



OOP with Java

3. The Eclipse Integrated Developer Environment

Thomas Weise · 汤卫思

twaise@hfu.edu.cn · <http://iao.hfu.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

- 1 Introduction to Eclipse
- 2 Installation
- 3 A First Eclipse Project
- 4 Summary



website



- one of the most popular integrated developer environments (IDEs) for Java

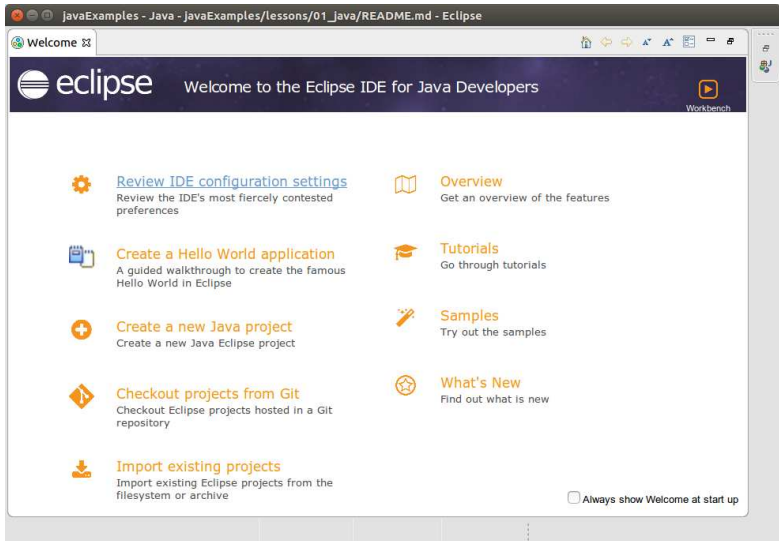


- one of the most popular integrated developer environments (IDEs) for Java
- syntax highlighting, online-compilation, refactoring, . . . : very comfortable editor



- one of the most popular integrated developer environments (IDEs) for Java
- syntax highlighting, online-compilation, refactoring, . . . : very comfortable editor
- comes with lots of tools (which we will talk about later, see, e.g., Lesson 13: *Debugging*, Lesson 27: *Testing with JUnit*, and Lesson 30: *Building with Maven*) integrated

- go to <http://www.eclipse.org>
- click “Download”
- select Eclipse package (currently “Eclipse Neon” and “Download 64 bit”)
- download the package
- unpack it into a folder of your liking
- inside that folder, you can find an executable named `eclipse` or `eclipse.exe` → run it



- So this below was our first program. We now want to edit and run it in Eclipse.

Listing: The "Hello World" Program.

```
/**
 * A simple program just printing "Hello World!" on the screen.<br/>
 * Execute using Eclipse: 1) right-click "HelloWorld.java", 2) click "Run
 * As", 3) click "Java Application"
 */
public class HelloWorld {

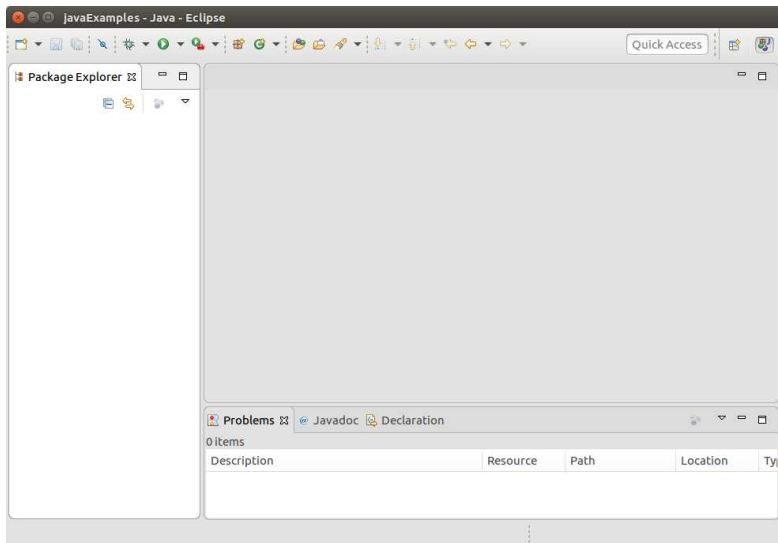
    /** The main routine
     * @param args
     *      we ignore this parameter for now */
    public static final void main(final String[] args) {
        System.out.println("Hello World!"); // print "Hello World!" to console
        //$NON-NLS-1$
    }
}
```


- We now want to try our “Hello World!” example in Eclipse

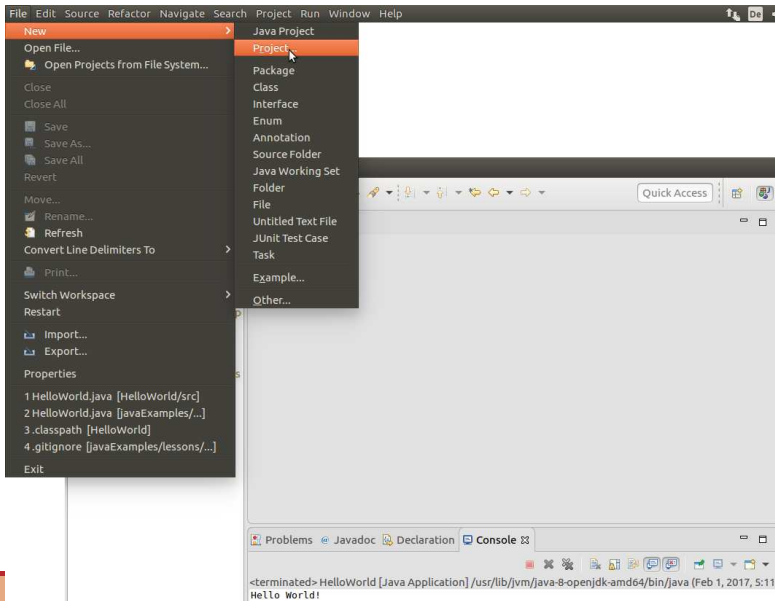
- We now want to try our “Hello World!” example in Eclipse
- In Eclipse, we always work on **Projects**.

- We now want to try our “Hello World!” example in Eclipse
- In Eclipse, we always work on **Projects**.
- A project may contain multiple source code files and resources.

- We now want to try our “Hello World!” example in Eclipse
- In Eclipse, we always work on **Projects**.
- A project may contain multiple source code files and resources.
- For this, we first need to create a new **Java Project**



Hello World



New Java Project

Create a Java Project
Create a Java project in the workspace or in an external location.

Project name:

☒ Use default location
Location: [Browse...](#)

JRE

☒ Use an execution environment JRE: [Configure JREs...](#)

☐ Use a project specific JRE:

☐ Use default JRE (currently 'java-8-openjdk-amd64')

Project layout

☐ Use project folder as root for sources and class files

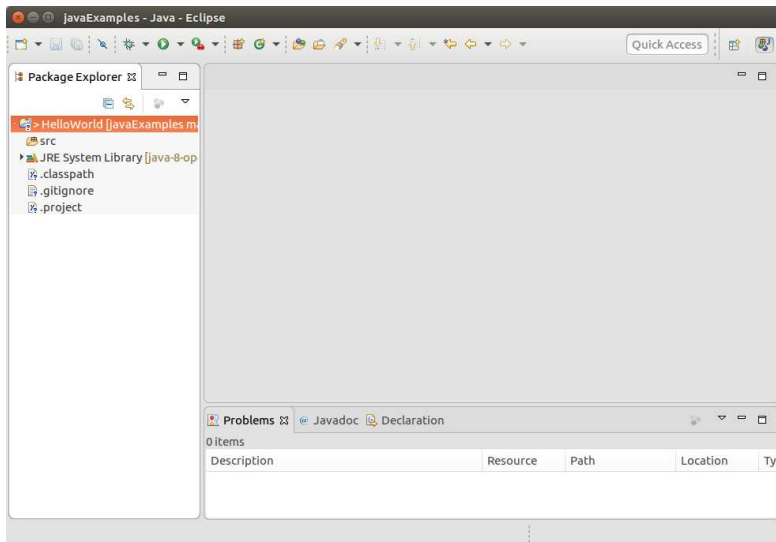
☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

☐ Add project to working sets [New...](#)

Working sets: [Select...](#)

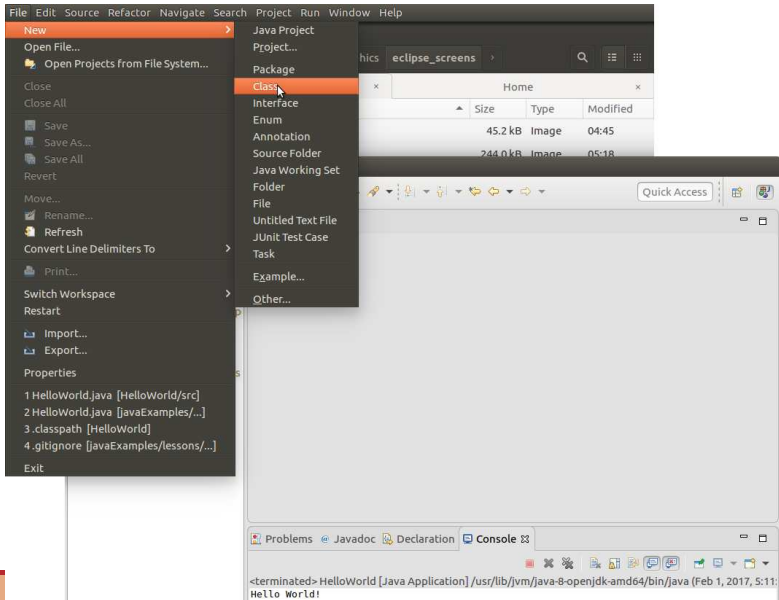
[?](#) [< Back](#) [Next >](#) [Cancel](#) [Finish](#)

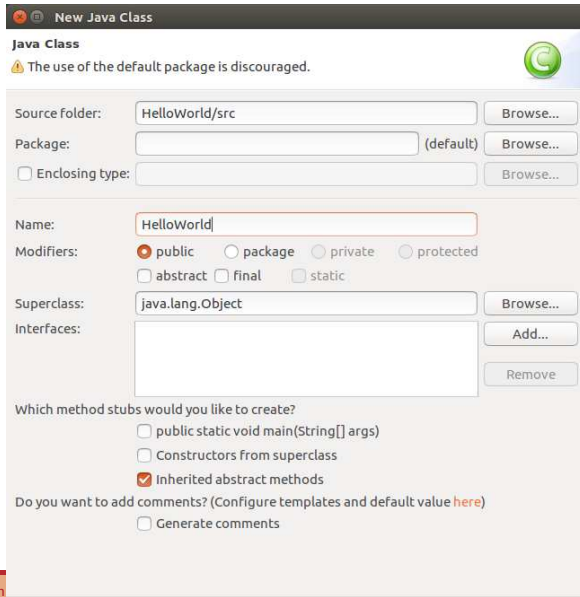


- We now want to try our “Hello World!” example in Eclipse
- In Eclipse, we always work on **Projects**.
- A project may contain multiple source code files and resources.
- For this, we first need to create a new **Java Project**
- OK, we now have an empty project.

- We now want to try our “Hello World!” example in Eclipse
- In Eclipse, we always work on **Projects**.
- A project may contain multiple source code files and resources.
- For this, we first need to create a new **Java Project**
- OK, we now have an empty project.
- We need to insert a new Java class


Hello World








New Java Class

Java Class

 The use of the default package is discouraged.


Source folder: HelloWorld/src 



Package: (default) 

☐ Enclosing type: 

Name: HelloWorld

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object 

Interfaces:  

Which method stubs would you like to create?

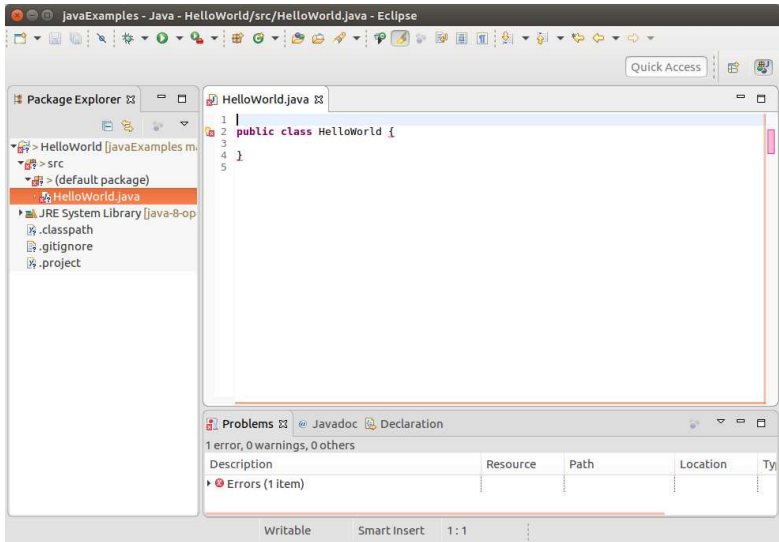
☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

A screenshot of the Eclipse IDE interface. The title bar reads "javaExamples - Java - HelloWorld/src/HelloWorld.java - Eclipse". The Package Explorer on the left shows a project named "HelloWorld" with a sub-package "src" containing "HelloWorld.java". The main editor window displays the code for "HelloWorld.java":

```
1 |  
2 | public class HelloWorld {  
3 |  
4 | }  
5 |
```

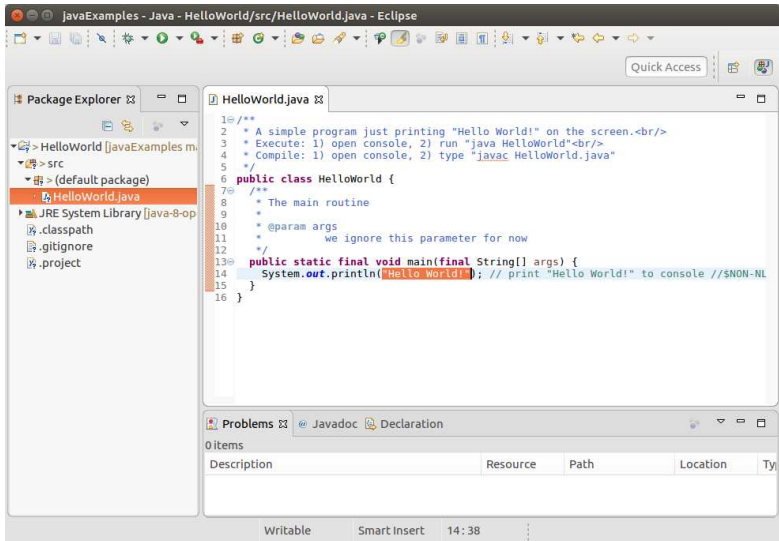
The Problems view at the bottom shows "1 error, 0 warnings, 0 others". Below this, a table lists the error details.

Description	Resource	Path	Location	Type
▶ Errors (1 item)				

The status bar at the bottom indicates "Writable", "Smart Insert", and "1:1".

- We now want to try our “Hello World!” example in Eclipse
- In Eclipse, we always work on **Projects**.
- A project may contain multiple source code files and resources.
- For this, we first need to create a new **Java Project**
- OK, we now have an empty project.
- We need to insert a new Java class
- We have now an empty `HelloWorld.java`

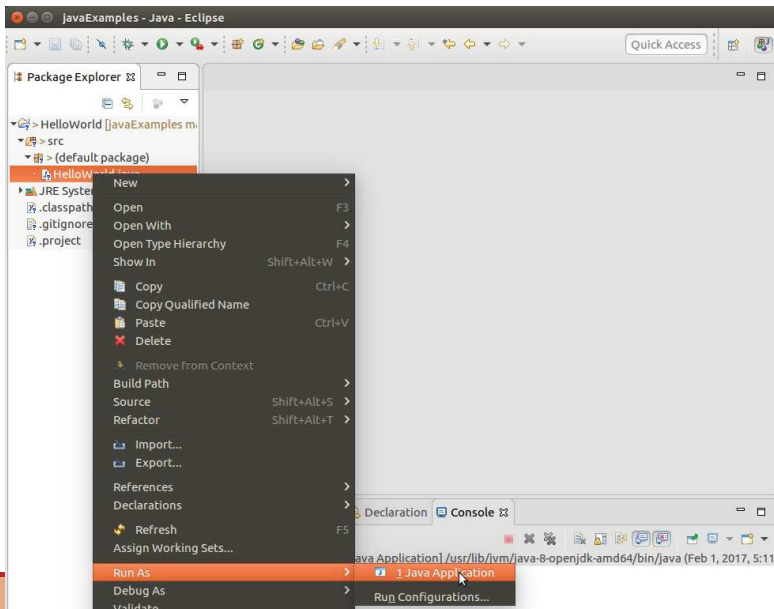
- We now want to try our “Hello World!” example in Eclipse
- In Eclipse, we always work on **Projects**.
- A project may contain multiple source code files and resources.
- For this, we first need to create a new **Java Project**
- OK, we now have an empty project.
- We need to insert a new Java class
- We have now an empty `HelloWorld.java`
- Let's fill in the code

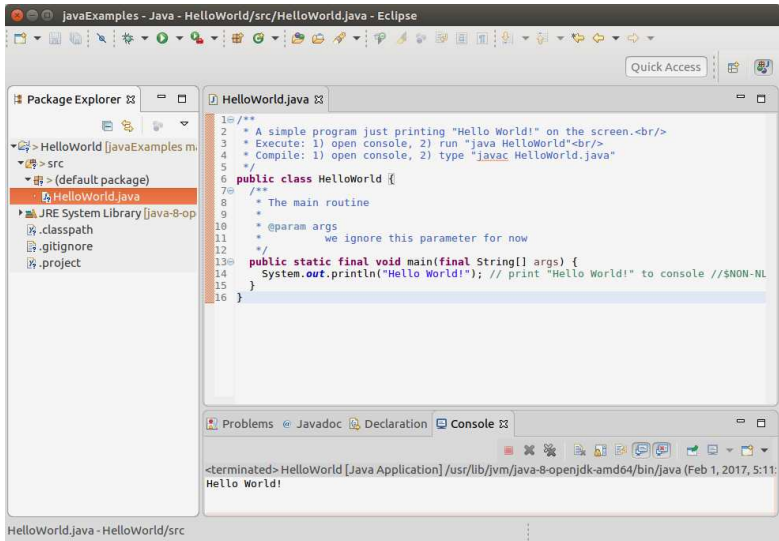
A screenshot of the Eclipse IDE interface. The title bar reads "javaExamples - Java - HelloWorld/src/HelloWorld.java - Eclipse". The Package Explorer on the left shows a project named "javaExamples" with a sub-package "src" containing "HelloWorld.java". The main editor displays the source code of "HelloWorld.java", which includes a class definition and a main method. The Problems view at the bottom shows "0 items". The status bar at the very bottom indicates "Writable", "Smart Insert", and the time "14:38".

```
1 /**
2  * A simple program just printing "Hello World!" on the screen.<br/>
3  * Execute: 1) open console, 2) run "java HelloWorld"<br/>
4  * Compile: 1) open console, 2) type "javac HelloWorld.java"
5  */
6 public class HelloWorld {
7     /**
8      * The main routine
9      *
10     * @param args
11     *     we ignore this parameter for now
12     */
13     public static final void main(final String[] args) {
14         System.out.println("Hello World!"); // print "Hello World!" to console //$NON-NLS
15     }
16 }
```


- We now want to try our “Hello World!” example in Eclipse
- In Eclipse, we always work on **Projects**.
- A project may contain multiple source code files and resources.
- For this, we first need to create a new **Java Project**
- OK, we now have an empty project.
- We need to insert a new Java class
- We have now an empty `HelloWorld.java`
- Let's fill in the code
- And run out program (which will automatically be compiled for us)

Hello World



A screenshot of the Eclipse IDE interface. The title bar reads "javaExamples - Java - HelloWorld/src/HelloWorld.java - Eclipse". The Package Explorer on the left shows a project named "javaExamples" with a sub-package "src" containing "HelloWorld.java". The main editor displays the source code of "HelloWorld.java", which includes a multi-line comment describing the program and its execution steps, followed by a Java class definition. The Console at the bottom shows the output "Hello World!".

```
1 /**  
2  * A simple program just printing "Hello World!" on the screen.<br/>  
3  * Execute: 1) open console, 2) run "java HelloWorld"<br/>  
4  * Compile: 1) open console, 2) type "javac HelloWorld.java"  
5  */  
6 public class HelloWorld {  
7     /**  
8      * The main routine  
9      *  
10     * @param args  
11     *     we ignore this parameter for now  
12     */  
13     public static final void main(final String[] args) {  
14         System.out.println("Hello World!"); // print "Hello World!" to console //$NON-NLS  
15     }  
16 }
```

<terminated> HelloWorld [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (Feb 1, 2017, 5:11:)
Hello World!

- We now want to try our “Hello World!” example in Eclipse
- In Eclipse, we always work on **Projects**.
- A project may contain multiple source code files and resources.
- For this, we first need to create a new **Java Project**
- OK, we now have an empty project.
- We need to insert a new Java class
- We have now an empty `HelloWorld.java`
- Let's fill in the code
- And run out program (which will automatically be compiled for us)
- Yay... everything worked well

- By default, Eclipse separates the source code `*.java` files from the compiled class files `*.class`

- By default, Eclipse separates the source code `*.java` files from the compiled class files `*.class`:
 - source code goes into the sub-folder `src`

- By default, Eclipse separates the source code `*.java` files from the compiled class files `*.class`:
 - source code goes into the sub-folder `src`
 - compiled files go into the sub-folder `bin`

- By default, Eclipse separates the source code `*.java` files from the compiled class files `*.class` :
 - source code goes into the sub-folder `src`
 - compiled files go into the sub-folder `bin`
- Eclipse creates two more files, `.project` and `.classpath` (sometimes also a `.gitignore`) in the root folder of the project... .. Let's ignore them for now

- Like most IDEs, Eclipse has a wide set of tools that make a programmer more productive

- Like most IDEs, Eclipse has a wide set of tools that make a programmer more productive, including:
 - ① syntax highlighting
 - ② refactoring
 - ③ compilation of code while you type it
 - ④ auto-completion
 - ⑤ debugger
 - ⑥ several other tools that we will use, such as Maven, git, JUnit. . .

- Like most IDEs, Eclipse has a wide set of tools that make a programmer more productive, including:
 - ① syntax highlighting
 - ② refactoring
 - ③ compilation of code while you type it
 - ④ auto-completion
 - ⑤ debugger
 - ⑥ several other tools that we will use, such as Maven, git, JUnit. . .
- In software engineering, we always use IDEs

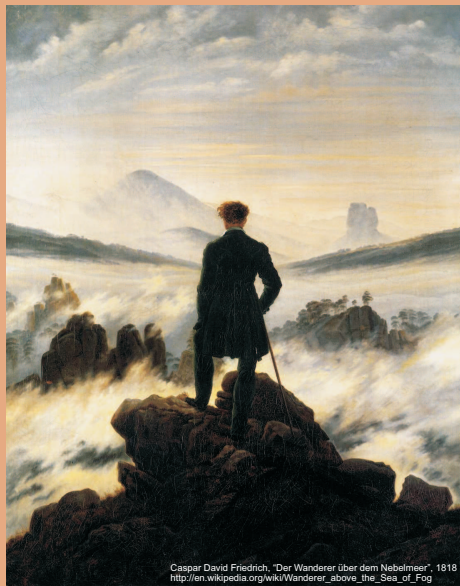
- ① We have learned what an IDE like Eclipse is.
- ② We have downloaded and installed Eclipse.
- ③ We have created a first Eclipse project.
- ④ We have compiled and executed a program with Eclipse.
- ⑤ We have learned about the basic file structure of Eclipse projects.

谢谢

Thank you

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