合肥學院
HEFEI UNIVERSITY

OOP with Java
10. Static 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

1. Introduction
2. Method Definitions and Implementations
3. Structuring Programs: Methods in Different Classes
4. Recursion
5. Summary

website

(Static) Methods

- Sometimes, a program uses the same code, but at *different places* (so we cannot use loops)

(Static) Methods

- Sometimes, a program uses the same code, but at *different places* (so we cannot use loops)
- Having multiple copies of the same code is a very bad software design

(Static) Methods

- Sometimes, a program uses the same code, but at *different places* (so we cannot use loops)
- Having multiple copies of the same code is a very bad software design:
  - if the code needs to be changed, all copies need to be changed

(Static) Methods

- Sometimes, a program uses the same code, but at *different places* (so we cannot use loops)
- Having multiple copies of the same code is a very bad software design:
  - if the code needs to be changed, all copies need to be changed
  - if there is an error, there are multiple errors that need to be fixed
- We want one only copy of the code and "invoke" it from different places

(Static) Methods

- Sometimes, a program uses the same code, but at *different places* (so we cannot use loops)
- Having multiple copies of the same code is a very bad software design:
  - if the code needs to be changed, all copies need to be changed
  - if there is an error, there are multiple errors that need to be fixed
- We want one only copy of the code and "invoke" it from different places
- This can be done by putting it into a method

(Static) Methods

- Sometimes, a program uses the same code, but at *different places* (so we cannot use loops)
- Having multiple copies of the same code is a very bad software design:
  - if the code needs to be changed, all copies need to be changed
  - if there is an error, there are multiple errors that need to be fixed
- We want one only copy of the code and "invoke" it from different places
- This can be done by putting it into a method
- You already know two methods

(Static) Methods

- Sometimes, a program uses the same code, but at *different places* (so we cannot use loops)
- Having multiple copies of the same code is a very bad software design:
  - if the code needs to be changed, all copies need to be changed
  - if there is an error, there are multiple errors that need to be fixed
- We want one only copy of the code and "invoke" it from different places
- This can be done by putting it into a method
- You already know two methods:
  - the `public static final void main(...` method of your programs

(Static) Methods

- Sometimes, a program uses the same code, but at *different places* (so we cannot use loops)
- Having multiple copies of the same code is a very bad software design:
  - if the code needs to be changed, all copies need to be changed
  - if there is an error, there are multiple errors that need to be fixed
- We want one only copy of the code and "invoke" it from different places
- This can be done by putting it into a method
- You already know two methods:
  - the `public static final void main(...` method of your programs
  - things like `System.out.println(...`

(Static) Methods

- Sometimes, a program uses the same code, but at *different places* (so we cannot use loops)
- Having multiple copies of the same code is a very bad software design:
  - if the code needs to be changed, all copies need to be changed
  - if there is an error, there are multiple errors that need to be fixed
- We want one only copy of the code and "invoke" it from different places
- This can be done by putting it into a method
- You already know two methods:
  - the `public static final void main(...` method of your programs
  - things like `System.out.println(...`
- (`static` methods are methods that belong to a class, there are also other types of methods, but we will ignore this for now)

Method Signature

- A method has

Method Signature

- A method has
  - a name

Method Signature

- A method has
  - a name
  - a list of parameters, where each parameter has a type and a name (similar to variable declarations)

Method Signature

- A method has
  - a name
  - a list of parameters, where each parameter has a type and a name (similar to variable declarations)
  - a return type (such methods are called functions) or `void` if it returns nothing (such methods are called procedures)

Method Signature

- A method has
  - a name
  - a list of parameters, where each parameter has a type and a name (similar to variable declarations)
  - a return type (such methods are called functions) or `void` if it returns nothing (such methods are called procedures)
- Example: `static double position(double x0, double v0, double t)` declares

Method Signature

- A method has
  - a name
  - a list of parameters, where each parameter has a type and a name (similar to variable declarations)
  - a return type (such methods are called functions) or `void` if it returns nothing (such methods are called procedures)
- Example: `static double position(double x0, double v0, double t)` declares
  - a static method named "position"

Method Signature

- A method has
  - a name
  - a list of parameters, where each parameter has a type and a name (similar to variable declarations)
  - a return type (such methods are called functions) or `void` if it returns nothing (such methods are called procedures)
- Example: `static double position(double x0, double v0, double t)` declares
  - a static method named "position", which
  - return a `double` as its result

Method Signature

- A method has
  - a name
  - a list of parameters, where each parameter has a type and a name (similar to variable declarations)
  - a return type (such methods are called functions) or `void` if it returns nothing (such methods are called procedures)
- Example: `static double position(double x0, double v0, double t)` declares
  - a static method named "position", which
  - return a `double` as its result and
  - takes three parameters

Method Signature

- A method has
  - a name
  - a list of parameters, where each parameter has a type and a name (similar to variable declarations)
  - a return type (such methods are called functions) or `void` if it returns nothing (such methods are called procedures)
- Example: `static double position(double x0, double v0, double t)` declares
  - a static method named "position", which
  - return a `double` as its result and
  - takes three parameters:
    1. a `double` value called `x0`

Method Signature

- A method has
  - a name
  - a list of parameters, where each parameter has a type and a name (similar to variable declarations)
  - a return type (such methods are called functions) or `void` if it returns nothing (such methods are called procedures)
- Example: `static double position(double x0, double v0, double t)` declares
  - a static method named "position", which
  - return a `double` as its result and
  - takes three parameters:
    1. a `double` value called `x0`,
    2. a `double` value called `v0`

Method Signature

- A method has
  - a name
  - a list of parameters, where each parameter has a type and a name (similar to variable declarations)
  - a return type (such methods are called functions) or `void` if it returns nothing (such methods are called procedures)
- Example: `static double position(double x0, double v0, double t)` declares
  - a static method named "position", which
  - return a `double` as its result and
  - takes three parameters:
    1. a `double` value called `x0`,
    2. a `double` value called `v0`, and
    3. a `double` value called `t`

Calling Methods

- Methods can be called by writing their name and providing the necessary parameters

Calling Methods

- Methods can be called by writing their name and providing the necessary parameters
- `static double position(double x0, double v0, double t)` can be called as `position(0.9d*2, 10d, 0.5d)` which invokes `position`

Calling Methods

- Methods can be called by writing their name and providing the necessary parameters
- `static double position(double x0, double v0, double t)` can be called as `position(0.9d*2, 10d, 0.5d)` which invokes `position` and provides
  1. `x0=1.8d`

Calling Methods

- Methods can be called by writing their name and providing the necessary parameters
- `static double position(double x0, double v0, double t)` can be called as `position(0.9d*2, 10d, 0.5d)` which invokes `position` and provides
  1. `x0=1.8d`,
  2. `v0=10d`

Calling Methods

- Methods can be called by writing their name and providing the necessary parameters
- `static double position(double x0, double v0, double t)` can be called as `position(0.9d*2, 10d, 0.5d)` which invokes `position` and provides
  1. `x0=1.8d`,
  2. `v0=10d`, and
  3. `t=0.5d`

Calling Methods

- Methods can be called by writing their name and providing the necessary parameters
- `static double position(double x0, double v0, double t)` can be called as `position(0.9d*2, 10d, 0.5d)` which invokes `position` and provides
  1. `x0=1.8d`,
  2. `v0=10d`, and
  3. `t=0.5d`
- If a method has a return type `T`, this method can be used as an expression of type `T`

Method Implementation

- The code of the method follows after the signature inside `{..}`

Method Implementation

- The code of the method follows after the signature inside `{..}`, just like in the `main` methods we did so far

Method Implementation

- The code of the method follows after the signature inside `{..}`, just like in the `main` methods we did so far
- Inside the code, you can access the method parameters as if they were local variables

Method Implementation

- The code of the method follows after the signature inside `{..}`, just like in the `main` methods we did so far
- Inside the code, you can access the method parameters as if they were local variables, just like we did with `args` in our `main` methods

Method Implementation

- The code of the method follows after the signature inside `{..}`, just like in the `main` methods we did so far
- Inside the code, you can access the method parameters as if they were local variables, just like we did with `args` in our `main` methods
- If the method has a return type `T`, then its last instruction must be `return <expression of type T>;`

Method Implementation

- The code of the method follows after the signature inside `{..}`, just like in the `main` methods we did so far
- Inside the code, you can access the method parameters as if they were local variables, just like we did with `args` in our `main` methods
- If the method has a return type `T`, then its last instruction must be `return <expression of type T>;`
- Actually, `return` can be called anywhere in the method, if it is called, the method exists

Method Implementation

- The code of the method follows after the signature inside `{..}`, just like in the `main` methods we did so far
- Inside the code, you can access the method parameters as if they were local variables, just like we did with `args` in our `main` methods
- If the method has a return type `T`, then its last instruction must be `return <expression of type T>;`
- Actually, `return` can be called anywhere in the method, if it is called, the method exists
- Methods without return value can also use `return` to exit, but they cannot specify a expression whose result is to be returned.

The Vertical Ball Throw, now as Function

Listing: The Vertical Ball Throw, now as Function

```
/**
 * A ball is thrown vertically upwards into the air by a 1.8m tall person<br/>
 * with velocity 10m/s. Where is it after t = 0, 0.2, ..., 2.2 seconds?<br/>
 * x(t) = x_0 + v_0 * t - 0.5 * g * t^2
 */
public class VerticalBallThrowFunction {

  /** Compute the position of a ball (good style: these comments document
   *                                                what the method does)
   * @param x0 the height of the thrower, i.e., the initial vertical position
   * @param v0 the vertical upward velocity with which the ball is thrown
   * @param t the time at which we want to get the position x(t)
   * @return the position x(t) of the ball at time step t
   */
  static double position(double x0, double v0, double t) {
    return x0 + (v0 * t) - 0.5d * 9.80665d * t * t;
  }

  /** The main routine
   * @param args
   *          we ignore this parameter */
  public static final void main(String[] args) {
    for (int i = 0; i < 12; i++) { // using an integer for counting
      System.out.println(position(1.8d, 10d, 0.2d * i)); // prints the current position
    }
  }
}
```

Listing: Multiplying and Printing Matrices

```java
/** An example program printing and multiplying matrices.  */
public class MultiplyMatrices {

  static void print(double[][] matrix, String name) {
    System.out.println("Matrix␣" + name + ':'); //$NON-NLS-1$
    for (double[] row : matrix) {// fast read-only iteration over matrix rows
      for (double value : row) { // fast read-only iteration of values in row
        System.out.print('␣');
        System.out.print(value);
      }
      System.out.println();
    }
  }

  static double[][] multiply(double[][] a, double[][] b) {
    int aColumns = a[0].length;
    int bColumns = b[0].length;

    double[][] result = new double[a.length][bColumns]; //allocate and initialize all values to 0

    for (int i = 0; i < a.length; i++) { // iterate over the rows of a
      for (int j = 0; j < bColumns; j++) { // iterate over the columns of b
        for (int k = 0; k < aColumns; k++) { // iterate over the columns of A
          result[i][j] += a[i][k] * b[k][j];
        }
      }
    }

    return result; // return result
  }

  /** The main routine
   * @param args
   *          we ignore this parameter */
  public static final void main(String[] args) {
    double[][] a = { { 4d, 3d }, { 2d, 1d } }; // allocate and initialize first matrix
    double[][] b = { { -0.5d, 1.5d }, { 1d, -2d } }; // allocate and initialize second matrix

    print(a, "a");// call a procedure printing a //$NON-NLS-1$
    print(b, "b");// call a procedure printing b //$NON-NLS-1$
    print(multiply(a,b), "a*b");// call a procedure printing the result of the multiplication //$NON-NLS-1$
  }
}
```

Structuring Programs: Methods in Different Classes

- A class can have (almost) arbitrarily many methods.

Structuring Programs: Methods in Different Classes

- A class can have (almost) arbitrarily many methods.
- But if we have many methods in one class, the code gets much harder to understand.

Structuring Programs: Methods in Different Classes

- A class can have (almost) arbitrarily many methods.
- But if we have many methods in one class, the code gets much harder to understand.
- Actually, we can also call methods specified in another class!

Structuring Programs: Methods in Different Classes

- A class can have (almost) arbitrarily many methods.
- But if we have many methods in one class, the code gets much harder to understand.
- Actually, we can also call methods specified in another class!
- In this case, we cannot just use the name of the method, but need to specify "`canonical-name-of-class.name-of-method`" instead

Using Methods from another Class

Listing: Using Methods from another Class: Matrices

```
/** An example program printing and multiplying matrices
 *  using our other program MultiplyMatrices.  */
public class MultiplyMatricesUsingMethodsFromOtherClass {

  /** The main routine
   * @param args
   *          we ignore this parameter */
  public static final void main(String[] args) {
    double[][] a = { { 4d, 3d }, { 2d, 1d } }; // allocate and initialize first matrix
    double[][] b = { { -0.5d, 1.5d }, { 1d, -2d } }; // allocate and initialize second matrix

    MultiplyMatrices.print(a, "a");// call a procedure printing a //$NON-NLS-1$
    MultiplyMatrices.print(b, "b");// call a procedure printing b //$NON-NLS-1$
    MultiplyMatrices.print(MultiplyMatrices.multiply(a,b), "a*b");// multiply and print //$NON-NLS-1$
  }
}
```

Example: The (Java-Provided) Class `Math`

Listing: Using the Static Methods of the (Java-Provided) Class `Math`

```
/** An example program using the methods of java.lang.Math */
public class MathMethods {

  /** The main routine
   * @param args
   *             we ignore this parameter */
  public static final void main(String[] args) {
    System.out.println(Math.exp(Math.sin(6)));
    System.out.println(Math.atan(Math.tan(1)));
    System.out.println(Math.hypot(3, 4));
  }
}
```

Methods of the Same Name

- You can define multiple methods of the same name

Methods of the Same Name

- You can define multiple methods of the same name
- If they are in different classes, they can have the exactly same signature (since we know which one is called because of the prepended class name)

Methods of the Same Name

- You can define multiple methods of the same name
- If they are in different classes, they can have the exactly same signature (since we know which one is called because of the prepended class name)
- If they are in the same class, they need to have different parameter types

Methods of the Same Name

- You can define multiple methods of the same name
- If they are in different classes, they can have the exactly same signature (since we know which one is called because of the prepended class name)
- If they are in the same class, they need to have different parameter types
- You cannot have two methods with the same name and same parameter types in the same class, even if the parameter names are different

Methods of the Same Name

- You can define multiple methods of the same name
- If they are in different classes, they can have the exactly same signature (since we know which one is called because of the prepended class name)
- If they are in the same class, they need to have different parameter types
- You cannot have two methods with the same name and same parameter types in the same class, even if the parameter names are different
- You cannot have two methods with the same name and parameter types, even if the method's return type is different

Example: Two Methods with Same Name

Listing: Two Methods with Same Name but Different Parameters

```
/** An example program specifying two methods of the same name
 *  (but, of course, with different parameters) */
public class MethodsOfSameName {
  // compute lnnumber
  static double log(final double number) {
    return Math.log(number);
  }
  // compute log_base number
  static double log(final double base, final double number) {
    return log(number) / log(base);
  }

  /** The main routine
   * @param args
   *             we ignore this parameter */
  public static final void main(String[] args) {
    System.out.println(log(8d));      // ln8
    System.out.println(log(2d, 8d)); // log_2 8
  }
}
```

Recursion

- We can put arbitrary code inside a method.

Recursion

- We can put arbitrary code inside a method.
- We can also call other methods from within a method (obviously, think `System.out.print`)

Recursion

- We can put arbitrary code inside a method.
- We can also call other methods from within a method (obviously, think `System.out.print`)
- We can also call the method *itself*, which is called *recursion*

Recursion

- We can put arbitrary code inside a method.
- We can also call other methods from within a method (obviously, think `System.out.print`)
- We can also call the method *itself*, which is called *recursion*
- If a method calls itself, we need to make sure that this does not repeat infinitely

Listing: Recursion: The Fibonacci Numbers $F(i) = F(i-1) + F(i-2)$, stopping condition $F(1) = F(2) = 1$

```
/** An example program computing Fibonacci numbers F(n) = F(n-1) + F(n-2) with
 *  F(1) = F(2) = 1 recursively. */
public class FibonacciRecursive {

  /** Recursively compute the ith Fibonacci number
   * @param i the index of the number to compute
   * @return ith Fibonacci number
   */
  static long F(int i) {
    if((i == 1L) || (i == 2L)) {
      return 1L; // take care of cases F(1) and F(2)
    }
    return F(i-1) + F(i-2); // recurse
  }

  /** The main routine
   * @param args
   *          we ignore this parameter */
  public static final void main(String[] args) {
    for(int i = 1; i <= 40; i++){ // print the first 40 Fibonacci numbers
      System.out.print("F("); //$NON-NLS-1$
      System.out.print(i);
      System.out.print(") = "); //$NON-NLS-1$
      System.out.println(F(i));
    }
  }
}
```

Summary

- We have learned what static methods are.
- We have learned how to define them, how to call them, and how to implement them.
- One class can have (almost) arbitrarily many methods.
- We have learned that we can put methods into different classes and call methods from different classes.
- We have even used recursion.

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