OOP with Java

17. Packages and Import

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

- Classes allow us to structure our program code

Introduction

- Classes allow us to structure our program code
- A program is usually a big heap of data structures and algorithms

Introduction

- Classes allow us to structure our program code
- A program is usually a big heap of data structures and algorithms
- We can take the algorithms that work on one specific data structure and put the data structure and algorithm into one single class together

Introduction

- Classes allow us to structure our program code
- A program is usually a big heap of data structures and algorithms
- We can take the algorithms that work on one specific data structure and put the data structure and algorithm into one single class together
- We can find related data structures and algorithms and try to generalize them in order to put their common parts in common super class

Introduction

- Classes allow us to structure our program code
- A program is usually a big heap of data structures and algorithms
- We can take the algorithms that work on one specific data structure and put the data structure and algorithm into one single class together
- We can find related data structures and algorithms and try to generalize them in order to put their common parts in common super class
- This way, we can make the code shorter and structure the program more clearly

Introduction

- But even then, we might end up with a huge mess

Introduction

- But even then, we might end up with a huge mess
- Classes can have dozens of methods

Introduction

- But even then, we might end up with a huge mess
- Classes can have dozens of methods
- There may be hundreds of classes

Introduction

- But even then, we might end up with a huge mess
- Classes can have dozens of methods
- There may be hundreds of classes
- Programs often have 1’000’000 to 10’000’000 lines of code

Introduction

- But even then, we might end up with a huge mess
- Classes can have dozens of methods
- There may be hundreds of classes
- Programs often have 1'000'000 to 10'000'000 lines of code
- We need more structure!

Packages: What are they good for?

- A lot of classes are for entirely unrelated stuff

Packages: What are they good for?

- A lot of classes are for entirely unrelated stuff:
    - some classes represent the “business logic” of the software

Packages: What are they good for?

- A lot of classes are for entirely unrelated stuff:
    - some classes represent the “business logic” of the software
    - others may be for I/O, i.e, reading input and writing output

Packages: What are they good for?

- A lot of classes are for entirely unrelated stuff:
    - some classes represent the “business logic” of the software
    - others may be for I/O, i.e, reading input and writing output
    - others may render a graphical user interface on the screen

Packages: What are they good for?

- A lot of classes are for entirely unrelated stuff:
    - some classes represent the "business logic" of the software
    - others may be for I/O, i.e, reading input and writing output
    - others may render a graphical user interface on the screen
    - yet others may be general algorithms such as sorting

Packages: What are they good for?

- A lot of classes are for entirely unrelated stuff:
  - some classes represent the “business logic” of the software
  - others may be for I/O, i.e, reading input and writing output
  - others may render a graphical user interface on the screen
  - yet others may be general algorithms such as sorting
  - others may represent collections like lists or hash maps used

Packages: What are they good for?

- A lot of classes are for entirely unrelated stuff:
    - some classes represent the “business logic” of the software
    - others may be for I/O, i.e, reading input and writing output
    - others may render a graphical user interface on the screen
    - yet others may be general algorithms such as sorting
    - others may represent collections like lists or hash maps used
- We should put these into different places

Packages: What are they good for?

- A lot of classes are for entirely unrelated stuff:
    - some classes represent the “business logic” of the software
    - others may be for I/O, i.e, reading input and writing output
    - others may render a graphical user interface on the screen
    - yet others may be general algorithms such as sorting
    - others may represent collections like lists or hash maps used
- We should put these into different places
- This is what packages are good for

Packages: What are Packages?

- Packages offer something like a hierarchical file system for classes

Packages: What are Packages?

- Packages offer something like a hierarchical file system for classes:
  - imagine the unix file system with `.` instead of `/`

Packages: What are Packages?

- Packages offer something like a hierarchical file system for classes:
    - imagine the unix file system with `.` instead of `/`
    - imagine the windows file system with `.` instead of \

Packages: What are Packages?

- Packages offer something like a hierarchical file system for classes:
  - imagine the unix file system with `.` instead of `/`
  - imagine the windows file system with `.` instead of \
  - packages then are like directories

Packages: What are Packages?

- Packages offer something like a hierarchical file system for classes:
  - imagine the unix file system with `.` instead of `/`
  - imagine the windows file system with `.` instead of \
  - packages then are like directories
  - actually, they do 1:1 correspond to source folders!

Elaborate Example

- We make a more elaborate example based on our “Person” classes from Lesson 14: *Objects, Instance Variables, and* `new`

Elaborate Example

- We make a more elaborate example based on our "Person" classes from Lesson 14: *Objects, Instance Variables, and* `new`
- In Java, you usually choose a base package which represents your organization

Elaborate Example

- We make a more elaborate example based on our “Person” classes from Lesson 14: *Objects, Instance Variables, and* `new`
- In Java, you usually choose a base package which represents your organization:
    - Usually, this is the server part of the URL of your organization in reverse, so that the structure is globally unique

Elaborate Example

- We make a more elaborate example based on our “Person” classes from Lesson 14: *Objects, Instance Variables, and* `new`
- In Java, you usually choose a base package which represents your organization:
  - Usually, this is the server part of the URL of your organization in reverse, so that the structure is globally unique
  - We thus choose `cn.edu.hfuu.iao`, since `http://iao.hfuu.edu.cn/` is the URL of my institute

Elaborate Example

- We make a more elaborate example based on our "Person" classes from Lesson 14: *Objects, Instance Variables, and* `new`
- In Java, you usually choose a base package which represents your organization:
    - Usually, this is the server part of the URL of your organization in reverse, so that the structure is globally unique
    - We thus choose `cn.edu.hfuu.iao`, since http://iao.hfuu.edu.cn/ is the URL of my institute
    - This means our code's root folder is not just the `src` folder in the Eclipse project, but `src/cn/edu/hfuu/iao`

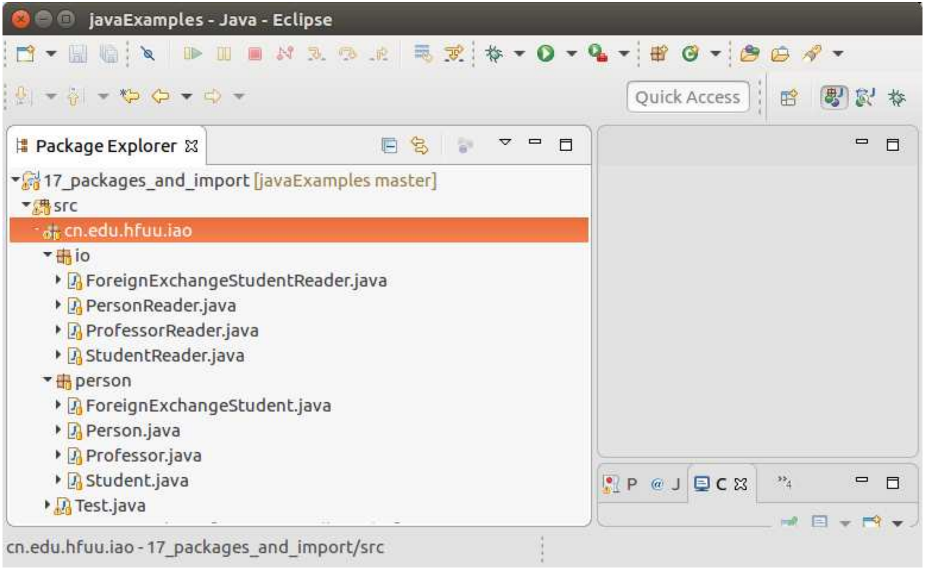
Elaborate Example

- We make a more elaborate example based on our "Person" classes from Lesson 14: *Objects, Instance Variables, and* `new`
- In Java, you usually choose a base package which represents your organization:
  - Usually, this is the server part of the URL of your organization in reverse, so that the structure is globally unique
  - We thus choose `cn.edu.hfuu.iao`, since `http://iao.hfuu.edu.cn/` is the URL of my institute
  - This means our code's root folder is not just the `src` folder in the Eclipse project, but `src/cn/edu/hfuu/iao`
- We then put the "Person" class hierarchy into sub-package `cn.edu.hfuu.iao.person` (equivalent to folder `src/cn/edu/hfuu/person`)

Elaborate Example

- We make a more elaborate example based on our "Person" classes from Lesson 14: *Objects, Instance Variables, and `new`*
- In Java, you usually choose a base package which represents your organization:
  - Usually, this is the server part of the URL of your organization in reverse, so that the structure is globally unique
  - We thus choose `cn.edu.hfuu.iao`, since `http://iao.hfuu.edu.cn/` is the URL of my institute
  - This means our code's root folder is not just the `src` folder in the Eclipse project, but `src/cn/edu/hfuu/iao`
- We then put the "Person" class hierarchy into sub-package `cn.edu.hfuu.iao.person` (equivalent to folder `src/cn/edu/hfuu/person`)
- We add some I/O classes in sub-package `cn.edu.hfuu.iao.io` (equivalent to folder `src/cn/edu/hfuu/io`)

Elaborate Example


javaExamples - Java - Eclipse

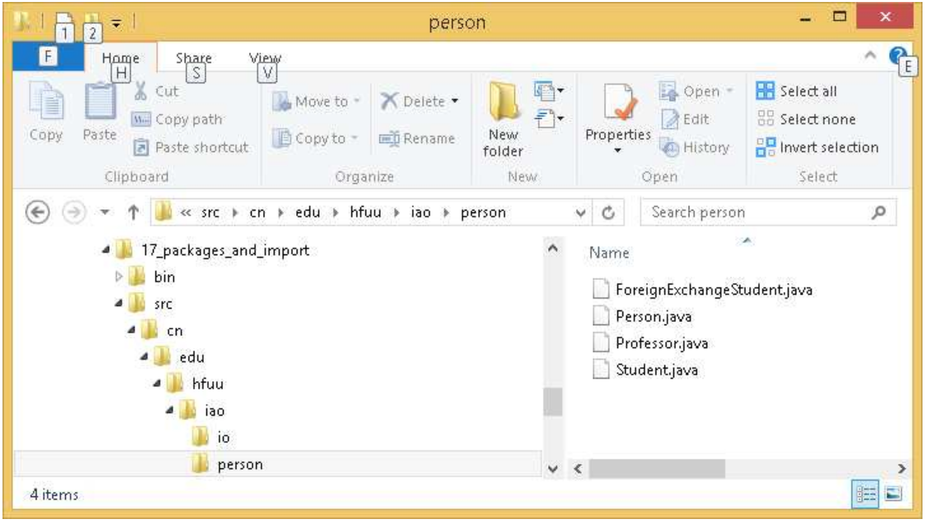
Quick Access

Package Explorer

```
▾ 17_packages_and_import [javaExamples master]
  ▾ src
    cn.edu.hfuu.iao
      ▾ io
        ▸ ForeignExchangeStudentReader.java
        ▸ PersonReader.java
        ▸ ProfessorReader.java
        ▸ StudentReader.java
      ▾ person
        ▸ ForeignExchangeStudent.java
        ▸ Person.java
        ▸ Professor.java
        ▸ Student.java
      ▸ Test.java
```

cn.edu.hfuu.iao - 17_packages_and_import/src

Elaborate Example



« src ▸ cn ▸ edu ▸ hfuu ▸ iao ▸ person

- 17_packages_and_import
  - bin
  - src
    - cn
      - edu
        - hfuu
          - iao
            - io
            - person

ForeignExchangeStudent.java
Person.java
Professor.java
Student.java

Package Declaration in Class

- If I put a class into a package `a.b.c`, then the first line in the Java file must be `package a.b.c;`

Package Declaration in Class

- If I put a class into a package `a.b.c`, then the first line in the Java file must be `package a.b.c;`

Listing: A Person class in package `cn.edu.hfuu.iao.person`

```java
package cn.edu.hfuu.iao.person; // declare the package cn.edu.hfuu.iao.person

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

  /** create a person record and set its name */
  public Person(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;
  }
}
```

Professor Class

Listing: A Professor class in package cn.edu.hfuu.iao.person

```java
package cn.edu.hfuu.iao.person; // declare the package cn.edu.hfuu.iao.person

/** A class representing a professor */
public class Professor extends Person { // class Processor extends class Person
  /** create a person record and set its name */
  public Professor(String _familyName, String _givenName) {
    super(_familyName, _givenName);
  }

  /** return "Prof. " + result of super.toString() = Person.toString() */
  @Override // mark this method explicitly as overridden
  public String toString() {
    return "Prof.␣" + super.toString(); //$NON-NLS-1$
  }

}
```

Student Class

Listing: A Student class in package cn.edu.hfuu.iao.person

```java
package cn.edu.hfuu.iao.person; // declare the package cn.edu.hfuu.iao.person

/** A class representing a student */
public class Student extends Person { // class Student extends class Person
  /** the id of the student */
  String id;

  /** create a student record and set its name and student id */
  public Student(String _familyName, String _givenName, String _id) {
    super(_familyName, _givenName);
    this.id = _id;
  }

  /** return a string representation of this student record */
  @Override // mark this method explicitly as overridden
  public String toString() {
    return "student␣" + super.toString(); //$NON-NLS-1$
  }
}
```

Foreign Exchange Student Class

Listing: A Foreign Exchange Student class in package `cn.edu.hfuu.iao.person`

```java
package cn.edu.hfuu.iao.person; // declare the package cn.edu.hfuu.iao.person

/** A class representing a foreign exchange student */
public class ForeignExchangeStudent extends Student {
  /** the home country of the student */
  String homeCountry; // we add a new field

  /** create a student record and set its name, student id, and home country */
  public ForeignExchangeStudent(String _familyName, String _givenName,
                                String _id,         String country) {
    super(_familyName, _givenName, _id);
    this.homeCountry = country;
  }

  /** override toString() from Person */
  @Override // mark this method explicitly as overridden
  public String toString() {
    return super.toString() + "␣from␣" + this.homeCountry;//$NON-NLS-1$
  }
}
```

Canonical Class Names

- A class `MyClass` is specified doing `class MyClass { ... }`

Canonical Class Names

- A class `MyClass` is specified doing `class MyClass { ... }`
- `MyClass` is the simple name of the class, used as type, used for calling constructors, and for invoking static methods

Canonical Class Names

- A class `MyClass` is specified doing `class MyClass { ... }`
- `MyClass` is the simple name of the class, used as type, used for calling constructors, and for invoking static methods
- If the class is in package `a.b.c`, then its canonical name becomes `a.b.c.MyClass`

Canonical Class Names

- A class `MyClass` is specified doing `class MyClass { ... }`
- `MyClass` is the simple name of the class, used as type, used for calling constructors, and for invoking static methods
- If the class is in package `a.b.c`, then its canonical name becomes `a.b.c.MyClass`
- The canonical name of our new `Person` class is `cn.edu.hfuu.iao.person.Person`

Canonical Class Names

- A class `MyClass` is specified doing `class MyClass { ... }`
- `MyClass` is the simple name of the class, used as type, used for calling constructors, and for invoking static methods
- If the class is in package `a.b.c`, then its canonical name becomes `a.b.c.MyClass`
- The canonical name of our new `Person` class is `cn.edu.hfuu.iao.person.Person`
- You can use classes in different packages by using their canonical name

Using Canonical Class Names: A Person Reader

Listing: Person Reader in package `cn.edu.hfuu.iao.io`

```
package cn.edu.hfuu.iao.io;

/** a class to read a person record from stdin: using canonical class names*/
public class PersonReader {

  /** the constructor */
  public PersonReader(){
  }

  /** Read a person record from stdin. All class names are specified canonically
   * @return the new person record  */
  public cn.edu.hfuu.iao.person.Person read(java.util.Scanner scanner) {
    System.err.println("Enter␣person's␣family␣name:"); //$NON-NLS-1$
    String familyName = scanner.nextLine(); // read a string from scanner
    System.err.println("Enter␣person's␣given␣name:"); //$NON-NLS-1$
    String givenName = scanner.nextLine(); // read a string from scanner

    return new cn.edu.hfuu.iao.person.Person(familyName, givenName);
  }
}
```

`import`

- If a class `A` is in the same package as your class `B`, then you can use it by its simple name

`import`

- If a class `A` is in the same package as your class `B`, then you can use it by its simple name
- Otherwise, you have to use its canonical name

`import`

- If a class `A` is in the same package as your class `B`, then you can use it by its simple name
- Otherwise, you have to use its canonical name
- Always writing canonical names makes the code hard to read

`import`

- If a class `A` is in the same package as your class `B`, then you can use it by its simple name
- Otherwise, you have to use its canonical name
- Always writing canonical names makes the code hard to read
- You can “import” a class `MyClass` by its canonical name `a.b.c.MyClass` into your class by doing `import a.b.c.MyClass`

`import`

- If a class `A` is in the same package as your class `B`, then you can use it by its simple name
- Otherwise, you have to use its canonical name
- Always writing canonical names makes the code hard to read
- You can "import" a class `MyClass` by its canonical name `a.b.c.MyClass` into your class by doing `import a.b.c.MyClass`
- You then can refer to imported classes by their simple name

`import`

- If a class `A` is in the same package as your class `B`, then you can use it by its simple name
- Otherwise, you have to use its canonical name
- Always writing canonical names makes the code hard to read
- You can “import” a class `MyClass` by its canonical name `a.b.c.MyClass` into your class by doing `import a.b.c.MyClass`
- You then can refer to imported classes by their simple name
- This is just a shorthand syntax, the compiled code is identical to using the canonical name

`import`

- If a class `A` is in the same package as your class `B`, then you can use it by its simple name
- Otherwise, you have to use its canonical name
- Always writing canonical names makes the code hard to read
- You can "import" a class `MyClass` by its canonical name `a.b.c.MyClass` into your class by doing `import a.b.c.MyClass`
- You then can refer to imported classes by their simple name
- This is just a shorthand syntax, the compiled code is identical to using the canonical name
- You cannot import two classes with the same simple name and you cannot import a class with the same simple name as your class

Using `import`: A Professor Reader

Listing: Professor Reader in package `cn.edu.hfuu.iao.io`

```java
package cn.edu.hfuu.iao.io;

import java.util.Scanner; // import class Scanner from java.util

//import class Professor from package cn.edu.hfuu.iao.person
import cn.edu.hfuu.iao.person.Professor;

/** a class to read a professor record from stdin*/
public class ProfessorReader extends PersonReader {

  /** the constructor */
  public ProfessorReader(){
  }

  /** read a profesor record from scanner (pointing to stdin)
   * @return the new person record */
  @Override
  public Professor read(Scanner scanner) {
    System.err.println("Enter␣professor's␣family␣name:"); //$NON-NLS-1$
    String familyName = scanner.nextLine(); // read a string from scanner
    System.err.println("Enter␣professor's␣given␣name:"); //$NON-NLS-1$
    String givenName = scanner.nextLine(); // read a string from scanner

    return new Professor(familyName, givenName);
  }
}
```

Using `import`: A Student Reader

Listing: Student Reader in package cn.edu.hfuu.iao.io

```java
package cn.edu.hfuu.iao.io;

import java.util.Scanner; // import class Scanner from java.util
//import class Student from package cn.edu.hfuu.iao.person
import cn.edu.hfuu.iao.person.Student;

/** a class to read a student record from stdin*/
public class StudentReader extends PersonReader {

  /** the constructor */
  public StudentReader(){
  }

  /** read a student record from scanner (pointing to stdin)
   * @return the new person record */
  @Override
  public Student read(Scanner scanner) {
    System.err.println("Enter␣student's␣family␣name:"); //$NON-NLS-1$
    String familyName = scanner.nextLine();   // read a string from scanner
    System.err.println("Enter␣student's␣given␣name:"); //$NON-NLS-1$
    String givenName = scanner.nextLine(); // read a string from scanner
    System.err.println("Enter␣student's␣ID:"); //$NON-NLS-1$
    String id = scanner.nextLine(); // read a string from scanner

    return new Student(familyName, givenName, id);
  }
}
```

Using `import`: A Foreign Exchange Student Reader

Listing: Foreign Exchange Student Reader in package cn.edu.hfuu.iao.io

```java
package cn.edu.hfuu.iao.io;

import java.util.Scanner; // import class Scanner from java.util
//import class ForeignExchangeStudent from package cn.edu.hfuu.iao.person
import cn.edu.hfuu.iao.person.ForeignExchangeStudent;

/** a class to read a student record from stdin*/
public class ForeignExchangeStudentReader extends PersonReader {

  /** the constructor */
  public ForeignExchangeStudentReader(){
  }

  /** read a foreign exchange student record from scanner (pointing to stdin)
   * @return the new person record */
  @Override
  public ForeignExchangeStudent read(Scanner scanner) {
    System.err.println("Enter␣exchange␣student's␣family␣name:"); //$NON-NLS-1$
    String familyName = scanner.nextLine(); // read a string from scanner
    System.err.println("Enter␣exchange␣student's␣given␣name:"); //$NON-NLS-1$
    String givenName = scanner.nextLine(); // read a string from scanner
    System.err.println("Enter␣exchange␣student's␣ID:"); //$NON-NLS-1$
    String id = scanner.nextLine(); // read a string from scanner
    System.err.println("Enter␣exchange␣student's␣home␣country:"); //$NON-NLS-1$
    String country = scanner.nextLine(); // read a string from scanner

    return new ForeignExchangeStudent(familyName, givenName, id, country);
  }
}
```

A Main Class Reading and Printing Person Records

Listing: Main class in package `cn.edu.hfuu.iao`

```java
package cn.edu.hfuu.iao;

import java.util.Scanner; // import class Scanner from the java.util package

import cn.edu.hfuu.iao.io.ForeignExchangeStudentReader; // import all needed data structure
import cn.edu.hfuu.iao.io.PersonReader;                 // and I/O classes from our sub-packages
import cn.edu.hfuu.iao.io.ProfessorReader;
import cn.edu.hfuu.iao.io.StudentReader;
import cn.edu.hfuu.iao.person.Person;

/** testing our package structure */
public class Main {
  /** The main routine reading person records of a certain type from stdin
   *  @param args we ignore this parameter */
  public static void main(String[] args) {
    PersonReader reader;
    Scanner scanner = new Scanner(System.in); // create a structured text reader

    System.err.println("Do␣you␣want␣to␣read␣(p)rofessors,␣(s)tudents,␣or␣(e)change␣students:␣"); //$NON-NLS-1$

    switch (scanner.nextLine().charAt(0)) { // check the first character entered by the user
      case 'p': { reader = new ProfessorReader(); break; } // p -> read professors
      case 's': { reader = new StudentReader();   break; } // s -> read students
      default: {  reader = new ForeignExchangeStudentReader(); break; } // otherwise: read exchange students
    }

    for (;;) { // loop forever, see loop condition at bottom of loop
      Person person = reader.read(scanner); // use the person read to read a person
      System.out.println("You␣entered:␣" + person); // print person.toString //$NON-NLS-1$
      System.out.println("Type␣enter␣to␣continue,␣Ctrl-D␣to␣exit."); //$NON-NLS-1$
      if (scanner.hasNextLine()) { // if user pressed enter
        scanner.nextLine();          // we read the line and continue
        continue;                    // and do another iteration
      }                              // if she instead pressed Ctrl-D or stdin ends, there is
      return;                        // no next line and we exit the main routine
    }
  }
}
```

Summary

- We have learned about packages:
  - which are something like a folder structure (with `.` instead of `/`) to arrange our code and
  - actually correspond to folders
- They allow us to cleanly organize even large projects.
- By using the server part of the URL (with `.` instead of `/`) of our organization as root package, we can achieve globally unique canonical class names.
- This allows us to mix our code with code from arbitrary other sources.
- Canonical class names have the form "packagename.simpleClassName", where simpleClassName is the name we specify after the `class` keyword.
- We can refer to classes in other packages using their canonical name
- We can `import` classes via their canonical name and then refer to them using their simple name

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