are methods working directly on the instance variables using the `this` statement

Listing: A class for complex numbers, i.e., $\mathbb{C}$ and math operators

```
/** a class representing a complex number z∈ℂin rectangular form z=α+βi, now also implementing the math routines */
public class ComplexNumber {
  double realPart; // the real part α of the complex number
  double imaginaryPart; // the imaginary part β

  /** create a new complex number setting the real part, leaving imaginary part 0 */
  public ComplexNumber(final double _realPart) { this.realPart = _realPart;  }

  /** create a new complex number setting both real and imaginary part */
  public ComplexNumber(final double _realPart, final double _imaginaryPart) {
    this(_realPart); // first invoke the one-parameter constructor setting real part
    this.imaginaryPart = _imaginaryPart;
  }

  /** return new complex number with result {@code this + x} */
  public ComplexNumber add(ComplexNumber x) {
    return new ComplexNumber((this.realPart + x.realPart), (this.imaginaryPart + x.imaginaryPart)); // (αx+αy) + (βx+βy)i
  }

  /** return new complex number with result {@code this - x} */
  public ComplexNumber subtract(ComplexNumber x) {
    return new ComplexNumber((this.realPart - x.realPart), (this.imaginaryPart - x.imaginaryPart)); // (αx−αy) + (βx−βy)i
  }

  /** return new complex number with result {@code this*x} */
  public ComplexNumber multiply(ComplexNumber x) {
    double a1 = this.realPart, b1 = this.imaginaryPart, a2 = x.realPart, b2 = x.imaginaryPart;
    return new ComplexNumber(((a1 * a2) - (b1 * b2)), ((a1 * b2) + (b1 * a2))); // (αxαy−βxβy) + (αxβy+βxαy)i
  }

  /** return new complex number with {@code this/x} */
  public ComplexNumber divide(ComplexNumber x) {
    double a1 = this.realPart, b1 = this.imaginaryPart, a2 = x.realPart, b2 = x.imaginaryPart;
    return new ComplexNumber((((a1 * a2) + (b1 * b2)) / ((a2 * a2) + (b2 * b2))), // αxαy+βxβy / α²y+β²y
        (((a2 * b1) - (b2 * a1)) / ((a2 * a2) + (b2 * b2)))); // αyβx−βyαx / α²y+β²y i
  }

  /** print this complex number */
  public void println() {
    System.out.print(this.realPart);      System.out.print("␣+␣"); //$NON-NLS-1$
    System.out.print(this.imaginaryPart); System.out.println('i');
  }
}
```

A class implementing mathematical operations over $\mathbb{C}$

Listing: A class implementing mathematical operations over $\mathbb{C}$

```
/** testing the new complex number class: almost the same as in last lesson, just with instance methods and
    toString */
public class ComplexNumberTest {

  /** The main routine
   *  @param args
   *          we ignore this parameter */
  public static final void main(String[] args) {
    ComplexNumber a, b, res;

    a = new ComplexNumber(20d); // instantiate a = 20 + 0i
    a.println(); // print 20 + 0i

    b = new ComplexNumber(1d, -2d); // create b = 1 - 2i
    b.println(); // print 1 + 2i

    (res = a.multiply(b)).println(); // set res = a * b = 20 * (1 - 2i) = 20 - 40i and print

    res = a.multiply(b).subtract(b); // we can chain methods by applying subtract to the result of multiply
    res.println(); // print the result of the above computation: (20 * (1 - 2i)) - (1 - 2i) = 19 * (1 - 2i) = 19 - 38i

    res.divide(b).println();// print the result of (19-38i)/(1-2i) = 19 = 19 + 0i, but don't store it:
                            // created object becomes immediately subject to GC (same as a.multiply(b) above)

    res.multiply(res).divide(res.multiply(// (19-38i)*(19-38i) / (19-38i)*(1-i) = 28.5 - 9.5i
        new ComplexNumber(1d, -1d))).println();
  }
}
```

- Objects can have a "special" function called `toString()` taking no parameter and returning a `String`

- Objects can have a "special" function called `toString()` taking no parameter and returning a `String`
- This function is called whenever the object is used in an expression where a `String` is expected

- Objects can have a "special" function called `toString()` taking no parameter and returning a `String`
- This function is called whenever the object is used in an expression where a `String` is expected
- E.g., in a `String` concatenation expression, or in `System.out.println(...)`

Person class with toString

Listing: A Person class with toString Method

```
/** A class representing a person with constructor and toString method. */
public class PersonWithToString {
  /** the family name of the person */
  String familyName;
  /** the given name of the person */
  String givenName;

  /** create a person record and set its name */
  PersonWithToString(String _familyName, String _givenName) {
    this.familyName = _familyName;
    this.givenName = _givenName;
  }

  /** return a string representation of this person record */
  public String toString() {
    return this.givenName + '␣' + this.familyName;
  }

  /** The main routine
   *  @param args
   *           we ignore this parameter */
  public static final void main(String[] args) {
    PersonWithToString weise = new PersonWithToString("Weise", "Thomas"); //$NON-NLS-1$//$NON-NLS-2$
    System.out.println(weise.toString()); // print a string representing the weise object
    System.out.println(weise); // the same as above: a String is expected, weise.toString is  called

    System.out.println(new PersonWithToString("Chan", "Jacky")); //$NON-NLS-1$//$NON-NLS-2$
  }
}
```

`static` vs. instance methods

- In Lesson 10: *Static Methods*, we learned about `static` methods

`static` vs. instance methods

- In Lesson 10: *Static Methods*, we learned about `static` methods
- `static` methods are different from instance methods

`static` vs. instance methods

- In Lesson 10: *Static Methods*, we learned about `static` methods
- `static` methods are different from instance methods
- An instance method is always invoked in the context of one specific object

`static` vs. instance methods

- In Lesson 10: *Static Methods*, we learned about `static` methods
- `static` methods are different from instance methods
- An instance method is always invoked in the context of one specific object
- It can access the member variables of this object via the `this` keyword

`static` vs. instance methods

- In Lesson 10: *Static Methods*, we learned about `static` methods
- `static` methods are different from instance methods
- An instance method is always invoked in the context of one specific object
- It can access the member variables of this object via the `this` keyword
- It can invoke other instance methods on the same instance also using `this.methodName(...)`

`static` vs. instance methods

- In Lesson 10: *Static Methods*, we learned about `static` methods
- `static` methods are different from instance methods
- An instance method is always invoked in the context of one specific object
- It can access the member variables of this object via the `this` keyword
- It can invoke other instance methods on the same instance also using `this.methodName(...)`
- A static method is invoked without an object context, it cannot use `this`

Using `static` and instance methods

Listing: A program using both `static` and instance methods

```
/** A class representing a person with constructor and toString method. */
public class PersonWithToStringAndStatic {
  /** the family name of the person */
  String familyName;
  /** the given name of the person */
  String givenName;

  /** create a person record and set its name */
  PersonWithToStringAndStatic(String _familyName, String _givenName) {
    this.familyName = _familyName;
    this.givenName = _givenName;
  }

  /** return a string representation of this person record */
  public String toString() {
    return this.givenName + '␣' + this.familyName;
  }

  /** return a string representation of a person record */
  public static String toString(PersonWithToStringAndStatic person) {
    return person.givenName + '␣' + person.familyName;
  }

  /** The main routine
   *  @param args
   *          we ignore this parameter */
  public static final void main(String[] args) {
    PersonWithToStringAndStatic weise = new PersonWithToStringAndStatic("Weise", "Thomas"); //$NON-NLS-1$//$NON-NLS-2$
    System.out.println(weise.toString()); // print a string representing the weise object, obtained from instance method toString()
    System.out.println(weise); // the same as above: a String is expected, weise.toString is  called
    System.out.println(toString(weise)); // invoke static method toString and print its result

    System.out.println(new PersonWithToStringAndStatic("Chan", "Jacky")); // using instance method toString //$NON-NLS-1$//$NON-NLS-2$
    System.out.println(toString(new PersonWithToStringAndStatic("Chan", "Jacky"))); // using static toString //$NON-NLS-1$//$NON-NLS-2$
  }
}
```

Summary

- We have learned what instance methods are.
- With them, we can directly work on the fields (instance variables) of an object.
- The special function `String toString()` can be implemented to return a text representation of an object.
- `String toString()` is always called when the object is used in a `String`-typed expression.

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog